Industrial Control System Coffee Pot

An Impromptu Compilation of Notes Regarding the Creation of My ICS Testbed

Cameron Williams

A document created partially for the fulfillment of
Masters of Cybersecurity

Department of Computer Science
University of Idaho
5 May 2022

Note from the author: This text was compiled somewhat last-minute from a collection of Word documents that were thrown together throughout the school year as I assembled my project. As such, the bulk of this document is assembled chronologically with mostly minor edits and corrections. Please forgive any errors you come across. The LaTeX source for this document can be found at https://github.com/kwilliamscameron/ICS-Coffeepot.

Contents

1 Problem Statement & Approach

Attacks against Industrial Control Systems are a growing problem [MKK+21] [Lan13] [JCK+17] [Dra22]. There is a shortage of talent in the cybersecurity space to combat these types of threats [See22]. As such, efforts should be made to teach about ICS technologies in cybersecurity curriculum.

Large scale industrial systems are too complex, too expensive, or too industry specific for newcomers or professionals to fully grasp. The goal of this project is to make cyber-physical systems research accessible and fun by using a coffee maker as an affordable, small-scale model of an industrial system. To this end, a coffee maker modified to allow for control over a network via a PLC.

2 Project Materials

The following table shows all the tools and components I used in constrcuting the ICS Coffeepot test bed. Note that the total cost does not necessarily reflect the total money spent since some of the tools and equipment were already on hand either through myself or within RADICL-IF. Additionally, you'll notice some duplicate equipment due to some mistakes made during design and assembly. Read on to find out more about these problems. In particular, I'd point out Sections 5.1 and 5.7 for details on duplicate coffee pot purchases and an extra CLICK PLC module purchase.

Materials	Count	Unit Cost	Cost
Mr. Coffee BVMC-EVX Series	1	$25.00	$25.00
Soldering Iron	1	$50.00	$50.00
Electrical Solder	1	$8.79	$8.79
Electrical Tape	1	$5.29	$5.29
Assorted 16AWG Wire	1	$28.95	$28.95
Raspberry Pi 4B 8GB Variant	3	$109.99	$329.97
Raspberry Pi Cluster Case	1	$19.99	$19.99
32 GB SD Card	3	$7.99	$23.97
CLICK Standard Ethernet PLC	1	$168.00	$168.00
CLICK AC Power Supply	1	$44.50	$44.50
Linksys WRT AC1900 Router (or equiv. OpenWRT)	1	$229.00	$229.00
5 Port Dumb Switch	2	$18.98	$37.96
KVM HDMI/USB Switch	1	$34.99	$34.99
Keyboard	1	$6.99	$6.99
Mouse	1	$9.99	$6.99
Surge Suppressor	1	$24.99	$24.99
HDMI to DVI Adapter	1	$7.87	$7.87
Micro HDMI to HDMI Adapter	3	$7.99	$23.97
Type-K Thermocouple	1	$18.50	$18.50
CLICK PLC Analog Input Module	1	$165.00	$165.00
Utility Cart	1	$89.99	$89.99
CLICK Discrete Output Module (Sourcing)	1	$47.00	$47.00
USB to Ethernet Adapter	1	$13.77	$13.77
24 V Relay Modules	1	$6.89	$6.89
USB Wall Charger	1	$29.99	$29.99
USB Type A to Type C Charging Cables	2	$6.99	$13.98
Black+Decker Coffeepot	1	$19.99	$19.99
Total			$1,482.33

3 Explanation of Materials

This section provides a brief explanation of the choice of materials for my ICS coffee pot project. This project is heavily inspired and informed by Matthew Kirkland's thesis which he wrote about his CacTiE water testbed [Kir20]. The design of my project has changed somewhat since I presented about it in CS 531, so the following list will seek to explain the changes I have made since then in addition to any of the larger purchases that may require more explanation

- CLICK Ethernet Standard PLC, Power Supply, and 16 AWG wire

  In my original design, I was not going to include any real PLCs, but have their functionality entirely virtualized by using Arduinos and/or Raspberry Pis to simulate them since OpenPLC supports their use. However, after discussing further with Dr. Haney, I decided that getting a CLICK PLC [Dir22] such as the one Matt used in CacTiE would be highly beneficial to the realism of my project and my educational experience in constructing it [Kir20]. As such, I intend to purchase at least one CLICK Ethernet PLC. There are many variants in this product line which has made selection somewhat confusing. At time of writing, the best option seems to be the C0-11DD1E-D model (Note from the future: this was a slightly incorrect choice for my purposes; see Section 5.7. This is a CLICK Ethernet Standard PLC with discrete outputs that are sinking and have 5V in their possible output range. This is important since the circuits in the coffee maker I am using implement 5V logic. Using this PLC will also require the purchase of a CLICK AC power supply (model C0-01AC) [Dir22] and 16 AWG wires are necessary to connect things to the PLCs I/O. I needed these pieces very urgently for my final project in CS 543 (in addition to my master's project), so I ordered them myself and will request reimbursement.

- 3 x Raspberry Pi 4Bs, cluster case, and 32GB micro SD cards

  In my original design conception, I intended to use only one Raspberry Pi as a stand in for the PLC using OpenPLC. With this design, the rest of the networking and workstation elements would be included via RADICL-IF. However, after more extensive discussion with Dr. Haney I have decided to more closely follow Matt's design [Kir20] such that my implementation is not dependent on RADICL-IF (though it could later be integrated into RADICL-IF at no additional cost). In doing this, I require more Raspberry Pis so that I can implement the various network zones and workstations as Matt did with CacTiE. These will each require an additional microSD card. After discussing with Dr. Haney, we decided that three Raspberry Pi 4B 8GB models would be appropriate. However, after searching online I have found extreme shortages and high prices. I have therefore selected the 4GB model as a cost saving measure and for the purposes of availability. Purchasing a miniature tower case to contain Raspberry Pis also seems prudent to keep things organized and cool, so I have put it on the parts list.

- Type-K Thermocouple and Thermocouple Analog Input Module

  One possible design extensions to my project is to add a more complex input element to my system in the form of a temperature sensor. Dr. Haney suggested I order these parts sooner rather than later and I found some useful references for determining the appropriate CLICK PLC module (model C0-04THM) and sensor (model THMK-D08L04-01) [Aut] [Dir22].

- Linksys WRT AC3200ACM Router and 2 x 5 port dumb switches

  Similar to CacTiE, I require a router that supports OpenWRT so that I can implement the various network zones (Control Network, Supervisory Network, and Enterprise Network) by using the vLAN feature included in OpenWRT. The switch will be used in achieving this as well.

- KVM Switch, Surge Suppressor, HDMI to DVI Adapter, Micro HDMI to HDMI Adapter, and Utility Cart

  The purpose of these components are somewhat self-explanatory, so I've lumped their explanation together. The KVM switch will allow for using one set of keyboard, monitor, and mouse across the three Raspberry Pis. I plan on using a keyboard, monitor, and mouse from RADICL-IF since there are many spares there. The monitors support DVI only, however, so I will need an HDMI to DVI adapter to connect the HDMI output on the KVM switch to the monitor. Additionally, the Raspberry Pis only have micro HDMI output, so I need three micro HDMI to HDMI cables.

Naturally, everything will need to be powered, making the purchase of a surge suppressing power strip necessary. Finally, after discussing with Dr. Haney, we decided that a cart would be useful since I intend to leave my project behind in RADICL-IF.

4 Testbed Organization & Design

4.1 ICS Layers

There are several architectural models for dividing the hardware of an industrial network into various levels/layers. Among these are the Purdue Enterprise Reference Architecture (PERA), the subsequent ISA-95 that it inspired (also called the IEC 62264 series), and later ISA-99 (renumbered to the ISO/IEC 62443 series) [KL15]. Since my coffeepot testbed is heavily inspired by Matt Kirkland's CacTiE testbed that largely followed the ANSI/ISA-99 definition of ICS levels, I will label and manage my ICS levels likewise [Kir20]. A brief discussion of the ISA99 levels relevant to my implementation follows.

- Level 3 – Enterprise Network: Composed of (who would have guessed) enterprise services found on a typical network. The most common of these would be a Windows Active Directory Domain Control to provide Role-based Access Control to the industrial network through the firewall. Knapp & Langill note that this is not at all best practice – the industrial network should have its own separate domain controller – but it is nonetheless a fairly common [KL15]. A data historian that records values measured and output by the control network devices (e.g. PLCs) may be present or backed up here.
- Level 2 – Supervisory Network: Composed of systems for overseeing the control network and the industrial process as a whole. Includes devices such as an engineering workstation which may be running software to provide a Human Machine Interface (HMI) for process engineers to interact with the PLCs and the process. This is another location where a data historian machine might reside.
- Level 1 – Control Network: Composed of devices such as Programmable Logic Controllers (PLCs), Remote Terminal Units (RTUs), Intelligent Electronic Devices (IEDs) that manage individual industrial process zones by interacting with sensors and actuators. In this instance, a process zone is one stage of an industrial process such as the chlorination of water at a water treatment plant [Kir20].
- Level 0 – I/O Network: Composed of endpoint sensor and actuation devices controlled by devices in the control network (e.g. PLCs). Examples of this include heating elements (actuators) and thermometers (sensors).

4.2 The Layers with respect to the Testbed

- Level 3 – Enterprise Network: One of the three Raspberry Pis will serve as an Enterprise Workstation. This may be the most appropriate location to implement the client side of the Application Gateway interface I envisioned. This does not necessarily reflect best practices in actual industrial networking, but this is how the coffee pot will work.
- Level 2 – Supervisory Network: One of the Raspberry Pis will reside in this zone and run the ScadaBR software [Sca22]. This zone would typically receive information from multiple control devices belong to multiple zones, but for the purposes of this testbed it will only be connected to one. Matt configured ScadaBR to "provide alters, watch set points, host a web-based HMI, and provide log aggregation" [Kir20]. I will likely do the same. He noted that the ScadaBR project had compatibility issues with Rasbian, so I may have to change the OS on this Pi. This also may be the best location to implement the application gateway server that converts incoming HTCPCP requests into Modbus commands that are sent to the PLC.
- Level 1 – Control Network: The Automation Direct CLICK PLC will reside in this zone. Matt's implementation had a separate HMI directly connected to the PLC via a serial port, but I have forgone this since the coffee pot essentially has its own HMI in the form of the control panel attached to it. In this way, the Control Zone and Field Device are somewhat intermingled since the HMI and heating element are closely coupled together in the form of the modified coffee maker.
- Level 0 – Field Devices: For the purposes of this project, the field devices are the heating element embedded in the modified coffee maker and the type-K thermocouple temperature sensor I intend to add to the system if all goes well.

4.3 Application Gateway

To facilitate the usage of the Hyper Text Coffee Pot Control Protocol (HTCPCP) [Mas98] as intended from the original conception of this project, an application gateway will be developed and deployed (most likely in Level 2 with an accompanying client program residing in Level 3) in the testbed. The application will be developed in Python and utilize the Pymodbus [Rip22] library to provide Modbus capabilities. The Python sockets

4.4 The 3rd Raspberry Pi

I originally thought that Matt's testbed used one Raspberry Pi in each zone, but upon reviewing his thesis it is not clear. In his control network, the diagram shows the HMI and the PLC. While the HMI could be implemented in software on a Rasbperry Pi, this would be redundant since this will already be done in the Supervisory Network using ScadaBR. While this would make sense in a more realistic industrial network with multiple control networks from multiple process zones each with their own PLCs and HMIs reporting to the Supervisory system with a large supervisory software-based HMI, it doesn't make sense in my testbed that has only a single PLC. As such, another software-based HMI in my Control Network is not a good idea. Reviewing Matts' virtual design, he implements a pfSense firewall between the Level 3 Enterprise Network and the rest of the industrial network. This is likely the best use of the third Raspberry Pi despite it not being explicitly stated in the design of his physical CacTiE testbed.

4.5 Diagram of Testbed Layout

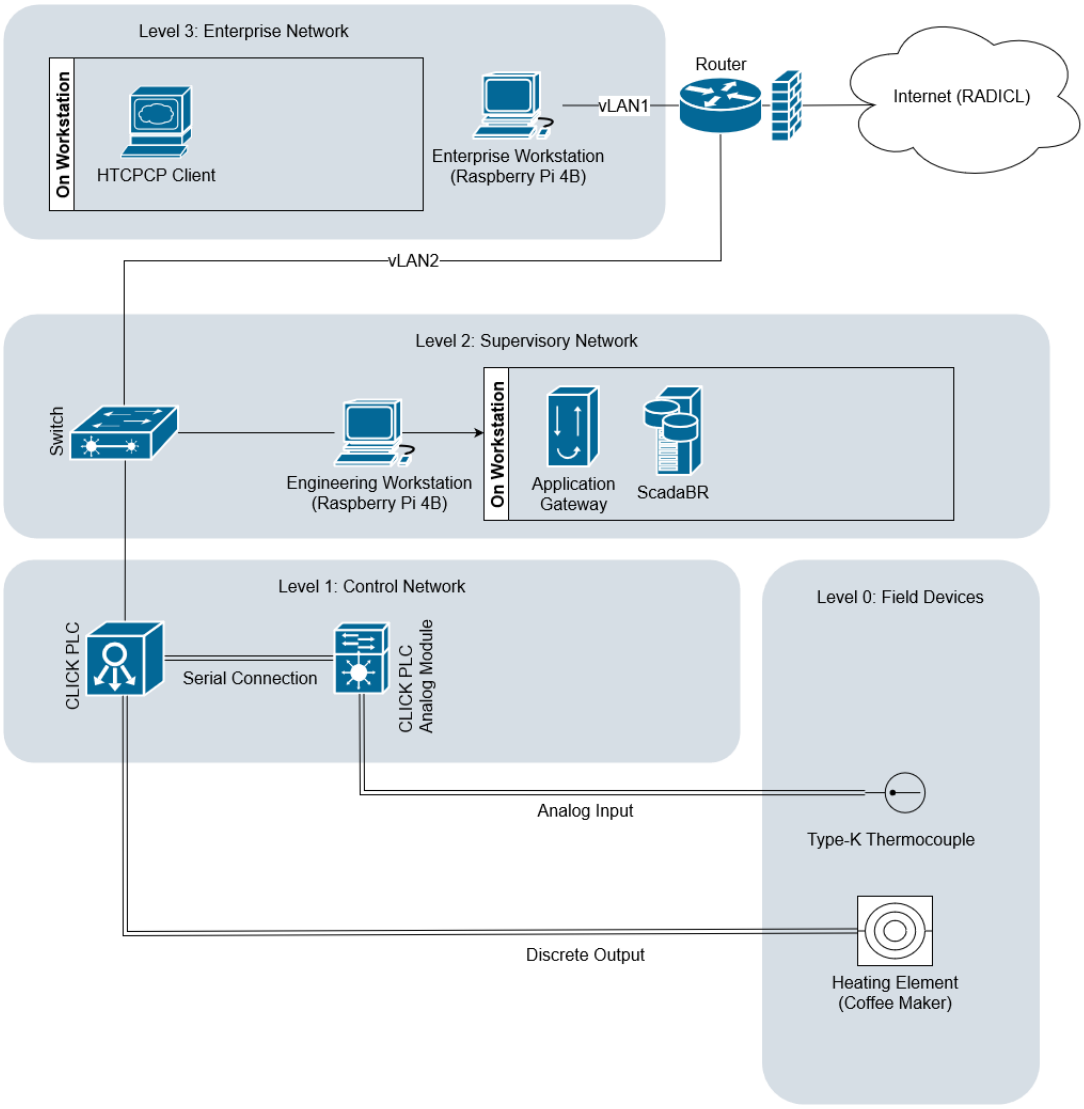


Figure 1: Depiction of the initial vision of organization of the testbed. Of the three Raspberry Pis, one will serve the purpose of the Engineering Workstation on which the Hyptertext Coffeepot Control Protocol (HTCPCP) Client will reside. It will reside in the Enterprise Network that will have it's own vLAN facilitited by the OpenWRT router. A second Raspberry Pi will serve as a dedicated firewall device whose location may change. Two example deployment locations of the dedicated firewall are between the Supervisory Network or between the Supervisory Network and the Control Network. The third Raspberry Pi serves as the Engineering Workstation on the Supervisory Network. The Application Gateway that serves as the converter between HTCPCP and Modbus resides on the Engineering Workstation as does ScadaBR, an application that provides virtual HMI, data logging, and visualization functionalities. Finally, the CLICK PLC resides in the Control Network and the sensors it is attached to (the relay controlling the heating element in the cofeepot and the Type-K Thermocouple) reside in the Field Device level.

Figure 2: The second coffee pot I used in the project – the Black+Decker CM0700.

5 Coffeepot Teardown

5.1 The Coffee Maker – The Center of the Testbed

The coffeepot itself is the at the very center of this testbed since it was the core of the original conception and provides the primary ISA-99 Level 0 functionality – that of the industrial process actuator. Last semester, I did a teardown and attempted to modify the Mr. Coffee BVMC-EVX23 coffee maker that I had laying around my home. This portion of the project doubled as a part of my final project for CS 543 (Embedded Systems) regarding an exploration into Programmable Logic Controllers. I attempted to modify the coffeepot so that my CLICK PLC could interface with it to turn the coffee maker on and off. This ended in failure since a mishap regarding a loose screw burnt out several of the electrical components on the coffee maker's printed circuit board (see presentation file labelled A1). As such, I needed a new coffee maker to use for the project.

5.2 Enter the Black+Decker CM0700

The Black+Decker CM0700 was purchased at Fred Meyer's for $20 dollars (and it retails online for $35 – what a steal!). As can be seen in Figure 2, this coffee maker has a single rocker switch. This is different from the previous coffee maker I intended to use that had a toggle button that would switch between multiple states (Off, Clock-based Power On, and On). The rocker switch does incorporate an LED behind it that I may be able to tap into to use as a signal about whether the coffee maker switch is on (e.g. tell the PLC that the coffee maker is on even though the PLC didn't send the on signal). To gather more info about how I can modify the internals of the coffee maker, I'll have to tear it apart and get a look at the PCB.

5.3 The Teardown

1. Two tri-point screws on the bottom hold it together. Some prying was required to free the bottom from the rest of the chassis once these were unscrewed (Figure 3).

2. Pretty much done. The internals are now revealed and they are much simpler than the internals of the Mr. Coffee maker that I tore down last time thanks to the lack of clock and scheduling functionality

Figure 3: The screws on the bottom of the Black+Decker coffee pot (left) and the coffee pot with the bottom plate removed (right).

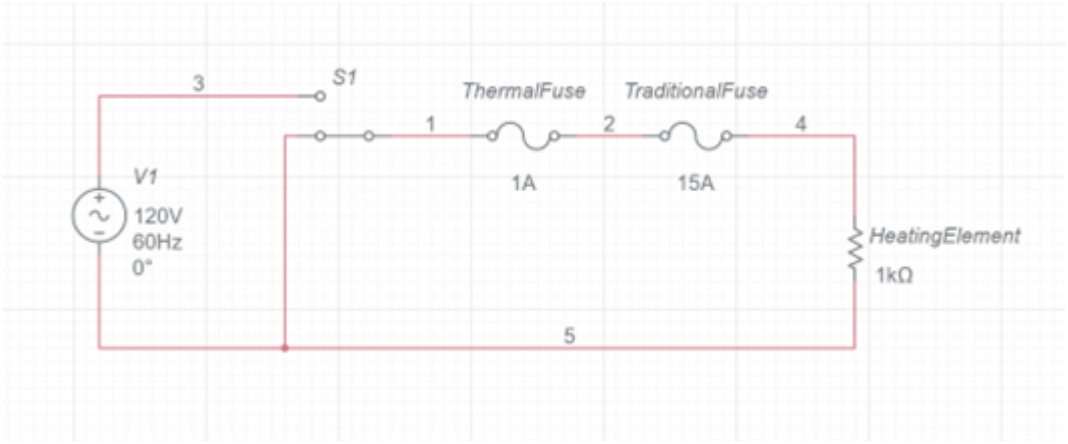


Figure 4: A model of the circuitry observed inside the Black+Decker coffee maker. Note that the component labels are not necessarily accurate. Circuit model was created using NI Multisim Live.

5.4 Notes on Proceeding

Since the internals are so simple, it would be viable to make a full diagram of what is going inside the coffee maker before proceeding with modification. The black wire of the standard two-prong wall outlet cable goes directly to one of the terminals of the rocker switch. The other two terminals of the rocker switch connect to a red wire and a white wire. The white wire is connected to what appears to be the heating element as well as the white wire of the outlet cable. The red wire connects to two elements that appeared similar to diodes, but at a closer look one is a thermal fuse. The other component in series with the thermal fuse may be a second thermal fuse, but apparently of a different make and model – it is difficult to read the text on it since it is on the underside of the component. The end of this series is in turn in series with the other end of what I think is the heating element. A schematic based on my observations may be found in Figure 4. I'm unable to confirm what the second component is that looks like the other thermal fuse, but I figure it is either another thermal fuse or a regular fuse, so I have marked it as such. Note: the heating element probably has much higher Ω.

5.5 The Intended Modification

I purchased a 24V relay previously since I knew that the PLC's outputs are 24V. The relay in question is a Songle SRD-24VDC-SL-C. According to the data sheet [Rel], this means the nominal coil voltage

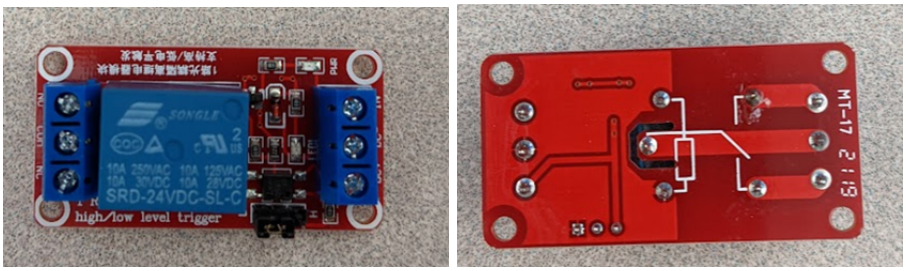


Figure 5: Songle relays actually require a number of surrounding componentes to function properly, though this was not mentioned on the datasheet (damn you Songle!). To make this easier, I purchased a small board that came with the appropriate Songle relay and surrounding components.

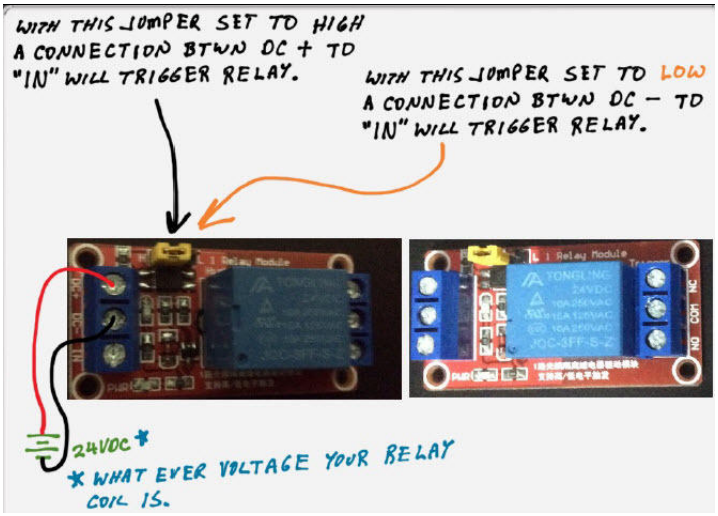


Figure 6: Description of undocumented relay board functionality courtesy of Amazon user "b1". Thanks random internet stranger!

(the voltage to activate the relay) is 24V. The S indicates that the relay is of the "sealed" type, the L indicates the sensitivity of the coil (0.36 W to activate), and the C indicates the contact type. The data sheet further indicates that the Form C contact type supports 7A at 28VDC, 10A at 125VAC, or 7A at 240VAC. This should be sufficient for my purposes since the coffee maker is powered from the wall outlet directly at 120VAC, though I'm not entirely sure how many amps it draws. I think it is safe to assume that it takes less than 10A since this would be an insane power draw for a small coffee maker. As a standard relay, the PLCs 24V output should be applied across the coil (DC+ and DC-) terminals. When the PLC output is enabled, this will allow current to pass through from IN terminal to the NO (Normally Open) terminal, turning the coffee maker on. The NC (Normally Closed) terminal could be used to reverse this behavior, but this is not appropriate for this use case. Thus far, I have only described the operation of the relay itself and not the surrounding PCB. It's difficult to know for sure what is going on with the rest of the PCB since there is no official documentation from the provider. I have resorted to the Amazon reviews to determine the functionality it. A reviewer named "bl" describes the functionality of the jumper in Figure 5. It should work in my case whether the jumper is set to high or low. I should insert the relay into the coffee maker circuit such that the regular switch in the coffee maker can still shut it off in the event of an issue occurring. To accomplish this, I will connect the red wire coming off of the switch to the IN terminal on the relay and connect the NO terminal to the thermal fuse. The PLC output will connect to DC+ and DC- on the relay board. See Figure 7.

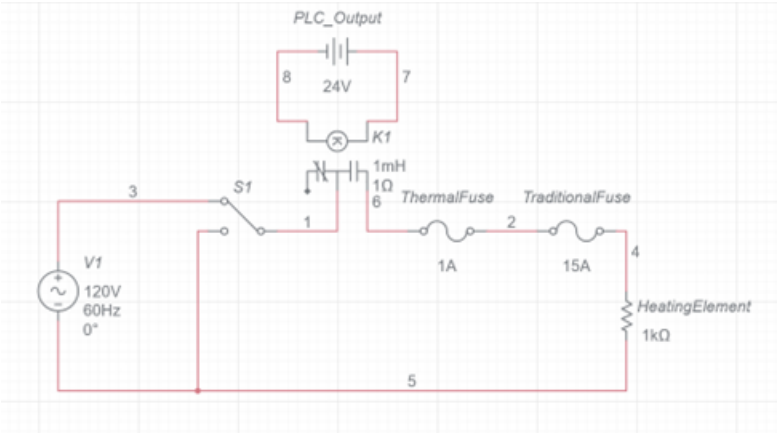


Figure 7: A modification of the previously created circuit model to incorporate the relay into the design.

5.6 Testing My Design

The first thing I want to test before mangling the internal wiring of the coffee maker is the relay itself. I will first attempt hooking the relay up to the PLC and testing that toggling the output will appropriately activate the relay.

5.7 Big Problems in Little RADICL - Sourcing vs. Sinking PLC Outputs

I could not get the relay to toggle using the integrated I/O modules on the PLC that I purchased (See Figure 8. After puzzling over this for about a day and half (with help from colleague and friend George Makrakis), I determined that the type of outputs on the module were not appropriate for my purposes. It turns out that the "sinking" label on the outputs indicates that the outputs essentially function as active low. In other words, when an output terminal is enabled, it toggles from floating to connecting to a ground, thus allowing current to flow. I figured that I could get the relay to work with this setup, but it wouldn't really be correct. Instead of having the relay powered and occasionally changing the IN signal to toggle the connection between the COM and NO terminals, I would connect the DC- terminal to the output such that toggling the output would pull the DC- terminal low, powering the relay. I would then have the IN terminal permanently connected to the +24V of the power supply to toggle the relay appropriately. This would probably have worked, but it was also a dirty hack, so I opted to go with the more correct solution – buying a new I/O module for the PLC that has sourcing rather than sinking outputs.

5.8 The New Module

As mentioned, the main problem was that the discrete outputs integrated with the PLC were sinking rather than sourcing. To address this with the new I/O module, I simply needed sourcing outputs. I purchased the CLICK C0-08TD2 I/O module from Automation Direct [Dir22]. A picture captured from the data sheet may be seen below in Figure 9.

5.9 Testing the New Module

The new module arrived within two days of ordering (praise Automation Direct!) and I tested it today (3/25/2022). After some fiddling, I realized that the new module requires direct connection to the power supply in order to function which is what enables it to function using a range of voltages. I used a blue wire to connect the power supply +24V to the V1 terminal on the new output module and a white cable to connect the power supply 0V to the C (Common Ground) terminal on the module. I then daisy chained additional blue and white wires to connect the module terminals to the DC- and DC+ terminals of the relay as well. Next, I connected a yellow wire from terminal 1 to the IN terminal of the relay and set the jumper on the relay to the H (High) position. Using the CLICK Programming

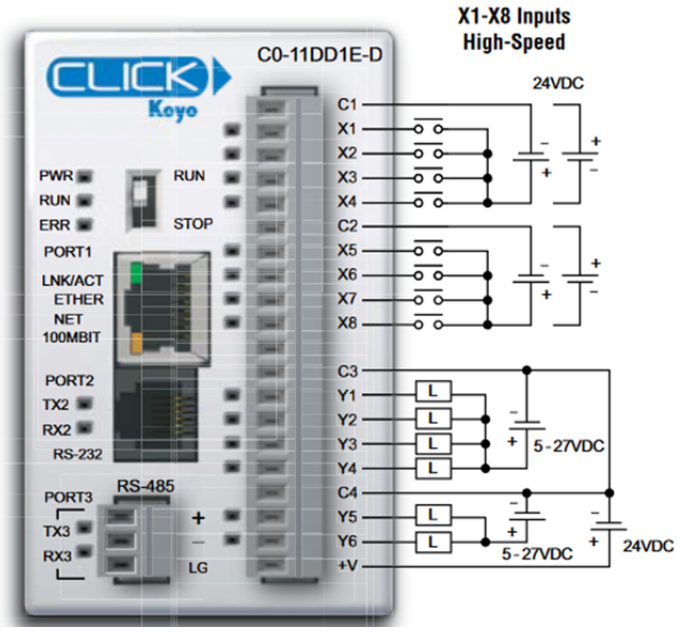
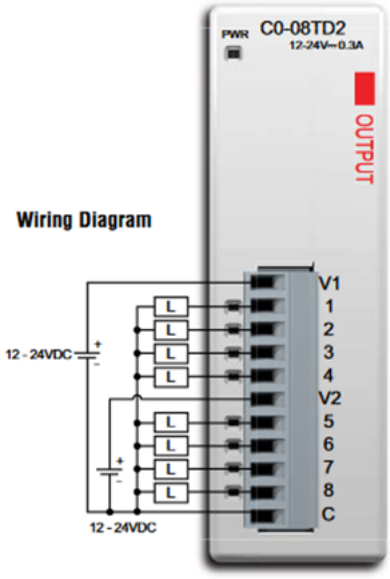


Figure 8: Schematic of the CLICK PLC I purchased. I turns out that the sinking outputs are not necessarily appropriate for controlling the relay.

C0-08TD2 – 8-Point Sourcing DC Output Module

8-point 12–24VDC current sourcing output module, 1 common, 0.3 A/pt, removable terminal block included.



Output Specifications	
Outputs per Module	8 (Source)
Operating Voltage Range	12–24VDC
Output Voltage Range	9.6–30 VDC
Maximum Output Current	0.3 A/point , 1.2 A/common
Minimum Output Current	0.5 mA
Maximum Leakage Current	0.1 mA @ 30.0 VDC
On Voltage Drop	1.5 VDC @ 0.3 A
Maximum Inrush Current	1A for 10ms
OFF to ON Response	< 1ms
ON to OFF Response	< 1ms
Status Indicators	Logic Side (8 points, red LED) Power Indicator (green LED)
Commons	1 (8 points/common)
Bus Power Required (24VDC)	Max. 50mA (All Outputs On)
Terminal Block Replacement	AutomationDirect p/n C0-8TB
Weight	2.8 oz (80g)

Figure 9: The sourcing output module I purchased and used to control the relay.

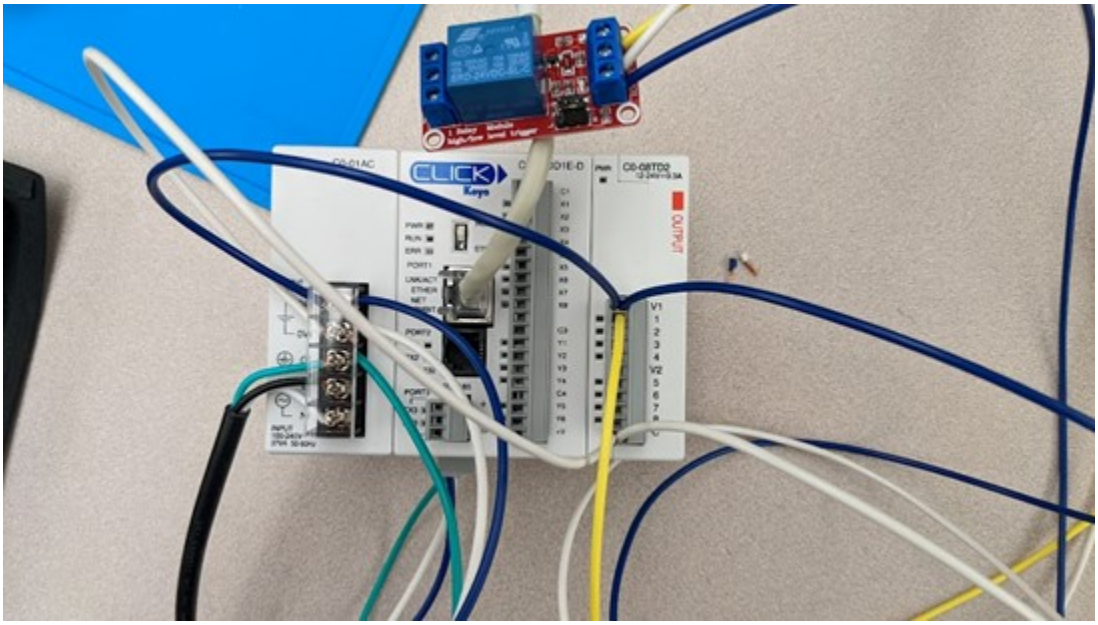


Figure 10: The new sourcing output module hooked up to the relay board for testing.

Figure 11: The wiring connected to the rocker switch on the Black+Decker coffee maker.

software [Dir], I found that the terminals of the new output module were mapped to addresses Y101-Y108 (it varies depending on the location of the module in the stack). I made a simple ladder logic program to toggle Y101 every 5 seconds. I could see and hear the relay activating periodically as expected as it made a characteristic click upon activation and lit up the red active LED on the relay board.

5.10 Finally, Modifying the Coffeepot

I should be able to proceed with my earlier-conceived plans. I will begin by clipping the power cable's attachments to the rocker switch in the coffee maker (See Figure 11). I will then use a multimeter in continuity testing mode to ensure that the various terminals on the coffee maker work exactly as I expect. Then, I will cut some new, shorter wires to attach the relay to the circuit as depicted in my diagram. The layout of the wires connected to the button may be seen above. By testing with the multimeter, I found that the bottom two terminals are connected when the switch is in the off position and the top two terminals are connected when the switch is in the on position. Comparing my planned diagram to the existing coffeepot wiring, the red wire that connects to the heating element and fuses will be cut and become the wires labeled 1 and 6 in the diagram (See Figure 7.

5.11 Success!

After modifying the coffee pot successfully, I tested the system by brewing a cup of coffee. I found it odd that the blue LED inside the button turns on when the relay is not activated since I thought the whole circuit is not closed and thus unable to power the LED. It had me worried at first that I had misunderstood something and that I may have wired it wrong. However, in testing I found this not to be the case. When the coffee maker is plugged in and the switch is on, the light turns on but the coffee doesn't brew until the relay is explicitly activated. To test this functionality, I used the CLICK programming software's manual override to enable and disable the relay. Using this feature, I brewed a cup of coffee. Hurray!

6 Network Configuration

6.1 The Router

I purchased the Linksys WRT ACM3200 for the project since it supports the OpenWRT firmware that Matt used in his testbed and it was reasonably priced [Kir20]. As with Matt's testbed, this is an affordable way to acquire vLAN functionality since most enterprise switches that would perform this functionality are prohibitively expensive. The first step in configuring the router is flashing the OpenWRT firmware.

6.2 Firmware Flashing

Flashing the firmware was fortunately incredibly easy. A quick Google search yielded precisely the instructions I was looking for [Opea]. I download the latest OpenWRT firmware image intended for the ACM3200, put it on a flash drive, accessed the router configuration page through one of the Raspberry Pis, and selected the firmware to upgrade to. I received a warning that the firmware was possibly not for my router, but after triple checking I determined that it was in fact correct. The warning was mostly likely issued because it was not an official Linksys firmware distribution. I proceeded with flashing and after a few minutes I made it to the new OpenWRT firmware configuration page.

6.3 Router Settings

- Password

  I set the password for the root account on the router to "Password27". While this is a weak password, this is necessary for others to actually be able to remember and use the password when the project is complete.

- OpenWRT UCI System

  OpenWRT uses the Unified Configuration Interface system [Opeb]. While many settings can be accessed via the browser portal, with UCI all the configuration files are stored at /etc/config on the router's root file system.

- Disabling DHCP

  I wanted to disable DHCP, so I accessed the /etc/config/dhcp file and removed entries related to providing DHCP on the Local Area Network. After doing this, I noticed that the web UI still showed active leases for the connected Raspberry Pis. To remedy this, I restarted the router. Afterwards, the DCHP leases were no longer active and the Raspberry Pis could not connect to the router. To fix this, I needed to assign static IPs to each of them.

- Static IPs for the Raspberry Pis [Kin22]
  - Edited /etc/resolv.conf to point to the DNS server running on the OpenWRT router
  - Added static IP settings for each Raspberry Pi
    - tb-enterprise – 192.168.1.2
    - tb-firewall – 192.168.1.3
    - tb-engineering – 192.168.1.4
    - tb-plc – 192.168.1.27
  - Note: I encountered a bizarre issue with sudo after changing host names on the Pis. Sudo would take a long time executing a command, time out, give an error about resolving the host name, and then execute the command (sometimes). I attempted to fix this by replacing the old hostname in /etc/hosts with the new one, and this seemed to work.

- Added port forwarding rules to the router config in **OpenWRT** > **Network** > **Firewall** > **Port Forwards**. Added the following rules that **SHOULD BE REMOVED LATER** to more accurately emulate a realistic environment:
  - WAN 23 → tb-enterprise 2222 (SSH)

- WAN 24 → tb-firewall 2223 (SSH)
- WAN 25 → tb-engineering 2224 (SSH)
- WAN 5905 → tb-engineering 5900 (VNC)
- Note: I also ensured that ssh was in fact enabled on each of the Pis so that this could work. For reference, the router itself was assigned the address **172.29.101.190** from RADICL Red Zone's DHCP server. This is liable to change since it's not statically assigned. I will be able to access each of the devices via ssh on their respective ports at this address. This is straightforward when in the lab physically – I just plug my laptop into a Red zone Ethernet port. To access remotely, I would use sshuttle to connect to the lab as usual, ssh into OldGlory, then ssh to Eagle, then finally ssh to my router (IP above).
- Mystery (Partially Solved): With the previously mentioned configurations, I was able to connect via SSH to tb-enterprise and tb-firewall, but not to tb-engineering. This was quite the mystery for a while. I tried restarting the router, restarting the device, and verifying the IP and listening status of tb-engineering (ifconfig and netstat). Everything was correct, but it still would not work. Scott pointed out a setting called "Masquerading" in **OpenWRT > Network > Firewall > General Settings > Zones**. It seems Masquerading was not enabled in the **LAN → WAN** zone. With masquerading disabled on this zone, it seems the router was dropping tb-engineering's packets on the way out. Mystery partially solved. However, it is still unclear why two of three devices worked without masquerading. They are all working now with Masquerading enabled.

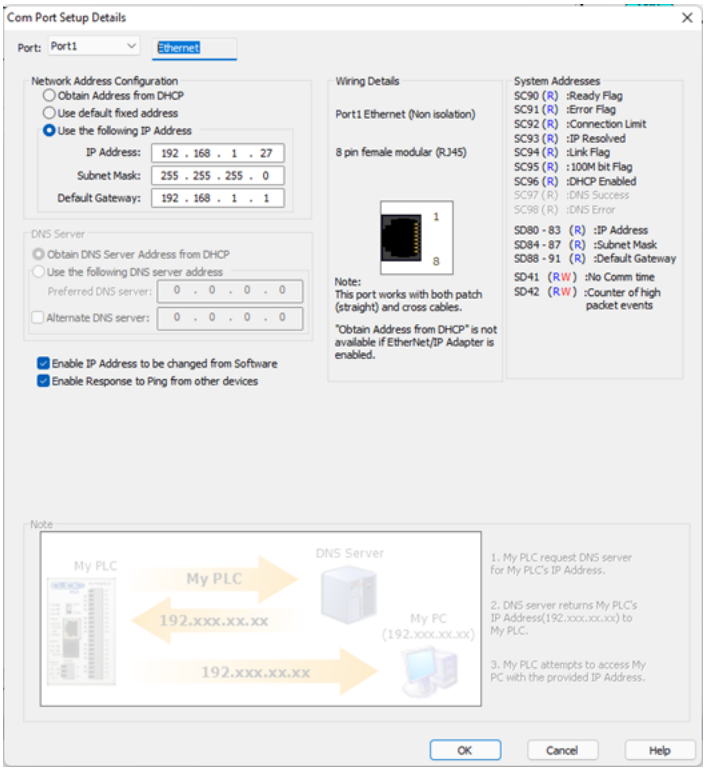


Figure 12: The Com Port Setup Details window found in the CLICK Programming Software.

7 Modbus Test Configuration & Pymodbus

Setting up the Modbus server on the CLICK PLC seemed very intuitive at first. After exploring the CLICK programming software and testing with Pymodbus, the program wasn't working. I ended up looking into the documentation further and discovered that the MODBUS addresses don't correspond to what they immediately seem to. Relevant documentation is in Chapter 4 of the CLICK PLC User Manual [Dir21].

7.1 Com Port Configuration

In the CLICK Programming Software, I went to **Setup** > **Com Port...**. The Com Port configuration window may be seen in Figure 12. From there, I selected **Port 1** since it corresponds to the Ethernet port. It appears the PLC may be assigned a static IP here, among other network settings. I left the IP, subnet mask, and default gateway on what it had defaulted to already (**192.168.1.27**, **255.255.255.0**, and **192.168.1.1**, respectively). I also left the other settings on their default, enabling the IP address to be changed from the software and allowing the PLC to respond to pings. These could be changed later to more closely align with realistic PLC configurations (allowing pings in a real environment would possibly allow adversaries to discover the PLC via ping scanning).

7.2 Modbus TCP Configuration

In the CLICK Programming Software, I visited **Setup** > **Modbus TCP...** The resulting dialogue box can be used to configure the Modbus TCP settings including which port to operate over (Port 1 in my case since this corresponds to the Ethernet port I had just configured), the configuration for the Modbus TCP client (also called the Master; this corresponds to both the "application gateway" that I will write in Python and ScadaBR), and the configuration for the Modbus TCP server. A screenshot of these settings may be seen in Figure 13. Of the settings, the most important to me was ensuring that the server was enabled. I left the port number on the default Modbus number – 502.

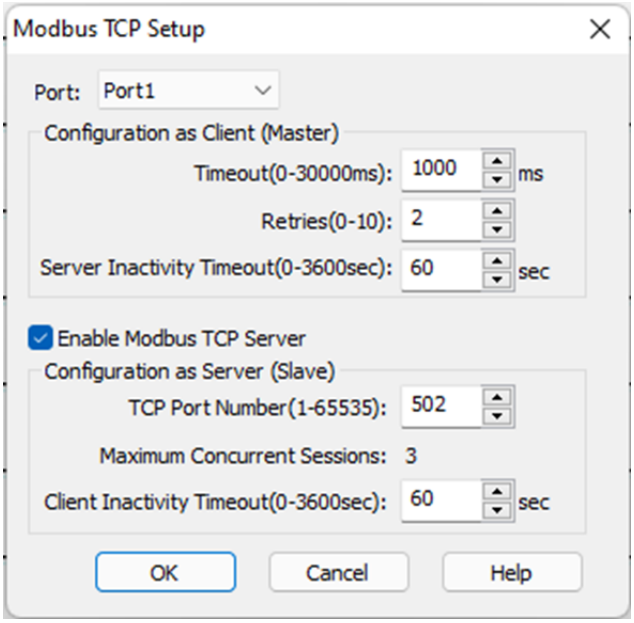


Figure 13: The Modbus TCP Setup window found in the CLICK Programming software.

7.3 Modbus Address

As detailed in the intro, initial testing with Pymodbus was unsuccessful. It turned out that the Modbus addresses in use were not what they seemed to be. In order to determine the real Modbus addresses in use, the **Address Picker** in the CLICK Programming Software needs to be examined. The Address Picker can be found in the CLICK Programming Software at **Program** > **Address Picker...** This opens a dialogue box displaying all the different internal addresses (inputs, outputs, and registers). By checking the **Display MODBUS Addresses** box, the Modbus addresses corresponding to each internal address may be seen. A screenshot of this may be seen in Figure 14. I already know from my hardware configuration (and by verifying in **Setup** > **System Configuration...**) that the output I've connected to the coffee pot relay corresponds to PLC output Y101. The Modbus address corresponding to Y101 is 8225. This may be used in the Pymodbus program to read/write to Y101, allowing me to programmatically check the state of the coffeepot relay and change it between off and on.

7.4 A Simple Test with Pymodbus

I created a simple Python program using Pymodbus that reads the status of some coils, writes to them, then reads them again to confirm that the write worked as intended [Rip22]. In testing I found that as part of the part of the Modbus standard that Pymodbus implements, reads return a byte-long response even if the read was to a single coil (one bit) [MO12]. If one bit is read, the rest of the byte will be padded with 0 (False) independently of whether the addresses following the selected address are 0 or 1 (True or False). This behavior is demonstrated in the example program by reading all 8 coils starting at the target coil (PLC address Y101, MODBUS address 8225), turning all of them on with a write, then reading only the first 4 bits. See Figure 15 for the test program code and the associated output. With write and read capability to and from the PLC via Modbus TCP, I have the only primitive I will need to develop the Modbus portion of the Application Gateway. I also captured the exchange in Wireshark just for fun (see Figure 16).

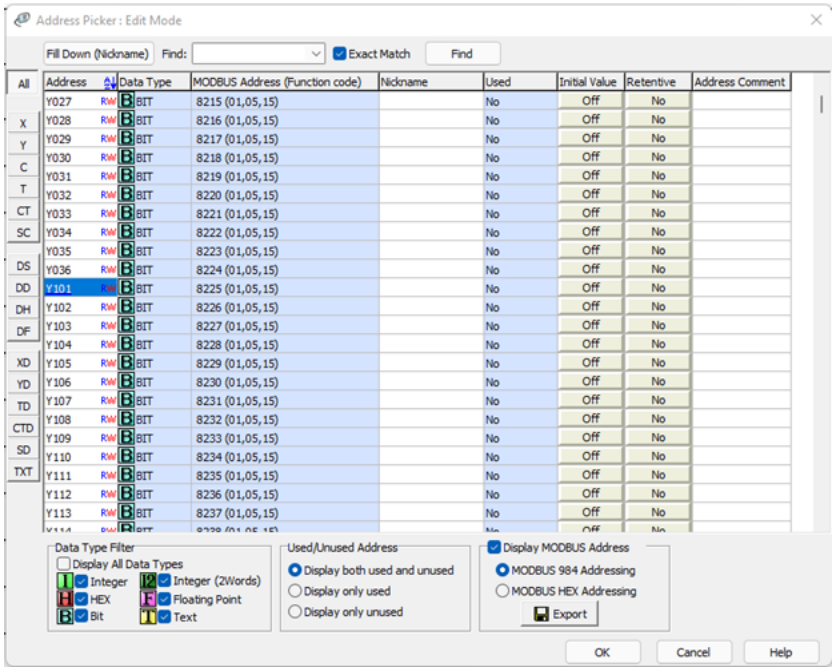


Figure 14: The Address Picker window in the CLICK Programming software.

```python
import scapy
from pymodbus.client.sync import ModbusTcpClient

client = ModbusTcpClient('192.168.1.27')

result = client.read_coils(8225, 8, unit=0x1)
if not result.isError():
    print(result.bits)
else:
    print("Error on initial read.")

result = client.write_coils(8225, [True]*8, unit=0x1)
if not result.isError():
    print(result)
else:
    print("Failure on write.")

result = client.read_coils(8225, 4, unit=0x1)
if not result.isError():
    print(result.bits)
else:
    print("Error on final read.")
```

```
[False, False, False, False, False, False, False, False]
WriteNCoilResponse(8225, 8)
[True, True, True, True, False, False, False, False]
```

Figure 15: Test program written in Python using the Pymodbus library to read from and write to a coil of the PLC (top) and the output from running the program (bottom).

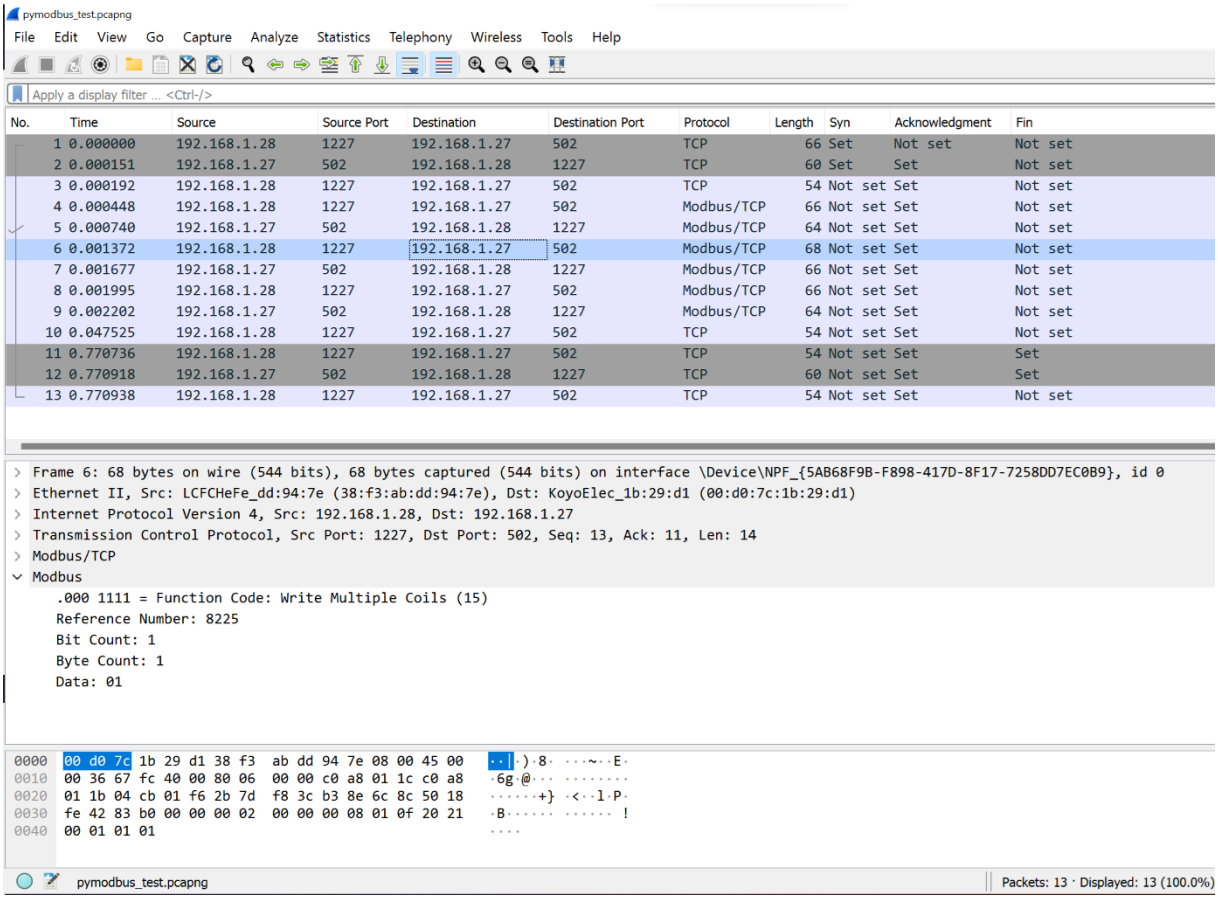


Figure 16: A Wireshark network traffic capture showing the packets produced from the Pymodus program in Figure 15.

```
Accept-Additions = "Accept-Additions" ":"
                   #( addition-range [ accept-params ] )

 addition-type    = ( "*"
                    | milk-type
                    | syrup-type
                    | sweetener-type
                    | spice-type
                    | alcohol-type
                    ) *( ";" parameter )
 milk-type        = ( "Cream" | "Half-and-half" | "Whole-milk"
                    | "Part-Skim" | "Skim" | "Non-Dairy" )
 syrup-type       = ( "Vanilla" | "Almond" | "Raspberry"
                    | "Chocolate" )
 alcohol-type     = ( "Whisky" | "Rum" | "Kahlua" | "Aquavit" )
```

Figure 17: The list of Accept-Addtions as laid out in RFC 2324.

8 Application Gateway

I initially considered two approaches for implementing the Hypertext Coffee Pot Control Protocol (HTCPCP) portion of the application gateway. The first approach was to use the existing Python http library and modify it to support the various extensions that the HTCPCP RFC lays out [Mas98]. The second approach (and the one I ended up going with) involved using Python's socket library to facilitate networking and manually implementing a bare-bones HTTP message generator and parser for the HTCPCP client and server ends, then building the requirements laid out by the HTCPCPC RFC into it. The HTCPCP client and server (applciation gateway) will be uploaded to my GitHub at https://github.com/kwilliamscameron/ICS-Coffeepot alongside this PDF and the LaTeX source for this PDF.

8.1 RFC 2324 - HTCPCP

HTCPCP stands for Hypertext Coffee Pot Control Protocol. It is based on HTTP and was originally published as RFC 2324 on April 1st, 1998. HTCPCP adds the BREW method which is used to tell the coffee pot to brew some coffee. The HTTP POST method will achieve this as well, but its use with coffee pots is deprecated according to the RFC as of 1998, so my HTCPCP appication gateway should probably not accept POST. The GET method works similarly to in HTTP, but in this case it will get the status of the coffee pot (is it on or off?). The RFC also discusses the use of the PROPFIND and WHEN methods. PROPFIND is supposed to return metadata about the "brewed resource" and WHEN is used for saying "when" there is enough milked poured into the coffee. My testbed does not have the physical means to support these methods, so I will either not implement or I will send a response accordingly.

The RFC also dictates the use of an existing header field and proposes a few new header fields. First, it describes the use of the "Safe" header field that represents whether it is safe to repeat the previous request, e.g. if you submit a web form and it has some effect, you wouldn't want it to submit twice. Requesting that the coffee pot turn on/off or checking if it is on/off is safe to repeat, so the Safe header will be set. A new proposed header is "Accept-Additions". In regular HTTP, the Accept header lists the type of media are acceptable in a response. Accept additions lets the coffee pot know what types of things may be added to the coffee (e.g. milk or cream). The list of Accept-Additions the protocol definition supports is in Figure 17. Once again, the testbed is not equipped to actually deliver any of these, but support for these fields will be included.

I intend to implement a terminal user interface (TUI) for the HTCPCP client program. I found a library that I can use for this called Pytermgui [Bcz22]. The client program should implement all of the Accept-Additions options as seen in Figure 17. If an option that is physically unsupported (read: all of them), then the gateway HTCPCP server should respond with 406 Not Acceptable and give a

list of supported additions.

9 ScadaBR Setup

This document discusses the setup of ScadaBR on the testbed's engineering workstation Raspberry Pi.

9.1 How NOT to Install ScadaBR

Since I don't have an active connection to the real internet, I followed the directions provided on the OpenPLC site [Alv22] to clone the git repository onto a flash drive, then transferred them to the engineering workstation. When running the installer, I got a Java path error. I also then realized that this installer was made for x86 machines. I will need to follow the manual installation steps given on the wiki page of the ScadaBR Source Forge distribution [cel21].

9.2 Installing Java

The first step in installing ScadaBR is to get the correct version of Java installed on the Raspberry Pi. ScadaBR uses Apache Tomcat which in turn uses some version of the Java Development Kit (JDK), depending on the version. If I use the latest stable version of Tomcat (10.0.20 at time of writing [Fou22]), I will need some version 8 or later of the JDK. I downloaded the Arm 64 Compressed Archive of JDK 18 [Ora22]. I then installed this version of Java [Sho21].

9.3 Installing Apache Tomcat

I downloaded Apache Tomcat 10.0.20 and uploaded it to the home folder of the machine using SSH (now that I got that working on tb-engineering; see the mystery discussed in the 6.3). When I tried extracting it, tar spat out an error about the timestamps on the files being in the future. The Raspberry Pi clock was off by a lot, so I disabled NTP syncing and manually set the time [Fro20]. I was then able to extract the file and start Tomcat via the {apache_home_dir}/bin/startup.sh file. I verified that Tomcat was working by visiting `http://localhost:8080/` on the tb-engineering.

9.4 Finally, Installing ScadaBR

I downloaded the latest release of ScadaBR and used scp to copy it to tb-engineering. I then moved the file to {apache_home_dir}/webapps/ and restarted Tomcat. I attempted to verify it was working by visiting `http://localhost:8080/ScadaBR/` on tb-engineering, but the page was broken. A brief inspection of the logs revealed many errors. While the SourceForge documentation states to use the latest version of Apache Tomcat, the latest release of ScadaBR found on the GitHub says to use Java 8 and Apache Tomcat 9 with ScadaBR 1.2 (the latest and only currently available release).

9.5 Running it Back

I removed Java 18 and Tomcat 10, then repeated the above steps, this time installing Java 8 and Tomcat 9. ScadaBR is now working.

References

[Alv22] Thiago Alves. Openplc. https://www.openplcproject.com/, 2022.

[Aut] AutomationDirect.com. Click plc - how to setup and use a thermocouple module. https://www.automationdirect.com/videos/video?videoToPlay=xJS-NsvfU58.

[Bcz22] BczSalba. pytermgui 5.0.0. https://pypi.org/project/pytermgui/, Apirl 2022.

[cel21] celsou. Scadabr, 2021.

[Dir] Automation Direct. Free click software. https://www.automationdirect.com/clickplcs/free-software/free-click-software.

[Dir21] Automation Direct. Click plc user manual. https://cdn.automationdirect.com/static/manuals/c0userm/c0userm.html, October 2021.

[Dir22] Automation Direct. Click family of plcs. https://www.automationdirect.com/clickplcs, 2022.

[Dra22] Dragos. Pipedream: Chernovite's emerging malware targeting industrial control systems. Technical report, Dragos, 2022.

[Fou22] The Apache Software Foundation. Apache tomcat versions. https://tomcat.apache.org/whichversion.html, 2022.

[Fro20] Patrick Fromaget. How to sync time with a server on raspberry pi? https://linuxhint.com/install-oracle-java-jdk-16-raspberry-pi/, 2020.

[JCK+17] B. Johnson, D. Caban, M. Krotofil, N. Brubaker, and C. Glyer. Attackers deploy new ics attack framework "triton" and cause operational disruption to critical infrastructure. Technical report, Mandiant, December 2017.

[Kin22] Phil King. How do i set a static ip adress on raspberry pi? https://openwrt.org/docs/techref/uci, March 2022.

[Kir20] Matthew J. Kirkland. Designing and developing virtual and real testbeds for industrial control systems education and training. Master's thesis, University of Idaho, 2020.

[KL15] Eric D. Knapp and Joseph T. Langill. *Industrial Network Security*. Syngress, 2015.

[Lan13] Ralph Langner. To kill a centrifuge. Technical report, The Langner Group, 2013.

[Mas98] L. Masinter. The hypertext coffee pot control protocol (htcpcp/1.0), April 1998.

[MKK+21] G. M. Makrakis, C. Kolias, G. Kambourakis, C. Rieger, and J. Benjamin. Industrial and critical infrastructure security: Technical analysis of real-life security incidents. *IEEE Access*, IX(2021):165295–165325, 2021.

[MO12] Inc. Modbus Organization. Modbus application protocol specification v1.1b3. https://www.modbus.org/specs.php, April 2012.

[Opea] OpenWRT. Linksys wrt3200acm. https://openwrt.org/toh/linksys/wrt3200acm.

[Opeb] OpenWRT. Uci (unified configuration interface) - technical reference. https://openwrt.org/docs/techref/uci.

[Ora22] Oracle. Java downloads. https://www.oracle.com/java/technologies/downloads/#java16, 2022.

[Rel] Songle Relay. https://components101.com/sites/default/files/component_datasheet/5V%20Relay%20Datasheet.pdf.

[Rip22] RiptideIO. https://github.com/riptideio/pymodbus, 2022.

[Sca22] ScadaBR. https://github.com/ScadaBR/ScadaBR, 2022.

[See22] Cyber Seek. Cybersecurity supply/demand heatmap. https://www.cyberseek.org/heatmap.html, 2022.

[Sho21] Shahriar Shovon. How to install oracle java jdk 16 on raspberry pi. https://linuxhint.com/install-oracle-java-jdk-16-raspberry-pi/, 2021.