Frozen Pipe Detection and Prevention System

IoT Application with Arduino Uno and AWS

Joseph Luciano
Electrical and Computer Engineering
State University of New York at Binghamton
Binghamton, New York, USA
jlucian3@binghamton.com

ABSTRACT

Frozen pipes can build up pressure and burst, annually resulting in millions of dollars in damages to homes. Despite the simplicity in preventing this issue, it still afflicts many households in the United States. This paper introduces an approach to prevent frozen pipes by utilizing an Arduino Uno, temperature sensor, two networked devices, and external services. The Arduino Uno powers the temperature sensor and provides real time temperature data over serial connection to one of the networked devices. This first device processes the data into time stamped comma separated values (CSVs) and hosts a server on the local area network (LAN). The second device monitors the real time temperature data and downloads the hourly data before backing it up on a server hosted by Amazon Web Services (AWS). When the temperature drops below a threshold, it automatically activates a light emitting diode (LED) to simulate a heating element. This system sets a blueprint for a more complex system that can seamlessly integrate with a home's heating system or power a separate heating system altogether.

ACM Reference format:

Joseph Luciano. 2021. Frozen Pipe Detection and Prevention System: IoT Application with Arduino Uno and AWS. In *Proceedings of ACM Woodstock conference (WOODSTOCK'18). ACM, New York, NY, USA, 3 pages.* https://doi.org/10.1145/1234567890

KEYWORDS

Comma Separated Values (CSV), Local Area Network (LAN), Amazon Web Services (AWS), Light Emitting Diode (LED), Universal Serial Bus (USB), Internet of Things (IoT), React JavaScript (ReactJS), Anaconda, Python, Arduino Uno, Arduino Integrated Development Environment (IDE), Serial Connection, Visual Studio Code (VSCode)

1 INTRODUCTION

The Frozen Pipe Detection and Prevention System monitors the temperature of vulnerable pipes to provide valuable information to the user while automatically activating heat sources to prevent frozen pipes. This is done by monitoring the ambient (or water) temperature with a DS18B20 sensor connected to an Arduino Uno. The Arduino Uno checks to see if any actions should be taken immediately and sends the temperature data as CSVs to the local computer over a serial connection. The Arduino connected computer hosts a server on the LAN to broadcast real time temperature data with the ability to download the daily temperature data. The local server and temperature data can be accessed via any locally networked device to view the ReactJS website. A virtual machine, also hosted on this computer, records the data from the server's webpage and transmits the data to be saved in an AWS server. Meanwhile, the Arduino monitors the temperature data and lights an LED to simulate a heating system. In a commercial application, the computer would be replaced by a streamlined process that directly stores the data in the cloud where this system could be integrated into a building's heating system. However, in this system the user will also be able to see the temperature data in real time and can take appropriate actions ahead of the system. All the data transmission steps have been automated as well. This paper will first review the system design as well as all the hardware and software components. Shortfalls of the project as well as future improvements will be addressed after reviewing some of the results.

1.1 Motivation

Americans are no stranger to frozen pipes and the devastating costs they bring on. According to Tritan Plumbing, "More than 250,000 families and home experience frozen and/or burst pipes each year in the U.S." There is great potential to prevent these disasters. Considering that, "on average, freezing pipes and water damage insurance claims cost around $7,307".[1] Preventing frozen pipes could save American's millions of dollars every year. Insurance companies could be incentivized to prevent these claims since, "it's estimated that anywhere from 11% - 20% (or more) of homeowner's insurance claims each year are due to water damage and freezing".[1] This paper will demonstrate a reliable and cost-effective approach to prevent frozen pipes in the future.

2 System Design

The system incorporates the following devices and software applications: Arduino Uno, DS18B20 temperature sensor, red LED, all associated cables, breadboard, PC connected to a home network, AWS server, Anaconda to manage and run Python code, Visual Studio Code with Node.js for local server hosting, Arduino IDE to upload the Arduino code, and Oracle VM VirtualBox for the Ubuntu virtual machine. A simplified data flow diagram of the core features can be referenced in Figure 1: System Data Flow Diagram.

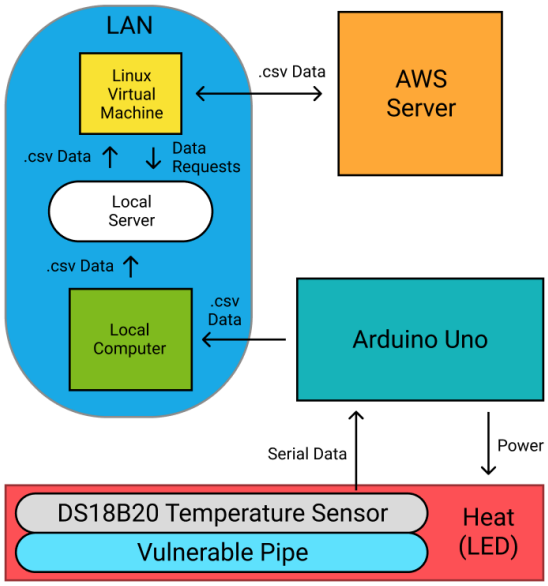

Figure 1: System Data Flow Diagram

2.1 HARDWARE

The Frozen Pipe Detection and Prevention System incorporates an Arduino Uno, a basic wiring kit with LEDs, a DS18B20 temperature sensor, a universal serial bus (USB) cable, and a computer with networking capabilities. A view of the essential hardware components can be seen in Figure 2: Essential Hardware Components.

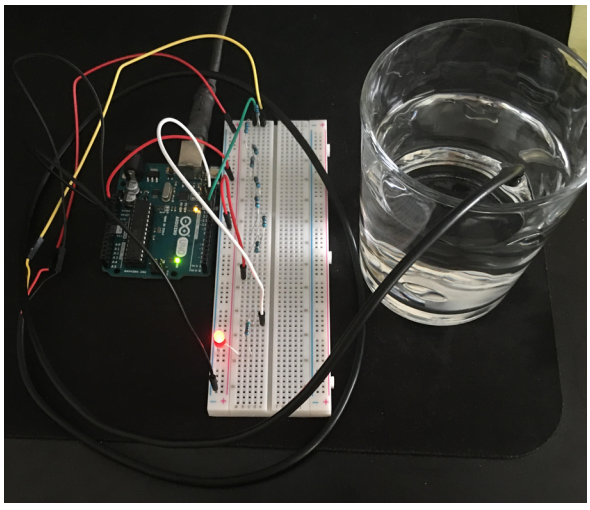

Figure 2: Essential Hardware Components

The DS18B20 temperature sensor has 3 wired connections: power, ground, and data. The data wire needs to be combined with the power source across a 4.7K ohm resistance to provide accurate data. An output port on the Arduino Uno controls the power to an LED across a 1K ohm resistor. All the data exchange is accomplished via USB connection to the computer.

2.2 SOFTWARE

First, the Arduino Uno must correctly interface with the DS18B20 temperature sensor and send the collected data as CSVs so that the computer can interpret the data. Figure 3: Arduino Uno Temperature Sensor Script demonstrates how this is accomplished.

Figure 3: Arduino Uno Temperature Sensor Script

Once powered, the Arduino Uno will then report temperature data every second to the connected computer. The computer runs a Python script to translate the CSV serial data to a locally stored CSV file. This script will automatically timestamp the data and reset the CSV file at midnight. Please reference Figure 4: Serial to CSV Python Conversion Script for more details on how the script works.

Now that the temperature data is stored in an accessible format, it is possible to store and preview it on a server. The server is hosted using Node.js and the webpage code is written in ReactJS. The entire server supports a webpage that reads the CSV data every time it updates before displaying the most

recent line as well as files' contents as a chart. The webpage also monitors the current time to see when it should initiate a data transfer to web user's computer. This is done once every hour and names the file accordingly so that the Linux VM can transfer the file.

```python
import os.path
from os import path
import serial
import datetime
import time

arduino_port = "COM3"   # USB port is COM3
baud = 9600             # Arduino baud rate is 9600
# Location of the CSV file is in the server's directory
fileName = "./react-router-website/src/components/temp-data.csv"
current_datetime = datetime.datetime.now()      # Get the current date and time
# Save the current day for detecting a new day
current_day = current_datetime.day

# Set up serial connection with COM port
ser = serial.Serial(arduino_port, baud)
print("Connected to Arduino port:" + arduino_port)
# Open the file path and append new data
file = open(fileName, "a")
print("Created file")

while True:
    if current_day != datetime.datetime.now().day:
        print("New day, resetting temperature data")
        time.sleep(60)  # Sleep for 60 seconds so VM can download csv
        file.close()    # Close the file to prevent errors

        os.remove(fileName)  # Delete yesterday's data

        file = open(fileName, "a")  # Reopen the file
        print("Created file")
        current_day = datetime.datetime.now().day  # Reset the current day variable

    # Gets the current line generated by the Arduino Uno at the current time
    getData = str(ser.readline())
    current_datetime = datetime.datetime.now()

    # Extracts the timestamp and relevant data values
    data = str(str(current_datetime)[:-7]) + "," + getData[2:][:-5]
    print(data)

    file.write(data + "\n")  # Writes the data

print("Data collection complete!")
file.close()
```

Figure 4: Serial to CSV Python Conversion Script

A Linux VM runs with the webpage open in the background. Figure 5: Webpage Temperature Data Display shows what the CSV data display looks like on the webpage. The button manually downloads the hourly temperature data, and it is automatically downloaded when the hour changes.

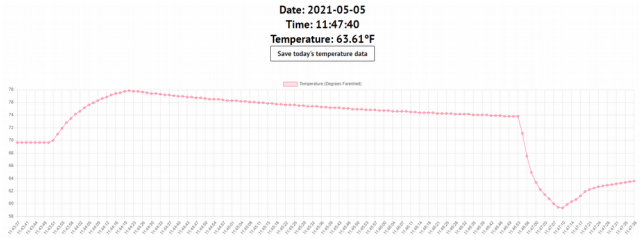

Figure 5: Webpage Temperature Data Display

The VM acts as a bridge between the Arduino connected computer and the AWS server. Using crontab, the Linux VM runs a shell script every hour. This shell script uses the SCP protocol to transfer the automatically downloaded CSV file to the AWS server. The protocol requires a locally saved key pair, the desired file, the destination server, and the destination file location. Figure 6: Shell Script to Send CSV Data to AWS displays the scripts that uses the current date and time to find the correct CSV file and send it to the AWS server.

sendTempData.sh × sendTest ×

```
#!/bin/bash
echo "Hourly Data Backup to AWS"

CURRENT_YEAR=$( date '+%Y' )
CURRENT_MONTH=$( date '+%m' )
CURRENT_DAY=$( date '+%d' )
CURRENT_HOUR=$( date '+%H' )

scp -i ./Downloads/AWS_Broker_key_pair.pem ./Downloads/${CURRENT_YEAR}-
${CURRENT_MONTH}-${CURRENT_DAY}_Hour_${CURRENT_HOUR}.csv
ubuntu@ec2-35-170-248-185.compute-1.amazonaws.com:/home/ubuntu/
${CURRENT_YEAR}-${CURRENT_MONTH}-${CURRENT_DAY}_Hour_${CURRENT_HOUR}.csv
```

Figure 6: Shell Script to Send CSV Data to AWS

3 POTENTIAL IMPROVEMENTS

The original design of the system utilized a Wi-Fi module that could directly transmit temperature data from the Arduino Uno to AWS. However, a series of issues arose, and a new system design needed to be conceived. The use of a Linux VM adds unnecessary complexity to the system but is necessary to demonstrate how data can be wirelessly transmitted across a home network. Additionally, no wirelessly integrated heating elements were implemented in this system. A commercial application would require a heating element to fully prevent frozen pipes.

The use of numerous devices to transfer data in different formats introduces many points of failure. With a system that is expected to run 24 hours a day, 365 days a year, too many possibilities of error can be devastating.

Ideally, this entire system could be integrated into a single device that can be attached to piping within a home. The device setup would require the credentials to the local network. This connection would provide a gateway to a web server where information can be stored and analyzed. This web server could then send commands to a home heating system. For example. AWS could send commands to a Smart Thermostat. Alternatively, the device could be connected to an external heating element that prevents the frozen pipes.

4 CONCLUSION

The Frozen Pipe Detection and Prevention System presents a cost effective and reliable method of avoiding thousands of dollars in property damages. Even the cost of a plumber to unfreeze pipes that have not yet burst ranges from $100 to $500.[2] Although many improvements could be made to the design, the system functionally operates as well as stores and displays valuable information for the user.

5 ACKNOWLEDGMENTS

Many thanks to Professor Mo Sha and Junyang Shi, without them this project would not have been possible.

6 REFERENCES

[1] Tritan Plumbing. 2018. Frozen Pipes: Facts and Prevention. (January 2018). Retrieved April 28, 2021 from https://www.tritan-plumbing.com/blog/2018/january/frozen-pipes-facts-and-prevention/

[2] Anon. 2020. Burst Pipe Repair Cost: Frozen Pipes, Corrosion, Age. (March 2020). Retrieved April 28, 2021 from https://www.fixr.com/costs/burst-pipe-repair#:~:text=For%20example%2C%20thawing%20frozen%20pipes,remaining%20costs%20going%20to%20materials.