HyperMesh (AKA “The Mesh”)
Regional Mesh Networking Standard

Michael Quiniola

12 DEC 2017

I. Introduction

The Internet has become the world’s primary source for information, and the primary medium for communication and the free exchange of ideas. In recent years this environment has been threatened by government surveillance and threats to net neutrality. This means that not only do organizations monitor the traffic, but the flow of Internet traffic through Internet service providers (ISPs) can be throttled based on content type and the information about it can be sold to the highest bidder. The Internet could become censored to the point where usability is nil, lose its ability to provide privacy due to surveillance legislature, or get turned off altogether (as has been seen during government crackdowns, e.g. Cameroon). This is coupled with the fact that the Internet is running on the power grid and is susceptible to natural disasters, limiting or cutting communications between people and emergency services.

However, networking technology has become powerful and readily accessible to the point that it can facilitate access to wider area networks and bypass the need for an ISP, as well as government wiretapping and censorship. This paper proposes a mesh networking technology stack that can not only be replicated across the globe, but can seamlessly and immediately integrate with other mesh networks that utilize the same build.

II. Summary

There are many other mesh solutions that solve a multitude of problems including censorship, automatic routing, encryption, and physical linkage. The issue with these solutions is not that they don’t work, but that none of them solve all of the problems. The proposed solution is to combine existing mesh technologies to allow scalability, decentralization, and encryption, over a physical connection that the ISPs do not control. This can be done with a combination of mesh networking methods that are already available and currently being used.

III. Mesh Technologies Used

LibreMesh (LiMe)

LibreMesh is a router and wireless access firmware based upon the OpenWRT/LEDE firmware. It is built around the idea of auto-configurable wireless mesh networks and is currently used by many local meshes. LiMe is an actively developed firmware and is a suitable replacement for QMP (Quick Mesh Project). The routing protocols used by LiMe are B.A.T.M.A.N-adv on OSI layer 2 and BMX7 on OSI layer 3.

Issues with using LiMe as a sole technology include:

- DHCP is facilitated by a distributed table, but is sent between all nodes using A.L.F.R.E.D., which hurts scalability and speed.
- Anecdotal evidence suggests that some mesh devices can't deal with overcrowding (> ~5 devices).
- Traffic between nodes is not encrypted.

CJDNS (Caleb James DeLisle Network Suite)

CJDNS is a networking protocol that implements encrypted end-to-end traffic over IPv6, utilizing public-key cryptography for network address allocation and a distributed hash-table for routing. In order to connect to a CJDNS network you must be connected to another node in that network using either connection credentials on that node, or ethernet beaconing on the same physical link. The largest network of CJDNS nodes is called “Hyperboria”, from which “HyperMesh” derives its name.

Issues with using CJDNS/Hyperboria include:

- Hyperboria does not utilize its own physical connections; it connects over the standard Internet.
- In order to connect to a network, you need to have credentials from a node already in the network.

There are other perceived disadvantages of CJDNS having to do with security that fall under the realm of security through obscurity, but those are not relevant to this whitepaper.

IEEE 802.11s

IEEE 802.11s is a Wireless LAN standard and an IEEE 802.11 amendment for mesh networking which defines how wireless devices can create a WLAN mesh network. IEEE 802.11s requires Hybrid Wireless Mesh Protocol (HWMP) to be supported by default, which is based on Ad-Hoc On-Demand Distance Vector Routing (AODV) and tree-based routing. Every wireless device is a Mesh Station and allows multi-hop to route traffic between devices that are not in range of each other, but are in range of an intermediary Mesh Station. It relies on the other wireless standards (i.e. IEEE 802.11a/b/g/n/ac) for its physical layer connections. Peer authentication is done via a pre-shared key.

Issues with using 802.11s include:

- Only an OSI layer 2 protocol.
- Requires IP address management.
- Only the wireless signal is encrypted, not the traffic.

These are only some of the technologies being used in this guide. There are two separate examples of a home/client node listed in the appendices; the first is based on LibreMesh and the second is based on Raspberry Pi.

IV. The Mesh Stack

Combining the different technologies above creates a mesh networking software stack that will allow automatic connections between mesh devices within range of each other.

Layer 3	CJDNS
Layer 2	LibreMesh + 802.11s
Layer 1	Wifi (802.11a/b/g/n/ac)

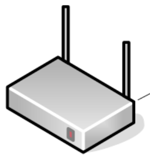

Figure 1: Mesh Software Stack

The routers and wireless access point hardware are loaded with LibreMesh firmware. Any device that is OpenWRT/LEDE-capable can have LibreMesh installed on it since LibreMesh is based on OpenWRT/LEDE. LibreMesh is capable of using IEEE 802.11s for its OSI layer 2 routing and mesh connection protocol. LiMe can also ad hoc utilizing B.A.T.M.A.N., but IEEE 802.11s is preferred.

CJDNS is installed on the LiMe firmware using the respective opkg package. Peer information is visible in the LuCi web interface under LiMe. For the purposes of this paper, installing CJDNS on LiMe is sufficient for functionality of the mesh net. For the purpose of higher throughput, it is better to use CJDNS on a device behind LiMe that has a proper CPU (or other hardware) for cryptographic functions.

Using a basic wireless SSID (for the purposes of this paper "HyperMesh" is used) with IEEE 802.11s, wireless nodes will automatically connect to each other if they are within range. Since all of the encryption, routing, and IP addressing is handled via CJDNS, there is no need to implement pre-shared keys, DHCP, or 802.11s routing. With CJDNS set up using its "ethernet beaconing" feature, CJDNS nodes will automatically peer with each other when on the same physical link (the wireless signal). If there is a dedicated physical cable, it can be used as well.

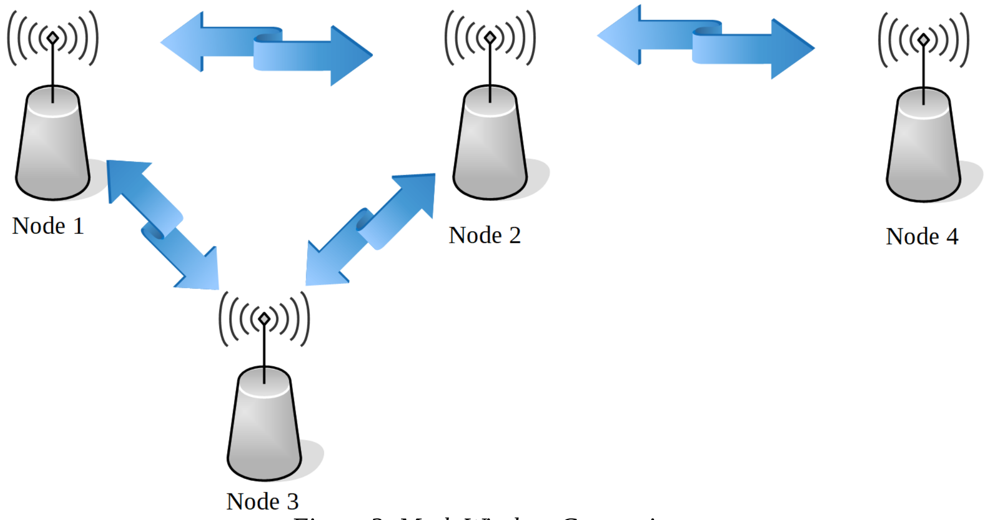

Figure 2: Mesh Wireless Connections

In Figure 2, all of the nodes are running the Mesh Stack (LiMe, 802.11s, CJDNS), all nodes are using the “HyperMesh” SSID, and all nodes have “ethernet beaconing” configured on CJDNS. Nodes 1, 2, and 3 are within range of each other and all automatically connect to the 802.11s mesh. The CJDNS beaconing can detect the other nodes on that mesh and automatically set up encrypted tunnels between them. Node 4 is within range of Node 2 but not the other nodes. Node 4’s CJDNS will connect to Node 2 as a peer and Node 2 will route traffic from Node 4 to the other nodes and vice versa. Even though Node 2 is an intermediary, it cannot see the encrypted traffic going between Node 4 and any other node due to the end-to-end encrypted nature of CJDNS. The setup above can be replicated using Raspberry Pis and wireless adapters that support IEEE 802.11s.

V. Infrastructure

One of the primary goals of the mesh is to be independent of ISPs and censorship. This can only be done by having active physical links that are separate from the standard Internet. By owning the physical links, natural disaster resistance can be built in and facilitate quick redeployment after an event. Due to the ever-increasing storage capacity and decreasing size of batteries, the entire mesh can and should be run on solar power, potentially with a battery backup. This document will describe a solar usage setup and provide different examples of node set-ups in order to have a starting point from which a community can build its network and adjust to different needs.

There are two major types of nodes that will be used in the mesh: an infrastructure node and a home/client node. Infrastructure nodes are purposed solely for the connection of nodes which cannot see each other due to topography or distance. Home/client nodes can still provide intermediary traffic routing between non-linked nodes, but their main purpose is to connect a user/user’s network to the mesh.

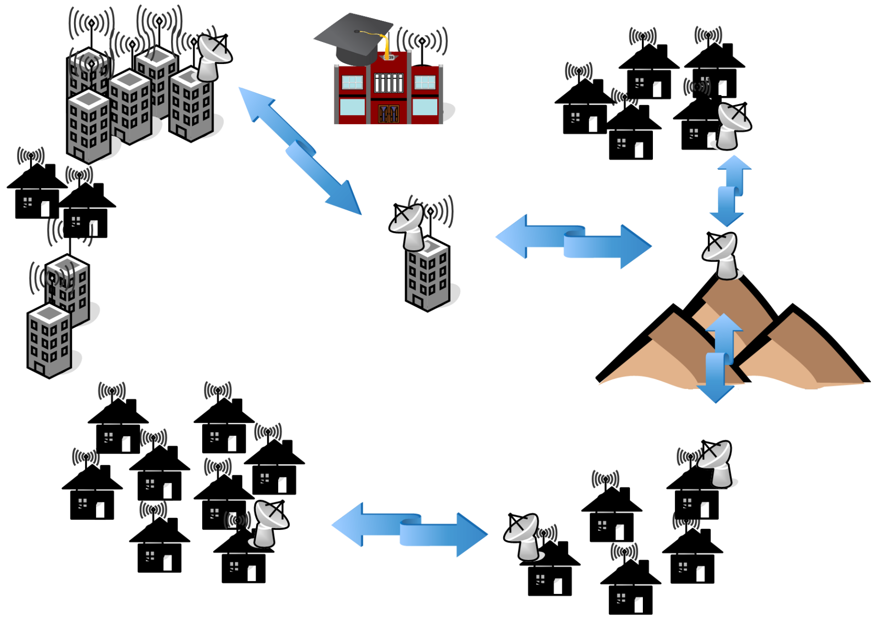

Figure 3: Example Regional Wireless Mesh Topology

Infrastructure nodes are intended to connect to the mesh when no other mesh node (infrastructure, home, or otherwise) is within range of a node trying to connect. In the event that any intermediary node becomes inaccessible, traffic will be routed around the downed node through another available link. The more dense a network becomes with mesh nodes, the more infrastructure nodes are made redundant, used for backup, or reconfigured for longer-distance connections.

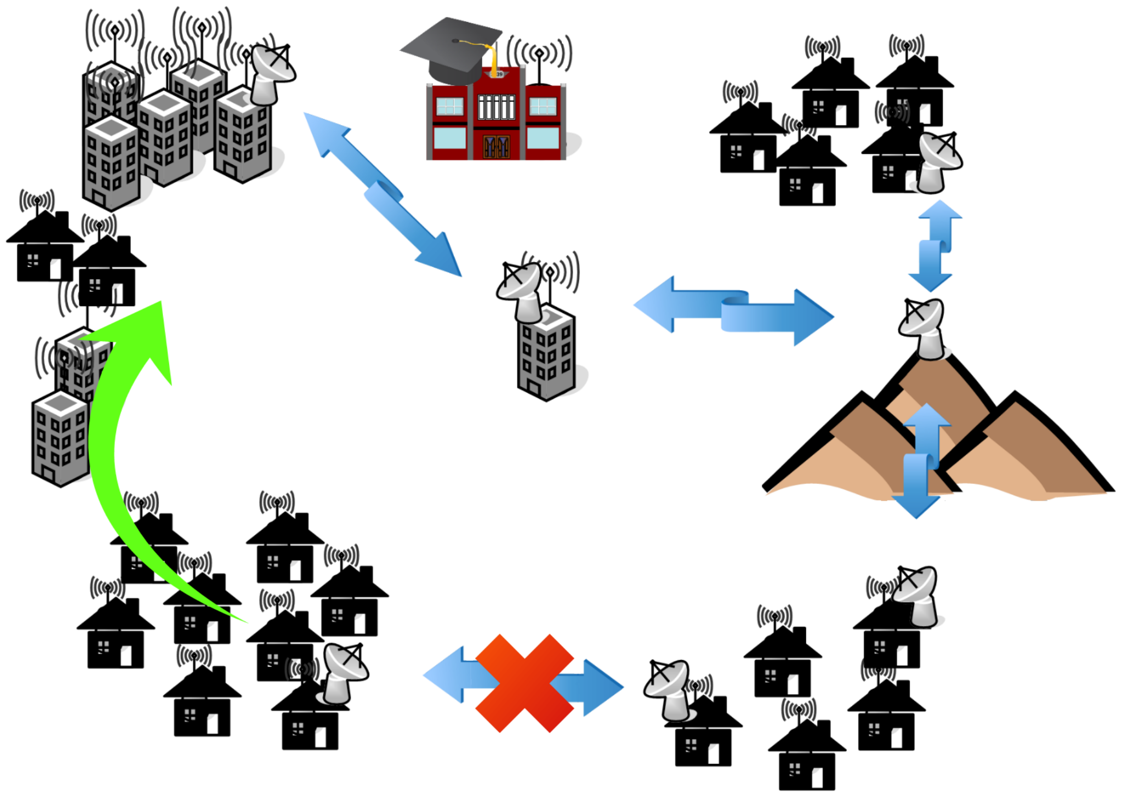
Figure 4: Example Topology with a Downed Link

In the diagram above, the southernmost link has gone down due to an unknown circumstance and traffic is routed around the blockage through other links.

Due to its automatic configuration and standard OSI layer 3 build, if one community/city/region happens to be connected to another community/city/region, traffic will automatically be routed between the regions and both will have access to the other region's resources.

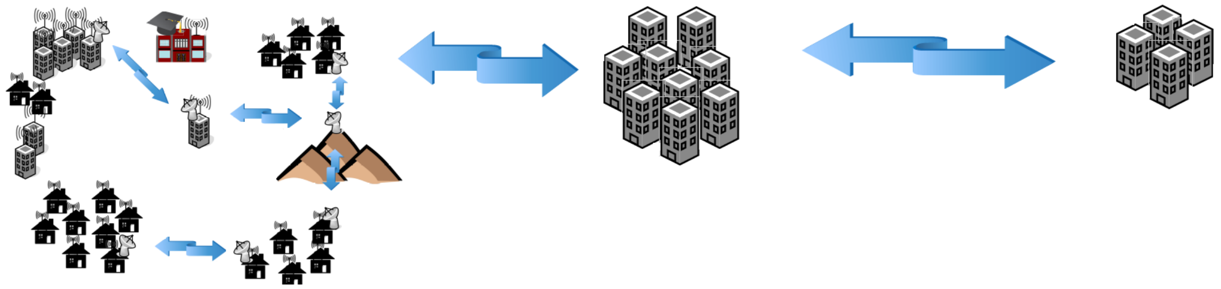
Figure 5: Example Topology Connected to Other Meshes

VI. Self-sustainability

The infrastructure design and mesh software stack previously described still requires power. In the event of a natural disaster, power outage, or tripped circuit breaker, a power solution that doesn’t require connection to the grid is needed. In home nodes, a simple UPS (Uninterruptible Power Supply) is all that may be required to power the network devices. The UPS in this situation should *only* be used to power the connectivity hardware, so as to prolong uptime in the event of an outage. In most urban cities, unintended power outages last no longer than one day with an average of 2 hours of interrupted service, or 3 hours including major events (EIA, 2016). For infrastructure nodes, the power grid may be used to supplement electricity, but should not be the primary solution.

The answer to this problem is solar power. Solar has become significantly less expensive, dropping in cost by an average of 25% over the course of 2017. At the time of this writing, a 100w 12v solar panel can be purchased for $100-$120(USD). Battery capacity should last an absolute minimum of 24 hours for the equipment that is running the node. It is recommended that battery capacity should be able to run the infrastructure node for a minimum of 72 hours without sunlight.

For every 15 watts of solar power rating, the panel outputs roughly 1 amp. This guide errs on the side of caution and assumes 8 hours of sunlight during the day, and that cloudy weather does not equal sunlight. This is difficult to accomplish, with capacity and size depending on how large the infrastructure node setup will be. The solar power output must be able to charge the batteries to full during the day despite draw from the mesh equipment. Figure 6 shows an example setup and calculation based on power requirements for an infrastructure node with 2 separate unidirectional transmitters and 1 omnidirectional transmitter using Ubiquiti Nanostation M5 series and a standard LibreMesh capable wireless router with a high-gain antenna (omnidirectional router is for nearby access only):

2x Ubiquiti Nanostation M5 @ 24v 0.5 amps = 1 amp @ 24v		=	**2 amps @ 12v**
1x LibreMesh router @ 12v 1 amps		=	**1 amps @ 12v**
	Total power	=	**3 amps @ 12v**
Minimum battery requirement	= 3 amps * 24 hours	=	**72 amps @ 12v**

Night time amperage usage = 3 amps * 16 hours = 48 amps

Required amperage = 48 amp nighttime loss + 24 amp equipment usage = 72 amps

72 amps / 8 hour day = **9 amps per hour solar requirement**

9 amps * 15 w = **minimum 135 watt solar panel needed**

Minimum required power equipment:

Battery – 72 amps at 12v
Solar – 12v 135 watt solar panel

Figure 6: Infrastructure Node Power Calculations

The above is just an example of an infrastructure node setup in a populated area with home nodes nearby (the reason for the omnidirectional antenna). As previously stated, the solar option can be supplemented/replaced with a power grid solution, but the power grid cannot be depended upon in the event of an outage or natural disaster.

VII. Mesh Services

The reason for the need to connect to the Internet has to do with the services that people, organizations, and companies provide. Any service that can be hosted on the standard Internet can be hosted over the mesh net. This includes but is not limited to websites, game servers, cloud solutions, VoIP, video calls, and messaging services. Listed are some of the services that can be hosted on (and in some cases federated to) the mesh net to replace services on the clearnet. Standard HTTP/web-site hosting programs are not listed.

- Mastodon - https://joinmastodon.org/

  Mastodon is an open-source and federated social networking platform that is similar to Twitter. A Mastodon server allows users to create accounts on that instance and follow users on other servers due to its federation capabilities. Once a server is federated with another, a feed is created that pulls from any other servers that a given Mastodon server has a relation to.

- Matrix – https://matrix.org

  Matrix is a federated, open-source communications platform similar to IRC, Slack, and Discord that allows for real-time synchronization. It has decentralized, cryptographically-signed conversation history (timeline and key-value stores) replicated over all the servers that participate in a room. Some of its features include VoIP signaling, voice calls, video chat, end-to-end encryption, group and 1:1 messaging, read receipts, and a decentralized content repository. It is currently the platform of choice for local mesh networks to communicate.

- IPFS – https://ipfs.io

  IPFS is a peer-to-peer hypermedia protocol. It aims to replace HTTP and reduce bandwidth usage by distributing high volumes of data in a highly efficient manner. It removes data duplication across the network and tracks version history of every file. Each file and all of the blocks in it are identified with a unique cryptographic hash.

Any service that can be hosted on the standard Internet can be hosted on the mesh. These are just a few examples of software that can replace standard Internet services, provided owners of the existing Internet-equivalent services decide not to host on the mesh themselves.

VIII. Conclusion

An open-source mesh network that is completely encrypted end-to-end, decentralized, community-owned, self-powered, and which operates seamlessly between regions can reduce if not eliminate dependency on Internet service providers. If needed, access to the standard Internet can be facilitated through the mesh much like a VPN (Virtual Private Network). The provided examples are just a few of the methods of connecting to a wireless mesh that is running CJDNS. Once built, it is imperative that the mesh be absolutely free to access. As wireless technology becomes faster in the years to come, the modular nature of the node builds allows for them to be quickly and easily upgraded as needed. These software and hardware technologies are inexpensive, highly configurable, and are what the world needs in order to be independent of governments and corporations that would regulate our freedom of speech and freedom of open communication across the globe.

References

- https://wiki.exarcheianet.gr/images/0/08/MeshNetworkingTalkSlides.pdf
- https://www.libremesh.org/
- https://github.com/cjdelisle/cjdns
- https://github.com/cjdelisle/cjdns/blob/master/doc/Whitepaper.md
- https://joinmastodon.org/
- https://matrix.org/
- https://ipfs.io
- https://github.com/SeattleMeshnet/meshbox
- https://www.eia.gov/todayinenergy/detail.php?id=27892
- https://diasporafoundation.org/

Revision History:

None

Appendix A – LibreMesh Installation

The following is the documentation from LibreMesh's Quick Starting Guide
(http://libremesh.org/docs/en_quick_starting_guide.html):

Compatible Hardware

It is recommended that the router has at least 8 MB of flash memory. For 4 MB routers use the special firmware named *`-mini`*.

These wireless routers have been tested with LibreMesh and have 8 MB of flash memory:

- TP-Link WR842ND
- TP-Link WR1043ND
- TP-Link WDR3500
- TP-Link WDR3600
- TP-Link WDR4300
- Dragino MS14
- Alix 2d2
- Ubiquiti Unifi AP
- Ubiquiti AirRouter
- Ubiquiti AirGateway
- Ubiquiti NanoStation M5 XW
- Ubiquiti NanoStation M5 XM
- Ubiquiti NanoBridge M5
- Ubiquiti NanoStation LoCo M2
- Ubiquiti PicoStation M2
- Ubiquiti Bullet M2

Also models with 4 MB have been tested, using the *`-mini`* images:

- TP-Link WR740N
- TP-Link WR741ND
- TP-Link WR841ND

For detailed information on these routers check out our http://libremesh.org/docs/hardware/index.html page.

Many other models are supported even if not tested yet (as far as we know), see through our firmware images at http://downloads.libremesh.org/dayboot_rely/17.06/targets/.

Get the Firmware

Choose a source for your firmware

- For a precompiled firmware with default configuration (e.g. wireless AP name LibreMesh.org) you can use our http://downloads.libremesh.org/dayboot_rely/17.06/targets/ site;
- You can compile LibreMesh firmware on your computer using https://github.com/libremesh/lime-sdk (advanced, supports custom community profiles);
- Our https://chef.libremesh.org/ platform allows you to build an image of the firmware online. The platform supports custom community profiles.

For more options check the http://libremesh.org/getit.html page.

Download the correct firmware image

Find the download for your router by name or model number. You may need to search different variations or aliases. You can find more detailed router model instructions at https://wiki.lede-project.org/toh/start.

If you are installing for the first time (a router with stock firmware), choose the link ending with `-factory.bin`. If there’s no `-factory.bin` image or you are upgrading an existing install of LibreMesh, OpenWrt or LEDE, choose the link ending with `-sysupgrade.bin`.

Installation Procedure

Open your router web interface

Using an ethernet cable connect to a LAN port on your router. Make sure that the ethernet cable is the only active network interface on your computer (e.g. disable the wireless interface).

If the router is running stock firmware, follow manufacturer instructions to connect to the router. Its IP should be written on the original box or under the router. Usually just opening 192.168.0.1 or 192.168.1.1 in a web browser lets you reach the router web interface. If you can’t connect to the router because you can’t find its IP, you can try the IP address of your gateway. For getting it go to the terminal and use netstat -rn (mac), or ip route show default (Linux). More details on finding your router’s IP can be found at http://www.howtogeek.com/233952/how-to-find-your-routers-ip-address-on-any-computer-smartphone-or-tablet/ and http://www.computerworld.com/article/2474776/wireless-networking/network-security-find-the-ip-address-of-your-home-router.html.

If the router is running OpenWrt or LEDE the instructions in the previous paragraph should apply.

If the router is running LibreMesh just opening http://thisnode.info should get you to the web interface.

Then you can log in as admin (if unchanged, the username and password will be on the router’s box, on OpenWrt by default there’s no admin password, on LEDE by default it’s an empty password).

For more connection options see http://libremesh.org/docs/en_connecting_nodes.html.

If you suspect you can’t connect to your router because of a damaged configuration, follow the Troubleshooting guide at http://libremesh.org/docs/en_troubleshooting.html.

Flashing

For Ubiquiti AirMax series routers, flashing on top of AirOS versions 5.6.x will brick your device (the recovery procedure requires opening the router chassis and connecting directly to its serial port). If your router has AirOS 5.6.x, you will have to find and download an AirOS 5.5.x version and use it to downgrade your router.

Once you’ve logged in as root or admin in your router, reach the firmware upgrade page.

If there’s a Keep Settings option, take care to UNCHECK it. It is checked by default on OpenWrt/LEDE.

Upload the firmware image file you’ve downloaded and click Flash Image. Wait a couple of minutes for the process to complete. Reconnect to the ethernet interface (for getting the new IP) and open http://thisnode.info.

Congratulations, you have a working LibreMesh router!

Connect to Your LibreMesh Router

Just connect to the router via its wireless AP interface or via ethernet cable on its LAN port and open http://thisnode.info in the web browser.

If is the first time you connect to the router, you will have to set an admin/root password.

Leaving a LibreMesh router with no admin password is a huge security risk.

For more connection options see How to connect to nodes page at http://libremesh.org/docs/en_connecting_nodes.html.

If you suspect you can’t connect to your router because of a damaged configuration, follow the Troubleshooting guide at http://libremesh.org/docs/en_troubleshooting.html.

Share the Internet Connection with the LibreMesh Network

LibreMesh is automatically sharing with the rest of the mesh network any internet connection is connected to the router WAN port. There’s no problem if more than one internet gateway is connected to the LibreMesh network, likely the one closest to the client will be used.

If the LibreMesh router has no WAN port (just LAN ports, or just one ethernet port), one of the ethernet ports has to be configured as WAN port in order to share the internet connection. Refer to next section for configuration.

Configuration

WORK IN PROGRESS

Refer to the LibreMesh config file page for detailed information at http://libremesh.org/docs/en_config.html.

Using the Console Interface (optional)

Until here we went through the installation, connection and configuration procedures using the LibreMesh web interface.

As in every Linux-based system there’s the availability of a textual console interface for advanced configuration and hardcore users.

This part of the guide should not be needed for normal LibreMesh use.

Flashing Via the Console Interface (optional)

This is possible just if you’re upgrading an existing OpenWrt, LEDE or LibreMesh installation, not from stock firmware.

Copy the downloaded firmware image to the /tmp directory on your target router using the scp command.

Do not try to copy the firmware image to directories different from /tmp. They have limited memory access.

In case the router already has LibreMesh you can do this with

```
scp /LOCAL/PATH/TO/BUILD.bin root@thisnode.info:/tmp/
```

Otherwise (upgrading from OpenWrt or LEDE) you will need to insert the router IP address in

```
scp /LOCAL/PATH/TO/BUILD.bin root@ROUTERIPADDRESS:/tmp/
```

When upgrading from OpenWrt, in order to connect via ssh/scp you will need to have an admin/root password set via the web interface or via telnet and the passwd command. Check out Connecting to your own node for help on this at http://libremesh.org/docs/en_connecting_nodes.html.

Now connect to the console interface using ssh, if LibreMesh is already running with ssh root@thisnode.info or with ssh root@ROUTERIPADDRESS if OpenWrt or LEDE are running.

Then enter the /tmp directory where the firmware is present with cd /tmp, check the presence of the file with ls and install it with

```
sysupgrade -n lede-ROUTERMODEL-squashfs-sysupgrade.bin
```

The -n option for sysupgrade command is needed for discarding the previous configuration files. Omitting the -n option is never a good idea when flashing LibreMesh.

For more information on the sysupgrade process, see these OpenWRT instructions at http://wiki.openwrt.org/doc/howto/generic.sysupgrade.

Configuring Via the Console Interface (optional)

Rather than using the web interface, modifying directly /etc/config/lime file allows to access more advanced options but increases the risk of writing broken configuration.

You can use the vi or vim text editor for editing /etc/config/lime, the settings in this file will override the default ones in /etc/config/lime-defaults.

You can find examples and documentation in the /docs/lime-example file (you can find it online here at https://github.com/libremesh/lime-packages/blob/develop/packages/lime-docs/files/lime-example) as well as in LibreMesh config file page at http://libremesh.org/docs/en_config.html.

After saving the edits to the lime file, apply the changes to system configuration files launching the lime-config command. Next reboot the router with reboot && exit to apply the new settings.

Appendix B – CJDNS Installation and Configuration on LibreMesh

The older versions of LibreMesh (before 17.06 DaybootRely) do not have the cjdns opkg in the repositories, which is required by the luci-app-cjdns package. In order to install on older versions, the package must be manually transferred and installed to the node.

This installation can be done from the web interface
1. Log in to the terminal via ssh, then update the repositories and install the luci-app-cjdns. This will automatically install the cjdns package as well, which is a prerequisite for the luci app. Optionally, you should install iperf3 for network testing

```
opkg update && opkg install luci-app-cjdns
opkg install iperf3
```

2. Ethernet beaconing is turned on by default for all interfaces.

Appendix C – Home/Client Node – LibreMesh Based

The purpose here is to replace the cable router an ISP would provide in a standard Internet service installation. A home node does not necessarily require LibreMesh if it is not being run on a router.

A LibreMesh-based home node requires:

- Completion of the steps to install LibreMesh on a capable router and configure CJDNS.
- A high-gain outdoor mountable antenna with cable long enough to reach it.
- An UPS dedicated to network equipment.
- (Optional) A unidirectional access point or antenna for the router.

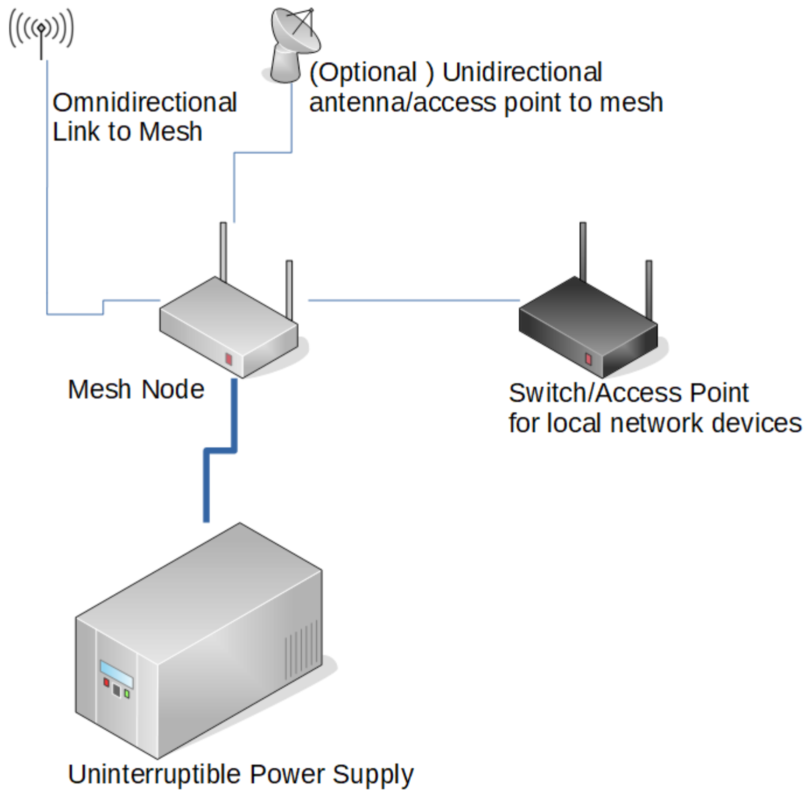
Omnidirectional Link to Mesh
(Optional) Unidirectional antenna/access point to mesh
Mesh Node
Switch/Access Point for local network devices
Uninterruptible Power Supply

Figure 7: Example Home Node Setup

1. Install the package **kmod-ipt-nat6**

```
opkg update && opkg install kmod-ipt-nat6
```

2. Enter the following ip6tables rules into the file **/etc/firewall.user**:

```
ip6tables -t nat -A POSTROUTING -o tuncjdns -j MASQUERADE
ip6tables -A FORWARD -i tuncjdns -o br-lan -m state --state RELATED,ESTABLISHED -j
ACCEPT
ip6tables -A FORWARD -i br-lan -o tuncjdns -j ACCEPT
```

3. Under Network > Interfaces, click **Edit** next to **LM_NET_BR_LAN_ANYGW_IF.** Scroll down to DHCP Server and click the IPv6 Settings tab. Change DHCPv6-Service to **server mode**, DHCPv6-Mode to **stateful-only**.

4. After rebooting the device, verify that the computer can retrieve an IPv6 address from the access point. To test CJDNS connectivity, ping a CJDNS connected address from the computer.

Appendix D – Home/Client Node – Raspberry Pi Based

This is a home node setup courtesy of the Toronto Mesh Net group. It works on the Raspberry Pi 1, 2, 3, and Orange Pi Zero.

After initial pi setup, the following commands are to be run in the terminal:

```
wget https://raw.githubusercontent.com/tomeshnet/prototype-cjdns-pi/develop/scripts/install /
&& chmod +x install && WITH_MESH_POINT=true WITH_WIFI_AP=true WITH_IPFS=false /
WITH_PROMETHEUS_NODE_EXPORTER=true WITH_PROMETHEUS_SERVER=false WITH_GRAFANA=false / WITH_H_DNS=true
WITH_H_NTP=true WITH_FAKE_HWCLOCK=true WITH_EXTRA_TOOLS=true / TAG_PROTOTYPE_CJDNS_PI=develop ./install

sudo systemctl stop hostapd
sudo systemctl disable hostapd
sudo sed -i s/wlan0/eth0/ radvd.conf
sudo sed -i s/wlan0/eth0/ dnsmasq.conf
sudo sed -i s/eth0/eth1/   /etc/network/interfaces
sudo sed -i s/wlan0/eth0/ /etc/network/interfaces
sudo sed -i s/eth0/tun0/   /etc/hostapd/nat.sh
sudo sed -i 's/^exit 0/\/etc\/hostapd\/nat.sh\nexit 0/' /etc/rc.local
```

The first line will cause the machine to automatically reboot after it has completed running. Then, after entering the rest of the commands, the ethernet interface will provide DHCP to the network and NAT all traffic into CJDNS and through the mesh.

Appendix E – Infrastructure Node

An infrastructure node’s primary purpose is to connect nodes that cannot communicate with each other either due to distance or due to topographical obstructions. This set-up is a minimal infrastructure node with one omnidirectional transmitter and one unidirectional transmitter. Power requirements will expand with more transmitters. Infrastructure nodes can be as small using single omnidirectional antenna, using two directional antennas, or as large as 4-directional antennas, a PoE camera, and a Raspberry Pi. All infrastructure node builds should be tailored to a specific need. A small area might only need an infrastructure node to get around a canyon edge, which would only need two directional antennas. A node might only need one strong omnidirectional antenna.

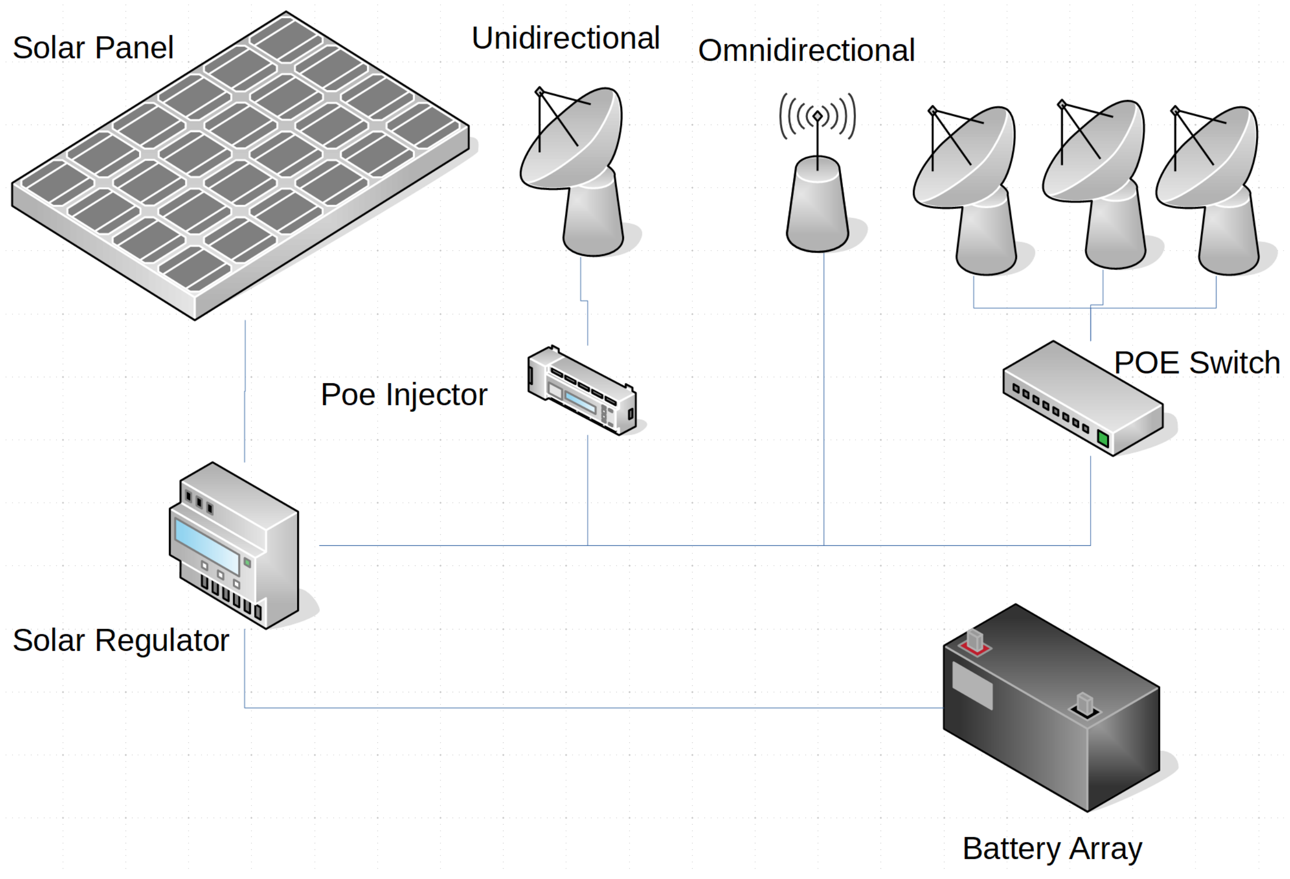

Figure 7: Example Infrastructure Node Options

Figure 7 is a diagram of a simple solar power setup with possible options for network equipment. A larger infrastructure node can be used to service a larger area, but will require a much larger battery and solar array. If using ethernet bridging on the wireless devices, then a device such as a Raspberry Pi behind the nodes can be used as the CJDNS box for that node to increase throughput. The diagrams only show power connections, not network connections. If POE injectors are utilized then any number of transmitters larger than 2 will require a switch, which will factor into the power usage. Figure 8 is an example of a POE injector setup, but utilizing a Raspberry Pi.

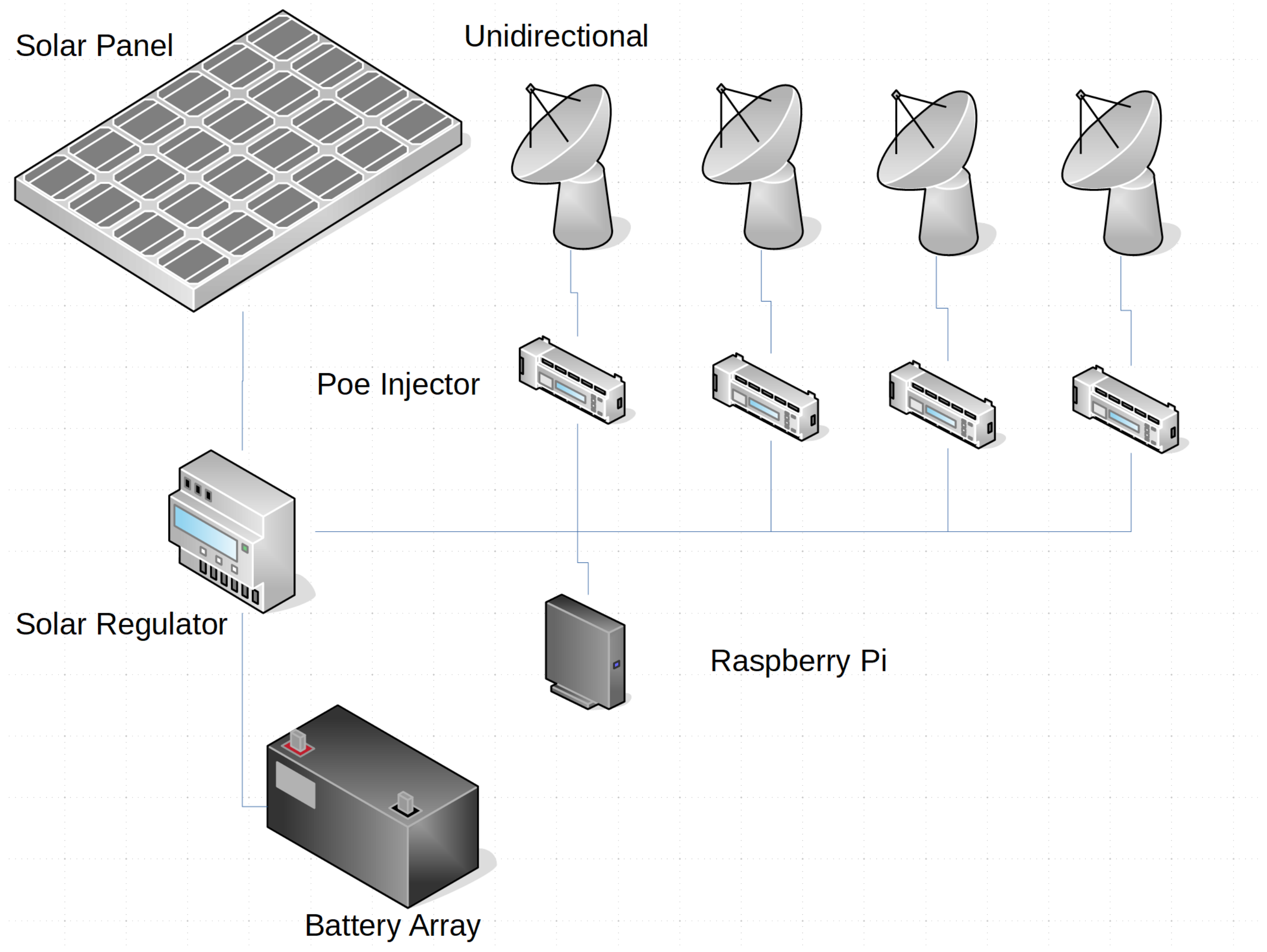

Figure 8: Example Infrastructure Node with Raspberry Pi

The following is an example case for a build design for a central mesh node.

This node requires an omnidirectional and a unidirectional transmitter, and happens to be the first link between the main area mesh network and a suburban area. This central mesh node can be pointed to from multiple locations in the suburban community, but also needs the range to reach the next nearest mesh node. Because of this, it is ideal to use 2.4GHz for the community link. For this set-up, a Raspberry Pi is used to guarantee throughput.

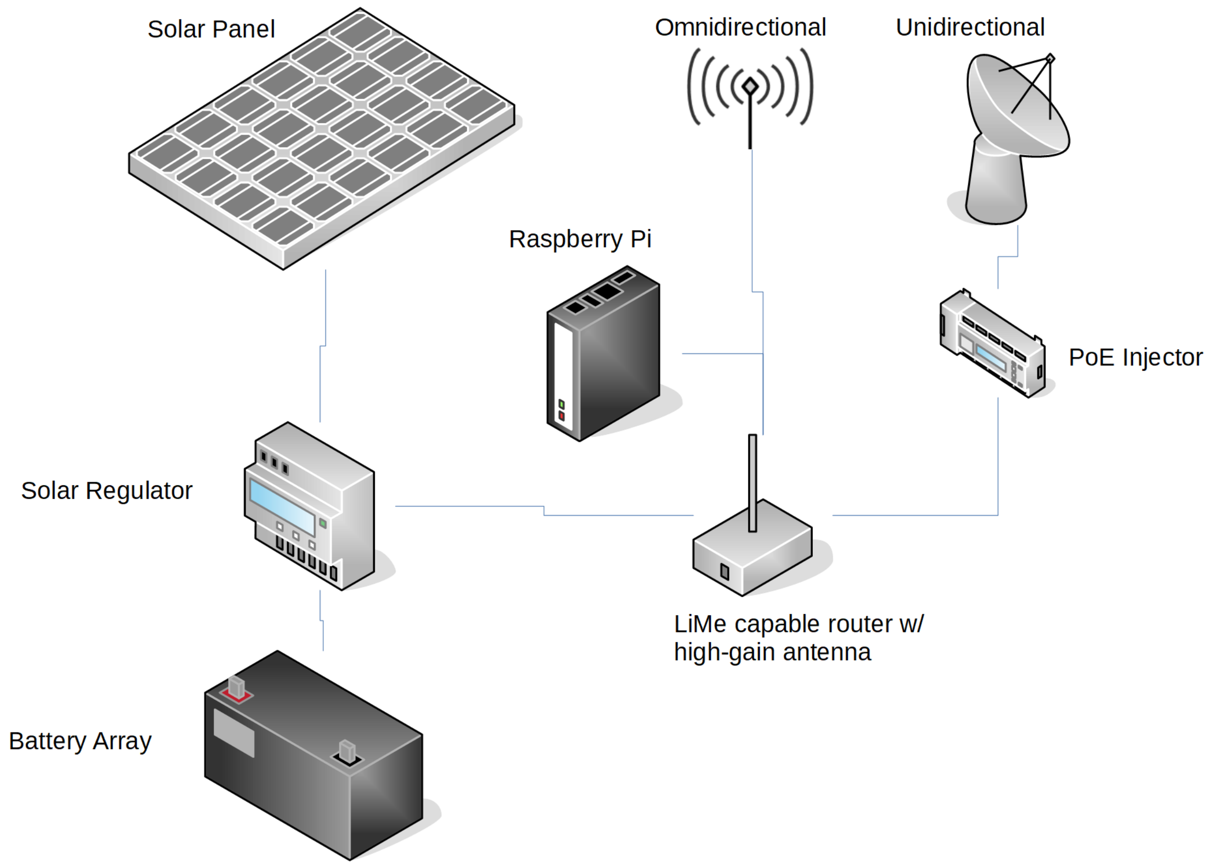

Figure 9: Example Node build for Suburban Community

Power Requirements:

Unidirectional Transmitter	= .5 amps @ 24v	=	**1 amp @ 12v**
Router w/ Omnidirectional	= 1 amp @ 12v	=	**1 amp @ 12v**
Raspberry Pi	= 2.5 amp @ 12v	=	**1 amp @ 12v**
	Total	=	**3 amps @ 12v**

Minimum battery requirement 3 amp * 24 hours = **72 amps @ 12v**

Night time amperage usage = 3 amps * 16 hours = 48 amps

Required amperage = 48 amp night time loss + 24 amp equipment usage = 72 amps

72 amps / 8 hour day = 9 amps per hour solar requirements

9 amps * 15 watts = **minimum 135 watt solar panel**

Figure 10: Example Node Power Requirements

This node doesn’t have any line-of-sight, so it will have to sit on top of a building in the area. The following parts list can be used to construct the node:

- Cinder block weighted pole base
- Cinder blocks
- 8 foot pole
- Top-of-pole solar panel mount
- Weatherproof electrical Box for equipment storage
- Steel clamps for mounting equipment
- LibreMesh capable router and High-Gain Antenna
- Unidirectional Transmitter (i.e. Ubiquiti nanostations)
- Raspberry Pi
- DC PoE Injector
- 150 Watt solar panel
- 12v to 24v transformer
- 12v to 5v step downgrade
- 72 amp battery (or batteries in parallel to equal 72 amps)
- Weatherproof Battery Box
- Cat 6 cable
- Solar cables

The community infrastructure node ultimately looks like the diagram below:

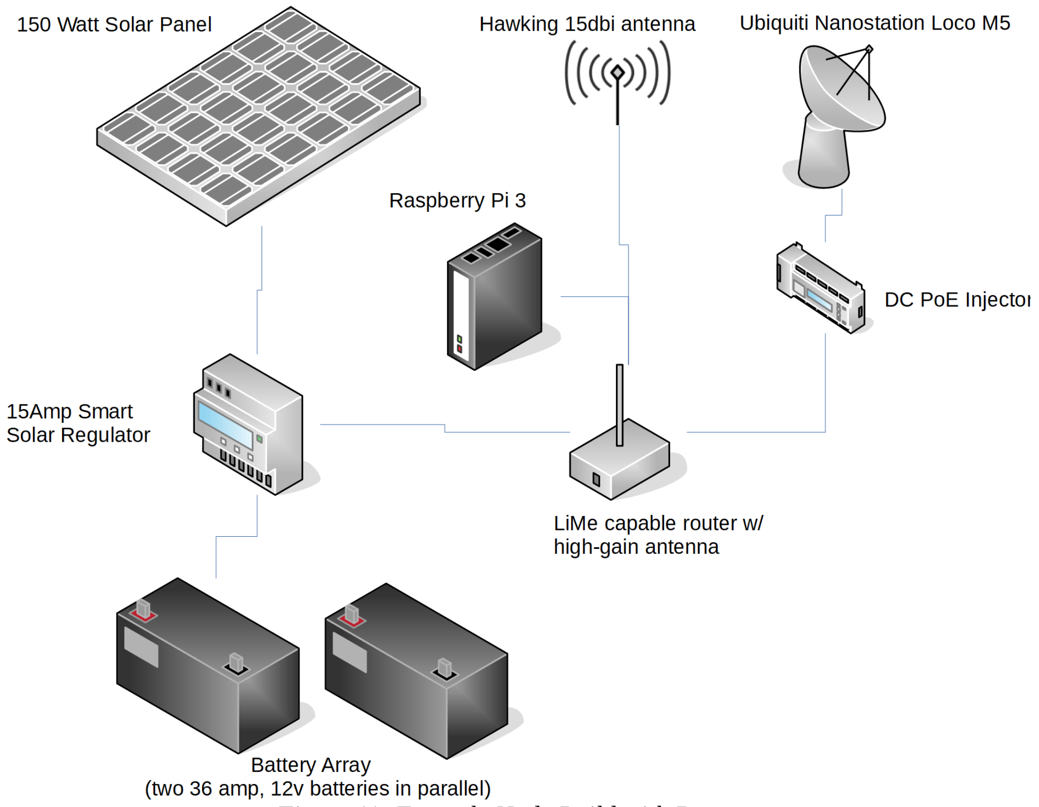
150 Watt Solar Panel
Hawking 15dbi antenna
Ubiquiti Nanostation Loco M5
Raspberry Pi 3
DC PoE Injector
15Amp Smart Solar Regulator
LiMe capable router w/ high-gain antenna
Battery Array
(two 36 amp, 12v batteries in parallel)

Figure 11: Example Node Build with Parts