Cost of Securing IEEE 802.11s Mesh Networks Using CJDNS*

Thomas Hastings[1]

Abstract— **The Internet is weak, it is broken, and we are not doing anything to fix it. The Internet can be affected by natural disasters, wars, governments, and surveillance. It is running out of address space and the internet service providers are not incentivized to fix it. Mesh networks, using the IEEE standard 802.11s, may one day provide a more robust and resilient infrastructure. Although mesh networking is not a new idea or a new concept, wireless mesh networking is stripping previous barriers to entry. IEEE 802.11s makes mesh networks a reality for users who otherwise would never have been able to setup such a distributed network.**

Applications like cjdns are making it easier than ever to create secure wireless mesh network among communities. This paper will look at the system costs associated with using cjdns. How much performance are we willing to sacrifice for ease of use and security?

I. INTRODUCTION

The Internet is weak, it is broken, and we are not doing anything to fix it. The Internet can be affected by natural disasters, wars, governments, and surveillance [1]. It is running out of address space and the internet service providers are not incentivized to fix it [2]. Mesh networks, using the IEEE standard 802.11s, may one day provide a more robust and resilient infrastructure. Although mesh networking is not a new idea or a new concept, wireless mesh networking is stripping previous barriers to entry. IEEE 802.11s makes mesh networks a reality for users who otherwise would never have been able to setup such a distributed network.

Further reducing barriers to entry, open source applications such as cjdns are openly developed and shared across the Internet. Cjdns boasts a couple of features that make implementing a wireless mesh network simple. Cjdns implements an encrypted IPv6 network using public-key cryptography for address allocation and a distributed hash table for routing. This provides near-zero-configuration networking, and prevents many of the security and scalability issues that plague existing networks [3].

There are a growing number of users on the Internet joining mesh network communities that utilize cjdns. The communities are typically based on geographic location. For example, Philadelphia has a thriving mesh network group called Philly Mesh [4]. Toronto also has a strong group at Toronto Mesh [5]. These geographically separated mesh networks are utilizing an open source network, Hyperboria, that leverages the open Internet to connect to each other using cjdns [6]. At what cost do these networks provide convenient setup and privacy?

*This work was done as a semester project for CS 5960 at the University of Colorado at Colorado Springs school of Applied Science and Engineering

[1]Thomas Hastings is an MESE graduate student at the University of Colorado at Colorado Springs `thasting@uccs.edu`

II. BACKGROUND

In this section I explain the basics of IEEE 802.11s, some of the history of the protocol, and I explain how cjdns works at a high level.

A. IEEE 802.11s

IEEE 802.11s is an amendment to IEEE 802.11 Wireless Local Area Network. In 2012 the amendment was officially added to the specification. The protocol integrates mesh networking services and protocols with 802.11 at the MAC layer. The primary scope of the amendment was to create a wireless distribution system with automatic topology learning and wireless path configuration. The amendment includes dynamic data delivery via unicast and broadcast as well as multicast [12].

B. cjdns

Cjdnss creator, Caleb James DeLisle, has a vision and a mission for cjdns. He wants everyone to be able to setup and host mesh networks if they wish to do so. He wants the network to be secure from espionage and forgery. He also wants to make it easier for individuals or companies to provide Internet services to the masses. Cjdns is made up of three main components. The components are dependent on the other components. If one component malfunctions or fails to operate correctly then the entire node is disabled.

1) The Switch: The first component is the switch. The switch passes packets to a director based on the least significant bits of the routing label. The director, a component within the switch, knows the next interface in the path and passes the data along.

Example: Supposing Alice wanted to send a packet to Fred via Bob, Charlie, Dave and Elinor, she would send a message to her switch with the first Director instructing her switch to send down its interface to Bob, the second Director instructing Bob's switch to send to Charlie and so on [2].

2) The Router: The second component is the router. The router is responsible for identifying the most efficient path for packets to take in order to reach the requested node. The router listens for searches that are sent out by other routers. The searches provide network information to other routers on the mesh network. The router must also respond to searches from other routers in order to help map the mesh network. Lastly, the router forwards packets based on the responses previously collected.

3) The CrytoAuth: The third, and final component, is the crytpoauth. The crytpoauth provides a wrapper for interfaces. The interface allows users to send and receive encrypted packets. The cryptoauth is also responsible for the initial handshake packets which allows nodes to join the mesh network. The overhead for the packets passing through the cryptoauth is 120 bytes [2].

III. ANALYSIS DESIGN

The final goal of this study is to determine the cost associated of cjdns on mesh networks. In this section I provide an overview or my analytical procedure in two phases.

A. Collecting Phase

The first step in the collecting phase is to identify common metrics that I could use across the hardware. I decided to use the bandwidth throughput as a measurement. One of the challenges I ran into was that bandwidth throughput as a measurement was not the only measurement that I needed to track. I found that my hardware was limited in terms of CPU cycles when running cjdns. In addition to collecting throughput measurements I also collected CPU measurements.

B. Analysis Phase

After the collecting phase I began to group analyze the measurements. This was a fairly simple process due to the nature of the measurements. The outcome, however, was surprising.

IV. IMPLEMENTATION

I used two Raspberry Pis, one version 1 model B and one version 2 Model B, two TP-Link N150 wireless high gain adapters, one Linksys WRT54G router, and a laptop to implement the experiment.

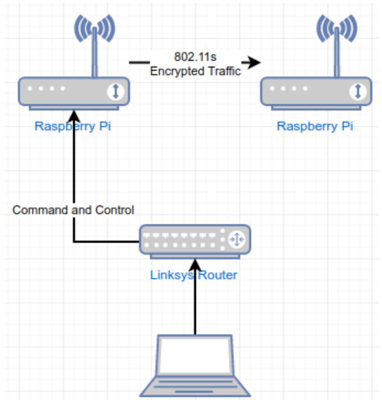
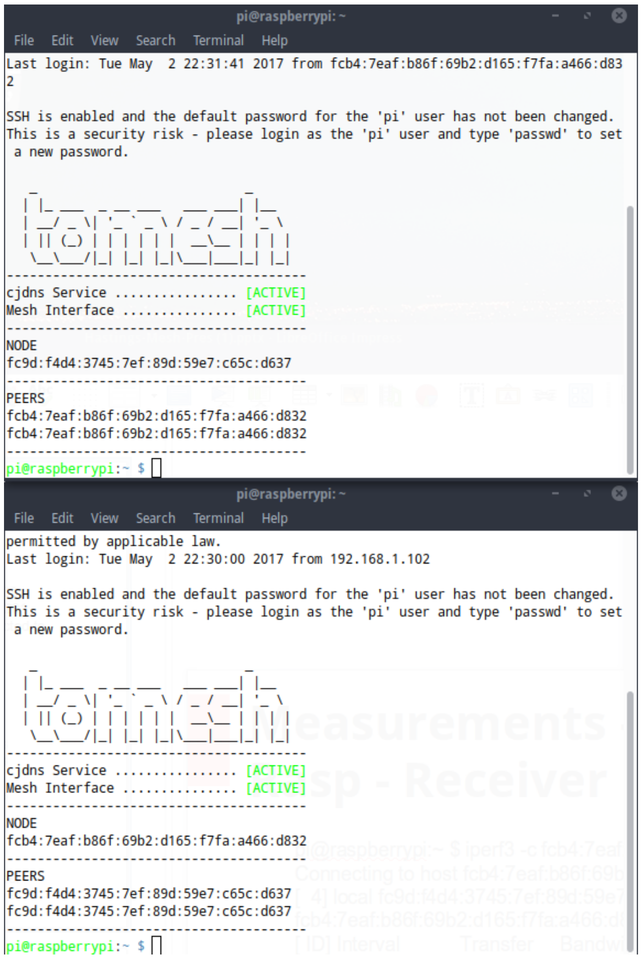


Fig. 1. Implementation Diagram

A. Raspberry Pis

I flashed the Raspberry Pis with the Raspian Jessie operating systems. Raspian is based off of the popular Ubuntu Linux. I then installed cjdns on each of the Pis. I placed one of the TP-Link N150 wifi usb dongles in each of the Pis and configured the dongles to broadcast using 802.11s. Both Pis broadcast an SSID of tomsmeshnetwork. Due to the how I configured cjdns and because both Pis shared the same SSID the Pis auto connected and formed a simple two node mesh network. A screenshot of the Pis' console is in Fig 2.



```
Last login: Tue May  2 22:31:41 2017 from fcb4:7eaf:b86f:69b2:d165:f7fa:a466:d83
2

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
 a new password.

  _                            _
 | |_ ___  _ __ ___   ___  ___| |__
 | __/ _ \| '_ ` _ \ / _ \/ __| '_ \
 | || (_) | | | | | |  __/\__ \ | | |
  \__\___/|_| |_| |_|\___||___/_| |_|
---------------------------------------
cjdns Service ................ [ACTIVE]
Mesh Interface ............... [ACTIVE]
---------------------------------------
NODE
fc9d:f4d4:3745:7ef:89d:59e7:c65c:d637
---------------------------------------
PEERS
fcb4:7eaf:b86f:69b2:d165:f7fa:a466:d832
fcb4:7eaf:b86f:69b2:d165:f7fa:a466:d832
---------------------------------------
pi@raspberrypi:~ $
```

```
permitted by applicable law.
Last login: Tue May  2 22:30:00 2017 from 192.168.1.102

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
 a new password.

  _                            _
 | |_ ___  _ __ ___   ___  ___| |__
 | __/ _ \| '_ ` _ \ / _ \/ __| '_ \
 | || (_) | | | | | |  __/\__ \ | | |
  \__\___/|_| |_| |_|\___||___/_| |_|
---------------------------------------
cjdns Service ................ [ACTIVE]
Mesh Interface ............... [ACTIVE]
---------------------------------------
NODE
fcb4:7eaf:b86f:69b2:d165:f7fa:a466:d832
---------------------------------------
PEERS
fc9d:f4d4:3745:7ef:89d:59e7:c65c:d637
fc9d:f4d4:3745:7ef:89d:59e7:c65c:d637
---------------------------------------
pi@raspberrypi:~ $
```

Fig. 2. Raspberry Pi Console

B. Command and Control

I used the Linksys WRT54G as a hub to interact with the Pis. I plugged each Pi into the router and then I attached my laptop to the router as well. This gave me the ability to ssh into one of the nodes and interact with the other node of the 802.11s mesh. This is also how I was able to collect the required metrics needed to identify the cost of of cjdns.

C. Iperf3

I was able to ssh onto one of the nodes using the Linksys router. Once there I used an application that I installed on both Pis called Iperf3. Iperf3 is a client / server application that measures bandwidth throughput. I was able to use the Pis IPv6 address from the node to connect one node to the other and pass data across.

D. Top

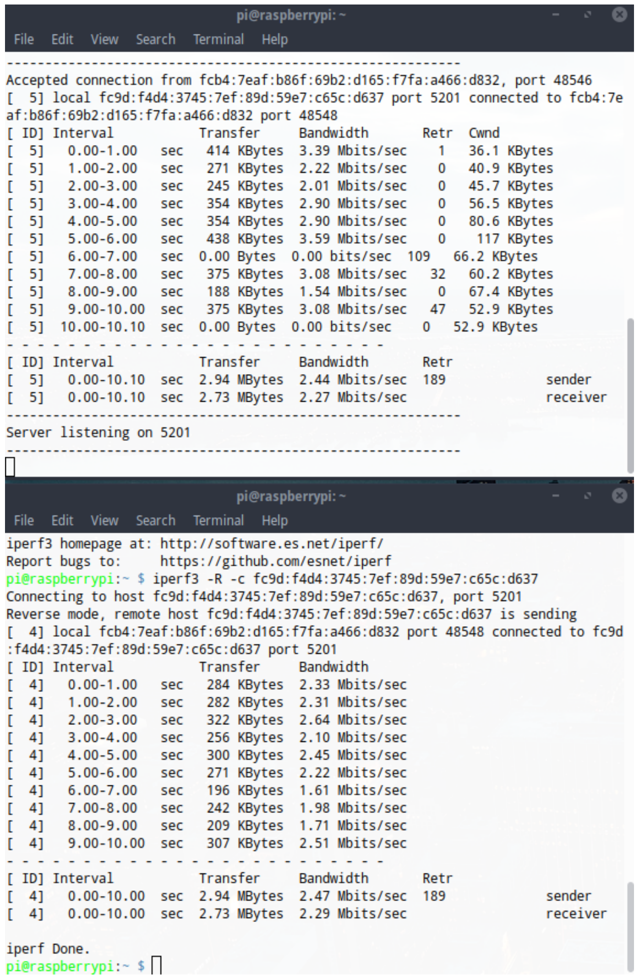
Top is a Linux command that displays processor activity and tasks that are managed by the kernel in real-time. Its similar to the Application Manager on Windows that shows processes and hardware / system utilization.

E. Data Collection

I ran 10 total tests across each of the Pis. Each test consists of passing data from node 1 to node 2 and then from node 2 back to node 1. In addition to the bandwidth throughput tests I took samples of the CPU usage and recorded the measurements.

V. MEASUREMENTS

Sample measurements can be seen in Fig 3 and Fig 4.



```
-----------------------------------------------------------
Accepted connection from fcb4:7eaf:b86f:69b2:d165:f7fa:a466:d832, port 48546
[  5] local fc9d:f4d4:3745:7ef:89d:59e7:c65c:d637 port 5201 connected to fcb4:7e
af:b86f:69b2:d165:f7fa:a466:d832 port 48548
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  5]   0.00-1.00   sec   414 KBytes  3.39 Mbits/sec    1   36.1 KBytes
[  5]   1.00-2.00   sec   271 KBytes  2.22 Mbits/sec    0   40.9 KBytes
[  5]   2.00-3.00   sec   245 KBytes  2.01 Mbits/sec    0   45.7 KBytes
[  5]   3.00-4.00   sec   354 KBytes  2.90 Mbits/sec    0   56.5 KBytes
[  5]   4.00-5.00   sec   354 KBytes  2.90 Mbits/sec    0   80.6 KBytes
[  5]   5.00-6.00   sec   438 KBytes  3.59 Mbits/sec    0    117 KBytes
[  5]   6.00-7.00   sec  0.00 Bytes  0.00 bits/sec  109   66.2 KBytes
[  5]   7.00-8.00   sec   375 KBytes  3.08 Mbits/sec   32   60.2 KBytes
[  5]   8.00-9.00   sec   188 KBytes  1.54 Mbits/sec    0   67.4 KBytes
[  5]   9.00-10.00  sec   375 KBytes  3.08 Mbits/sec   47   52.9 KBytes
[  5]  10.00-10.10  sec  0.00 Bytes  0.00 bits/sec    0   52.9 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  5]   0.00-10.10  sec  2.94 MBytes  2.44 Mbits/sec  189             sender
[  5]   0.00-10.10  sec  2.73 MBytes  2.27 Mbits/sec                  receiver
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

```
iperf3 homepage at: http://software.es.net/iperf/
Report bugs to:     https://github.com/esnet/iperf
pi@raspberrypi:~ $ iperf3 -R -c fc9d:f4d4:3745:7ef:89d:59e7:c65c:d637
Connecting to host fc9d:f4d4:3745:7ef:89d:59e7:c65c:d637, port 5201
Reverse mode, remote host fc9d:f4d4:3745:7ef:89d:59e7:c65c:d637 is sending
[  4] local fcb4:7eaf:b86f:69b2:d165:f7fa:a466:d832 port 48548 connected to fc9d
:f4d4:3745:7ef:89d:59e7:c65c:d637 port 5201
[ ID] Interval           Transfer     Bandwidth
[  4]   0.00-1.00   sec   284 KBytes  2.33 Mbits/sec
[  4]   1.00-2.00   sec   282 KBytes  2.31 Mbits/sec
[  4]   2.00-3.00   sec   322 KBytes  2.64 Mbits/sec
[  4]   3.00-4.00   sec   256 KBytes  2.10 Mbits/sec
[  4]   4.00-5.00   sec   300 KBytes  2.45 Mbits/sec
[  4]   5.00-6.00   sec   271 KBytes  2.22 Mbits/sec
[  4]   6.00-7.00   sec   196 KBytes  1.61 Mbits/sec
[  4]   7.00-8.00   sec   242 KBytes  1.98 Mbits/sec
[  4]   8.00-9.00   sec   209 KBytes  1.71 Mbits/sec
[  4]   9.00-10.00  sec   307 KBytes  2.51 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  2.94 MBytes  2.47 Mbits/sec  189             sender
[  4]   0.00-10.00  sec  2.73 MBytes  2.29 Mbits/sec                  receiver

iperf Done.
pi@raspberrypi:~ $
```

Fig. 3. Bandwidth Throughput - Measurement

VI. EVALUATION

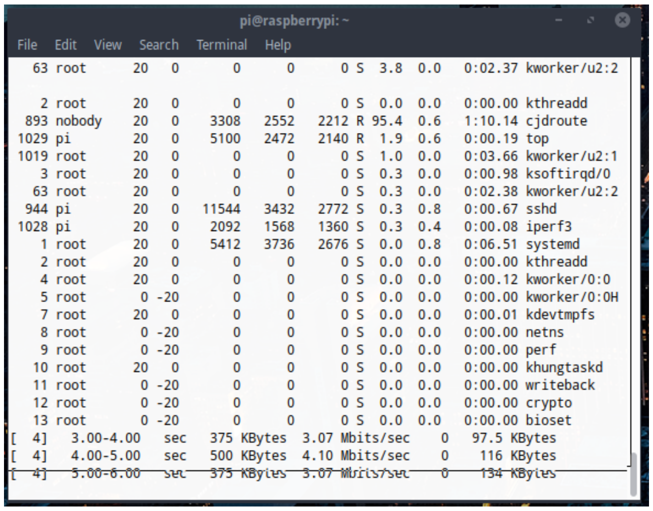
This section provides the evaluation of the measurements that were taken previously during the implementation phase.

A. Unencrypted

Raspberry Pi Model B – 37.9 Mbit/s
Raspberry Pi 2 Model B - 75.3 Mbit/s

B. cjdns

Raspberry Pi Model B - 2.27 Mbit/s
Raspberry Pi 2 Model B - 3.10 Mbit/s



```
   63 root      20   0       0      0      0 S  3.8  0.0   0:02.37 kworker/u2:2

    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
  893 nobody    20   0    3308   2552   2212 R 95.4  0.6   1:10.14 cjdroute
 1029 pi        20   0    5100   2472   2140 R  1.9  0.6   0:00.19 top
 1019 root      20   0       0      0      0 S  1.0  0.0   0:03.66 kworker/u2:1
    3 root      20   0       0      0      0 S  0.3  0.0   0:00.98 ksoftirqd/0
   63 root      20   0       0      0      0 S  0.3  0.0   0:02.38 kworker/u2:2
  944 pi        20   0   11544   3432   2772 S  0.3  0.8   0:00.67 sshd
 1028 pi        20   0    2092   1568   1360 S  0.3  0.4   0:00.08 iperf3
    1 root      20   0    5412   3736   2676 S  0.0  0.8   0:06.51 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    4 root      20   0       0      0      0 S  0.0  0.0   0:00.12 kworker/0:0
    5 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S  0.0  0.0   0:00.01 kdevtmpfs
    8 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 netns
    9 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 perf
   10 root      20   0       0      0      0 S  0.0  0.0   0:00.00 khungtaskd
   11 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 writeback
   12 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 crypto
   13 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 bioset
[  4]   3.00-4.00   sec   375 KBytes  3.07 Mbits/sec    0   97.5 KBytes
[  4]   4.00-5.00   sec   500 KBytes  4.10 Mbits/sec    0    116 KBytes
[  4]   5.00-6.00   sec   375 KBytes  3.07 Mbits/sec    0    134 KBytes
```

Fig. 4. Top - CPU Usage - 93%

C. Limiting Factors

Both Raspberry Pis during the test maintained an average of 99.35% CPU utilization. This leads me to believe the limiting factor may not necessary be the radio or transmission of packets. Rather, I believe the limiting factor in this experiment was the hardware.

VII. CHALLENGES

I faced a couple of challenges while working on this research project. The first challenge was with the hardware that I picked. Cjdns no longer supports the Raspberry Pi 1 Model B. In order to overcome this challenge I had to use cjdns version 18 on that Raspberry Pi. I was also limited to 2.4GHz for this exercise due to the USB WiFi dongle I choose. I dont believe this really impacted the outcome of the project because of the hardware limitations. Lastly, I reached the threshold for the hardware. The CPU was under so much load that it makes it hard to determine if the low bandwidth throughput is due to the 802.11s standard or due to the limited processing power on the Raspberry Pis.

VIII. RELATED WORK

IEEE 802.11s is a growing topic of research due to the growing popularity of mesh networks. Many have researched and presented topics on 802.11s. There are a few good papers with ideas on how to secure 802.11s but few get to the implementation.

A. Security Based Analysis

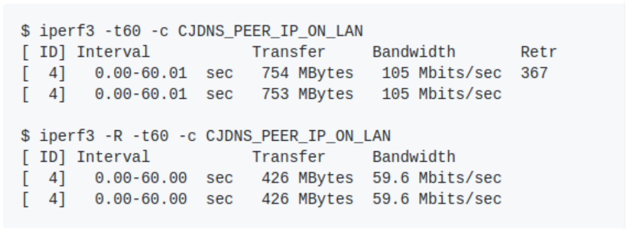
One of the few papers that provide an implementation offers a protocol called SHWMP [9]. SHWMP differs from cjdns because it is only using crytographic extensions to provide authenticity which is similar to cjdns but is lacking features that cjdns provides. In addition to encryption cjdns provides distributed hash tables for routing and auto configuration features that allow nodes to automatically connect and join mesh networks.

B. State of the World Based Analysis

Hal Hodson published an article in the New Scientist titled, Lets Start The Net Again which was published on August 10, 2013 [8]. Hal doesnt dive into the protocol but he discusses how mesh networks are beginning to take shape from coast to coast. Hal talks about cjdns and how the software is core to the mesh networks.

C. Benchmarks

The Philly Mesh group has benchmarks for multiple devices including the Raspberry Pis. Their most promising benchmark is with a device called an Odroid-c2 [5]. You can see their benchmark in fig 5.



```
$ iperf3 -t60 -c CJDNS_PEER_IP_ON_LAN
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-60.01  sec   754 MBytes   105 Mbits/sec  367
[  4]   0.00-60.01  sec   753 MBytes   105 Mbits/sec

$ iperf3 -R -t60 -c CJDNS_PEER_IP_ON_LAN
[ ID] Interval           Transfer     Bandwidth
[  4]   0.00-60.00  sec   426 MBytes  59.6 Mbits/sec
[  4]   0.00-60.00  sec   426 MBytes  59.6 Mbits/sec
```

Fig. 5. Philly Mesh Odroud-c2 Benchmark

IX. FUTURE WORK

For future work I would like to use hardware with better specifications. Perhaps using an Odroid C2 would make a difference. The Odroid has a faster processor, more RAM, USB 3.0 ports, and a gigabit ethernet port. I would also like to try and connect multiple devices to one node and pass data to other nodes in the network using the one node. With the current setup only one device can be connected to a node at a time. I would like to add a WiFi access point that provides devices access to a single node and use the node as a point of contact for the devices to communicate on the mesh network.

X. CONCLUSIONS

The main purpose of this study is the analyze the cost of using cjdns to secure mesh networks over IEEE 802.11s. The cost is high when using a Raspberry Pi due to the Pis hardware. The bandwidth throughput dropped over 90% when using cjdns on Raspberry Pis. I believe applications like cjdns are going to usher in a new era of connected communities. We need to put more resources towards mesh networks and more resources towards developing applications like cjdns.

REFERENCES

[1] Ïts Time To Take Mesh Networks Seriously (And Not Just For The Reasons You Think)ẅww.wired.com/2014/01/its-time-to-take-mesh-networks-seriously-and-not-just-for-the-reasons-you-think/ 2014. Web. 10 May 2017.

[2] ”cjdelislecjdns”, GitHub, 2017. [Online]. Available: github.com/cjdelisle/cjdns/blob/master/doc/Whitepaper.md. [Accessed: 10- May- 2017].

[3] ”cjdelislecjdns”, GitHub, 2017. [Online]. Available: github.com/cjdelisle/cjdns. [Accessed: 10- May- 2017].

[4] ”IEEE SA - 802.11s-2011 - IEEE Standard for Information Technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 10: Mesh Networking”, Standards.ieee.org, 2011. [Online]. Available: standards.ieee.org/findstds/standard/802.11s-2011.html. [Accessed: 10- May- 2017].

[5] Phillymesh.net, 2017. [Online]. Available: phillymesh.net/. [Accessed: 10- May- 2017].

[6] ”Toronto Mesh”, Tomesh.net, 2017. [Online]. Available: tomesh.net/. [Accessed: 10- May- 2017].

[7] ”Hyperboria”, Hyperboria.net, 2017. [Online]. Available: hyperboria.net/. [Accessed: 10- May- 2017].

[8] Hodson, Hal, New Scientist. 8/10/2013, Vol. 219 Issue 2929, p20-21. 2p.

[9] Hodson, Hal, New Scientist. 8/10/2013, Vol. 219 Issue 2929, p20-21. 2p.

[10] M. S. Islam, M. A. Hamid, and C. S. Hong, SHWMP: A Secure Hybrid Wireless Mesh Protocol for IEEE 802.11s Wireless Mesh Networks, Transactions on Computational Science VI Lecture Notes in Computer Science, pp. 95114, 2009.