# IoTea: IoT Capabilities With a Tea Kettle

Mark Rizko

*Abstract*—**IoT devices allow consumers and hobbyists to engineer their own creations for the purpose of fulfilling any behavioral requirement of technology that solves a problem. For my IoT project, I created a system that allows me to control the operation of my tea kettle in the morning to ensure it turns on at the perfect time in the morning to start my day. This project was done by using a Raspberry Pi to communicate with my personal computer and a relay board using established technologies such as GPIO and Python programming language.**

## I. Introduction

THE advancement of IoT technology in the 21$^{st}$ century allows anyone with slight engineering knowledge to automate processes in their lives with no limit on creativity. The idea I have chosen to pursue to begin my IoT experience is an automated tea kettle system, which saves me a trip to the kitchen each morning. Besides saving distance, I am also saving time in setting up the tea kettle and this automation will allow me to transition from my bed to my first cup of tea with very little thought and high reliability. I anticipate that even on the hardest of mornings, this solution will aid me in getting a strong and energetic start to the day, which is very important for productivity.

While this is not the first IoT project I have worked on, it is the first one I have done by myself without using the implementation of others as a base. There is a challenge in learning how to connect the tea kettle to my Raspberry Pi, especially regarding the electrical components of the process, yet it is a fun and rewarding challenge as well. Since standard tea kettles are not designed to be used in this way, I had to take great care in the electrical setup of the project, making sure to take the necessary precautions and figure out a solution that would allow the Raspberry Pi to reliably control a switch for the tea kettle's operation.

### A. Existing Solutions

The automation process that I have used for my project is definitely unique, however there are already implementations of "Smart Kettles" on the consumer market. Companies like COSORI and KOREX have created tea kettles that act in conjunction with a smartphone application, allowing for variable temperature control as well as some basic timing options for the kettle's operation; however, the focus of these tea kettles is mainly on temperature control, and do not offer a smart scheduling system. I aim to establish a solution that schedules the kettle based on my indirect action rather than simply using time.

### B. Proposed Solution

My proposed solution for this project involved many different ideas. The basis of the sensor aspect of my project was focused around finding a reliable trigger every morning that wouldn't slow down the momentum of my day. I did not want the kettle to trigger the moment I woke up (as an alarm kettle would), as I sometimes enjoy sitting in bed for a while before I begin my day. After going through many ideas, I eventually settled on a sensor that works reliably with my workflow, which is the powering on of my PC. I will get into more details about the sensor component in a later section. The system will function as follows: between the hours of 9am and 11am, they Raspberry Pi will await me turning on my PC for the first time of the day, which will trigger the kettle and brew boiling water. In a fully setup environment, I will have pre-filled the kettle and toggled the button the night before, as well as prepared my mug of tea. This preparation is minimal, and it is more favorable for me to prepare everything the night before, for automatic tea in the morning when I am tired and have less energy.

## II. Sensors

In this section, I will describe the process I took to find and set up an optimal sensor for this project, one that would work reliably as well as provide the most effective timing for me. Early in the project's development, I was eager to explore how the Raspberry Pi interacts with different sensors such as microphones and load cells (weight sensors). The leading idea in the beginning was to install a few load cells under my bed to detect when I have left my bed in the morning (in the specified range); however, after working on an IoT beehive scale about a year ago, I decided that the load cells would require too much wiring and optimization to become effective and reliable so I decided to pivot to a different sensor idea. After shutting down the load cell idea, the sensor that was next in line was a voice recognition system that would trigger the kettle through a command phrase, such as "Good morning!" I was very close to implementing this idea, but I knew that it would require a high-quality microphone array in order to be reliable, which was outside of the budget I had set for the project. In addition to the value of the microphone array, I was also reluctant to begin experimenting with google's voice recognition APIs with how busy my quarter had become at this point in time. There is no regret in rejecting these ideas, because my final idea was too good and feasible for me to dwell on what could have been. For my final idea, I went back to the drawing board and began thinking more about when I wanted the kettle to fire, instead of how. I realized that the timing was more important to me than the sensor itself was, and started to think about actions that I take every single morning when I am ready to get to work. As it turns out, the perfect idea was sitting in front of me this whole time, if I configured the system to trigger when I turned my PC on every morning, the timing would be absolutely perfect. For this method, I did not even need a sensor, rather, I knew I could just use the local network in my house to implement this idea. During the hours of operation, the Raspberry Pi will send a ping to my PC every minute. If there is a successful ping, the tea kettle will activate, providing me with the best timing for my situation. In order to make the sensor reliable, I set my PC to a static IP address, rather than allowing DHCP to handle the assignment.

## III. Hardware Components

As I learned earlier during the hardware sections of this class, I could not simply pass the +1000W tea kettle through the Raspberry Pi, and needed a module in between them to absorb power and act as a switch. For this application, I decided to use a relay board that was equipped to handle the power requirements of the tea kettle. The relay board that I purchased came with two relays, although I only used one, and each relay was equipped with three different ports. The normally open (NO) port of the relay board is used for applications in which the system is on for a short time, and remains off for the rest of the day. Alternatively, there is a normally closed (NC) port that is used for the opposite application. In between these ports is a common terminal (COM), which must be plugged in regardless for the operation of this relay board. The input to the relay board is an Active LOW, meaning the device will be on when a LOW signal is sent to the board through GPIO, and a HIGH signal will turn off the relay.

## IV. System Architecture

This section will describe the main circuit used for this project and will outline the connections made between the components of the system. Below is an abstraction of the circuit.

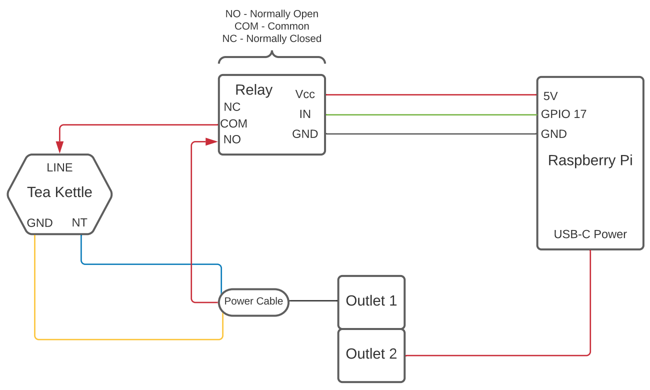


Fig. 1. Abstracted Circuit Diagram

I will begin by describing the connection between the Raspberry Pi and the relay board. As with most devices that are compatible with micro-controllers, I needed to provide power to the relay board through the 5V and GND pins of the Raspberry Pi, as well as communicating ata with GPIO pin 17. Connecting the relay board to the tea kettle was a challenge at first, as I was unsure which of the three wires in the tea kettle's power cable were to be fed through the relay board. My first attempt at making this connection involved feeding the ground wire into the normally open port of the relay board, and the neutral through the common port. My reasoning for making this connection was based on a source online that I had misinterpreted, and it was back to square one. Luckily in my case, I have a relative who is my age that is studying electrical engineering, and he was able to quickly correct my mistake by informing me that I needed only to feed the line through the relay board. After making the right connection between the tea kettle and the relay board, all that was left was powering both devices. Since the tea kettle is very high powered at a little over 1000W, the Raspberry Pi and tea kettle had to be plugged into separate wall outlets in order to ensure proper power delivery to the tea kettle and safety of the system.

## V. Code Analysis

This section will describe the components used in the python code of this project and how they made this system work.

### A. Initializing GPIO

The first task done by the python code was initializing the GPIO ports on the Raspberry Pi.

```
import RPi.GPIO as GPIO
# initialize GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)
# starts off HIGH = open circuit
GPIO.output(17, GPIO.HIGH)
```

The first step in this process was importing the python GPIO library. After that, I had to specify the addressing mode of the GPIO, which can either be GPIO.BOARD for the literal pin numbers or GPIO.BCM which corresponds to the Broadcom SOC channel number, which I decided to use since this is what I followed on the pinout when I was making physical GPIO connections. The next step was to set up GPIO port 17 as an output port, and immediately initializing the port with a HIGH signal, as HIGH for the relay board corresponds to inactive.

### B. Getting Date and Time

```
from datetime import datetime
import pytz
# initialize time zone
tz_LA = pytz.timezone('America/Los_Angeles')
```

In order to get the date and time, I had to import two libraries, *datetime* for basic date-time functions and *pytz* to get the time zone. After that, I simply had to initialize the time zone using pytz.timezone.

### C. Ping

Since the system relies on a ping to one of my home devices, the next component of my python code was getting ping to work properly.

```
from pythonping import ping
responses = ping('IP Address')
```

I used the *ping* function to save the (four by default) responses to a python variable, to be acted on at a later point.

### D. Combining The Components

Finally, I combined all of these components to create the logic that satisfies the intended behavior of the project, which I will describe now. Please see the full code in the appendix for further comments and corresponding line numbers. I will begin on line 20, where the system operates entirely in a loop once the variables and GPIO port are initialized. The first line of the loop [21] gets the current time based on the time zone variable. After getting the time, the main logical condition of the loop is checking whether the time is in the range of 9-11am. If the time is outside of the range, I put the Raspberry Pi to sleep for five minutes before checking again by simply using the *time* library. When it is between the hours of 9-11am approximately, my PC, with a static IP address of 192.168.1.46 on my local network, will be pinged by the Raspberry Pi. Immediately after, the responses are checked whether they are successful pings or not [30]. If there is a successful ping, the signal variable initialized on [17] will be set true, and the next block of code will be executed. Lines [35-37] of the code correspond to the operation of the tea kettle in the desired conditions, setting the GPIO output to LOW (turning on the relay board) and waiting five minutes for the water to boil before before turning it back off by sending a HIGH signal to the GPIO pin. The code that follows is a synchronization method after the kettle has been turned on once during the window, that synchronizes the program with the hour mark for the next day. Finally, line [45] sleeps for an entire two hours, taking the system completely out of the time window for the day, as the kettle should not be operational until the next day.

## VI. Performance Evaluation

This final section will describe results of testing the system, and why in one case it did not function properly. I was able to test the IoTea three times at the time of writing this report, however, instead of testing between the hours of 9-11am I instead changed the testing range to 9-11pm. This decision was so that I could observe the project during the daytime, rather than having it on during the middle of the night. I decided to test it in this fashion because of some uncertainties I still had about the safety of the circuit, and wanted to be able to troubleshoot if anything were to go wrong with the electronics. For all tests, I began running the program at 12pm, and decided to turn on my PC at around 9:10pm to compensate for the loose time restrictions I had in the program. The first test was unsuccessful, and this is where I realized that the IP address of my PC that I was using when testing the ping function had since changed due to router troubleshooting that we had to do in between. Instead of simply using the new IP address, I decided at this moment to set a static IP address for my PC as to not run into this again. The next two days I ran the same exact test, one with the Raspberry Pi connected to Ethernet and one with the to WiFi. I wanted to see the difference between WiFi and Ethernet in this situation because there is some distance from my room to the router, but the difference in delay was negligible. Had I done the project slightly earlier, I would have been able to run more tests, however, I believe after two successful tests I conclude that the system does in fact behave as intended.

## VII. Conclusion

I very much enjoyed doing this project, and learned quite a bit about the implementation of IoT systems and things to watch out for next time.

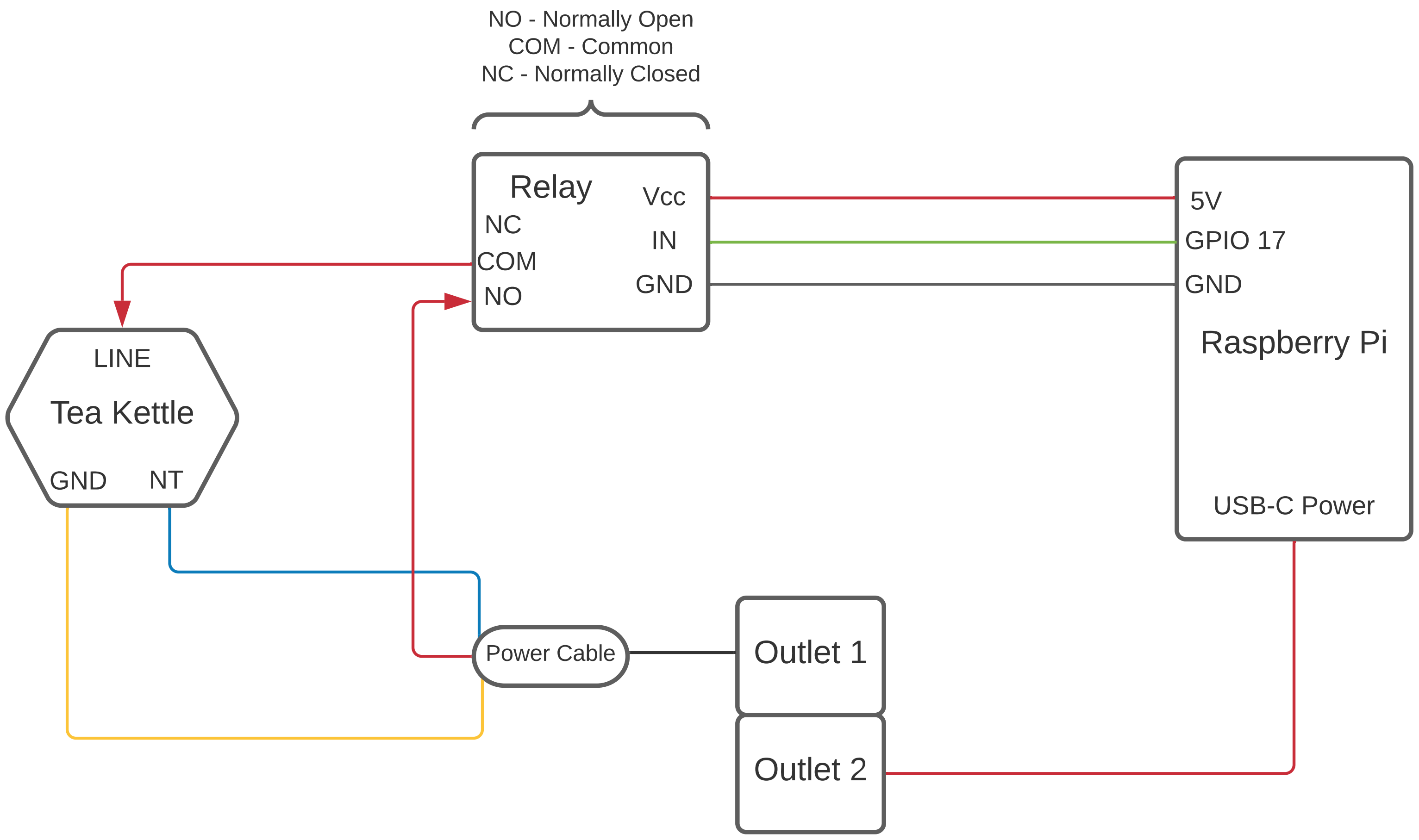
### A. Lessons Learned

The implementation of this project taught me a lot of important takeaways for doing these projects in the future. Initially, I was under the impression that in order to modify the data on the Raspberry Pi, I would have to connect the device to a display, mouse, and keyboard. It was not long, however, that I learned how useful the SSH protocol would be in this situation. Additionally, I also used the SFTP protocol to quickly transfer files so that I could write and edit code while the Raspberry Pi was off. Another important lesson learned from this project was the operation of a relay board, and how easily it can be used as a switch to power high-output devices such as the tea kettle. I definitely see myself using these again for future IoT projects that I come up with. My familiarity with completing an IoT project has grown, and I am excited to see what is in store for the future.

### B. Improvements

I also wanted to highlight some of the shortcomings of this project, and improvements that can be made to ensure proper operation of the system. The first major improvement to be made on this project is to purchase a higher-powered relay board. While the current relay board is designed to handle the power output of the tea kettle, it does not leave any breathing room, which for an engineer causes concern about the reliability of the project over time and continual use. Another improvement that can be made on this project resides in the code, as the synchronization methods and power efficiency can be improved by ensuring the system is operational in the exact window of time and sleeps as much as possible outside the window. I started to make some of these improvements in the code, but did not want to complicate the program too much, as I did not have much time to test. The final improvement to be made on this project would be to consolidate the electrical components into a more uniform system. This would include 3D printing a case for the Raspberry Pi and relay board as well as ensuring proper insulation of the soldered connections. These improvements will improve performance, reliability, and power efficiency of the project.

# Appendix A
## Abstracted Circuit Diagram



## APPENDIX B
## PYTHON CODE - DEMO.PY

```python
# Python code used for project demo
import RPi.GPIO as GPIO
from pythonping import ping
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)
GPIO.output(17, GPIO.HIGH)

while True:
  resp = ping('192.168.1.196')
  for re in resp:
    if re.success == True:
      GPIO.output(17, GPIO.LOW)
      time.sleep(300)
      GPIO.output(17, GPIO.HIGH)

```

## Appendix C
## Python Code - run.py

```python
import time
import RPi.GPIO as GPIO
from datetime import datetime
from pythonping import ping
import pytz

# initialize time zone
tz_LA = pytz.timezone('America/Los_Angeles')


# initialize GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)
# starts off HIGH = open circuit
GPIO.output(17, GPIO.HIGH)

signal = False # signal bool


while True: # system operational
    dt_LA = datetime.now(tz_LA) # initialize time
    # if out of range sleep and check every five minutes
    if (dt_LA.hour < 9) or (dt_LA.hour > 11):
        time.sleep(300)
    else:
        # pi is not sleeping, awaiting signal
        resp = ping('192.168.1.46')
        # if ping, set signal to true
        for re in resp:
            if re.success == True:
                signal = True
        if signal:
            # set it false for next time since it only fires once
            signal = False
            GPIO.output(GPIO.LOW) # activate kettle
            time.sleep(300) # wait 5 minutes for kettle to finish
            GPIO.output(GPIO.HIGH) # open circuit

            dt_LA = datetime.now(tz_LA)
            # check every minute until it's the hour mark, then go back to sleep
until out of range (2 hours)
            while dt_LA.minute != 0:
                time.sleep(58) # check every 58 seconds
                dt_LA = datetime.now(tz_LA)
                if dt_LA.minute == 0:
                    time.sleep(7200) # 2 hours takes us out of range and keeps us
close to the hour mark

```