Server Cooling Comparison Direct Liquid Immersion vs. Tradition Air Cooling

Jonathan Q. Askwig, Nathan R. Butler, Kyle M. Huglen, and Shane M. Sinda

***Abstract*—Traditional air cooling has reached its maximum server density at roughly 25kW per 42U rack. Direct liquid immersion cooling potentially allows for server densities in excess of 75kW per 42U rack. To test these numbers we designed a test bed that would allow for a comparison between tradition air cooling and direct liquid immersion cooling. Each server ran a minimal install of Debian 6.0 (Squeeze). Once the servers were configured, several benchmarking tests were to be conducted, which put the servers through daily loads (from light to extreme) over a period of a month and output the information to the MySQL database. To ease the data collection process, temperature sensors were utilized to monitor hotspots inside and around the server. After all the data was collected, the results were to be analyzed and compared. A conclusion was then be determined on which cooling solution is the most effective.**

***Index Terms*—air cooling, direct immersion cooling, liquid cooling, server cooling, submerged cooling**

I. Introduction

LARGE datacenters have been plagued with high power costs for decades. On average, datacenter power costs can consume 40% of the monthly expenses, with over 20% due to powering and cooling electronic equipment. For some data centers, this can lead to well over $1,000,000 per month in costs [8].

Traditional data centers utilize raised floor technology and large forced air HVAC systems to keep electronics cool. A direct contributor to both the noise and power draw is cooling electronic hardware. Traditional forced air cooling, while effective, has a capability of around 5 kW per rack over a sustained area. Today the same 5 kW rack can now draw up to 25kW. The increase in power density raises the question of how to cool these servers. The power density is coming close to the maximum cooling potential of the conventional air-cooled datacenter architectures [6].

Manuscript received April 28, 2011. This work was supported by Danny Miller, BS, Lecturer, Electrical Engineering Technology, Michigan Technological University, School of Technology.

J. Q. Askwig graduates summer 2011 from Michigan Technological University with a bachelors in Computer Network and Systems Administration. jqaskwig@mtu.edu.

N. R. Butler is a recent graduate of Michigan Technological University in Computer Network and System Administration. nrbutler@mtu.edu.

K. M. Huglen is a recent graduate of Michigan Technological University in Computer Network and System Administration. kmhuglen@mtu.edu.

S. M. Sinda is a recent graduate of Michigan Technological University in Computer Network and System Administration. smsinda@mtu.edu.

The potential solution the Submerged Cooling Senior Design team has spent months researching is the direct immersion of computer hardware in dielectric oil. Studies have shown that this method is the most efficient way to cool electronics due to its power saving capabilities, significant temperature changes, and even cooling distribution throughout the entire system.

Mineral oil has a high specific heat capacity, which means it can absorb a considerable amount of heat without having to vent it out to the surrounding air. It takes the system a significant amount of time to work up to its equilibrium temperature. According to Pugetsystems, the starting temperature on the CPU would start at 29°C and takes about an hour to hit a stable idle temperature of 37°C [7].

One of the companies implementing this solution is Green Revolution Cooling (GRC). For over 90 years, this company has been providing dielectric fluid submersion cooling solutions to several categories of industrial size electronics. Just recently, GRC started implementing submerged datacenters with their own unique blend of oil called GreenDEF. This unique blend has been claimed to perform better than air because it holds six times the dielectric constant, which prevents micro-arcing, improves electrical efficiency and connectivity. It also holds 1,200 times more heat by volume and is significantly better at heat transfer [6].

As far as saving energy goes, GRC has developed an efficient way to calculate the cost and savings of using the GreenDEF solution. This can be found at http://www.grcooling.com/wp-content/uploads/2010/06/Cost-Savings-Plugin2.htm. Just to give a perspective, using this calculator, one 42U rack containing 42 dual CPU submerged servers will save a company $22,664 dollars per year if their energy cost is $0.20 cents per kWh. This is very substantial savings for companies [6].

II. Materials

A. Hardware

The following segment will cover the technical aspects of the hardware that was used throughout the course of this project. Equipment that was utilized throughout the entirety of the study will be covered first followed by hardware used in their respective network implementations referenced to their associated figures shown later.

The control center for the testing environment was a Dell T410 with two Intel Xeon 5500 processors with 4GB of

memory providing ample computing power for its purpose. The Dell T410 housed two 150GB hard drives (15000 RPM Dell Cheetah), configured in Raid-5, offering more than enough storage to hold data retrieved from the test bed and the scripts used to procure it. Testing scripts were pushed from the Dell T410 to four Dell PowerEdge 750 servers. Each of the Dell PowerEdge 750s were equipped with a Pentium 3 processor along with 2GB of memory. To maintain consistency between the test results, all four machines were kept as identical as possible. For the submerged machines several irrelevant parts were stripped: including the internal fan casing, CPU fan, memory fan, USB fan, PCI fan, floppy drive, CD-ROM drive. The hard drive cables were extended and the hard drives were placed outside the submersion tank as to not be harmed by the oil.

In the original network design (Appendix A, Figure 1) two database servers were the initial choice to store the data, as there would be a layer of redundancy with a main and slave data storage server. These IBM xSeries4340s were running a Pentium 3 processor and contained three 36GB 10k rpm SCSI drives providing a total of 108GB of disk space on each server. Also present in the first version of network was a Foundry FastIron 800 switch.

Version 3 (Appendix A, Figure 3) of the network schematic replaces the Foundry FastIron 800 with a DLink DES 3624 switch. The DLink DES 3264 had 24 Ethernet ports to accommodate the network's needs. Another identical DLink DES 3624 is added in Version 5 (Appendix A, Figure5) to provide switching capability for the isolated testing area. The redundant database servers found in Version 1 and 2 were swapped for an EMC Clariion CX600 Storage Area Network. This machine has a total of ten useable hard drives. Each drive was 72GB giving a total of 720GB hard drive space. It also had a fiber connection to the storage server that interfaces with the existing Ethernet network. Since there was no longer a need for the database servers, one of them was used as the Storage Server and outfitted with a fiber channel NIC so it could communicate with the EMC Clariion CX600 SAN. Due to the relocation of the submerged Dell PowerEdge 750 servers, a Linksys WRT54G was required to wirelessly connect them to the rest of the testing environment.

Version 6 (Appendix A, Figure 6) introduces the Arduino Mega 2560 microcontroller in charge of collecting data from sensors placed inside the devices being tested. The Arduino Mega 2560 can be powered via USB or external PSU, has 256KB (8KB is for bootloader) of flash memory, a 16MHz clock speed, can support up to 54 Digital I/O pins and 16 Analog pins; each which can house a device such as a Maxim DS18S20 temperature sensor [1].

Maxim DS18S20 temperature sensors were used to obtain the external temperatures of the servers. Maxim DS18S20 sensors' are parasitically powered temperature sensors that are capable of detecting temperatures ranging from -55°C to +125°C. While between the ranges of -10°C to +85°C the sensors' deviation is only ±0.5°C whilst giving accurate temperature readings down to the hundredths of a degree. Each DS18S20 has a unique hardcoded 64-bit address which allows for multiple sensors to be placed on the same data bus line [3].

B. Software

lm_sensors was installed on the testing servers to obtain temperature data from the servers' onboard CPU and motherboard sensors. The control server stored "get" and "send" scripts for each testing server. The "get" scripts executed lm_sensors, output the data to temporary files, saved the CPU and motherboard temperature, sent the data to a .csv file on the control server, and emailed the group if the temperature surpassed 150°F. Each "send" script simply sent the "get" script to the corresponding testing server. The "send" scripts were put in the crontab and set to run every 5 seconds. Example scripts (get_air00.sh, send_air00.sh) and an example output file (air00.csv) can be found in Appendix E and Appendix F.

cpuburn is a hardware stability testing tool developed by Michael Mienik. Cpuburn heats up the processor to its maximum operating temperature by continually cycling floating point unit (FPU) intensive functions for a specified time. FPUs are also known as the math coprocessor which is responsible for carrying out floating point arithmetic.

C. Network Configuration

A small network was needed in order to conduct the testing. From concept to actual implementation, the network design underwent six revisions. This was due to over-ambitious goals at the start of the project, hardware issues, and physical relocation of the project.

Version 1 of the network (Appendix A, Figure 1) utilized nine servers, a remote computer, the campus network, and a switch. Six servers were dedicated to the testing subnet: 2 submerged, 2 liquid cooled, and 2 air cooled. One server of each cooling group would run Windows while the other would run a Linux distribution. The server subnet consisted of three servers: a control server, a database server to store temperature data, and a slave database server for redundancy. The control server was a frontend to the test-bed, sending scripts to the test servers, allowing remote access to the test servers through SSH. The remote computer would be used as an additional layer of redundancy to store temperature data and configuration files. This version of the network was purely conceptual, as no hardware had been obtained yet.

Version 2 of the network (Appendix A, Figure 2) saw a few changes as hardware was obtained. A Dell T410 running CEntOS was used as the control server and a Foundry FastIron 800 was used as the switch. An IBM and a Dell server were used for the database servers.

One public IP address would be allocated for the project. As a result, the control server was placed on the edge of the network and used as a gateway for the internal testing subnet (it had four Gigabit Ethernet ports). Iptables and DNS were configured to allow the internal subnet to access the Internet through the control server.

Worth mentioning is that there was only one available jack in EERC 328A, and another senior project also needed a public IP address. It was our job to configure the switch to split the connection, which is the reason for the EXTERNAL

VLAN shown in Figure 2. This is the setup used through the majority of the fall semester until early February.

Version 3 of the network (Appendix A, Figure 3) underwent a major revision. Due to the general lack of resources along with the time constraint, it was decided to drop the liquid cooling portion of the project along with the comparison between Windows vs. Linux. This would also allow for a more defined project scope. At this point, the project advisor–Professor Danny Miller–provided four Dell PowerEdge 750 servers for testing.

It was also decided to eliminate the two database servers. Instead, one would be used as a storage server connected to the CNSA department's EMC Clariion CX600 SAN. This would provide a much higher level of redundancy and would also allow us to eliminate the remote backup computer from the design.

The Foundry switch was needed for the Network Engineering class. A DLink DES 3624 was provided as a replacement and was reconfigured without any issues.

We believed that the hard drives could not be submerged in oil, so the control server would be utilized as a PXE server and boot CEntOS for the test servers through the network.

The team was informed by Professor Miller that the 40 gallon tank of oil would need to be located in a workshop on the ground floor of the Minerals and Materials Engineering Building due to safety concerns. This caused a major issue since the workshop had no network connection. However, a decent Wi-Fi signal was available. A Linksys WRT54G was configured to run OpenWRT and to operate in “client-bridge” mode, which would allow the two submerged servers to send their temperature data through the control server in EERC 328A to the SAN. This was far from ideal since the Wi-Fi network requires a manual logon through a web browser every 24 hours in order to obtain an IP address. As a result, someone would have to physically visit the workshop every morning during testing to re-activate the connection. Otherwise, no scripts would be able to pass to the submerged servers. Another issue is that the IP address would be obtained through DHCP, making automation of the scripts extremely difficult since the address specified in the script may change daily.

Version 4 (Appendix A, Figure 4) only saw a slight change. Since the test-bed was now segmented, it would not make sense to push the PXE boot across the campus network to the submerged servers. The Linksys WRT54G would be used as an additional PXE server instead.

Version 5 (Appendix A, Figure 5) underwent another slight change. The air cooled servers were moved out of EERC 328A down to the workshop, and the former database slave server was used as the sole PXE server. An additional switch was obtained since the Linksys WRT54G only had four switch ports.

The final revision was Version 6 (Appendix A, Figure 6). After many different troubleshooting steps, we could not get the storage server to successfully communicate with the SAN. Without the SAN, there was no reason for the control server to be in EERC 328A, so everything was consolidated in the workshop.

It was decided to not utilize PXE booting after realizing that the data cables were long enough to extend out of the oil. The power wires, however, did have to be extended. It was also decided to switch from CEntOS to Debian 6.0 (Squeeze) for the testing servers due to the smaller footprint. Only the packages required for the testing were installed.

Figure 6 also shows the Arduino control board connected to the control server via USB, with temperature sensor bundles being run to each of the test servers.

The networking problem mentioned with Version 3 was solved after discovering an abandoned cable run leading to an empty jack in an adjacent room. CAT5 cable was run and MTU Telcom was contacted to transfer the record for the jack in EERC 328A to the new jack at the end of the cable run.

The hosts file on the control server and each testing server was edited to reflect the hostname of each server. Public Key Authentication was configured between the control server and each testing server in order to allow automated communication during testing. Sendmail was also configured to send the group a warning email in case any of the servers started to overheat.

To handle the test results MySQL 5.0.77 was chosen. The user ‘subcooling’ was created to input data into the database. ‘subcooling’ was restricted to INSERT only on the subcooling database on the localhost. The subcooling user was used to input test results into the subcooling database only. The database itself was created to have one centrally controlling table that was linked to all of the other tables by way of foreign keys. This allowed for quick references to specific data points from any test.

The servers table was created to store static information about the servers that are being used for the entire project. All of the other tables grew as the tests were performed on the equipment. Once the tests were performed, all of the data was easily accessible by knowing the test_id, this would quickly display all pertinent test information by ways of the testing table. The overall database design can be seen in Appendix B. The tables and users were created with two SQL scripts, TableCreate.sql (Appendix C) and UserCreate.sql (Appendix D).

Raymond Saliga, B.S. Biologicial Plant Science, developed code to monitor greenhouse variables that was easily portable to the project. After talking to Raymond it was decided that two programs would be required for this project. The first program, AddressFinder.pde (Appendix G) , simply obtains the 64-bit address ({0x10, 0x5B, 0x18, 0x25, 0x02, 0x08, 0x00, 0xA7}) that uniquely identifies each DS18S20. For the Arduino to read the addresses “0x” had to be appended to each octet. The second program, GetTemps.pde, retrieves all of the temps that are on the bus line. Both programs utilize the OneWire.h and DallasTemperature.h libraries.

AddressFinder.pde works by first setting up a OneWire instance to communicate with sensors. After the communication channel is established the Arduino starts interrogating the sensors for its current address and temperature in both Celsius and Fahrenheit. The sensors were then tagged with the last octet of their address and recorded for future references (0xA7 -> A7).

GetTemps.pde (Appendix H) obtains all of the external temperatures from the 20 sensor temperature array. Before this program can run correctly, the Arduino needs to be configured. At "// Data wire is plugged into port 3 on the Arduino" there are six variables that need to be defined. ONE_WIRE_BUS variable needs to be set to the data port that the sensors data line is plugged into on the Arduino. TEMPERATURE_PRECISON variable sets the sensitivity of the sensors and is currently set to 9. THERMOMETERS_MAX variable sets the current number of sensors that are on the bus line. This variable is important because this number is used to ensure that all sensors have reported data before transmitting to the server. HUMAN/MACHINE variables are used later in the program and should not be changed. OUTPUT_TYPE defines the output format that is going to be sent to the server. OUTPUT_TYPE 0 outputs a human readable format and OUTPUT_TYPE 1 outputs the data into a .csv file for easier parsing and inputting to the database.

Once all variables are configured correctly the program can be uploaded to the Arduino board. Instantly the Arduino starts interrogating the sensors and reporting back to the server. For the server to interact with the Arduino, minicom was used. Minicom would open a session with the Arduino, normally on "/dev/ttyACM0", and output the data to an output file. This process proved cumbersome for sensor arrays over ten sensors. The general consensus was that the Arduino was sending too many characters across the serial line without a carriage return. To solve this problem GetTemps.pde could be modified to collect the data and send it in two chunks rather than one large one.

D. Oil

The two different types of oil suitable for cooling purposes are transformer oil and its less refined counterpart, mineral oil. Both transformer and mineral oil are dielectric fluids, which do not affect electrical parts because they are non-conductive. Transformer oil was proposed at first because of its exceptional cooling capability along with its high combustion temperature that would create a safe testing environment. It also does not cause any type of corrosion or rusting due to its ability to prevent oxidation. Even though transformer oil had all the qualities that were desired, mineral oil was the best solution because it is cost friendly and possessed the same traits as transformer oil just a lower quality. Also, transformer oil is more expensive due to the extra steps needed to refine it.

To test the solution, GRC's GreenDEF mineral oil blend was desired. Unfortunately, the oil was unavailable to us because GRC said they were not in a position to donate any oil, but they provided guidance by saying any brand of mineral oil should work for the testing procedure.

UPPCO was then contacted requesting transformer oil. Liability costs caused them to be hesitant about donating oil due to the possibility of a toxic chemical being in the oil called Polychlorinated Biphenyls (PCB). After many weeks of waiting, UPPCO denied the opportunity of donating oil.

We finally obtained a brand of oil called "AGRIpharm Mineral Oil 95 Viscosity" which was purchased from a local feed store. This mineral oil is used as an animal laxative. However, after testing it by submerging a WRT54G wireless router in oil, we found that it had no flaws and it was feasible for the experiment. This was beneficial because all other brands of mineral oil cost about $18.00/gallon as opposed to this brand which was $13.00 a gallon. The budget allowed for the group to get 31 gallons: 1 gallon for the initial test of oil quality and 30 gallons to fill the tank.

III. Testing Procedures, Problems and Solutions

To ensure that no devices critical to the testing environment would be damaged after submerging them into oil, a testing procedure was implemented to minimize loss. The first device that was chosen to be immersed in the oil was a Linksys WRT54G router. After a successful test, the decision was made to move forward with submerging the two testing servers into the tank.

Following the start of the last stage in the procedure, events did not unfold exactly as anticipated. After starting sub00 while submerged, the server proceeded to start its POST then immediately shutdown. After further inspection, it was realized that the PSU fans were spinning irregularly and may have caused the abrupt shutdown. To confirm this, a second test which immobilized the PSU fan was conducted on air01, and as expected produced the same results. The reasoning for sub01 unexpected shutdown was determined to be caused by the viscosity of the mineral oil.

The next course of action chosen to alleviate the problem of the malfunctioning PSUfans was to extend the power and data lines length so the fans could sit outside of the tank and spin freely. However, when these changes were applied, sub01 still did not start (although it did get to grub).

Due to this inconsistency in the boot sequence, the team determined that the only adequate solution would be newer servers that have a more flexible BIOS for modifications.

IV. Conclusion

Overall the group is very pleased with the accomplishments we made during the year on this project. Although we were not able to effectively test this cooling solution ourselves it is currently being used in both industry and in home environments. We believe that we have laid a good foundation for this project to be carried on in the future. As most of the problems that we encountered were directly related to old and outdated hardware.

Appendix

Appendix A: Network Diagrams
Appendix B: MySQL Database Diagram
Appendix C: MySQL DatabaseCreqte.sql
Appendix D: MySQL UserCreate.sql
Appendix E: get_air00.sh
Appendix F: send_air00.sh
Appendix G: GetAddress.pde
Appendix H: GetTemps.pde

REFERENCES

[1] Arduino. (2010, Oct. 26). Arduino – Arduino BoardMega2560 [Online]. Available http://arduino.cc/en/Main/ArduinoBoardMega2560

[2] Arduino. (2010, Oct. 26). Arduino – Home Page [Online]. Available http://www.arduino.cc

[3] Maxim. (2010, Nov. 16). DS18S20-PAR.pdf [Online]. Available http://pdfserv.maxim-ic.com/en/ds/DS18S20-PAR.pdf

[4] M. Mienik. (2011, Jan. 20). CPU Burin-in Homepage [Online]. Available http://www.cpuburnin.com

[5] A. Trujillo. "Data Center Liquid Cooling vs. Forced Air cooling." Data Center: Systems Management News and advice for Data Center Managers. Feb. 19, 2007

[6] GRCooling. (2010, Sept. 25). Green Revolution Cooling [Online]. Available http://grcooling.com/?page_id=70

[7] Pugetsystems. (2011, Mar. 15). Mineral Oil Submerged Computer; Our Most Popular Custom PC [Online]. Available http://www.pugetsystems.com/submerged.php

[8] APC. (2010, Oct. 26). "Determining Total Cost of Ownership for Data Center and Network Room Infrastructure." Linux Labs. Available http://www.linuxlabs.com/PDF/Data%20Center%20Cost%20of%20Ownership.pdf

Appendix A

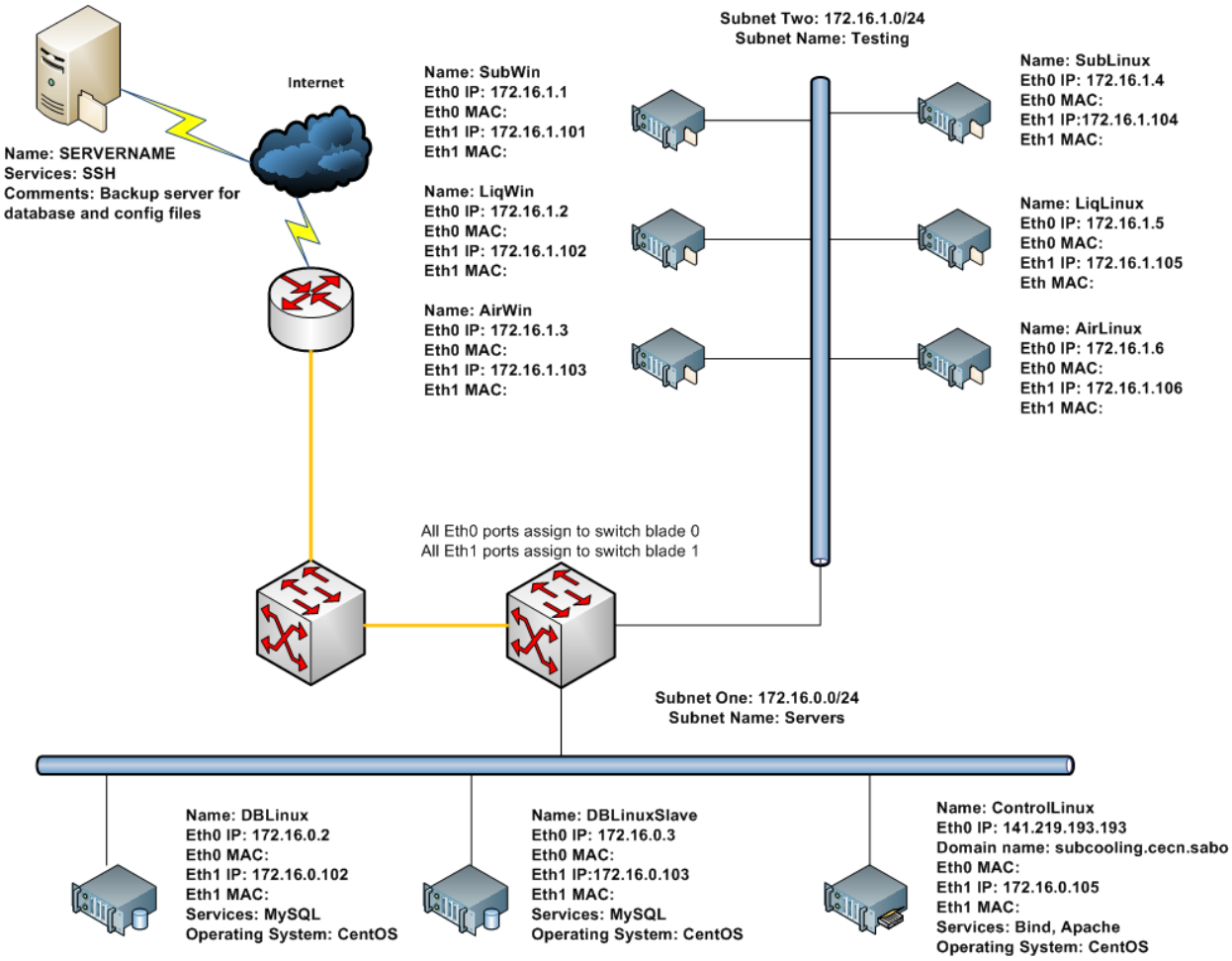


Figure 1

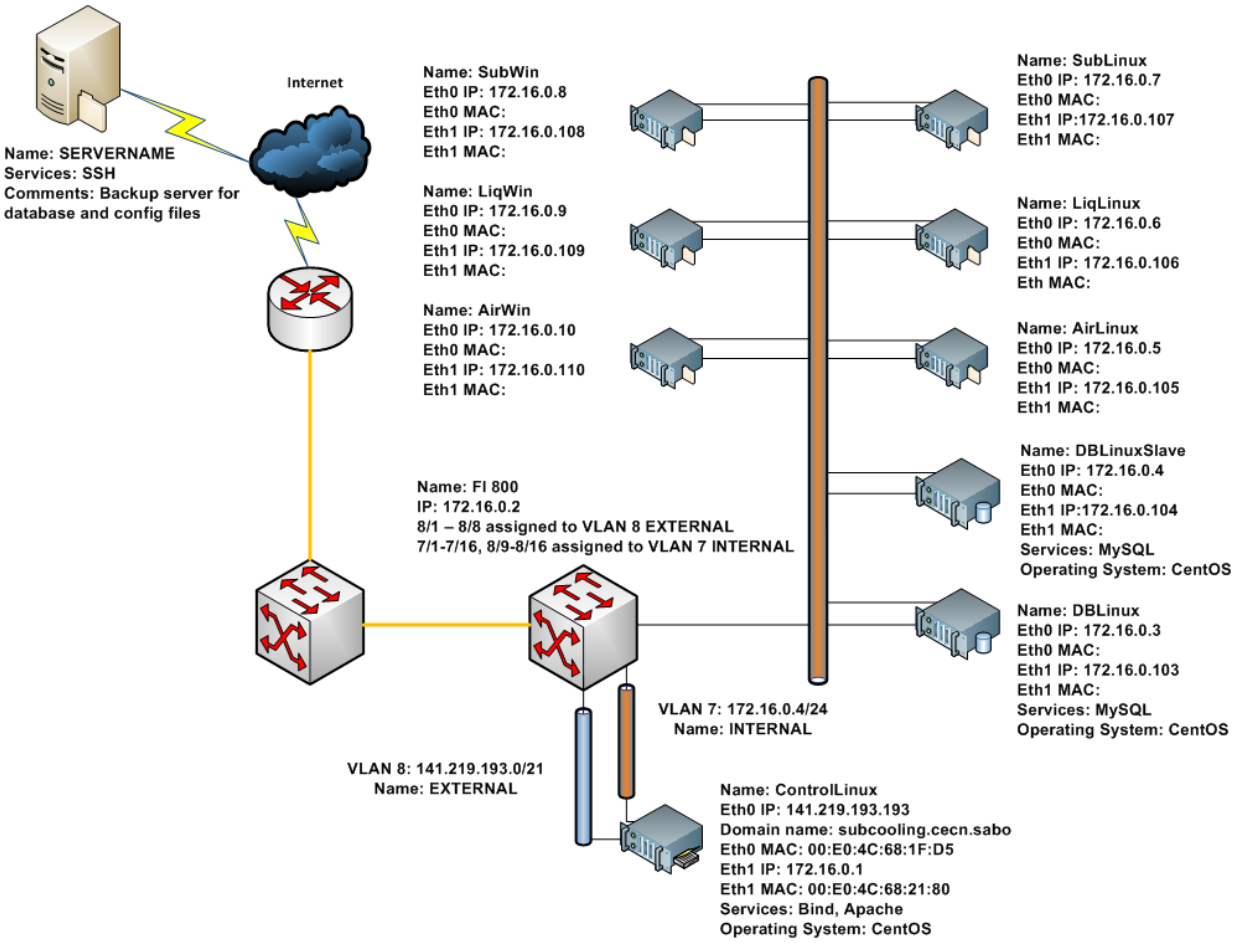

Name: SERVERNAME
Services: SSH
Comments: Backup server for database and config files
Internet
Name: SubWin
Eth0 IP: 172.16.0.8
Eth0 MAC:
Eth1 IP: 172.16.0.108
Eth1 MAC:
Name: LiqWin
Eth0 IP: 172.16.0.9
Eth0 MAC:
Eth1 IP: 172.16.0.109
Eth1 MAC:
Name: AirWin
Eth0 IP: 172.16.0.10
Eth0 MAC:
Eth1 IP: 172.16.0.110
Eth1 MAC:
Name: SubLinux
Eth0 IP: 172.16.0.7
Eth0 MAC:
Eth1 IP:172.16.0.107
Eth1 MAC:
Name: LiqLinux
Eth0 IP: 172.16.0.6
Eth0 MAC:
Eth1 IP: 172.16.0.106
Eth MAC:
Name: AirLinux
Eth0 IP: 172.16.0.5
Eth0 MAC:
Eth1 IP: 172.16.0.105
Eth1 MAC:
Name: DBLinuxSlave
Eth0 IP: 172.16.0.4
Eth0 MAC:
Eth1 IP:172.16.0.104
Eth1 MAC:
Services: MySQL
Operating System: CentOS
Name: DBLinux
Eth0 IP: 172.16.0.3
Eth0 MAC:
Eth1 IP: 172.16.0.103
Eth1 MAC:
Services: MySQL
Operating System: CentOS
Name: FI 800
IP: 172.16.0.2
8/1 – 8/8 assigned to VLAN 8 EXTERNAL
7/1-7/16, 8/9-8/16 assigned to VLAN 7 INTERNAL
VLAN 7: 172.16.0.4/24
Name: INTERNAL
VLAN 8: 141.219.193.0/21
Name: EXTERNAL
Name: ControlLinux
Eth0 IP: 141.219.193.193
Domain name: subcooling.cecn.sabo
Eth0 MAC: 00:E0:4C:68:1F:D5
Eth1 IP: 172.16.0.1
Eth1 MAC: 00:E0:4C:68:21:80
Services: Bind, Apache
Operating System: CentOS

Figure 2

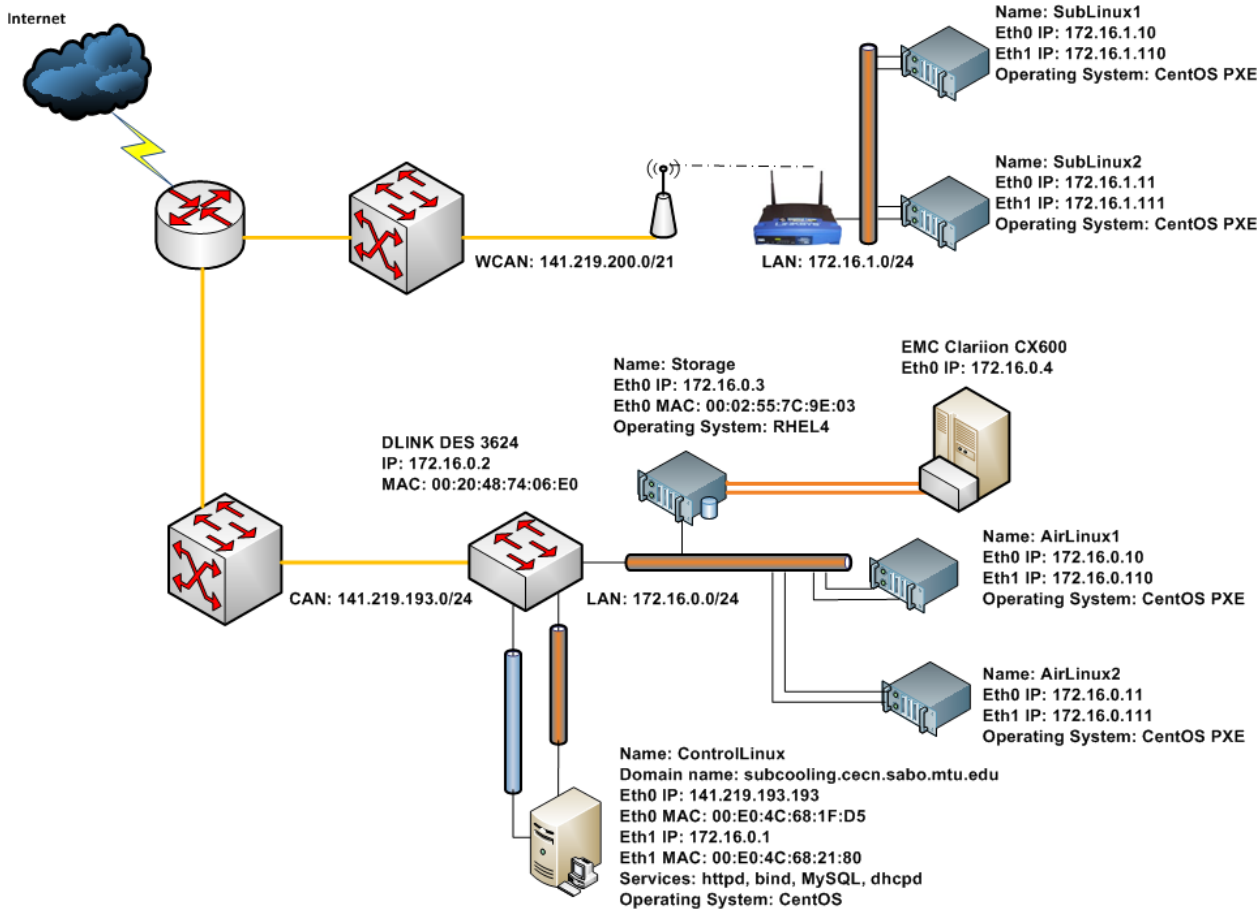

Internet
Name: SubLinux1
Eth0 IP: 172.16.1.10
Eth1 IP: 172.16.1.110
Operating System: CentOS PXE
Name: SubLinux2
Eth0 IP: 172.16.1.11
Eth1 IP: 172.16.1.111
Operating System: CentOS PXE
WCAN: 141.219.200.0/21
LAN: 172.16.1.0/24
EMC Clariion CX600
Eth0 IP: 172.16.0.4
Name: Storage
Eth0 IP: 172.16.0.3
Eth0 MAC: 00:02:55:7C:9E:03
Operating System: RHEL4
DLINK DES 3624
IP: 172.16.0.2
MAC: 00:20:48:74:06:E0
CAN: 141.219.193.0/24
LAN: 172.16.0.0/24
Name: AirLinux1
Eth0 IP: 172.16.0.10
Eth1 IP: 172.16.0.110
Operating System: CentOS PXE
Name: AirLinux2
Eth0 IP: 172.16.0.11
Eth1 IP: 172.16.0.111
Operating System: CentOS PXE
Name: ControlLinux
Domain name: subcooling.cecn.sabo.mtu.edu
Eth0 IP: 141.219.193.193
Eth0 MAC: 00:E0:4C:68:1F:D5
Eth1 IP: 172.16.0.1
Eth1 MAC: 00:E0:4C:68:21:80
Services: httpd, bind, MySQL, dhcpd
Operating System: CentOS

Figure 3

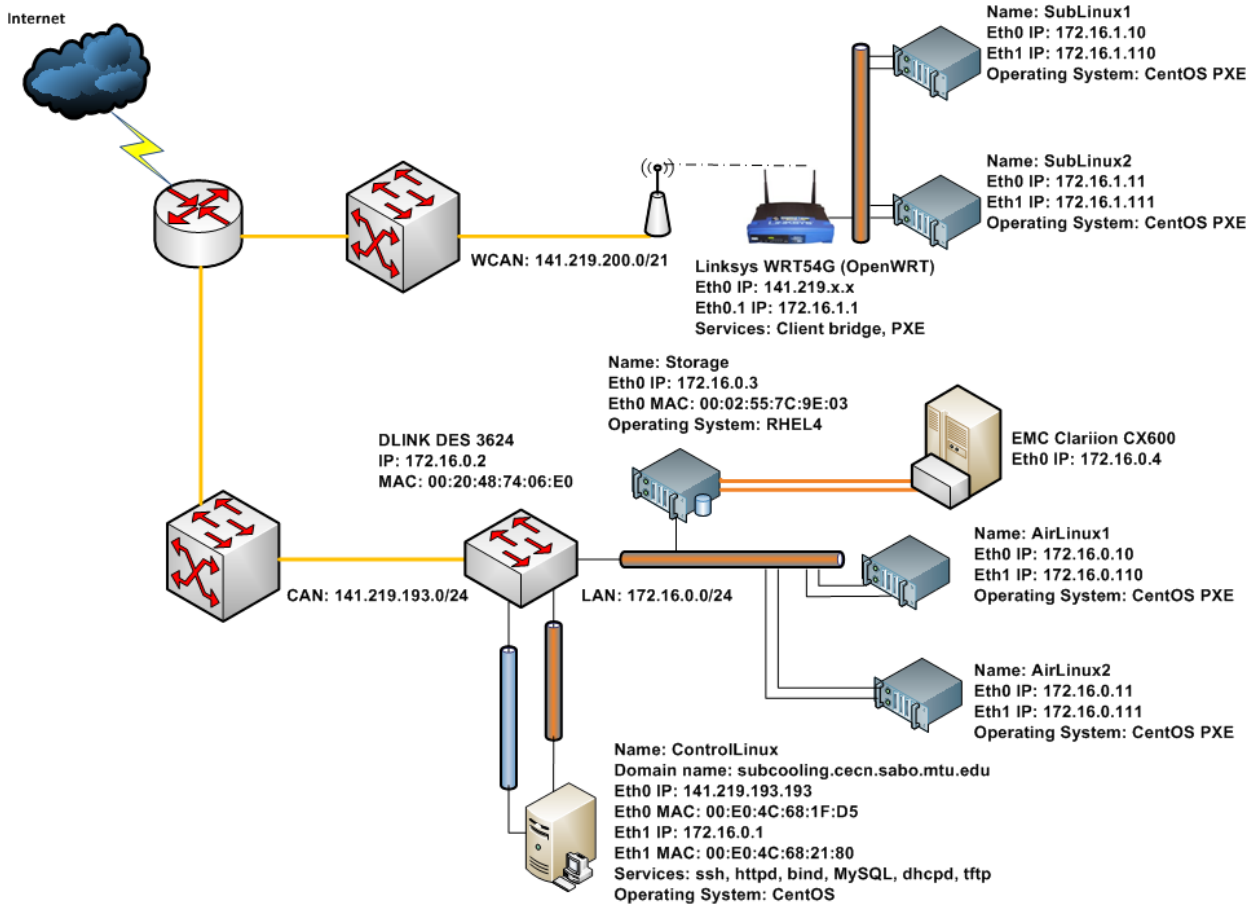

Internet
Name: SubLinux1
Eth0 IP: 172.16.1.10
Eth1 IP: 172.16.1.110
Operating System: CentOS PXE
Name: SubLinux2
Eth0 IP: 172.16.1.11
Eth1 IP: 172.16.1.111
Operating System: CentOS PXE
WCAN: 141.219.200.0/21
Linksys WRT54G (OpenWRT)
Eth0 IP: 141.219.x.x
Eth0.1 IP: 172.16.1.1
Services: Client bridge, PXE
Name: Storage
Eth0 IP: 172.16.0.3
Eth0 MAC: 00:02:55:7C:9E:03
Operating System: RHEL4
DLINK DES 3624
IP: 172.16.0.2
MAC: 00:20:48:74:06:E0
EMC Clariion CX600
Eth0 IP: 172.16.0.4
CAN: 141.219.193.0/24
LAN: 172.16.0.0/24
Name: AirLinux1
Eth0 IP: 172.16.0.10
Eth1 IP: 172.16.0.110
Operating System: CentOS PXE
Name: AirLinux2
Eth0 IP: 172.16.0.11
Eth1 IP: 172.16.0.111
Operating System: CentOS PXE
Name: ControlLinux
Domain name: subcooling.cecn.sabo.mtu.edu
Eth0 IP: 141.219.193.193
Eth0 MAC: 00:E0:4C:68:1F:D5
Eth1 IP: 172.16.0.1
Eth1 MAC: 00:E0:4C:68:21:80
Services: ssh, httpd, bind, MySQL, dhcpd, tftp
Operating System: CentOS

Figure 4

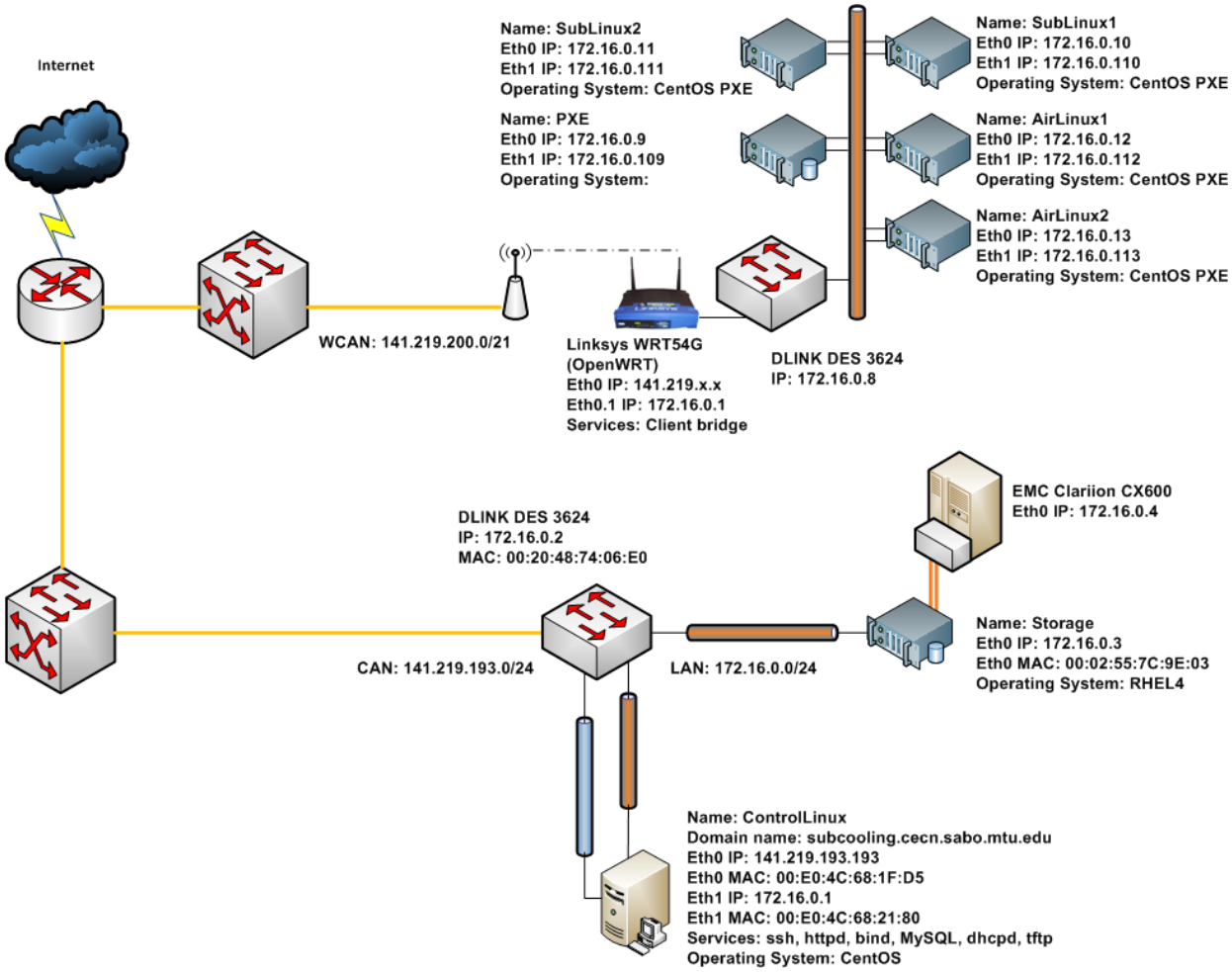

Internet
Name: SubLinux2
Eth0 IP: 172.16.0.11
Eth1 IP: 172.16.0.111
Operating System: CentOS PXE
Name: PXE
Eth0 IP: 172.16.0.9
Eth1 IP: 172.16.0.109
Operating System:
Name: SubLinux1
Eth0 IP: 172.16.0.10
Eth1 IP: 172.16.0.110
Operating System: CentOS PXE
Name: AirLinux1
Eth0 IP: 172.16.0.12
Eth1 IP: 172.16.0.112
Operating System: CentOS PXE
Name: AirLinux2
Eth0 IP: 172.16.0.13
Eth1 IP: 172.16.0.113
Operating System: CentOS PXE
WCAN: 141.219.200.0/21
Linksys WRT54G
(OpenWRT)
Eth0 IP: 141.219.x.x
Eth0.1 IP: 172.16.0.1
Services: Client bridge
DLINK DES 3624
IP: 172.16.0.8
DLINK DES 3624
IP: 172.16.0.2
MAC: 00:20:48:74:06:E0
EMC Clariion CX600
Eth0 IP: 172.16.0.4
Name: Storage
Eth0 IP: 172.16.0.3
Eth0 MAC: 00:02:55:7C:9E:03
Operating System: RHEL4
CAN: 141.219.193.0/24
LAN: 172.16.0.0/24
Name: ControlLinux
Domain name: subcooling.cecn.sabo.mtu.edu
Eth0 IP: 141.219.193.193
Eth0 MAC: 00:E0:4C:68:1F:D5
Eth1 IP: 172.16.0.1
Eth1 MAC: 00:E0:4C:68:21:80
Services: ssh, httpd, bind, MySQL, dhcpd, tftp
Operating System: CentOS

Figure 5

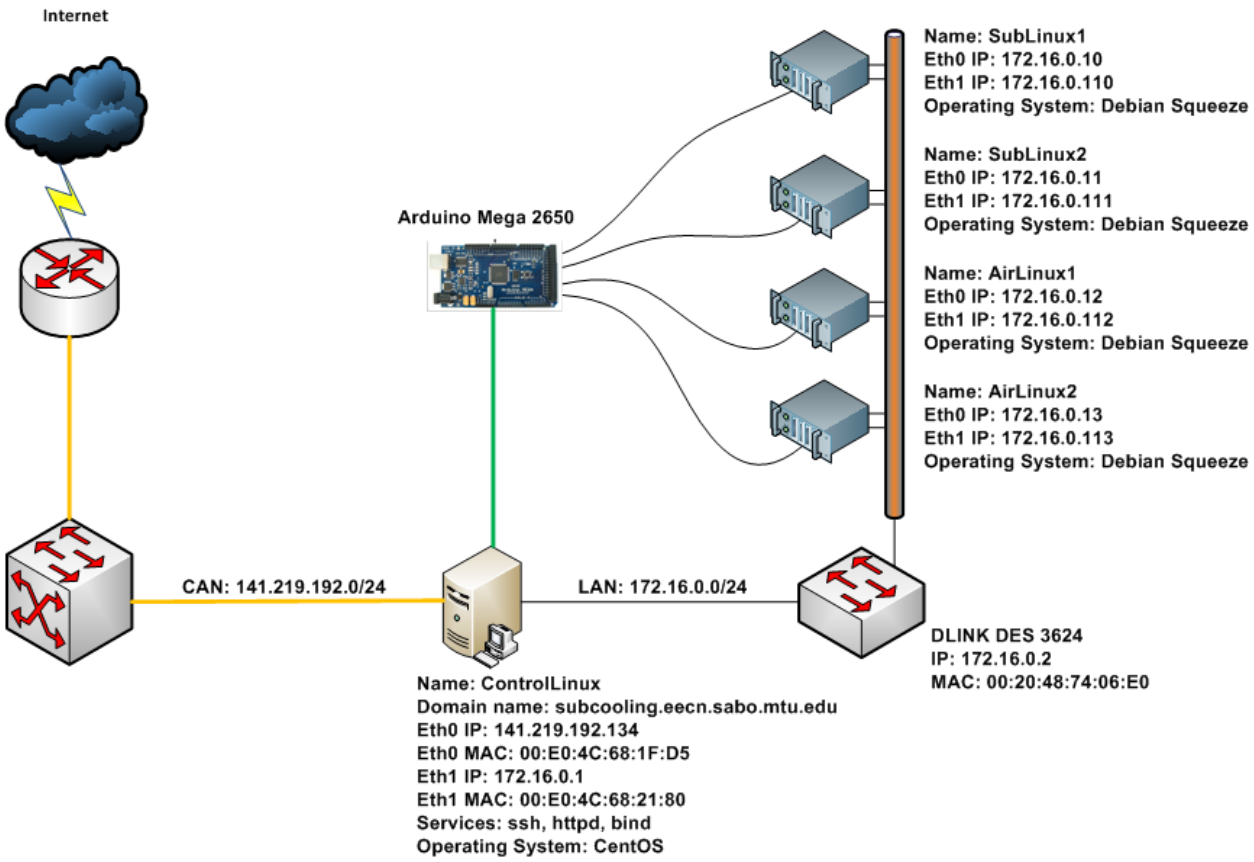
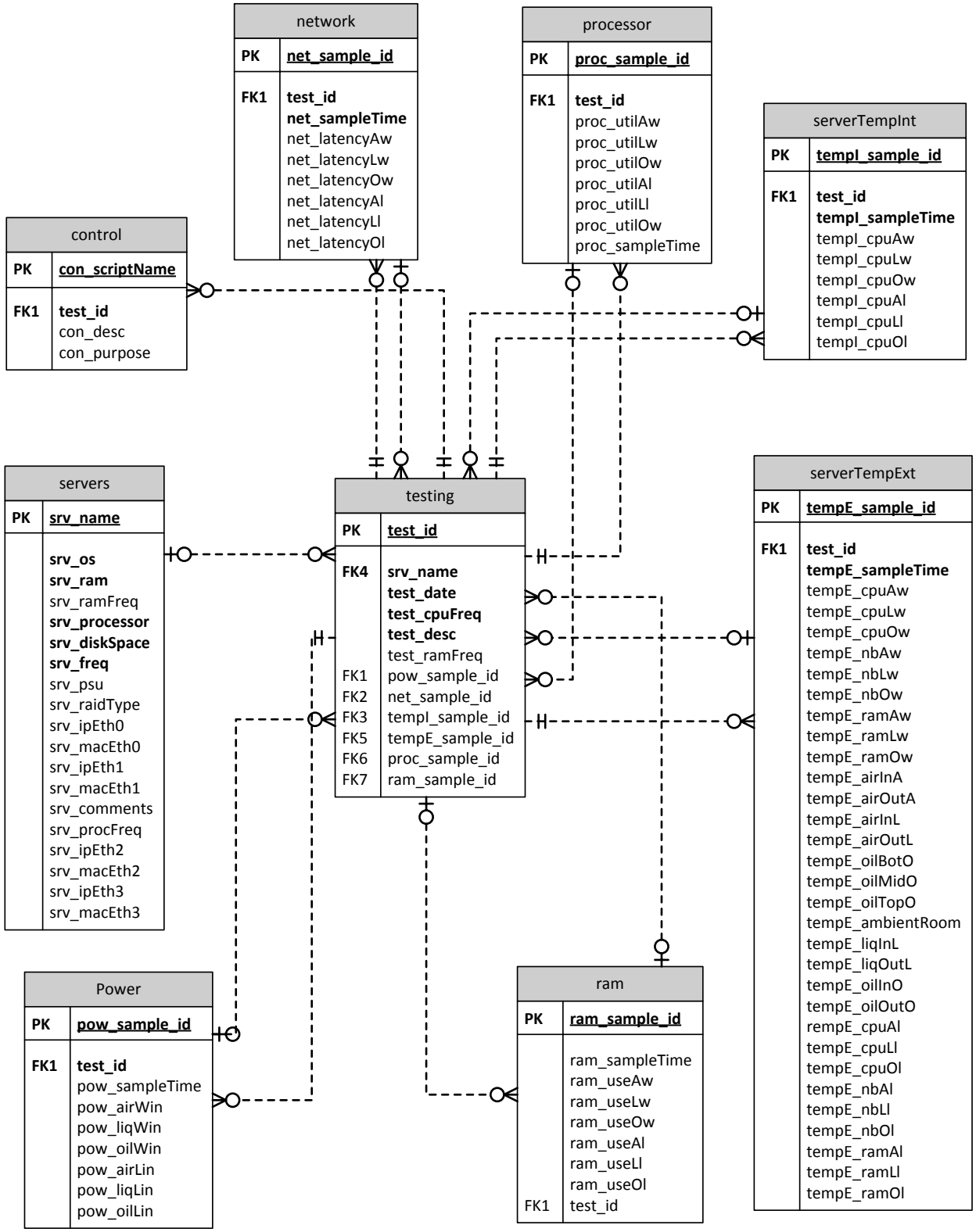

Internet
Arduino Mega 2650
Name: SubLinux1
Eth0 IP: 172.16.0.10
Eth1 IP: 172.16.0.110
Operating System: Debian Squeeze
Name: SubLinux2
Eth0 IP: 172.16.0.11
Eth1 IP: 172.16.0.111
Operating System: Debian Squeeze
Name: AirLinux1
Eth0 IP: 172.16.0.12
Eth1 IP: 172.16.0.112
Operating System: Debian Squeeze
Name: AirLinux2
Eth0 IP: 172.16.0.13
Eth1 IP: 172.16.0.113
Operating System: Debian Squeeze
CAN: 141.219.192.0/24
LAN: 172.16.0.0/24
DLINK DES 3624
IP: 172.16.0.2
MAC: 00:20:48:74:06:E0
Name: ControlLinux
Domain name: subcooling.eecn.sabo.mtu.edu
Eth0 IP: 141.219.192.134
Eth0 MAC: 00:E0:4C:68:1F:D5
Eth1 IP: 172.16.0.1
Eth1 MAC: 00:E0:4C:68:21:80
Services: ssh, httpd, bind
Operating System: CentOS

Figure 6

Appendix B


network
PK net_sample_id
FK1 test_id
net_sampleTime
net_latencyAw
net_latencyLw
net_latencyOw
net_latencyAl
net_latencyLl
net_latencyOl
processor
PK proc_sample_id
FK1 test_id
proc_utilAw
proc_utilLw
proc_utilOw
proc_utilAl
proc_utilLl
proc_utilOw
proc_sampleTime
serverTempInt
PK tempI_sample_id
FK1 test_id
tempI_sampleTime
tempI_cpuAw
tempI_cpuLw
tempI_cpuOw
tempI_cpuAl
tempI_cpuLl
tempI_cpuOl
control
PK con_scriptName
FK1 test_id
con_desc
con_purpose
servers
PK srv_name
srv_os
srv_ram
srv_ramFreq
srv_processor
srv_diskSpace
srv_freq
srv_psu
srv_raidType
srv_ipEth0
srv_macEth0
srv_ipEth1
srv_macEth1
srv_comments
srv_procFreq
srv_ipEth2
srv_macEth2
srv_ipEth3
srv_macEth3
testing
PK test_id
FK4 srv_name
test_date
test_cpuFreq
test_desc
test_ramFreq
FK1 pow_sample_id
FK2 net_sample_id
FK3 tempI_sample_id
FK5 tempE_sample_id
FK6 proc_sample_id
FK7 ram_sample_id
serverTempExt
PK tempE_sample_id
FK1 test_id
tempE_sampleTime
tempE_cpuAw
tempE_cpuLw
tempE_cpuOw
tempE_nbAw
tempE_nbLw
tempE_nbOw
tempE_ramAw
tempE_ramLw
tempE_ramOw
tempE_airInA
tempE_airOutA
tempE_airInL
tempE_airOutL
tempE_oilBotO
tempE_oilMidO
tempE_oilTopO
tempE_ambientRoom
tempE_liqInL
tempE_liqOutL
tempE_oilInO
tempE_oilOutO
rempE_cpuAl
tempE_cpuLl
tempE_cpuOl
tempE_nbAl
tempE_nbLl
tempE_nbOl
tempE_ramAl
tempE_ramLl
tempE_ramOl
Power
PK pow_sample_id
FK1 test_id
pow_sampleTime
pow_airWin
pow_liqWin
pow_oilWin
pow_airLin
pow_liqLin
pow_oilLin
ram
PK ram_sample_id
ram_sampleTime
ram_useAw
ram_useLw
ram_useOw
ram_useAl
ram_useLl
ram_useOl
FK1 test_id

Appendix C

```
/* DatabaseCreate.sql */
/* Create Subcooling database */


/* Drop tables */

DROP TABlE IF EXISTS testing CASCADE;
DROP TABLE IF EXISTS servers CASCADE;
DROP TABLE IF EXISTS network CASCADE;
DROP TABLE IF EXISTS processor CASCADE;
DROP TABLE IF EXISTS serverTempInt CASCADE;
DROP TABLE IF EXISTS serverTempExt CASCADE;
DROP TABLE IF EXISTS ram CASCADE;
DROP TABLE IF EXISTS power CASCADE;


/* Create test table. This table is used to organize all of the tests */

CREATE TABLE testing (
        test_id                 INT             AUTO_INCREMENT PRIMARY
KEY,
        test_date               DATE            NOT NULL,
        test_cpuFreq            FLOAT           NOT NULL,
        test_ramFreq            FLOAT,
        test_desc               VARCHAR(150),
        pow_sample_id           INT,
        net_sample_id           INT,
        tempI_sample_id         INT,
        tempE_sample_id         INT,
        proc_sample_id          INT,
        ram_sample_id           INT,
        srv_name                VARCHAR(20)
)ENGINE=INNODB;


/* Create servers table */

CREATE TABLE servers (
        srv_name                VARCHAR(20)     PRIMARY KEY,
        srv_os                  VARCHAR(20),
        srv_ram                 INT,
        srv_ramFreq             FLOAT,
        srv_processor           FLOAT,
        srv_procFreq            FLOAT,
        srv_raidType            INT,
        srv_ipEth0              VARCHAR(15),
        srv_macEth0             VARCHAR(20),
        srv_ipEth1              VARCHAR(15),
        srv_macEth1             VARCHAR(20),
        srv_ipEth2              VARCHAR(15),
        srv_macEth2             VARCHAR(20),
        srv_ipEth3              VARCHAR(15),
```

```
        srv_macEth3                 VARCHAR(20),
        srv_diskSpace               FLOAT,
        srv_psu                     FLOAT,
        srv_comments                VARCHAR(150)
)ENGINE=INNODB;


/* Create network table */

CREATE TABLE network (
        net_sample_id               INT             AUTO_INCREMENT PRIMARY
KEY,
        test_id                     INT,
        net_sampleTime              DATE            NOT NULL,
        net_latencyAw               FLOAT,
        net_latencyLw               FLOAT,
        net_latencyOw               FLOAT,
        net_latencyAl               FLOAT,
        net_latencyLl               FLOAT,
        net_latencyOl               FLOAT
)ENGINE=INNODB;


/* Create processor table. */

CREATE TABLE processor (
        proc_sample_id              INT
      AUTO_INCREMENT PRIMARY KEY,
        test_id                     INT,
        proc_sampleTime             DATE            NOT NULL,
        proc_utilAw                 FLOAT,
        proc_utilLw                 FLOAT,
        proc_utilOw                 FLOAT,
        proc_utilAl                 FLOAT,
        proc_utilLl                 FLOAT,
        proc_utilOl                 FLOAT
)ENGINE=INNODB;


/* Create serverTempInt table */

CREATE TABLE serverTempInt (
        tempI_sample_id             INT             AUTO_INCREMENT PRIMARY
KEY,
        test_id                     INT,
        tempI_sampleTime            DATE            NOT NULL,
        tempI_nbAw                  FLOAT,
        tempI_nbLw                  FLOAT,
        tempI_nbOw                  FLOAT,
        tempI_nbAl                  FLOAT,
        tempI_nbLl                  FLOAT,
        tempI_nbOl                  FLOAT,
        tempI_cpuAw                 FLOAT,
        tempI_cpuLw                 FLOAT,
        tempI_cpuOw                 FLOAT,
        tempI_cpuAl                 FLOAT,
        tempI_cpuLl                 FLOAT,
```

```
        tempI_cpuOl                 FLOAT
)ENGINE=INNODB;


/* Create serverTempExt table */

CREATE TABLE serverTempExt (
        tempE_sample_id             INT                 AUTO_INCREMENT PRIMARY
KEY,
        test_id                     INT,
        tempE_sampleTime            DATE                NOT NULL,
        tempE_nbAw                  FLOAT,
        tempE_nbLw                  FLOAT,
        tempE_nbOw                  FLOAT,
        tempE_nbAl                  FLOAT,
        tempE_nbLl                  FLOAT,
        tempE_nbOl                  FLOAT,
        tempE_cpuAw                 FLOAT,
        tempE_cpuLw                 FLOAT,
        tempE_cpuOw                 FLOAT,
        tempE_cpuAl                 FLOAT,
        tempE_cpuLl                 FLOAT,
        tempE_cpuOl                 FLOAT,
        tempE_ramAw                 FLOAT,
        tempE_ramLw                 FLOAT,
        tempE_ramOw                 FLOAT,
        tempE_ramAl                 FLOAT,
        tempE_ramLl                 FLOAT,
        tempE_ramOl                 FLOAT,
        tempE_airInAw               FLOAT,
        tempE_airInLw               FLOAT,
        tempE_airOutAl              FLOAT,
        tempE_airOutLl              FLOAT,
        tempE_oilBotO               FLOAT,
        tempE_oilMidO               FLOAT,
        tempE_oilTopO               FLOAT,
        tempE_oilInO                FLOAT,
        tempE_oilOutO               FLOAT,
        tempE_liqInL                FLOAT,
        tempE_liqOutL               FLOAT,
        tempE_extraAw               FLOAT,
        tempE_extraLw               FLOAT,
        tempE_extraOw               FLOAT,
        tempE_extraAl               FLOAT,
        tempE_extraLl               FLOAT,
        tempE_extraOl               FLOAT,
        tempE_ambTemp               FLOAT,
        tempE_ampTempO              FLOAT
)ENGINE=INNODB;


/* Create ram table. */

CREATE TABLE ram (
        ram_sample_id               INT                 AUTO_INCREMENT PRIMARY
KEY,
        test_id                     INT,
```

```
        ram_sampleTime              DATE              NOT NULL,
        ram_useAw                   FLOAT,
        ram_useLw                   FLOAT,
        ram_useOw                   FLOAT,
        ram_useAl                   FLOAT,
        ram_useLl                   FLOAT,
        ram_useOl                   FLOAT
)ENGINE=INNODB;


/* Create power table */

CREATE TABLE power (
        pow_sample_id               INT               AUTO_INCREMENT PRIMARY
KEY,
        test_id                     INT,
        pow_sampleTime              DATE              NOT NULL,
        pow_airWin                  FLOAT,
        pow_liqWin                  FLOAT,
        pow_oilWin                  FLOAT,
        pow_airLin                  FLOAT,
        pow_liqLin                  FLOAT,
        pow_oilLin                  FLOAT
)ENGINE=INNODB;

/* Create control table */

/* CREATE TABLE control (
        con_scriptName              VARCHAR(20)       PRIMARY KEY,
        test_id                     INT,
        con_desc                    VARCHAR(50),
        con_purpose                 VARCHAR(50)
)ENGINE=INNODB; */

/* Create table index */

CREATE INDEX idx_test_id ON testing(test_id);
CREATE INDEX idx_srv_name ON servers(srv_name);
CREATE INDEX idx_proc_sample_id ON processor(proc_sample_id);
CREATE INDEX idx_tempI_sample_id ON serverTempInt(tempI_sample_id);
CREATE INDEX idx_tempE_sample_id ON serverTempExt(tempE_sample_id);
CREATE INDEX idx_ram_sample_id ON ram(ram_sample_id);
CREATE INDEX idx_pow_sample_id ON power (pow_sample_id);


/* Add foreign keys */

ALTER TABLE testing     ADD FOREIGN KEY (net_sample_id) REFERENCES network
(net_sample_id) ON DELETE NO ACTION;
ALTER TABLE testing     ADD FOREIGN KEY (tempI_sample_id) REFERENCES
serverTempInt (tempI_sample_id) ON DELETE NO ACTION;
ALTER TABLE testing     ADD FOREIGN KEY (tempE_sample_id) REFERENCES
serverTempExt (tempE_sample_id) ON DELETE NO ACTION;
ALTER TABLE testing     ADD FOREIGN KEY (proc_sample_id) REFERENCES processor
(proc_sample_id) ON DELETE NO ACTION;
ALTER TABLE testing     ADD FOREIGN KEY (ram_sample_id) REFERENCES ram
(ram_sample_id) ON DELETE NO ACTION;
```

```
ALTER TABLE testing     ADD FOREIGN KEY (srv_name) REFERENCES servers
(srv_name) ON DELETE NO ACTION;
ALTER TABLE testing     ADD FOREIGN KEY (pow_sample_id) REFERENCES power
(pow_sample_id) ON DELETE NO ACTION;
ALTER TABLE network ADD FOREIGN KEY (test_id) REFERENCES testing (test_id) ON
DELETE NO ACTION;
ALTER TABLE serverTempInt ADD FOREIGN KEY (test_id) REFERENCES testing
(test_id) ON DELETE NO ACTION;
ALTER TABLE serverTempExt ADD FOREIGN KEY (test_id) REFERENCES testing
(test_id) ON DELETE NO ACTION;
ALTER TABLE processor ADD FOREIGN KEY (test_id) REFERENCES testing (test_id)
ON DELETE NO ACTION;
ALTER TABLE ram ADD FOREIGN KEY (test_id) REFERENCES testing (test_id) ON
DELETE NO ACTION;
ALTER TABLE power ADD FOREIGN KEY (test_id) REFERENCES testing (test_id) ON
DELETE NO ACTION;
```

Appendix D

```
/* UserCreate.sql */
/* Create users and set priviliges */

/* Create users */

CREATE USER 'subcooling'@'localhost' IDENTIFIED BY 'subcoolingisgreat';
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'Sub123Cooling!@#';

/* Set priviliges */

GRANT INSERT ON 'subcooling'.* TO 'subcooling'@'localhost';
GRANT INSERT,ALTER,CREATE,DELETE,DROP,INDEX,INSERT,SELECT,UPDATE ON *.* TO
'admin'@'localhost';
```

Appendix E

```
#!/bin/bash
############################################################
#get_air00.sh
#Script to get the CPU and motherboard temperature from air00 and send
#the data back to the control server
############################################################

#Timestamp variables
TIME="`date +"%H%M"`,"
DATE="`date +"%e%b"`,"

#Set the server that this script is being sent to
HOST=air00

#Get temperature data from lm-sensors program, -f option for output in
Fahrenheit
sensors -f > /tmp/raw_$HOST.dat

#Print second field from raw_$HOST.dat and store in data_$HOST.dat
awk '{print $2}' /tmp/raw_$HOST.dat > /tmp/data_$HOST.dat

#Delete lines 1-27 and 29-31 from temp_$HOST.dat, remove the +, °, and  F,
and store the result as CPUTEMP
CPUTEMP="`sed -e '1,27d' -e '29,31d' -e 's/+//' -e 's/°F//'
/tmp/data_$HOST.dat`,"

#Delete lines 1-26 and 28-31 from temp_$HOST.dat, remove the +, °, and  F,
and store the result as BTEMP
BTEMP="`sed -e '1,25d' -e '27,31d' -e 's/+//' -e 's/°F//'
/tmp/data_$HOST.dat`"

#Output data to a .csv file on the control server
echo $DATE$TIME$CPUTEMP$BTEMP | ssh root@172.16.0.1 "cat >>
/Testing/temps/results/$HOST.csv"

#Notify us if the server is overheating
if [ $CPUTEMP –gt 150 ]; then
echo "From: monitor" > /tmp/mail.txt
echo "To: sat4480s-l@mtu.edu" >> /tmp/mail.txt
echo "Subject:"$HOST" is  overheating" >> /tmp/mail.txt
echo " " >> /tmp/mail.txt
echo "Temperature has reached "$CPUTEMP"°F" >> /tmp/mail.txt
/usr/sbin/sendmail -t < /tmp/mail.txt
fi

#Clean up the temporary files
rm -f /tmp/data_$HOST.dat /tmp/raw_$HOST.dat /tmp/mail.txt
```

Appendix F

```
#!/bin/sh
############################################################
#send_air00.sh
#Script send the get_air00 script to the test server.
#The get script will send back the results
############################################################

cat /Testing/temps/scripts/get/get_air00.sh | ssh root@air00



#Example output file
#air00.csv
Date, Time, CPU, MB
24Mar,0106,73.4,68.0
24Mar,0109,73.4,68.0
24Mar,0110,73.5,68.0
24Mar,0110,73.5,68.0
```

Appendix G

/* GetAddress */

```
#include <OneWire.h>
#include <DallasTemperature.h>

// Data wire is plugged into port 2 on the Arduino
#define ONE_WIRE_BUS 3

// Setup a oneWire instance to communicate with any OneWire devices (not just
Maxim/Dallas temperature ICs)
OneWire oneWire(ONE_WIRE_BUS);

// Pass our oneWire reference to Dallas Temperature.
DallasTemperature sensors(&oneWire);

// arrays to hold device address
DeviceAddress thermometer;

void setup(void)
{
  // start serial port
  Serial.begin(9600);
  Serial.println("One Wire address Finder");

  // locate devices on the bus
  Serial.print("Locating devices...");
  sensors.begin();
  Serial.println("Found ");

  sensors.getAddress(thermometer, 0);
  printAddress(thermometer);
  Serial.println();

  // set the resolution to 9 bit (Each Dallas/Maxim device is capable of
several different resolutions)
  sensors.setResolution(thermometer, 9);

}

// function to print the temperature for a device
void printTemperature(DeviceAddress deviceAddress)
{
  float tempC = sensors.getTempC(deviceAddress);
  Serial.print("Temp C: ");
  Serial.print(tempC);
}

void loop(void)
{
  // call sensors.requestTemperatures() to issue a global temperature
  // request to all devices on the bus
  Serial.print("Requesting temperatures...");
```

```
  sensors.requestTemperatures(); // Send the command to get temperatures
  Serial.println("DONE");

  printTemperature(thermometer);
  Serial.println("");
}

// function to print a device address
void printAddress(DeviceAddress deviceAddress)
{
  for (uint8_t i = 0; i < 8; i++)
  {
    if (deviceAddress[i] < 16) Serial.print("0");
    Serial.print(deviceAddress[i], HEX);
  }
}
```

Appendix H

/* GetTemps.pde */

```
#include <OneWire.h>
#include <DallasTemperature.h>

// Data wire is plugged into port 3 on the Arduino
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 9
#define THERMOMETERS_MAX 1
#define HUMAN 0
#define MACHINE 1

//Select output type, 0 for human readable, 1 for CSV
#define OUTPUT_TYPE 0

// Setup a oneWire instance to communicate with any OneWire devices (not just
Maxim/Dallas temperature ICs)
OneWire oneWire(ONE_WIRE_BUS);

// Pass our oneWire reference to Dallas Temperature.
DallasTemperature sensors(&oneWire);
// arrays to hold device addresses
DeviceAddress thermometer[] = {{0x10, 0x5B, 0x18, 0x25, 0x02, 0x08, 0x00,
0xA7}};
// This is one device addres {0x10, 0x5B, 0x18, 0x25, 0x02, 0x08, 0x00, 0xA7}

int x;

void setup(void)
{

  // start serial port
  Serial.begin(9600);

  // Start up the library
  sensors.begin();

  if(OUTPUT_TYPE == HUMAN){
    // locate devices on the bus
    Serial.print("Locating devices...");
    Serial.print("Found ");
    Serial.print(sensors.getDeviceCount(), DEC);
    Serial.println(" devices.");

    //Print all devices found
    for(x=0; x<THERMOMETERS_MAX;x++)
    {
      Serial.print("Device ");
      Serial.print(x);
      Serial.print(" Address:");
      printAddress(thermometer[x]);
      Serial.println();
```

```
    }
  }else
  {
    Serial.print("DEVICES ");
    Serial.println(sensors.getDeviceCount(), DEC);
    //Print all devices found
    for(x=0; x<THERMOMETERS_MAX;x++)
    {
      printAddress(thermometer[x]);
      Serial.print(";");
    }
    Serial.println("");
  }
  //Set 9 bit resolution for all devices
  for(x=0; x<THERMOMETERS_MAX;x++) sensors.setResolution(thermometer[x], 9);
}

// function to print a device address
void printAddress(DeviceAddress deviceAddress)
{
  for (uint8_t i = 0; i < 8; i++)
  {
    // zero pad the address if necessary
    if (deviceAddress[i] < 16) Serial.print("0");
    Serial.print(deviceAddress[i], HEX);
  }
}

// function to print the temperature for a device
void printTemperature(DeviceAddress deviceAddress)
{
  float tempC = sensors.getTempC(deviceAddress);
  Serial.print("Temp C: ");
  Serial.print(tempC);
  Serial.print(" Temp F: ");
  Serial.print(DallasTemperature::toFahrenheit(tempC));
}
//function to print temp in degC for parsing
void printTemperatureCondensed(DeviceAddress deviceAddress)
{
  Serial.print(sensors.getTempC(deviceAddress));
  Serial.print(";");
}
// function to print a device's resolution
void printResolution(DeviceAddress deviceAddress)
{
  Serial.print("Resolution: ");
  Serial.print(sensors.getResolution(deviceAddress));
  Serial.println();
}

// main function to print information about a device
void printData(DeviceAddress deviceAddress)
{
  Serial.print("Device Address: ");
  printAddress(deviceAddress);
  Serial.print(" ");
```

```
  printTemperature(deviceAddress);
  Serial.println();
}

void humanLoop(){
    // call sensors.requestTemperatures() to issue a global temperature
  // request to all devices on the bus
  Serial.print("Requesting temperatures...");
  sensors.requestTemperatures();
  Serial.println("DONE");
  // print device info
  for(x=0; x<THERMOMETERS_MAX;x++) printData(thermometer[x]);
}
void machineLoop(){
    // call sensors.requestTemperatures() to issue a global temperature
  // request to all devices on the bus
  Serial.println("REQUEST");
  sensors.requestTemperatures();

  //print condensed list
  for(x=0; x<THERMOMETERS_MAX;x++) printTemperatureCondensed(thermometer[x]);
  Serial.println("");
}
void loop(void)
{
  if(OUTPUT_TYPE == MACHINE){
    machineLoop();}else humanLoop();
}
```