WIRELESS DOCKING STATION FOR MOBILE DEVICES

A THESIS

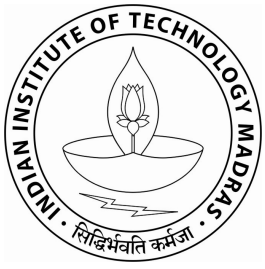
submitted by

SYED SUFIYAN

for the award of the degree

of

BACHELOR OF TECHNOLOGY



DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2014

THESIS CERTIFICATE

This is to certify that the thesis titled **WIRELESS DOCKING STATION FOR MOBILE DEVICES**, submitted by **Syed Sufiyan**, to the Indian Institute of Technology Madras, Chennai for the award of the degree of **Bachelor Of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Nitin Chandrachoodan
Research Guide
Associate Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 16th May 2014

ACKNOWLEDGEMENTS

14/03/1413 – a regular day in the life of the world but a particularly happy one for the people I wish to thank – my first and foremost thanks goes out to that day.

My life in this world has been one of constantly rebellion and I have always gotten myself into trouble because of my inability to accept failure and my next thanks goes out to those who supported me through this phase of rebellion and channeled my energies into making me a better human being – my parents – Abba and Ammi – Thank You.

I would like to thank my sister for helping me through and through and Chan Mama and Chan Mami for their support.

My next thanks goes to every one of those people who had to put up with my immaturity and rebellion – when I would come up with grandiose theories about a non-existent thing which in my strong sense of self was just something no one had bothered looking into. I thank all my school teachers for putting up with that.

I then reach the most beautiful four years of my life – a period of time where I realized how worthless I really was and how little I knew of the world around me. A period surrounded by such smart people, such talented people – a heaven for those inclined to achievement and glory, a place that with its every soul taunted my existence and with the same zeal surrounded me with those who I could look up to. This world – yes one removed from reality, one where despite dreamers being few and far between was a magical place. I wish to thank IIT Madras, insti as we lovingly call it and will remember it as through our life.

My life here would have been the most difficult unless I had the most amazing Faculty Advisor – Nitin Sir and I thank him most profusely for being there for me.

I also wish to thank every other professor who taught me prime among them – Prof.Ashwin, Prof. Raina, Prof. Chakravarthy, Prof. Aniruddhan, Prof. GV and Prof. Jhunjhunwala.

I thank all my friends for helping me in this stay – JB, Mitan, Tichku, Kavin, Sujan, Rohan all my lab mates – Karthi, Seetal, Ajmal, Vijay, Ramprasath, Jobin – you guys will be missed.

Also, there was someone who taught me what it meant to be an utter and complete failure in life and I thank her for teaching me that. I also thank her for guiding me through some tough times. I also take this opportunity to thank an amazing friend I made from Jordan who helped me in the past few months – without whom I would have never gotten the strength to do my project.

I have owed a lot of my life to the Internet and very recently came across some amazing communities where I learnt a lot – I thank all those strangers in the world who helped me.

I thank all my seniors who guided me through and through.

Last and most importantly, I wish to thank God Almighty for helping me through my journey and I hope to get better with time.

.

ABSTRACT

KEYWORDS: Android, Linux, Debian, Raspberry Pi, Embedded systems.

We design and develop a wireless docking station for mobile devices. We discuss the problem and the possible solutions and we also discuss the possible steps to overcome the same. We quantify the system performance and figure out bottlenecks.

We finally end with Future directions and how we intend on rolling it out as a consumer facing product and the possible steps toward the same.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

RPi	Raspberry Pi

CHAPTER 1

MOTIVATION AND STORY BEHIND THE PROJECT

While on an internship in Sony Japan, I saw most if not all of my colleagues using an extra screen for their laptops which were awfully small. A regular job for most of them would be to go from meeting to meeting and connect their laptops to a big screen using cables – then wait for the thing to show up on screen.

On one such regular sleepy day in the office, I was playing around with the long bunch of HDMI cables on my desk connecting to a TV. Now, Sony has a small cafeteria where you could get coffees and teas and all – most of them chilled despite living in weather of lesser than 26 C – an atrocity to a South Indian whose winters don't go lesser than 27 C.

So, I was sitting there at my desk with my tea and as I reach out to the Chilled Honey tea, I think in my head what if this tea spills over my desk. Well, that was the worst thought to have but just as luck would have it, my colleague sitting next to me has the misfortune of spilling tea all over as he was taking it off the desk and the culprit – well if you guessed HDMI cables, you guessed right!

That settled it – why not make a system to just connect wirelessly to these displays rather than through these long wires. The idea from here went through many stages of refining to get what it is at the moment.

Stage 0: Understanding the problem statement and the real reason for doing this.

Stage 1: Figure out possible means of connecting 2 different displays

Stage 2: Using the computational power of the mobile device and using keyboards and mice outside the system pluggable into a bigger system sitting on the stand alone monitor

Stage 3: Measuring the system performance and the bottlenecks associated with the existing design.

Stage 4: Future directions

This thesis is divided into these 5 stages and each chapter represents a stage and the subsequent efforts.

CHAPTER 2

STAGE 0: UNDERSTANDING THE PROBLEM STATEMENT AND THE REASONS FOR DOING THIS

The average smart phone that is used in everyday life costs about Rs. 9000/- and there are many smart phones which cost as low as Rs. 3000/- and high end cell-phones cost as much as Rs. 50000/-.

We compare the specs of the average and high end cell-phones in the market.

We discuss the Specifications for the Average cell-phone in the market in Appendix A and

This is the average cell-phone which is available in the market (we take the Nokia X cell-phone for the same):

	2G Network	GSM 850 / 900 / 1800 / 1900
GENERAL	3G Network	HSDPA 900 / 2100
	SIM settings	Optional Dual SIM (Micro-SIM)
BODY	Dimensions	115.5 x 63 x 10.4 mm, 73.2 cc
	Weight	128.7 g
DISPLAY	Type	IPS LCD capacitive touchscreen, 16M colors
	Size	480 x 800 pixels, 4.0 inches (~233 ppi pixel density)
	Multi touch	Available

SOUND	Loudspeaker	Available
	3.5mm jack	Available
MEMORY	Card slot	MicroSD, up to 32 GB
	Internal	4GB, 512 MB RAM
	GPRS	Upto 85.6 kbps
	EDGE	Upto 236.8 kbps
DATA	Speed	HSDPA, 7.2 Mbps; HSUPA, 5.76 Mbps
	WLAN	WiFi 802.11 b/g/n, WiFi – hotspot
	Bluetooth	Available, v3.0 with A2DP, HS (very important for media sink)
	USB	Yes, microUSB v2.0
CAMERA	Primary	3.15MP, 2048 x 1536 pixels, check quality
	Video	Available
	CPU	Dual core 1 GHz
COMPUTE	GPU	Adreno (rarely Mali is used in some average phones)
	Sensors	Accelerometer, Proximity
	GPS	A-GPS

And for the most high end cell-phone (we take the case of Samsung Galaxy S5) we have:

	2G Network	GSM 850/ 900/ 1800/ 1900
GENERAL	3G Network	HSDPA 850/ 900/ 1900/ 2100
	4G Network	LTE 800/ 850/ 900/ 1800/ 1900/ 2100/ 2600
	SIM	MicroSIM
	Dimensions	142,72.5x8.1 mm
BODY	Weight	145 g
	Type	Super AMOLED Capacitive touchscreen, 16M colors
DISPLAY	Size	1080x1920 pixels, 5.1inches (~432 ppi pixel density)
	Multi touch	Yes
	Protection	Corning Gorilla Glass 3
SOUND	Loudspeaker	Available
	3.5mm jack	Available
MEMORY	Card slot	MicroSD, upto 128GB
	GPRS	Available
	EDGE	Available
	Speed	HSDPA, 42.2 Mbps; HSUPA, 5.76 Mbps; LTE, Cat4,

		50Mbps UL, 150 Mbps DL
	WLAN	WiFi 802.11 a/b/g/n/ac, dual-band, DLNA, WiFi direct, WiFi hotspot
	Bluetooth	Available, v4.0 with A2DP, EDR, LE
	NFC	Available
	IR Port	Available
	USB	Yes, microUSB v3.0 (MHL 2.1), USB On-the-go, USB Host
CAMERA	Primary	16 MP, 5312 x 2988 pixels
	Video	2160p@30fps, 1080p@60fps, 720p@120fps
	CPU	Quad-core 2.5 GHz Krait 400 (or) Quad-core at 1.2 GHz and another Quad-core at 1.6 GHz
COMPUTE	GPU	Adreno (or Mali – very rarely)
	Sensors	Accelerometer, gyro, proximity, compass, barometer, gesture, heart rate
	GPS	Yes, with A-GPS support and GLONASS

Lets backup a little bit to 1961, Russia had just put the first man in space - Yuri Gargarin had just been the first man to enter space in the VOSTOK – 1.

The cost of 1 GFLOP in 1961 was USD 1.1 trillion (which with inflation in December 2013 costs USD 8.3 trillion) [1]

And in the period spanning from 1961 to end of 2013 we have sent almost 232 space missions with over 100 missions which happened in the era before the 1980's – this was the time where the most powerful super computer in the world was the CRAY – Y –

MP (released in 1988) and could have eight vector processes at 167 MHz with a peak performance of 333 Megaflops per processor. [2]

Compare this with the current average cell-phone – which has a Dual-core running at 1GHz (and it is very likely that the current high end cell-phones will become obsolete within the next 2 years) so we are realistically looking at cell-phones which can beat the supercomputers of the 1990's.

Also, the current eco system around mobile devices has made them the prime and only focus, with a few people using their desktop computers very rarely. Also the application eco system around these devices has become very good.

Total Available iPhone Apps:	1074634
Total Available iPad Apps:	652634
Total Available Mac Apps:	19894
Total Available Android Apps:	1 million+

Most people only use their mobile devices to access the Internet – and this trend is only bound to improve over the years.

At the same time, most of the users have to work with a very small screen and as much as there are laptops, people would rather use their cell-phones and carry around their applications and data everywhere they go.

This motivates the need for the current project. So, the initial design for the project will require the ability to put your display on a bigger screen and also take input from the user for the same.

This project seeks to use the computational power that is present in most modern cell-phones to provide a desktop like interface with a simple docking station that can be used for projecting the screen of the cell-phone onto a bigger screen connected to a keyboard and mice to make it usable as a stand-alone system

CHAPTER 3

STAGE 1: FIGURE OUT POSSIBLE MEANS OF CONNECTING TWO DISPLAYS

Consider the case of 2 separate screens – 1 connected with a computer and another which could either be connected to a computer or not.

We list out the many different ways of connecting these screens:

1. A simple client and server system which could be accomplished through:
    1. xWindows
    2. VNC
    3. orbX.js
    4. Windows Remote Desktop protocol
2. From the computer connect a hardware device which goes from the Video output and can push the same to the HDMI opening on the other screen wirelessly without using the existing hardware on the mobile device.

Let's start from the 2nd point onward.

Pros:

1. Platform independent – cell phone hardware agnostic
2. Easier to implement (given a hardware background)
3. No need to develop extra software

Cons:

1. Need to mimic a HDMI output port on the other end.
2. Need to match the impedances perfectly because HDMI is a very high speed protocol (~18Gbps)
3. Will drain cell-phone battery

Let us move on to the first point.

1. Xwindows:

   This is a windowing system for bitmap displays, commonly used in UNIX based operating systems. The client (user) simply tunnels to the server running the X based system and accesses the display through this. The user has keyboards, mice and a screen at their own terminal and can access the X server which draws to the Users screen. X displays are an ideal front end for a distributed computing environment. This means that in a classroom scenario, one can simply have a very powerful X Server to which others can ssh into and draw the same on their own displays.

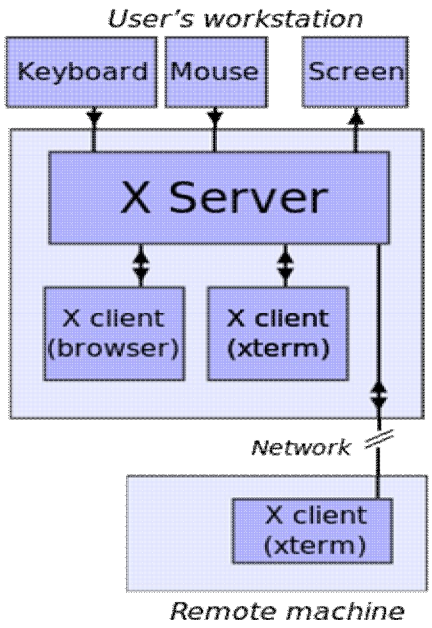


Fig 3.1 Simple diagram of the XWindows protocol

Typical usage: *ssh -X user@ipaddress* - and then use it as a regular terminal in the server machine

2. VNC – Virtual Network Computing

This is a desktop sharing application based on the Remote Frame Buffer protocol which can be used to remotely control another VNC server computer from a compatible VNC client

A VNC system consists of a client, a server, and a communication protocol

1. The VNC server is the program on the machine that shares its screen. The server passively allows the client to take control of it.

2. The VNC client (or viewer) is the program that watches, controls, and interacts with the server. The client controls the server.

3. The VNC protocol (RFB) is very simple, based on one graphic primitive from server to client ("Put a rectangle of pixel data at the specified X,Y position" – at the most basic level with a lot of improvements on top of this protocol for connections of different latencies) and event messages from client to server.

VNC by its very protocol is a very insecure protocol and someone sniffing the network for a reasonable amount of time can get the passwords for any VNC display. For better security, VNC is tunneled through SSH or VPN which adds an encrypted extra layer of security.

However, VNC is also good in that because it is based on a pixel based windowing system, it can be used with any system and provides great flexibility but, this is also the biggest reason for its inefficiencies because of a lack of understanding of the underlying graphic layout like X11 or windows Remote Desktop Protocol

3. OrbxJS

This is a relatively new thin client that is being made available to the public very soon and leverages the power of the cloud to seamlessly deliver content to any user who might be using the system.

Based on an Amazon EC2 instance, the library provides access to the EC2 through a web browser without using any add ons and is totally HTML5 based system.

As written on Amazon AWS:

"ORBX Cloud Game Console is the world's first turn-key high performance cloud desktop solution specifically designed for streaming high-end remote graphics and games. Use this AMI to stream a Windows based virtual

desktop, hosted in the cloud, to a web browser anywhere in the world. - Unleash the full power of the cloud on your favorite games and graphics applications. Offering nearly 2x the GPU power of an XBOX ONE on G2 instances, the ORBX Cloud Console delivers next generation gaming 'out of the box. - Access your performance intensive games and applications from any device, regardless of your operating system or device performance. - Your applications seamlessly render with clarity in HD due to OTOY's next generation ORBX(TM) Video Codec. - Stream your applications using OTOY's native client application (available for Windows, Linux and iOS), or go plugin-free with ORBX.js on any modern browser (including Chrome, Firefox, Safari, Opera, and Internet Explorer). - Includes OTOY WebCL(TM) remote graphics driver - the only OpenCL 1.2 GPU runtime for NVIDIA GRID"

However, the protocol is not yet fully available to the public and there are plans to open source the work on a very small scale by the end of the year (2014).

4. Windows Remote Desktop Protocol

RDP is a proprietary protocol developed by Microsoft, which provides a user with a graphical interface to connect to another computer over a network connection. The user employs RDP client software for this purpose, while the other computer must run RDP server software.

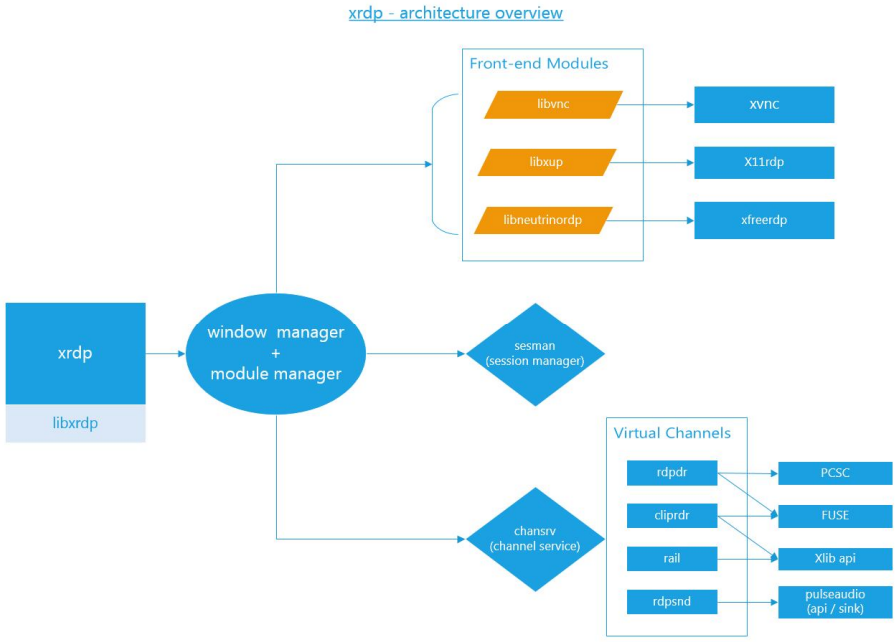


Fig. 3.2 Architecture Overview of the Windows Remote Desktop Protocol

So, it was decided to go for the Client server technologies because of the lesser overheads as regard to the need for dedicated hardware and also because it is a developed system with a lot of help from support forums.

It is also important to understand that most of these architectures work on the same scale without much of a difference. The only difference being that VNC being a pixel drawing based protocol, assumes that the Client connecting to it doesn't have great computational capabilities and doesn't use the power of the underlying OS at all by drawing to an altogether different window whereas the others rely on the same thus making them lossy at times.

Also, most client server technologies depend on having 2 computers running a modern OS such as Linux or Windows on them. This adds significantly to the cost.

Also, client server technologies have a lot of Licensing agreements that one must tread carefully over because it can cause a lot of pain when the product is released into the market. Considering also that most of the Android ecosystem is trying to move away from GPL and the other problems associated with the licensing that makes it difficult for different vendors to build on top of the same. This is a warning sign when it comes to using GPL licensed software.

Client server technologies are also associated with a lot of delays and latencies could make them unusable.

However, in the present project we will utilize the same and try to make some incremental changes and try to improve the system – making it usable at least for the short term goals of text processing and basic interaction.

We go for the Client Server Model

CHAPTER 4

STAGE 2: USE COMPUTATIONAL POWER OF MOBILE LAPTOP AND BUILD THE SYSTEM

Using 2 different techniques - Xwindows and VNC, we try to ascertain which is better.

But before we begin with that we need to first figure out how to transmit the data. For the moment let us have 2 computers, 1 acting as the SERVER and other acting as CLIENT.

We start by setting up a WiFi hotspot (DHCP server) on the computer that is going to be used as a Viewer. The device connecting to this viewer needs to run a server which will help in drawing to the viewer.

Designing the system to make it intuitive and easy to use is the most important thing that must be done in this system because it is a consumer facing application. And to this end we design the system such that it is extremely easy for the consumer to use it without very difficult set up and so that it is really easy for the end consumer to use.

To make matters simple we design the system such that the user simply has to connect to a WiFi hotspot and then has to do nothing more than that to get his system set up.

4.1. SETTING UP A WIFI HOTSPOT (refer APPENDIX C for further instructions)

The WiFi hotspot which the user connects to, as he would regularly connect, provides not just an internet connection but also sets up the viewer automatically.

This makes it relatively easier for people to use the same and make it easy for them to understand how to use it.

Setting up CLIENT viewer

In the terminal, type

sudo apt-get update && sudo apt-get install xtightvncviewer tightvncserver

Then, type in : **vncpasswd** in the terminal and enter a password.

Also, put a small script which automatically connects to your device when it is clicked on. On a terminal type in : **git clone https://github.com/SuFizz/Client.git**

Then, **cd Client && python Starter.py**

This will scan through the ports and connect you to the VNCserver.

4.2. SETTING UP MOBILE DEVICE SERVER (refer APPENDIX D for further instructions)

The Mobile server needs to be running for the screen to be mirrored. In most Linux devices, the X protocol exists however in Android we have the Surface Flinger module (which is provided as a system application to be used through the Binder IPC) which we use for the same.

Further on we discuss the speed and the bottlenecks associated with the same. We will also discuss the patterns that we see and the possible problems that might be indicated in the same.

CHAPTER 5

STAGE 3: MEASURING SYSTEM PERFORMANCE AND BOTTLENECKS ASSOCIATED WITH EXISTING DESIGN

Two utilities were used throughout the testing for the purpose of testing – top (runs as a regular user process) and nethogs (requires sudo privileges).

Testing was done for measuring CPU usage and bandwidth used for the server and viewer

1. At 6 different distances (Server and viewer separated by 10cm, 1m, 2m, 3m, 4m and finally 5m).
   We expect that the device will be used in only 2 of these settings – up to 1m and between 3m and 4m (which is about the average distance between a TV and a place for watching in most regular households); and once through a concrete wall (31cm x 2).
2. With 4 different encodings –Raw (R); Raw over SSH (S); Hextile (H) and Tight (T)
   We again expect that the device will be used only in the first 2 settings because of the huge bandwidth (approximately 54 Mbps) that we get over the connection.
3. 3 different sizes of the same video file (at 24 fps) that was being buffered – 426x228 (S); 640x344 (M) and 1280x688 (L).

In order to have better measurements, the room was sealed closed and shut to reduce interference from other WiFi networks (2 other very weak networks were detected after closing). The temperature of the room was at an average of 27 C and at a humidity of 65%.

The only other WiFi device which was ON at the time was the Laptop which was used to take measurements. The measurements were again randomized;

Nethogs consumes a significant amount of CPU and so the measurements were performed with Nethogs on the DHCP server where CPU was available for the same. We only show the CPU usage of totem (the Video player that was used in the measurements). Totem was used because it consumes on an average lesser CPU than VLC and is better suited for the limited amount of CPU power available on the RPi.

Distance	Size of Video 426x228 (S) 640x344 (M) 1280x688 (L)	Encoding Raw (R) Raw over SSH (S) Hextile (H) Tight (T)	CPU usage by TOTEM (in %)	CPU usage by VNC server (in %)	CPU usage by VNC viewer or SSH(Client side. Server side - SSH) (in %)	Speed of reception (in KBps)
10cm	S	R	68.1	18.7	8.1	703.675
10cm	M	R	83.3	6.1	6.5	498.744
10cm	L	R	68.1	18.7	8.1	601.937
10cm	S	S	56.6	5.3+14.8(ssh)	15.1	306.464
10cm	M	S	70.7	4.8+12.3(ssh)	11.2	111.670
10cm	L	S	75.3	5.0+10.3(ssh)	10.5	327.739
10cm	S	H	67.6	12.2	5.8	27.476
10cm	M	H	84.9	5.2	1.3	6.692
10cm	L	H	82.4	12.5	0.6	3.520
10cm	S	T	47.3	34.1	18.1	8.039
10cm	M	T	57.1	33.5	11.7	7.138
10cm	L	T	85.0	5.5	1.0	1.305
1m	S	R	64.1	13.8	12.3	857.164
1m	M	R	71.9	15.2	0.9	741.397
1m	L	R	80.9	7.0	6.4	458.876
1m	S	S	53.5	1.9+18.2(ssh)	16.1	382.864
1m	M	S	71.6	4.8+11.9(ssh)	11.5	353.589
1m	L	S	74.9	4.5+9.7(ssh)	9.5	300.702
1m	S	H	69.5	10.6	5.5	59.797
1m	M	H	73.4	3.1	0.3	4.129
1m	L	H	92.1	1.9	1.3	13.198
1m	S	T	47.6	12.8	3.9	5.813
1m	M	T	74.4	14.2	3.0	1.696
1m	L	T	66.4	0.3	0.5	2.368
2m	S	R	63.3	15.5	11.3	700.228
2m	M	R	80.9	7.0	5.2	384.516
2m	L	R	68.6	15.1	10.5	681.731
2m	S	S	50.8	2.2+21.5(ssh)	17.5	265.907
2m	M	S	70.2	4.7+1.3(ssh)	13.2	240.973
2m	L	S	88.0	1.6+3.5(ssh)	6.5	45.766
2m	S	H	72.8	7.7	2.2	79.758
2m	M	H	84.9	5.5	2.2	84.033
2m	L	H	86.7	3.2	1.9	82.119
2m	S	T	68.3	11.9	7.1	14.173

2m	M	T	78.7	11.0	7.3	12.273
2m	L	T	77.1	11.9	4.9	14.900
3m	S	R	66.2	11.6	10.2	772.810
3m	M	R	77.3	9.7	7.4	489.043
3m	L	R	79.0	8.4	5.1	504.616
3m	S	S	56.1	1.6+15.7(ssh)	16.4	303.844
3m	M	S	70.4	5.1+10.6(ssh)	10.0	303.799
3m	L	S	74.5	4.5+9.7(ssh)	11.2	311.495
3m	S	H	71.3	8.7	3.2	68.803
3m	M	H	84.5	6.5	2.6	104.050
3m	L	H	62.0	0.3	0.6	27.056
3m	S	T	62.1	18.7	9.5	7.754
3m	M	T	81.6	9.0	3.0	6.986
3m	L	T	79.3	10.6	0.9	3.105
4m	S	R	61.4	17.0	11.0	772.922
4m	M	R	77.2	9.4	8.0	576.495
4m	L	R	82.3	7.1	6.1	521.983
4m	S	S	55.7	6.1+16.1(ssh)	15.1	226.343
4m	M	S	67.5	4.5+9.7(ssh)	9.8	334.360
4m	L	S	68.1	5.1+14.4(ssh)	16.3	302.403
4m	S	H	68.0	9.0	2.2	103.914
4m	M	H	83.9	6.5	2.2	90.857
4m	L	H	81.0	6.1	2.5	136.693
4m	S	T	52.7	14.6	3.6	8.045
4m	M	T	62.8	5.0	0.6	1.637
4m	L	T	91.8	2.6	1.3	2.066
5m	S	R	66.9	12.9	9.7	622.430
5m	M	R	82.9	6.1	5.7	433.408
5m	L	R	81.8	7.1	4.9	447.617
5m	S	S	52.1	6.4+18.2(ssh)	14.8	290.927
5m	M	S	72.3	5.5+9.4(ssh)	9.9	232.081
5m	L	S	91.3	2.4+5.2(ssh)	5.2	70.612
5m	S	H	49.5	8.1	1.9	106.999
5m	M	H	84.2	6.1	2.2	100.204
5m	L	H	84.1	5.5	1.9	68.572
5m	S	T	65.6	17.4	10.5	23.802
5m	M	T	77.9	12.2	5.5	16.074
5m	L	T	76.8	12.6	7.3	18.063
Concrete	S	R	61.9	17.6	11.5	739.304
Concrete	M	R	82.6	6.5	4.5	393.512
Concrete	L	R	77.8	9.3	8.1	392.424

Concrete	S	S	54.4	6.9+15.6(ssh)	13.6	483.409
Concrete	M	S	59.0	8.0+14.4(ssh)	16.4	547.836
Concrete	L	S	55.8	7.7+20.9(ssh)	14.9	560.027
Concrete	S	H	70.2	9.3	2.5	87.045
Concrete	M	H	83.2	6.5	1.9	74.835
Concrete	L	H	82.5	6.4	1.6	70.343
Concrete	S	T	66.4	17.4	10.2	18.929
Concrete	M	T	72.9	13.9	7.1	18.333
Concrete	L	T	75.0	11.9	6.8	12.947

Rate seems pretty much independent of distance for a given encoding that was used. However, it was found that the viewer quality reduces over distance - for example at 10 cm there wasn't much flicker with the mouse. However, at 5m this was very apparent and sometimes very disturbing. SSH in comparison to RAW does reduce the data rate. It is likely that this might be because of the compression that SSH does.

These are the full properties of the Videos

426x228 (S) (.flv) -

Video Qualities:
Codec used : Sorenson Spark Video
Audio Qualities:
Codec used : MPEG-1 Layer 3 (MP3)
Sample rate : 22050 Hz
Bitrate : 64kbps

640x344 (M) (.mp4) -

Video Qualities:
Codec : H.264
Bitrate : 492 kbps
Audio Qualities:
Codec : MPEG-4-AAC
Sample Rate : 44100 Hz
Bitrate : 96 kbps

1280x688 (L) (.mp4) -

Video Qualities:
Codec : H.264
Bitrate : 1957 kbps
Audio Qualities:
Codec : MPEG-4-AAC
Sample Rate : 44100 Hz
Bitrate : 191 kbps

You can find the video here: https://www.youtube.com/watch?v=MOiyD26cJ2A - For the Birds 720p on YouTube

Considering the same, M must take up more bandwidth because it is relatively more amenable to being easily encoded by most of the standards which can send the data over more quickly. Even Raw encoding does a little bit of processing on the RFB to get the final data that is sent over the Internet.

Test was performed with this video because this sort of captures the average user considering that the video has over 21 million views and seemingly seems professionally made.

The quality deteriorates a lot when there is a lot of interference from many WiFi connections in the room due to packet loss (however, we haven't checked the same for our scenario)

CHAPTER 6

STAGE 4: FUTURE DIRECTIONS

We intend to roll this out as a consumer facing product through appropriate means. At present we have a very simple prototype which we will improve onward to a product.

For this to be realistically be possible, it would require massive amounts of work in reducing the delays. Most Android and Raspberry Pi's rolling out nowadays have hardware H.264 encoding and this will prove most important in reducing the lag that is seen. We hope to reduce the lag to at least less that 20 ms meaning it is realistically playable on even the worst of TV's which have Game play lags of 100ms . Also, it must be noted that most of the current solutions don't have a great encoding primarily because very little information is available with regard to the software to use H264 encoding (with regard to Raspberry Pi).

Also, making the device into something as concise and small as the Chromecast is something that we need to look into. And is the immediate focus of the authors work after the completion of the BTech requirements.

Also, we need to figure out sources of power in such a situation or come up with more innovative ways to reduce the power delivered to the WiFi dongle so that it consumes far lesser power than it must – which means we have to look into making the device for a smaller range than it currently is capable of.

Packaging is again another issue which we need to figure out. Eventually when we startup with the idea, then we should have enough people to join into the team.

Because the project is being done completely in a small setting, we will use a Maker Bot to prototype the same and also we can use a lot of open sourced designs already available online from: http://www.thingiverse.com/ and such websites.

APPENDIX A

NOOBS INSTALLATION INSTRUCTIONS

1. Insert an SD card that is 4GB or greater in size into your computer.
2. Format the SD card using the platform-specific instructions below:

    2.1. Windows

        2.1.1. Download the SD Association's Formatting Tool from https://www.sdcard.org/downloads/formatter_4/eula_windows/

        2.1.2. Install and run the Formatting Tool on your machine

        2.1.3. Set "FORMAT SIZE ADJUSTMENT" option to "ON" in the "Options" menu

        2.1.4. Check that the SD card you inserted matches the one selected by the Tool

        2.1.5. Click the "Format" button

    2.2. Mac

        2.2.1. Download the SD Association's Formatting Tool from https://www.sdcard.org/downloads/formatter_4/eula_mac/

        2.2.2. Install and run the Formatting Tool on your machine

        2.2.3. Select "Overwrite Format"

        2.2.4. Check that the SD card you inserted matches the one selected by the Tool

        2.2.5. Click the "Format" button

    2.3. Linux

        2.3.1. We recommend using gparted (or the command line version parted)

        2.3.2. Format the entire disk as FAT

3. Extract the files contained in this NOOBS zip file.
4. Copy the extracted files onto the SD card that you just formatted so that this file is at the root directory of the SD card. Please note that in some cases it may extract the files into a folder, if this is the case then please copy across the files from inside the folder rather than the folder itself.

5. Insert the SD card into your Pi and connect the power supply.

Your Pi will now boot into NOOBS and should display a list of operating systems that you can choose to install.

If your display remains blank, you should select the correct output mode for your display by pressing one of the following number keys on your keyboard:

1. HDMI mode - this is the default display mode.
2. HDMI safe mode - select this mode if you are using the HDMI connector and cannot see anything on screen when the Pi has booted.
3. Composite PAL mode - select either this mode or composite NTSC mode if you are using the composite RCA video connector.
4. Composite NTSC mode

If you are still having difficulties after following these instructions, then please visit the Raspberry Pi Forum (http://www.raspberrypi.org/phpBB3/) for support.

APPENDIX B

SAMPLE /etc/apt/apt.conf and CHANGES TO /etc/wgetrc

/etc/apt/apt.conf :

Acquire::http::proxy "http://ipaddress:port/";
Acquire::https::proxy "https://ipaddress:port/";
Acquire::ftp::proxy "ftp://ipaddress:port/";
Acquire::socks::proxy "socks://ipaddress:port/";

/etc/wgetrc changes :

Search for text which says ***use_proxy*** and change the settings around there to this:

https_proxy = http://ipaddress:port/
http_proxy = http://ipaddress:port/
ftp_proxy = http://ipaddress:port/
 # If you do not want to use proxy at all, set this to off.

use_proxy = on

APPENDIX C

SETTING UP A WIFI HOTSPOT

Hardware Required:

Raspberry Pi, USB WiFi dongle, Power adapter, USB hub typically non-powered (powered is Optional), Keyboard and Mouse, SD Card – typically 8 GB or greater, Display Monitor, optional computer for debugging (methods are shown for a PC running Ubuntu 13.10 but most of the steps are similar for other Oses as well)

Design choices:

RASPBERRY PI – 512 MB; Model B; 3.5W; ARM 1176JZF (ARMv6 instruction set) – S 700 MHz – provides the ability to quickly prototype and the design and schematics are open sourced so, it is easier to make your own system with the same.

USB WiFi DONGLE – Realtek USB WiFi dongle N150 with the RTL 8188CUS chipset which has Linux drivers provided by Netgear

POWER ADAPTER – 5V at 1.5 A (at least) if you use a lower rating power adapter then, you will need a Powered USB hub to get the WiFi adapter working because it needs at least 500 mA of current at 5V and considering as it is that the Pi itself takes 700 mA (3.5W at 5V).

USB HUB – Considering that the model B - Raspberry Pi has only 2 USB ports; our requirement needs 2 input devices (1 keyboard and 1 mouse)

KEYBOARD and MOUSE – To control the viewer and also for first time install.

SD CARD – The Raspberry Pi website recommends one to install the software using the NOOBS (New Out of Box) software that they provide to install Raspbian (Debian equivalent for the Raspberry Pi)

Software Required:

New Out Of Box Software (NOOBS) v1.3.4 or Raspbian with Debian Wheezy – You can find both of them on : http://www.raspberrypi.org/downloads/

And of course a good Internet connection for your Raspberry Pi

Initial Set up:

1. Install Raspbian using NOOBS or with the downloaded .img file of Raspbian
2. Expand the file system to occupy all of the SD card else you will run out of space.
3. Enable SSH on the device.

(For detailed information about the same refer Appendix A)

4. Boot the Pi and set up and test the Ethernet and WiFi connection by plugging in the USB WiFi dongle and testing through ***ifconfig*** to see wlan0 turning up in your results.
5. Now, double click on the terminal and figure out the IP address using ***ifconfig***
6. Change proxies in case you are working through a proxy. You will have to change proxies in : ***/etc/apt/apt.conf*** and ***/etc/wgetrc*** (Appendix B - find sample apt.conf and wgetrc files)
7. Then run : ***sudo apt-get update && sudo apt-get upgrade*** to update and upgrade your repositories

Install required software:

Now, that everything is ready with regard to the Pi

We will install the software on the Pi that will act as the "hostap" (host access point)

1. Run : ***sudo apt-get install hostapd isc-dhcp-server***
2. Set up the ***DHCP server***:

   1. sudo nano /etc/dhcp/dhcpd.conf

   2. Find the lines that say :

   option domain-name "example.org";
   option domain-name-servers ns1.example.org, ns2.example.org;

   and comment them out by adding a # in front of them

   3. Find the lines that say:

   # If this DHCP server is the official DHCP server for the local
   # network, the authoritative directive should be uncommented.
   # authoritative;

and remove the # so it says

 # If this DHCP server is the official DHCP server for the local
 # network, the authoritative directive should be uncommented.
authoritative;

4. Then scroll down to the bottom and add the following lines:

subnet 192.168.42.0 netmask 255.255.255.0 {
range 192.168.42.10 192.168.42.50;
option broadcast-address 192.168.42.255;
option routers 192.168.42.1;
default-lease-time 600;
max-lease-time 7200;
option domain-name "local";
option domain-name-servers 8.8.8.8, 8.8.4.4;
}

Save the file by typing in **ctrl+ x** then **Y**

5. Then, run : ***sudo nano /etc/default/isc-dhcp-server***

Scroll down to ***INTERFACES=""*** and update it to say ***INTERFACES="wlan0"*** (to indicate the WiFi port)

Save the file by typing in **ctrl+ x** then **Y**

3. Set up ***wlan0 for static IP***

If you happen to have wlan0 active because you set it up, run ***sudo ifdown wlan0*** There's no harm in running it if you're not sure

1. We will set up the wlan0 connection to be static and incoming. Let us edit the interfaces file. To do this run : ***sudo nano /etc/network/interfaces***

2. Find the line : ***auto wlan0*** and add a # in front of the line, and in front of every line afterwards. Remove any line containing any wlan0 configuration settings because we will be changing them later.

3. Add the lines:

iface wlan0 inet static
address 192.168.42.1
netmask 255.255.255.0

4. After ***allow hotplug wlan0*** add a # in front of all the subsequent lines

At this point your Interfaces file should look like this:

auto lo

iface lo inet loopback
iface eth0 inet dhcp
allow-hotplug wlan0

iface wlan0 inet static
address 192.168.42.1
netmask 255.255.255.0

#iface wlan0 inet manual
#wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
#iface default inet dhcp

Save the file

5. Assign a static IP address to the wifi adapter by running:

sudo ifconfig wlan0 192.168.42.1

4. ***Configure Access Point***. We will set up a password-protected network

so only people with the password can connect.

1. Create a new file by running ***sudo nano /etc/hostapd/hostapd.conf***

2. Paste the following text in. Change the **ssid** and **wpa_passphrase** to something of your liking

interface=wlan0
driver=rtl871xdrv
ssid=Pi_AP
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=Raspberry
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP

3. If you are not using the RTL8188CUS chipset you will have to change the ***driver=rtl871xdrv*** to say ***driver=nl80211*** or something along those lines.

Save the file as usual.

4. Now we tell the Pi where to find this configuration file. Run ***sudo nano /etc/default/hostapd***

5. Find the line ***#DAEMON_CONF=""*** and edit it so it says : ***DAEMON_CONF="/etc/hostapd/hostapd.conf"***

5. ***Configure Network Address Translation***. Setting up NAT will allow multiple clients to connect to the WiFi and have all the data 'tunneled' through the single Ethernet IP. (You should do it even if only client is ever going to connect)

    1. Run ***sudo nano /etc/sysctl.conf***

    2. Scroll to the bottom and add : ***net.ipv4.ip_forward=1*** on a new line. Save the file. This will start IP forwarding on boot up.

    3. Also run : ***sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"*** to activate it immediately

    4. Run the following commands to create the network translation between the ethernet port **eth0** and the wifi port **wlan0**

        sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
        sudo iptables -A FORWARD -i eth0 -o wlan0 -m state –state RELATED,ESTABLISHED -j ACCEPT
        sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT

        You can check to see whats in the tables with:

        sudo iptables -t nat -S
        sudo iptables –S

    5. To make this happen on reboot (so you dont have to type it in every time) run

        sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"

    6. Run **sudo nano /etc/network/interfaces** and add :

        up iptables-restore < /etc/iptables.ipv4.nat

        to the very end.

6. ***Update hostapd***:

   1. Go to the Realtek downloads page http://152.104.125.41/downloads/downloadsView.aspx?Langid=1&PNid=21&PFid=48&Level=5&Conn=4&ProdID=277&DownTypeID=3&GetDown=false&Downloads=true

   2. Download linux 3.4.4_4749

   3. Copy the zip to the SD card using any computer which will place it in the Pi's /boot directory (or somehow get that file onto your Pi)

   4. Boot the Pi from the SD card

   5. Thereafter run :

   sudo mv /boot/RTL8192xC_USB_linux_v3.4.4_4749.20121105.zip .
   unzip RTL8192xC_USB_linux_v3.4.4_4749.20121105.zip
   mv RTL8188C_8192C_USB_linux_v3.4.4_4749.20121105/ rtl
   cd rtl
   cd wpa_supplicant_hostapd
   unzip wpa_supplicant_hostapd-0.8_rtw_20120803.zip
   cd wpa_supplicant_hostapd-0.8/
   cd hostapd
   make
   When done, hostapd binary is in the directory
   sudo mv /usr/sbin/hostapd /usr/sbin/hostapd.ORIG
   sudo mv hostapd /usr/sbin/
   sudo chmod 755 /usr/sbin/hostapd
   sudo /usr/sbin/hostapd /etc/hostapd/hostapd.conf

7. You must be able to see the *ssid* that you created. Now, to ***set it up as a daemon*** – a program that runs when the Pi boots run the following commands:

   sudo service hostapd start
   sudo service isc-dhcp-server start

8. To ***start the daemon services***. Verify that they both start successfully (no 'failure' or 'errors')
   Then to make it so it runs every time on boot
   sudo update-rc.d hostapd enable
   sudo update-rc.d isc-dhcp-server enable

9. Depending on your distribution, you may need to ***remove WPASupplicant***. Do so by running this command:

sudo mv /usr/share/dbus-1/system-services/fi.epitest.hostap.WPASupplicant.service ~/ followed by rebooting : ***sudo reboot***

YOUR WIFI HOTSPOT IS READY TO USE. You can use it as a regular WiFi router now. Your IP addresses will range from 192.168.42.10 to 192.168.42.50 and you can find the leased devices on */var/lib/dhcp/dhclient.leases* along with the WiFi mac addresses.

APPENDIX D

SETTING UP A MOBILE DEVICE SERVER

We explore 2 aspects to this process – one that we will accomplish using a Raspberry Pi and another for a custom built Android running our application on a Samsung Galaxy S – i9000.

RASPBERRY PI:

Hardware Required:

Raspberry Pi, USB WiFi dongle, USB battery, SD Card – typically 8 GB or greater

Optional: USB hub, Keyboard and Mouse, Display Monitor, Computer for debugging

Design choices:

Our design was centered around getting a battery life of 6 hours on a full usage ie. At CPU usage of 100% and WiFi dongle used to always transmit or receive as the case maybe.

This means that we will be expending 700mA on the Raspberry Pi but at the same time because we are not using a lot of the other modules, we can realistically say that the current used is around 500mA on the Pi.

For the WiFi dongle, we can expect a realistic usage of less than 50% and worst case at 50% ie. On an average 250mA on the dongle. This means 950mA of current usage on an average. This implies 950 x 6 mAh = 5700 mAh

Now, in the store there were 3 options for batteries : 2500 mAh delivering 1.5A peak current; 5000mAh delivering 1.5A peak current and 10000mAh delivering 3A peak current.

Also, each of them doubled in costs so, we bought the average cost one.

Software Required:

New Out Of Box Software (NOOBS) v1.3.4 or Raspbian with Debian Wheezy – You can find both of them on : http://www.raspberrypi.org/downloads/

And of course a good Internet connection for your Raspberry Pi

INITIAL SET UP

1. Install Raspbian using NOOBS or with the downloaded .img file of Raspbian
2. Expand the file system to occupy all of the SD card else you will run out of space.
3. Enable SSH on the device.

(For detailed information about the same refer Appendix A)

4. Boot the Pi and set up and test the Ethernet and WiFi connection by plugging in the USB WiFi dongle and testing through ***ifconfig*** to see wlan0 turning up in your results.
5. Now, double click on the terminal and figure out the IP address using ***ifconfig***
6. Change proxies in case you are working through a proxy. You will have to change proxies in : ***/etc/apt/apt.conf*** and ***/etc/wgetrc*** (Appendix B - find sample apt.conf and wgetrc files for settings with proxy)
7. Then run : ***sudo apt-get update && sudo apt-get upgrade*** to update and upgrade your repositories

VNC SERVER SETTING UP

1. Log in to the device and in the terminal type in : ***sudo apt-get install tightvncserver***

(We choose the tightvncserver because it is capable of accomplishing a wide range of encryption techniques – copyrect; corre; hextile; raw; rre; tight *and* zlib)

2. Once that is done, in the terminal set up the server:

tightvncserver :1

in the next steps it will ask you to set up a password. Make sure you give a long password at least 8 characters long. If you use a different server you might not be allowed to input 8 characters.

This condition of having to use a long password is due to the inherent security issues associated with the VNC protocol which is prone to a man in the middle type attacks where someone who is sniffing the network can easily pick up the password and gain full access to the computer

3. Once the password is set up, you can change the same by running **vncpasswd**

4. Now, that your vncserver is running you will have to set it up to run every time on boot. So, we can do it in 2 different ways:

   1. **First method**

   1. Do ***nano /home/pi/.vnc/xstartup***

      It must look something like this:

      #!/bin/sh

      xrdb $HOME/.Xresources
      xsetroot -solid black
      /opt/azureus/azureus &
      k3b &
      icewm-session &

   2. Copy the following into ***/etc/init.d/vncserver***

      ```
      #!/bin/sh -e
      ### BEGIN INIT INFO
      # Provides:          vncserver
      # Required-Start:    networking
      # Default-Start:     3 4 5
      # Default-Stop:      0 6
      ### END INIT INFO

      PATH="$PATH:/usr/bin/"

      # The Username:Group that will run VNC
      export USER="pi"
      #${RUNAS}

      # The display that VNC will use
      DISPLAY="1"

      # Color depth (between 8 and 32)
      DEPTH="16"

      # The Desktop geometry to use.
      #GEOMETRY="<WIDTH>x<HEIGHT>"
      GEOMETRY="800x600"
      #GEOMETRY="1024x768"
      #GEOMETRY="1280x1024"

      # The name that the VNC Desktop will have.
      ```

```
NAME="my-vnc-server"

OPTIONS="-name ${NAME} -depth ${DEPTH} -geometry ${GEOMETRY} :${DISPLAY}"

. /lib/lsb/init-functions

case "$1" in
start)
log_action_begin_msg "Starting vncserver for user '${USER}' on localhost:${DISPLAY}"
su ${USER} -c "/usr/bin/vncserver ${OPTIONS}"
;;

stop)
log_action_begin_msg "Stoping vncserver for user '${USER}' on localhost:${DISPLAY}"
su ${USER} -c "/usr/bin/vncserver -kill :${DISPLAY}"
;;

restart)
$0 stop
$0 start
;;
esac
```

3. Finally, enter ***sudo chmod +x /etc/init.d/vncserver***

4. Then, ***sudo update-rc.d vncserver defaults***

   This file needs to be exactly as it is without any extra spaces, else it will fail. It is very difficult to get this right on the first shot but it is the best thing. However, we recommend you try the second method because it is far easier to get this running and you will not have to sit around and meddle with stuff like the vncserver starting too early on in the boot process which is something that can cause problems while booting.

2. **Second method**

   1. **sudo nano /etc/xdg/xsession/autostart.conf** or **sudo nano /etc/rc.local** (in this case avoid the @ that is used in the next step)

   2. Toward the end of the file add **@vncserver :1** and then save the file.

   3. Then, **sudo reboot** on the terminal

5. Once the Pi reboots, double click on the wpa_gui icon that you find on the desktop or if you have SSHed (with Xwindows set ie. ssh -X pi@xxx.xxx.xx.xx) to the system type in ***wpa_gui*** and hit Return. Click on SCAN with the mouse and when you find the original WiFi hotspot that you have created – ***Pi_AP*** and in the wpa_passphrase option enter the pass key that we created – ***Raspberry***

Now, your system is ready to rumble and whenever you switch on the 2 of them you will be able to see it automatically showing on your viewer.

ANDROID DEVICE

Now, we will set this system up with an Android. We will also show how to build a Cyanogen mod based ROM on top of which we will run the VNC server App.

We will show the steps to build the app for Ubuntu 13.10. Linux is the only build environment for Android and using any other build environment is not supported.

Hardware Required

Galaxy S – i9000 (galaxysmtd)

Powerful Desktop computer – ideally having > 12GB RAM (though there are other things this is really important)

So, Let us get started.

SOFTWARE INSTALL INSTRUCTIONS:

Many of you probably have some kind of wrong Java installed unless you're starting with a fresh Ubuntu base, and even then maybe.

sudo apt-get purge openjdk-* icedtea-* icedtea6-*

Follow the instructions to remove OpenJDK.

sudo add-apt-repository ppa:webupd8team/java

This will add the correct PPA to your system for updated builds of Java 6 JDK that are compatible with 13.10. No more unrecognized Java version errors! And it will update automatically with the rest of your system.

Next, we actually need to install the package.

sudo apt-get update && sudo apt-get install oracle-java6-installer

Follow the on-screen instructions. You have to *Accept* the Licensing Agreement to complete the install.

java -version

You should see something like the following:
java version "1.6.0_45″
Java(TM) SE Runtime Environment (build 1.6.0_45-b06)
Java HotSpot(TM) 64-Bit Server VM (build 20.12-b01, mixed mode)

When that is done, install more software:

sudo apt-get install git-core gnupg ccache lzop flex bison gperf build-essential zip curl zlib1g-dev zlib1g-dev:i386 libc6-dev lib32ncurses5 lib32z1 lib32bz2-1.0 lib32ncurses5-dev x11proto-core-dev libx11-dev:i386 libreadline6-dev:i386 lib32z-dev libgl1-mesa-glx:i386 libgl1-mesa-dev g++-multilib mingw32 tofrodos python-markdown libxml2-utils xsltproc readline-common libreadline6-dev libreadline6 lib32readline-gplv2-dev libncurses5-dev lib32readline5 lib32readline6 libreadline-dev libreadline6-dev:i386 libreadline6:i386 bzip2 libbz2-dev libbz2-1.0 libghc-bzlib-dev lib32bz2-dev libsdl1.2-dev libesd0-dev squashfs-tools pngcrush schedtool libwxgtk2.8-dev python bison build-essential curl flex git gnupg gperf libesd0-dev libncurses5-dev libsdl1.2-dev libwxgtk2.8-dev libxml2 libxml2-utils lzop openjdk-6-jdk openjdk-6-jre phablet-tools pngcrush schedtool squashfs-tools xsltproc zip zlib1g-dev g++-multilib gcc-multilib lib32ncurses5-dev lib32readline-gplv2-dev lib32z1-dev

After that is done do this :

sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 /usr/lib/i386-linux-gnu/libGL.so

just making a symbolic link here to save some things for later.

And more code to run on your terminal:

mkdir ~/bin && curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo && chmod a+x ~/bin/repo

The binary for a program called "*repo*" will let you talk to git servers and download all that precious source code. That second part after the && allows it to be executable.

sudo nano ~/.bashrc

At the very bottom, add the following line:

export PATH=~/bin:$PATH

source ~/.bashrc

mkdir -p ~/android/system
cd ~/android/system/

Initialize the build to get stuff from the official CyanogenMod repositories
repo init -u git://github.com/CyanogenMod/android.git

Download the source code. To start the download of all the source code to your computer:

repo sync

The CM manifests include a sensible default configuration for repo, which we strongly suggest you use (i.e. don't add any options to sync). For reference, our default values are -j 4 and -c. The -j 4 part means that there will be four simultaneous threads/connections. If you experience problems syncing, you can lower this to -j 3 or -j 2. -c will ask repo to pull in only the current branch, instead of the entire CM history.

Prepare to wait a long time while the source code downloads. Because this is over 10 GB huge.

Helpful Tip: The repo sync command is used to update the latest source code from CyanogenMod and Google. Remember it, as you can do it every few days to keep your code base fresh and up-to-date.

Get prebuilt apps. Next,

cd ~/android/system/vendor/cm

then enter:

./get-prebuilts

You won't see any confirmation- just another prompt. But this should cause some prebuilt apps to be loaded and installed into the source code. Once completed, this does not need to be done again.

Prepare the device-specific code. After the source downloads, ensure you are in the root of the source code (cd ~/android/system), then type:

source build/envsetup.sh
breakfast galaxysmtd

This will download the device specific configuration and kernel source for your device. An alternative to using the breakfast command is to build your own local manifest. To do

this, you will need to locate your device on CyanogenMod's GitHub and list all of the repositories defined in cm dependencies in your local manifest.

Note: You MUST be using the newest version of repo or you will encounter errors with breakfast! Run repo selfupdate to update to the latest.

Helpful Tip : If you want to know more about what " source build/envsetup.sh" does or simply want to know more about the breakfast, brunch and lunch commands, you can head over to the Envsetup_help page

Extract proprietary blobs. Now ensure that your Galaxy S is connected to your computer via the USB cable and that you are in the ~/android/system/device/samsung/galaxysmtd directory (you can cd ~/android/system/device/samsung/galaxysmtd if necessary). Then run the extract-files.sh script:

./extract-files.sh

You should see the proprietary files (aka “blobs”) get pulled from the device and moved to the right place in the vendor directory. If you see errors about adb being unable to pull the files, adb may not be in the path of execution. If this is the case, see the adb page for suggestions for dealing with "command not found" errors.

Note: Your device should already be running the branch of CyanogenMod you wish to build your own version of for the extract-files.sh script to function properly. If you are savvy enough to pull the files yourself off the device by examining the script, you may do that as well without flashing CyanogenMod first.

Note: It’s important that these proprietary files are properly extracted and moved to the vendor directory. Without them, CyanogenMod will build without error, but you’ll be missing important functionality, such as the ability to see anything!

Turn on caching to speed up build. If you want to speed up subsequent builds after this one, type:

export USE_CCACHE=1

Helpful Tip Instead of typing cd ~/android/system every time you want to return back to the root of the source code, here’s a short command that will do it for you: croot . To use this command, as with brunch, you must first do “. build/envsetup.sh” from ~/android/system. Notice there is a period and space (“. ”) in that command.

Start the build. Time to start building! So now type:

croot
brunch galaxysmtd

The build should begin.

Helpful Tip : If the build doesn't start, try lunch and choose your device from the menu. If that doesn't work, try breakfast and choose from the menu. The command make galaxysmtd should then work.

Helpful Tip : A second, bonus tip! If you get a command not found error for croot or brunch or lunch, be sure you've done the " . build/envsetup.sh" command in this Terminal session from the ~/android/system directory.

If the build breaks…

If you experience this not-enough-memory-related error:

ERROR: signapk.jar failed: return code 1 make: *** [out/target/product/galaxysmtd/cm_galaxysmtd-ota-eng.root.zip] Error 1

...you may want to make the following change to:

system/build/tools/releasetools/common.py

Change: java -Xmx2048m to java -Xmx1024m or java -Xmx512m

Then start the build again (with brunch).

If you see a message about things suddenly being "killed" for no reason, your (virtual) machine may have run out of memory or storage space. Assign it more resources and try again.

Install the build

Assuming the build completed without error (it will be obvious when it finishes), type:

cd OUT

in the same terminal window that you did the build. Here you'll find all the files that were created. The stuff that will go in /system is in a folder called system. The stuff that will become your ramdisk is in a folder called root. And your kernel is called... kernel.

But that's all just background info. The two files we are interested in are (1) recovery.img, which contains ClockworkMod recovery, and (2) cm-[something].zip, which contains CyanogenMod.

Install CyanogenMod

Back to the OUT directory on your computer-- you should see a file that looks something like: *cm-11-20140515-UNOFFICIAL-galaxysmtd.zip*

Install that on your i9000.

And once you are done with this process, we are proud to welcome you to an elite bunch of self builders

Thereafter, now let's build the VNC server app.

Type in : **mkdir ~/AOSP && cd ~/AOSP**

Then, **repo init -u https://android.googlesource.com/platform/manifest**

Then**, repo sync** Again this is going to take a lot of time.

Parallel to this, open up another terminal window and type in: **git clone https://www.github.com/SuFizz/droid-vnc-server.git**

After that, download the NDK - https://developer.android.com/tools/sdk/ndk/index.html#Downloads

Also, you will have to download the Android ADT - https://developer.android.com/sdk/index.html for the Java part of the code and to publish the .apk file

The droid-VNC-server projects consists in three main modules parts: the daemon, wrapper libs and the GUI.

Daemon: Provides the vnc server functionality, injects input/touch events, clipboard management, etc. Available in jni/ folder

Wrapper libs: Compiled against the AOSP so everyone can build the daemon/GUI without having to fetch +2GB files. Currently there are 2 wrappers, gralloc and flinger. Available in nativeMethods/ folder, and precompiled libs in nativeMethods/lib/

GUI: GUI handles user-friendly control. Connects to the daemon using local Binder IPC.

COMPILE C DAEMON

On project folder ie on the /path/to/droid-vnc-server:

```
$ ndk-build
$ ./updateExecsAndLibs.sh
```

COMPILE WRAPPER LIBS

```
$ cd <aosp_folder>
$ . build/envsetup.sh
$ lunch
$ ln -s <droid-vnc-folder>/nativeMethods/ external/
```

To build:

$ cd external/nativeMethods
$ mm .
$ cd <droid-vnc-folder>
$ ./updateExecsAndLibs.sh

COMPILE GUI

Import using eclipse as a regular Android project and then run the same to get final output apk which you can flash on your system.

You will need root privileges for the application though.

At this point open the app and just hit START after connecting to the WiFi hotspot that you created and follow the same instructions that you did with the Pi

REFERENCES

1. https://en.wikipedia.org/wiki/TeraFLOPS#Hardware_costs
2. http://en.wikipedia.org/wiki/Cray_Y-MP
3. http://en.wikipedia.org/wiki/App_Store_%28iOS%29#Number_of_launched_applications
4. Chromecast on thingiverse: http://www.thingiverse.com/thing:129631
5. Building Cyanogen Mod for galaxysmtd: http://wiki.cyanogenmod.org/w/Build_for_galaxysmtd
6. Setting up a build system on Ubuntu 13.10 - http://forum.xda-developers.com/showthread.php?t=2464683
7. Specification of Nokia X - http://www.gsmarena.com/nokia_x-6067.php
8. Specification of Samsung Galaxy S5 - http://www.gsmarena.com/samsung_galaxy_s5-6033.php
9. Android VNC server support – http://code.google.com/p/fastdroid-vnc/
10. Android VNC help - https://play.google.com/store/apps/details?id=org.onaips.vnc
11. Android VNC server - https://code.google.com/p/android-vnc-server/
12. Mirror casting - https://plus.google.com/110558071969009568835/posts/iWcpAhBha5F
13. Android Transporter - https://esrlabs.com/blog/android-transporter/
14. Setting up WiFi hotspot - https://learn.adafruit.com/setting-up-a-raspberry-pi-as-a-wifi-access-point/overview
15. More WiFi hotspot help - http://andypi.co.uk/?page_id=220
16. NOOBS setup - http://www.raspberrypi.org/help/noobs-setup/
17. Android Play Store - https://play.google.com/store/apps
18. Apple App Store - http://www.apple.com/itunes/
19. Android Binder IPC - http://developer.android.com/reference/android/os/Binder.html
20. Karim Yaghmour – Embedded Android by O'Reilly
21. Source for Android - https://source.android.com/
22. Developer for Android - http://developer.android.com/index.html
23. Realtek Drivers - http://152.104.125.41/downloads/downloadsView.aspx?Langid=1&PNid=21&PFid=48&Level=5&Conn=4&ProdID=277&DownTypeID=3&GetDown=false&Downloads=true
24. OrbxJS - https://aws.amazon.com/marketplace/pp/B00FGB60MK