**4 Physical Deployment on the Raspberry Pi**

One of the earliest problems screenPerfect has been compromises in how the software is served to players and artists. Reliance on public internet is difficult, because the internet is not always available. WiFi is taken for granted in most institutions, but it is not always reliable, and the most interesting installation zones, such as the forest, may not have internet available at all.

There are other challenges to public deployment, such as leaving a valuable production environment out in public, or requiring a technician to look in on specialized equipment. Both of these restrict the venues permitted for public display of work. Also limiting are instances where work should be deployed near people consuming alcohol, which is notoriously bad for electronics.

This chapter addresses my central idea on how to repair the gap between excellent idea for web-based deployment, and the physical reality of gallery spaces with sharply limited resources available for persistent software deployment.

**4.1 Problems and Complications in Display**

These issues have been revealed through various installations of screenPerfect in public space. Exhibiting in a van, in a park, and in an institution with limited bandwidth revealed a clear set of questions around improving installation circumstances. screenPerfect needed to be an arcade box, similar to those used for years in fighting games in bars.

Figure 4.1: psXXYborg van installation

Figure 4.2: Street Fighter 2 Cabinet

Some brainstorming resulted in the following scenarios that a reasonable arcade machine would need to address in order to present advanced work to the new, highly exclusive exhibit scenarios.

1. Data service to an external source cannot be assumed to be available.

2. The exhibit is assumed to be displayed in public

3. The environment is assumed to be meterologically hostile - hot or cold, wet or very dry, and to be hosting at least one party, such as an art opening, possibly with music

4. The exhibition is assumed to be supervised by technically untrained people.

5. The emphasis of the work should be on the work’s display, rather than on a laptop screen.

6. The collectors of the work are assumed to have extremely limited resources for ruggedized workstations.

7. Any host-provided data carriage for external connection - wiFi - is assumed to be overloaded by default.

These are all very real constraints that impact display of new media art. We use computers for work and play, but we still separate our lives into periods when we pursue one or the other, and we still have boundaries between our personal and public lives. To use the same machines to display art as we do to build the work is to reduce the work from something approachable to any other tab in a computer. New media works especially must be seen within their exhibition context to be understood.

4.2 Subnod.es and Public Private Space

This project has a precursor using similar technology builtat Eyebeam in New York in 2013. Subnod.es uses a captive portal similar to my own, based on the inexpensive Raspberry Pi framework, to display a chat client to only the local environment. The differences are substantial, although mainly located within the code. Subnod.es relies on an external DNS being made available via the actual subnod.es software, and depends on a different collection of software to serve the portal proper. It is also built such that those library dependencies are inseparable from the main project script.

The chief concern of subnod.es I have not yet mentioned: subnod.es was built as a response to concerns about communications privacy in North America under the NSA. Specifically, the author is concerned that people behave differently when they are watched, a subset of the concerns generally associated with panoptica and totalitarianism. While I have not specifically structured screenPerfect’s Art Portal to address these concerns, it has been built to be largely private. It serves an application to a limited selection of a public space.

The assumption of the Captive Portal Art Machine is that galleries have limited resources, but that people who go to art galleries almost certainly have access to a smart phone, which is a form of private space. Smart phones are people’s own homes, and are built to assume that they will stay with their owners at all times. This means that to install an app is to ask a lot of a viewer: specifically, it is to ask someone to bring an application into their private space without getting to sample it first. To contrast, serving that same application on the broad internet is to entirely delimit the context the art may be experienced within, which reduces its scarcity value to almost nothing while simultaneously removing the curator’s ability to set the context of an exhibition experience. This means that it’s unlikely an artist can be compensated in any conventional sense, despite their large audience, and also means that the curation of the exhibit is no different than the “curation” found on Tumblr. This seems to me to be a negative outcome.

A better outcome might be to make a limited public space available in a private context, and this is what we are doing when we ask that people open their phones and look at a website. The Internet is, famously, the new public space. By presenting a web application using public technology within the exclusive context of the gallery - or desert, or forest - we take control again over how our art is presented, and from there, how it can be consumed. A gallery or exhibit space can be set up very specifically for the benefit of an audience in a way that the internet in general cannot be, and web technologies are uniform and affordable in the way that more custom projection design software is not.

This sense of limited private space is key to the code-switching that human communication relies on. We are not the same people in public as we are in private, and we are again different people when we are in different publics, work to the street to school to the gallery. Technology that sensitively addresses these different code contexts seems likely to benefit its authors and its users both.

**4.3 Materials and Supplies**

**Raspberry Pi**

The Raspberry Pi is a full linux computer the size of a large credit card. A Raspi runs Debian linux off of a common SD card.

**32Gb SD Card for Raspberry Pi**

This is where we place the operating system and software for the Pi.

**USB wiFi dongle**

Edimax-based wiFi USB dongle, for serving wiFi hotspot on the Pi.

**USB flash memory**

For transferring or storing complete programs authored on external systems.

**Keyboard and Mouse**

For initial computer setup.

**Ethernet Cable**

Standard cat5 ethernet cable for programming remote.

**HDMI TV and cable**

Used as a monitor for the Raspberry Pi.

**Micro USB and power supply**

Power for the pi.

**Mac or PC computer with USB ports, ethernet port, SD Card reader**

Required for raspberry pi setup.

**4.4 Background for Linux Commands**

sudo means “do this now even if I appear to have insufficient user permissions” in Linux

apt-get is an inherited ”package manager” from Debian linux. ”Dependencies” are the software your software requires to run, Debian uses apt-get to manage them.

Things that follow sudo are commands.

**4.5 Setting Up The Raspberry Pi**

**4.5.1 Windows 7 SD Card setup and first boot**

This section is written for a Windows 7 environment, and is based on the common tutorial at http://learn.adafruit.com/adafruit-raspberry-pi-lesson-1-preparing-and-sd-card

1. Connect your main computer to the internet.
2. Download the most recent Raspbian distribution image from <http://www.raspberrypi.org/down>
3. Download Win32DiskImager from the greater internet. This is preferable because it allows you to write image backups to your harddrive.
4. Using Win32DiskImager, write your Raspbian distro to your SD card on your main computer.
5. Eject the microSD card and stick it into your RasPi.
6. Plug in your keyboard, and plug a mouse into your keyboard.
7. Plug in your HDMI cable and monitor. Turn them on.
8. Plug in the MicroUSB cable for power to your RasPi.

**4.5.2 Configuring Raspbian**

Once the RasPi is turning on, it needs to be set up to include all of its software. Turn the Pi on, and wait until the blue configuration screen comes up.

**Figure 4.3: Early RasPi Configuration Screen**

1. expandrootfs Expand the boot system so that you will not run out of onboard memory for software.
2. memorysplit Reduce the GPU to minimum, because we will be using the raspi as a headless server from the command line.
3. changepass Change the password so that your raspi will be less easy to hack.
4. ssh Enable SSH so that the pi will be accessible from an external computer.

When done, select finish to exit.

Type sudo reboot to restart the raspi.

**4.6 Software Setup for External WiFi Access**

A wiFi antennae can be used for one purpose at a time: it can either be used to access the external internet, for acquiring software to install into the raspi, or it can be used for serving a hotspot. It cannot do both at the same time. To load the pi up requires external access, so we will be loading that first. You must configure your wiFi before plugging in your wiFi antenna.

In Linux, there is warning you if you mistype a folder name, say, adding an ”s” to ”network” to make it ”networks.” If you would like to confirm your folder name is correct, try typing ”ls /etc/” to list the contents of that directory. Network is a default folder, and Interfaces is already present at first boot, so you can make sure your things are all there before you really get started. The way to tell you have done something wrong is if you type the below command and an empty new file opens. You are editing a file here, not creating one.

At your console prompt, type the following:

1 sudo nano / etc / network / interfaces

This opens a text editor. Enter the following into it.

1 auto lo

2

3 iface lo inet loopback

4 iface eth0 inet dhcp

5

6 allow - hotplug wlan0

7 auto wlan0

8

9 iface wlan0 inet dhcp

10 wpa - ssid " network name , commonly called an ssid , goes here "

11 wpa - psk " password "

Then type CTRL-X and Y to save your file.

1 sudo halt

Plug in your wifi antennae, pull your raspi’s power cable, and plug it back in. This should make the raspi’s antennae turn blue as it turns on. This little blue LED will frequently be the only way to tell something is going correctly or incorrectly, so it is an excellent tell that your machine is running.

Figure 4.4: Raspberry Pi with functioning wiFi antenna

If all went well, you’ve now connected to your own supply of wireless internet. This will not work if you are using an 802.1x network, such as those within OCADu. On your own home network, however, type:

1 sudo apt - get upgrade ; sudo apt -get update

This will upgrade your rasppi to whatever the latest agreed-upon package lists are, then update those packages to their most recent approved version.

**4.7 Installing Node.JS**

**4.7.1 Why Node?**

I’ve chosen to install Node because it is the software framework I selected to run the new game engine built in Part 1 of this thesis. Node is a new framework designed to get Javascript running on a server. There are advantages and disadvantages to this approach. The advantages are that JavaScript is a beautiful, minimal language that is relatively easy to learn. The disadvantages are that there is a heavy public bias against JS due to its years as a client-only language designed to manipulate what are known as Document Object Model (DOM) elements in-browser.

The brilliance of Node is that it replaces the need for a specific input-output window, re- placing that definition requirement with any internet browser. Node, backed by Google’s V8 engine, currently works best on Chrome, but it can interact with any browser.

Node is therefore easy to use, and easy to program for from the perspective of a mainly web based development chain.

**4.7.2 Installation Instructions for Node.JS**

Create a directory for Node to live in by typing the following at prompt.

1 sudo mkdir / opt / node

Acquire the node ”tarball” - compressed framework files - via the internet.

1 wget http :// nodejs . org / dist /v0 .10.2/ node -v0 .10.2 - linux -arm -pi. tar.gz

Unzip (desticky from tarball) it:

1 tar xvzf node -v0 .10.2 - linux -arm -pi. tar .gz

Copy the contents of the newly unzipped folder and paste them to your new directory. This leaves a copy of the tar and a copy of the unzipped tar at their original locations. You can probably remove them using sudo rm when you’re sure everything is where it should be.

1 sudo cp -r node -v0 .10.2 - linux -arm -pi /\* /opt/ node

Edit - or create - a .bashprofile file, which is a type of script that runs when you turn on the pi. In this case, it runs and tells Node that it exists on your computer, so that typing node runthisprogram will do something.

From your root directory, to open a new nano text file:

1 sudo nano . bash\_profile

Then add the following and save it to your new .bashprofile file...

1 PATH = $PATH :/ opt/ node / bin

2 export PATH

Control-X, Y to save it.

Node lives in the /opt/node directory you created above. This adds the commands ”node” and ”npm” to what are called ”environment variables.” If you are curious, and god knows you must be to play with a raspi, you can type ls /opt/node/bin and see the little programs sitting there in their bin.

**4.8 Testing Node**

Node will need to be able to fetch its own packages separately from the raspi from the internet in order to run some of the monitoring software I’ve chosen to use. Particularly, you will need the forever package.

**4.8.1 Selecting Monitoring Software**

forever has ultimately been the software I’ve decided on to monitor and run screen- Perfect, because it is a node-native package that keeps things running even when they crash. There are other software packages used for broader deployment, such as Monit, which installs to your Debian parcel rather than to Node. Monit typically runs with what is called an HTTP Proxy, which can be written directly in Node or installed inde- pendently. In a full deployment build, Monit and HAProxy would be preferable to Node alone, because this follows the best practice of separating out different programming elements from one another in production. Monit and HAProxy can also deploy applic- ations above and beyond Node itself, which is preferable for things written in Python, for example.

For this example, though, forever works well. It provides monitoring to tell us what the application is doing, and automatically restarts node applications when they crash. Were I deploying this such that it could keep an eye on the internet, which I am not, I would also include nodemon, as is recommended by the Subnod.es project. nodemon monitors your development code and pushes changes from a central server to your deployment automatically.

That is outside the scope of this paper at present.

**4.8.2 Installation of Node Modules**

To install a node package - or ”module” - you type

1 npm install PACKAGENAME

To install a package globally, type

1 npm install PACKAGENAME -g

To force install:

1 sudo su

2 PATH =/ opt / node / bin /: $PATH

3 npm install PACKAGENAME -g

4 exit

To install forever and nodemon

1 npm install forever -g

2 npm install nodemon -g

To run forever and nodemon together....

1 forever start / usr / local / bin / nodemon / path /to/ YOURAPP.js

**4.9 SSH via Direct Ethernet Connection and WiFi Internet Access**

Eventually, you will need both of the powered USB slots on the raspi for a USB key and for your wiFi. In addition, the raspi doesn’t have the power to drive a monitor and consistently serve wiFi out of its USB ports. To get around this, it is most convenient to be able to SSH in to your device. Although it appears to be best practice to use the wpasupplicant file to store how you wish your raspi to connect to the internet, I have had limited success with it, likely because I am not configuring a static IP for my raspi properly.

My /etc/network/interfaces file looks like this:

1 auto lo

2 iface lo inet loopback

3

4 auto eth0

5 iface eth0 inet static

6 address [MY MAIN TERMINAL 'S ETHERNET IP PLUS ONE ]

7

8 auto wlan0

9 allow - hotplug wlan0

10 iface wlan0 inet dhcp

11 wpa - ssid " network name here "

12 wpa - psk " dubiously secure password "

1 sudo nano / etc / default / ifplugd

2

3 ### MANY TALK , HOW COMMENT , SUCH WARNING ###

4 INTERFACES =" eth0 "

5 HOTPLUG\_INTERFACES =" eth0 "

6 ARGS ="-q -f -u0 -d10 -w -I"

7

8 SUSPEND\_ACTION =" stop "

Here is what your startup script should read. This ensures that your wiFi antenna turns on, which is likely not something it was doing when you plugged in your Ethernet directly.

1 sudo nano / etc /rc. local

2 #!/ bin /sh -e

3

4 # Print the IP address

5 \_IP =$( hostname -I) || true

6 if [ " $\_IP " ]; then

7 printf "My IP address is %s\n" " $\_IP "

8 fi

9

10 # Disable the ifplugd eth0

11 sudo ifplugd eth0 --kill

12 sudo ifup wlan0

13

14 exit 0

CTRL-X and Y to save, then sudo reboot open a terminal on your main laptop. On your laptop, at the prompt, enter:

1 ssh pi@ [ the static ip address you entered under eth0 static above ]

Your pi@[static ip] should appear in your terminal window, which means you can now talk to raspi. Per usual, to ensure your wifi is still working properly, try a sudo apt-get update or ping google.com, both should return you data.

**4.10 Backing Up Your RasPi**

Now that everything has been configured for the first steps, type sudo halt, and when your raspi turns off, remove the SD card from it. Place the SD card back in your main computer and reboot Win32DiskImager.

Create a new file folder somewhere within your Documents folder.

In the Write From section of the application, select your SD card, which is probably called boot. In the Write To section, select your new folder.

Write a copy of the kernel image from the boot card to the new backup directory. Then safely eject your SD Card and re-insert it in the RasPi. It is best practice to form these occasional backups as you proceed through set up. Many of these steps can cause your raspi distro to break badly, which a backup will help to ameliorate.

**4.11 Mount Your USB Flash Memory Stick to Your RasPi**

**4.11.1 Configuring Your Mount Drive**

This bears some thinking about, because the /media/ folder is for media, and you are instead choosing to run a program off of the drive. Subnod.es suggests making it your www drive, for world wide web. I picked /mnt/.

Find your USB memory by listing the the things plugged into dev:

1 sudo ls / dev /sd\*

If you’ve been following along, yours is almost certainly named ”/dev/sda1”.

So make a directory for it to be addressed at:

1 sudo mkdir / mnt / USBSTICKNAME ;

Then mount it to that directory

1 sudo mount -t vfat -o uid =pi ,gid=pi /dev/ sda1 / mnt / USBSTICKNAME /

2 sudo reboot

Rebooting will restart the raspi but also close your SSH session. Watch the lights on the raspi board until they’re stable again, about two minutes, then:

1 ssh pi@ [ static ip]

Your USB drive is currently mounted, but it will not automatically remount at boot.

**4.11.2 How to Boot Mount External Memory**

Find out the actual name of your external memory card:

1 ls -l / dev / disk /by - uuid

Write down the UUID of your USB stick.

This is the most manual way to run this operation, and there is software that handles automatic drive mounting. It is called usbmount and was discarded during this process because it ended up being more convenient to rely on my Node application being loaded directly onto the SD card, rather than from boot.

1 sudo chmod 775 / mnt / USBSTICKNAME

2 sudo sp / etc / fstab / etc / fstab . bak

3 sudo nano / etc / fstab

Add the following to /etct/fstab

1 UUID = YOURUUID / mnt / USBSTICKNAME vfat rw , defaults 0 0

CTRL-X, Y to save, then

1 sudo reboot

2 ls / mnt / USBSTICKNAME

This command should display the contents of your USB key when you go looking for it.

At this point, I have taken a copy of my Node application and moved it to the SD card in a separate directory. Although I have optimistically tried to make this a headless - no keyboard or monitor - box, realistically, lots can go wrong with the SSHing process. You will probably eventually want a keyboard, and it is much easier to store your access point as a single image per card, much like any other video game.

To store your games locally, rather than in the USB stick:

1 sudo cp -r / mnt / USBSTICKNAME / home /pi/ YOURDIRECTORYNAME

4.12 Set Up a wiFi Hotspot

To get started, you will need some more software.

1 sudo apt - get install hostapd dnsmasq

When everything is done installing, you will be converting your /etc/network/interfaces file to serve a hotspot, rather than connect to the internet.

Here is what my final /etc/network/interfaces file looks like:

1 auto lo

2 iface lo inet loopback

3

4 auto eth0

5 iface eth0 inet static

6 address 169.254.222. xx #xx is a stand -in for an actual address , not included

.

7

8 allow hotplug wlan0

9

10 ## wlan internet connect settings are commented out for easy swap .

11 # auto wlan0

12 # iface wlan0 inet dhcp

13 # wpa - ssid " network name "

14 # wpa - psk " network password "  
15  
16 iface wlan0 inet static  
17 address 192.168.42.1 #42 is a joke about Douglas Adams, as the number at this position can be anything between 0 and 255.  
 18 netmask 255.255.255.0

**4.13 Configuring HostAPD**

hostapd is the software that provides the access point using your raspi. It can be tricky, and in order to make it work, it needs to be compiled for one’s specific model of wiFi antennae. For the purposes of this paper, we are using an antenna sold and supported by Adafruit. The appropriate compile of the hostapd software is included in the supplementary files to this paper, but can also be found at http://www.adafruit.com/downloads/adafruit- hostapd.zip.

To install a valid copy of hostapd:  
  
1 wget http :// www . adafruit .com/ downloads / adafruit\_hostapd . zip

2 unzip adafruit\_hostapd .zip

3 sudo mv / usr / sbin / hostapd /usr/ sbin / hostapd . ORIG

4 sudo mv hostapd / usr / sbin

5 sudo chmod 755 / usr / sbin / hostapd

Now set up a daemon - a piece of automatic system software - to run the hostapd configuration file on boot.

1 sudo nano / etc / default / hostapd

Uncomment (remove the hash mark in front of ) DAEMON\_CONF="" and replace that line with DAEMON\_CONF="/etc/hostapd/hostapd.conf. Then type CTRL-X and Y to save your file.

My hostapd file is listed below.

1 sudo nano / etc / hostapd / hostapd . conf

2

3 interface = wlan0

4 driver = rtl871xdrv

5 ssid = piebox

6 hw\_mode =g

7 channel =6

8 macaddr\_acl =0

9 auth\_algs =1

10 ignore\_broadcast\_ssid =0

11 wpa =2

12 wpa\_passphrase = berrybox

13 wpa\_key\_mgmt =WPA - PSK

14 wpa\_pairwise = TKIP

15 rsn\_pairwise = CCMP

**4.14 Configuring DNS access via dnsmasq**

Configuring dnsmasq is straightforward. The installation package comes with an ex- tensive config file, which lives at /etc/dnsmasq.conf, and includes all of the options necessary to turn on a DNS routing service.

To configure your dnsmasq installation, enter sudo nano /etc/dnsmasq.conf and then add the following lines to the top of the configuration file. The configuration file contains all these values commented out already, and may be worth a separate read.

1 interface = wlan0

2 dhcp - range =192.168.42.2 , 192.168.42.50 ,255.255.255.0 ,12 h

3 address =/#/192.168.42.1 # redirect all DNS requests to 192.168.42.1

4 server =/ screenperfect /192.168.42.1#3003

5 address =/ apple . com /0.0.0.0

What the above does is tell the raspi to listen on the wlan0 interface, to the dhcp range between 192.168.42.2 and 42.50, for twelve hours per time a client connects to the wiFi point. In addition, the portal is supposed to redirect all DNS requests - things like ”google.com” - to the Pi’s main address, which is - as we can see in/etc/network/interfaces - 192.168.42.1, and from there to the port 3003, on which my particular Node application listens.

In addition, the portal serves a spoof address to apple.com, which helps us to pop up the appropriate page on the captive portal when it is turned on.

To date, this portion has not proven totally effective. Getting a page to pop up on a captive portal requires a series of correct internet handshakes per device, so it has so far been easier to set the URL by hand on client devices to the Raspi.