Basic Mesh Paper Not for Journals: DRAFT

Brian McGarvey, *Fellow, S&T,*

Abstract

This work describes the installation and configuration of a basic mesh network, based on the the B.A.T.M.A.N-adv topology developed and used in FreiFunk and other installations. There is no new work here, just a compilation and filtering of many internet sources. This is an introductory paper that describes the basic functionality of the mesh network options and how to set it up, with examples. The goal is to enable users to understand what many the options mean, and how to tweak some of them. This is not expected or intended to be a formal paper discovering anything new. But more of a simple paper describing the elements of the usable mesh network, and for simple implementation getting it running.

Index Terms

IEEE, OpenWRT, batman-adv, open-mesh, mesh networking, example mesh.

I. Introduction

B.A.T.M.A.N.-ADV is mesh-networking protocol it can run on a wide variety of internet protocol (IP) based commmunication media. We will be concentrating on the 802.11 or WiFi instantiation. B.A.T.M.A.N.-adv went through a change from user space to kernel space recently. With that change it moved from Layer 3 to Layer 2. It means that the routing is much more like a desktop switch in that all traffic is routed by *mac-addresses* not IP. This system has been widely implemented across parts of Europe. For example, Friefunk in Berlin has over 3000 nodes. More information regarding the inner workings of the source code, theory, and protocol information can be found at *http://www.open-mesh.org/projects/open-mesh/wiki*.

We are going to address the following issues. Installation of OpenWRT as a replacement operating system (OS) for a small home router. Configuration of each node in the mesh network, and networking 3 nodes together. The basic concept of the *batman-adv* idea is to have an open network, security is taken care of by the end users. This is because the original installation was to provide free internet service in high cost areas. We will connect the nodes together internally, and then connect clients to the network. An example might be a private LAN party in a small local area without wifi or internet service. Or augmenting an over utilized network for local usage. Or providing a private network for members of a small community.

Before we move forward, you will need to understand some basic concepts. The terms are listed below and links for further reading. **We are assuming basic concepts are understood by the user:**

- Mesh Networking: What it is and how it works in general: No specifics
- OpenWRT: What it is and begin able to install precompiled version on a wireless router
- Concept: Layer 2 (Ethernet MAC routing) vs. Layer 3 (IP Routing)
- Basic Command Line Interface (CLI) for linux: *ie. ls, chmod, rmdir, touch, etc*
- Small linux installations limitations such as (busybox, etc)

For Mesh Networking, you need to understand the idea of a self-forming and self-healing network. Advanced ideas such as loop avoidance will only come in after you connect the mesh to other networks.

OpenWRT: *http://www.openwrt.org*, OpenWRT is basically a very small instantiation of Linux with a few specialized optimizations. You will need to understand the Table of Hardware if you use a device other than the TP-linkv4 MR-3040.

Layer 2 vs Layer 3 Routing. One is IP, one is Mac Address based.

CLI: All command line interface (CLI) commands are fully detailed below, but basic linux shell familiarity will help.

Tiny Linux installations. Understanding you do not have an entire OS because of the tiny storage, ram, and processing power will help.

B. McGarvey was with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332 USA e-mail: bmcgarvey@gmail.com.

Manuscript received Dec 27 2014; revised Feb 2, 2015.

A. OpenWRT

OpenWRT is a very barebones installation of Linux that is designed to run on embedded routers with tiny amounts of processing power, RAM, and Flash. This is the version of Linux that we are using for our experiments. Specifically Barrier Breaker (14.07), Feb 2015.

B. TPlink MR-3040

For this example we are using the TP-Link MR3040. It has good and bad points. It is very cheap, battery powered, and supported by default OpenWRT distribution. It has USB, Ethernet, wifi, and the Atheros v7xxx chipset. The bad part is the small flash (4MB) and RAM (16MB). The RAM is less of a challenge then the Flash. The tiny flash is the main limitation for this box, but being battery powered and portable is fantastic. It has switches, LEDs, and can actually load the and run the drivers for a small R-Pi wireless adapter. Keep in mind this device is very small and is not a general purpose device. Keeping that in mind will help manage your expectation, and make you happy with the performance.

C. Outline of Action

There are a wide variety of possible network combinations that could be covered here, but for this initial paper, we will start with forming a basic mesh called PookieMesh. After that is successful, we will add additional complexity to the network. At each step, we will attempt to show the what you should do, how you do it, and why you do it to help you make changes on your own in the future. Our example hardware was selected because of availability, usefulness, and price: TP-Link MR-3040 battery powered travel router with 2.4GHz, 802.11g, usb, and Ethernet. Listed below are the steps described in this paper.

1) Replace stock OS with OpenWRT
2) Configure the TP-link MR-3040 for upgrade
3) Upgrade basic OpenWRT firmware to include *batman-adv*
4) Configure a single node as a PookieMeshNode
5) Configure the rest of the nodes
6) Create a 3 Node Stand Alone Network – PookieMesh
7) Mesh with a single gateway
8) Mesh with single non-mesh client
9) Mesh with a new mesh node joining
10) Mesh with a mesh node leaving

II. Example: Getting and understanding OpenWRT distributions

I wanted the example easy to follow, and cheap to duplicate. So, I chose the TPlink MR-3040 as the basis for learning and experimenting. My reasoning was, they are battery powered, small, cheap (30USD), 802.11n, USB, Ethernet (1GBit), and most importantly supported by OpenWRT out of the box. I had to verify the TP-Link MR-3040**v4** would work with precompiled OpenWRT and was able to update OpenWRT TOH with a small add.

The MR-3040 comes with a version of linux pre-installed to do things the way the vendor imagined. But our goal is to use the device in a non-traditional method, ie. mesh networking. The stock firmware is not up to the task. So as EEVBlog says.. ”Don’t turn it on. Take it apart!” This task is actually made easy, by the fact that the devices already uses linux to drive the system, and the GPL makes the vendor provide sufficient information to change the firmware. While this might be the scariest part, it is in fact the simplest part of this entire exercise.

We will have to download a few things as our first action as for some of this I would not expect you to have internet access due to the default configuration of OpenWRT. We will be downloading two (2) pre-compiled images of the OpenWRT firmware. The Barrier Breaker distribution is demonstrated here. Supported commands have changed over several of the major releases of OpenWRT, so I cannot guarantee this recipe will work for other releases.

Head on over to ***http://downloads.openwrt.org***. You should see the several different branches, such as Attitude Adujustment and Barrier Breaker. We are concerning ourselves with 14.xx of Barrier Breaker (Feb 2015). We have to browse through the directory structure to find the images we want. The first click is for *Barrier Breaker 14.07, Released: Thu, 02 Oct 2014*. Then you are presented with a large number of seemingly random links. These are all the different basic chipsets (hardware) that are supported directly. Select *ar71xx/*, that is the Atheros Chipset in the MR-3040. Next we select *generic*, as this is the type of flash that is installed for memory. Now you will see a giant list of images. Do a search for *3040*. It will show you four (4) different images.

```
openwrt-ar71xx-generic-tl-mr3040-v1-squashfs-fa..> 02-Oct-2014 07:28          3932160
```

```
openwrt-ar71xx-generic-tl-mr3040-v1-squashfs-sy..> 02-Oct-2014 07:28            3342340
openwrt-ar71xx-generic-tl-mr3040-v2-squashfs-fa..> 02-Oct-2014 07:28            3932160
openwrt-ar71xx-generic-tl-mr3040-v2-squashfs-sy..> 02-Oct-2014 07:28            3342340
```

You will want to download V2 both factory and sysupgrade. Those are highlighted below.

- *openwrt-ar71xx-generic-tl-mr3040-v1-squashfs-factory.bin*
- *openwrt-ar71xx-generic-tl-mr3040-v1-squashfs-sysupgrade.bin*

- ***openwrt-ar71xx-generic-tl-mr3040-v2-squashfs-factory.bin***
- ***openwrt-ar71xx-generic-tl-mr3040-v2-squashfs-sysupgrade.bin***

The URL addresses below were the direct links in Feb 2015, they may still work when you read this.

```
http://downloads.openwrt.org/barrier_breaker/14.07/ar71xx/generic/
openwrt-ar71xx-generic-tl-mr3040-v2-squashfs-factory.bin

http://downloads.openwrt.org/barrier_breaker/14.07/ar71xx/generic/
openwrt-ar71xx-generic-tl-mr3040-v2-squashfs-sysupgrade.bin}
```

The names of the various images seem very arbitrary at first. It is helpful to understand what all the different portions of the filenames describe. The filenames describe the basic distribution, chipset, flash type, etc.

openwrt:	Basic software name
ar71xx:	Wifi Chipset class
generic:	Flash type(generic, nand, nor)
tl-mr3040:	Specific hardware
v2:	Version 2 of the basic config for this hardware
factory:	Used to move from original software to new software will work for basic needs, but you should move to sysupgrade
sysupgrade:	Install this right after factory to make changes

You will need to download both the *sysupgrade.bin and and *factory.bin files for installation for your system, or if you are following the example these two (same as above):

- ***openwrt-ar71xx-generic-tl-mr3040-v2-squashfs-factory.bin***
- ***openwrt-ar71xx-generic-tl-mr3040-v2-squashfs-sysupgrade.bin***

If you are security conscious you should check the MD5 sums. This will give you the install with the least amount of issues. After completing the next task, you have the basic OS installed. Then it is time to begin making changes to the configuration.

III. Installing OpenWRT

Now that you have the base OS images downloaded. It is time to upgrade your router. I recommend using a direct Ethernet link from your laptop to the router seperate from your normal network. The default OpenWRT configuration has the main wireless radio **off** by default. This is to help keep you security conscious, and remind you to setup the security. Set the laptop to use DHCP, and watch until the laptop gets an address. Once the laptop has an address open a web browser and type in the URL/address bar 192.168.0.1 and press enter. A website should appear with the default user interface from the vendor. **At this point, I am assuming that you have the example hardware, or at least hardware that is compatible with OpenWRT.**

Now you are going to update the firmware for the first time with the *openwrt-ar71xx-generic-tl-mr3040-v2-squashfs-factory.bin* version of the OpenWRT operating system (OS). This version of the firmware has been designed to replace the stock firmware, and meet any error checking that the stock firmware checks. If you have errors check the forums: **TODO: provide link**

After you have installed the new software via the web interface, you will have to change the URL/address that you are currently pointed at from 192.168.0.1 to 192.168.1.1 to see the new OpenWRT interface called LUCI.

Now this may seem silly, but we are going to upgrade the system again pretty much immediately. You are going to upgrade *factory.bin to *sysupgrade.bin. This will free up some flash ram that you desperately need. The second image to install is ***openwrt-ar71xx-generic-tl-mr3040-v2-squashfs-sysupgrade.bin***. After that is complete, the LuCI website should reload. Your next tasks are important for security, and to be a tiny bit safer out on the internet.

A. Very Basic Security and Internet Settings

The settings that we change here will propagate as we switch from normal travel router to mesh network node.

1) Add a root password from the LuCI interface
2) Verify root password via *ssh root@192.168.1.1*
3) Set static IP – to one that is not being used on your network. I used (192.168.1.42)
4) Set hostname – Pick one, words all run together *ie. BatteryHorseStaple*
5) Set default gateway: Your internet gateway router (internet router)
6) Set default DNS: Same as your gateway or 8.8.8.8 or 8.8.4.4 (Google DNS)

Now you have the box setup to join your local network as a client. Unplug the router from the laptop, and plug both into your network that connects to the internet. Remember the radio is still off. SSH into the box from the laptop with the static IP that you just assigned to it. Generally, on the first SSH to a new box it, the ssh client asks if you want to remember the target's key. Reply yes, otherwise it will just boot you off.

B. Command Line Interface (CLI)

Now we have to install several specific pre-compiled packages to get the mesh working. We will have to download/install the kernel module and command line interface for batman-adv. You should ssh into your OpenWRT box with:

```
ssh root@192.168.1.42
```

When you ssh into you new box you should see the following

```
BusyBox v1.22.1 (2014-09-20 22:01:35 CEST) built-in shell (ash)
Enter 'help' for a list of built-in commands.
  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------
 BARRIER BREAKER (14.07, r42625)
 -----------------------------------------------------
  * 1/2 oz Galliano         Pour all ingredients into
  * 4 oz cold Coffee        an irish coffee mug filled
  * 1 1/2 oz Dark Rum       with crushed ice. Stir.
  * 2 tsp. Creme de Cacao
 -----------------------------------------------------
 root@OpenWrt:~#
```

This screen means that you have successfully installed OpenWRT on this tiny router. We can proceed from here, but we need to understand a command line utility called UCI.

C. UCI commands: What are they

UCI is a seemingly strange system that attempts to abstract the /etc/config/* files from the user. It combines command line interface with the web LUCI interface. It is to help the user with the formatting and automation of the configurations. You don't have to leave the command line to change the configuration of the box */etc/config/** For example I wanted to change the hostname of a box that I previously setup. I typed this in:

```
root@OpenWrt:~# uci set system.@system[0].hostname="MyExampleName"
root@OpenWrt:~# uci commit
root@OpenWrt:~# reboot
```

After it finishes rebooting, and you can log in you will see:

```
root@MyExampleName:~#
```

This successfully changed the *hostname*. It could have also been done by editing the */etc/config/system* file and changing the ***option hostname*** to MyExampleName. This example is provided to try out on your device and will get you familiar and confident that *uci* commands actually do change things, when your remember to commit. For the example used here, we will stick to the default hostname *OpenWRT* in our example.

IV. WIRELESS SETUP: MR-3040

By default the MR-3040 only has one radio, ***radio0***. By typing *cat /etc/config/wireless* at the command prompt on the router, we can see the default configuration that ships with OpenWRT.

```
root@OpenWrt:~# cat /etc/config/wireless
config wifi-device  radio0
option type     mac80211
option channel  11
option hwmode 11g
option path 'platform/ar933x_wmac'
option htmode HT20
# REMOVE THIS LINE TO ENABLE WIFI:
option disabled 1

config wifi-iface
option device   radio0
option network  lan
option mode     ap
option ssid     OpenWrt
option encryption none
```

This configuration has two main sections **wifi-device** and **wifi-iface** entries. The *wifi-device* configures the physical radio parameters, while the **wifi-iface** configures the logical network interfaces.

For the **wifi-device** radio configuration, we will use 802.11g (54Mbps, 2.4GHz) on channel 1, with the radio enabled. The logical interface, **wifi-iface**, will change the lan being shared over an an access point (AP) to a batman-adv mesh network in ad-hoc mode. These *uci* commands will change the */etc/config/wireless* file. Throughout this example, we will continue to show before and after any changes. The *uci* commands are not magic, they merely change the content in the associated configuration files.

A few notes regarding uci. The uci commands are white space sensitive, meaning the commands will fail with whitepace around the s̄ign. If you do not define a variable, the uci will also fail. I have not tested if one bad command in a batch will cause a failure of the whole batch.
Example:

```
root@OpenWrt:~# uci set wireless.radio0.macaddr =
```

will generate an error.

A. Wireless Configuration Changes

So let us configure the */etc/config/wireless* file, also known as the wireless for the mesh network.

```
root@OpenWrt:~# uci set wireless.radio0=wifi-device
root@OpenWrt:~# uci set wireless.radio0.channel='1'
root@OpenWrt:~# uci set wireless.radio0.disabled='0'
root@OpenWrt:~# uci set wireless.radio0.phy='phy0'

root@OpenWrt:~# uci set wireless.@wifi-iface[-1].device=radio0
root@OpenWrt:~# uci set wireless.@wifi-iface[-1].encryption=none
root@OpenWrt:~# uci set wireless.@wifi-iface[-1].network=mesh0
root@OpenWrt:~# uci set wireless.@wifi-iface[-1].mode=adhoc
root@OpenWrt:~# uci set wireless.@wifi-iface[-1].bssid='CA:CA:CA:CA:CA:00'
root@OpenWrt:~# uci set wireless.@wifi-iface[-1].ssid="PookieMesh"
root@OpenWrt:~# uci set wireless.@wifi-iface[-1].mcast_rate=11000

root@OpenWrt:~# uci commit
```

These commands are not loaded into the */etc/config/wireless* file until *uci commit* is entered. After committing, the */etc/config/wireless* file looks like:

```
config wifi-device 'radio0'
option type 'mac80211'
option hwmode '11g'
option path 'platform/ar933x_wmac'
option htmode 'HT20'
option channel '1'
option disabled '0'
option phy 'phy0'

config wifi-iface
option device 'radio0'
option encryption 'none'
option network 'mesh0'
option mode 'adhoc'
option bssid 'CA:CA:CA:CA:CA:00'
option ssid 'PookieMesh'
option mcast_rate '11000'
```

The individual node now has with wireless interface turned on, the Wifi channel, BSSID, and SSID for the mesh network defined. The hostname is not really important to the network as *batman-adv* uses mac addresses to route traffic.

You should notice some things are different. Under *wifi-iface* the network is now referenced as ***mesh0*** and the mode is ***ad-hoc***. This is the beginnings of having a mesh network. Before editing, the network was listed as *lan* and the mode was listed *ap*. This is the normal configsuration that allows wireless clients connection to an access point (AP), then to share the ethernet connection. Changing the mode to *adhoc* and the network to *mesh0* isolates the radio from the ethernet for now. These changes lead us to changes needed in */etc/config/network* file.

V. Network Setup

As we continue setting up this node, I have changed from the default DHCP on the ethernet port and set a password so I can ssh into the box with *ssh root@192.168.1.42*. You should use whatever IP is appropriate for your private network, and to have access to the internet for the next steps.

Below is the default content of network configuration before we make changes. Keep in mind that the wireless changes that we just make are still active, and the radio is currently on. Please notice that there is no mesh0 network, and therefore it is not routable. If someone were to connect with the mesh network, they should not be able to gain access to your local network without significant effort.

The default */etc/config/network* settings.

```
root@OpenWrt:~# cat /etc/config/network

config interface 'loopback'
option ifname 'lo'
option proto 'static'
option ipaddr '127.0.0.1'
option netmask '255.0.0.0'

config globals 'globals'
option ula_prefix 'fd91:61dd:8444::/48'

config interface 'lan'
option ifname 'eth0'
option force_link '1'
option type 'bridge'
option proto 'static'
option netmask '255.255.255.0'
option ip6assign '60'
```

```
option ipaddr '192.168.1.42'
```

VI. Getting Internet Connectivity Working Again

This will seem like a deviation, but it is required. We have to get connected to the internet to download some packages using *opkg*. It is similar to *apt-get* for ubuntu in it will download, and install new software packages to use.

As we need access to the internet for this next step, verify your connection by pinging your favorite site. I chose google.com, but as you can see below, I had a problem google was unreachable. I changed my IP address, but didn't add a gateway or DNS that is required to route IP traffic.

```
ping google.com
ping: bad address 'google.com'
```

This shows that I screwed that up. I changed the DHCP address to a static address, but did not add the gateway and DNS server. 192.168.1.1 is my gateway, and I typically use Google's DNS servers (8.8.8.8 and 8.8.4.4) to avoid my service providers crappy DNS servers.

```
uci set network.lan.gateway='192.168.1.1'
uci set network.lan.dns='192.168.1.1'
uci commit

restart networking without reboot.
/etc/init.d/network restart
```

Below you can see the changes that the uci made to the */etc/config/network* file. It just added the gateway and DNS to the lan interface.

```
root@OpenWrt:~# cat /etc/config/network

config interface 'loopback'
option ifname 'lo'
option proto 'static'
option ipaddr '127.0.0.1'
option netmask '255.0.0.0'

config globals 'globals'
option ula_prefix 'fd91:61dd:8444::/48'

config interface 'lan'
option ifname 'eth0'
option force_link '1'
option type 'bridge'
option proto 'static'
option netmask '255.255.255.0'
option ip6assign '60'
option ipaddr '192.168.1.42'
option gateway '192.168.1.1'
option dns '192.168.1.1'

root@OpenWrt:~# reboot

root@OpenWrt:~# ping google.com
PING google.com (65.199.32.21): 56 data bytes
64 bytes from 65.199.32.21: seq=0 ttl=250 time=12.850 ms
64 bytes from 65.199.32.21: seq=1 ttl=250 time=9.822 ms
```

Now it works and we can install the parts we need to make this thing work. I tried restarting the network with init.d, but I had issues. So I just rebooted it.

VII. Installing Kernel Module batman-adv, batctl, alfred

You will read throughout the published wiki's, forums, and websites that batman-adv is pre-compiled in the kernel. Well, I found that to be not quite true for this particular distribution. I was able to verify this by typing at the command prompt *lsmod — grep batman*. If the batman-adv module was installed and running this would show it. But, alas it is not installed, and reports nothing. If the module is installed, completing this task will not harm the system.

Now we need to install the missing bits. These first commands, just update the sources that OpenWRT can pull data from. After that we will install three main things: the batman-adv kernel module, the command line interface (batctl), and the network information tool (alfred). If you follow the flow below, it shows you what to type in and some of the output that you would expect to see when you have success.

```
root@PookieNode0:~# opkg update
Downloading ...
...
Updated list ...
```

This downloads and updates the list of available packages for OpenWRT to install. If it says it can't locate something check your internet connection again. The following steps depend on success here.

```
root@PookieNode0:~# opkg install kmod-batman-adv
....
kmod: failed to insert /lib/modules/3.10.49/batman-adv.ko
```

Here we downloaded the kernel module, the automatic script attempted to install it but failed. We will manually install it. It should work from then on even after reboots.

```
root@OpenWrt:~# modprobe batman-adv.ko
root@OpenWrt:~# modprobe batman-adv.ko
kmod: batman-adv is already loaded
```

We typed the command in twice to verify it actually got loaded. You could have accomplished the same thing with *modprobe* then *lsmod — grep batman*. Next we install the command line tool for interacting with the batman-adv kernel module: batctl

```
root@OpenWrt:~# opkg install batctl
Installing batctl ...
....
Configuring batctl.
```

Now we install alfred, which I am still figuring out, but it seems to be a useful module for batman-adv

```
root@OpenWrt:~# opkg install alfred
Downloading ...
Configuring libpthread.
Configuring librt.
Configuring alfred.
```

We have installed the software that we needed to download form the internet repositories. We are now able to break our connection with the internet to finish configuring the system. You will still need to ***ssh*** into the routers, so keep that in mind on how you change your setup. Leaving the nodes connected to your internal network will open a non-password protected wireless link to your network. Please, take care in what you do. I recommend breaking the connection.

VIII. Editing batman-adv configuration

We continue configuring the mesh network with */etc/config/batman-adv* modifications. We will be hand-editing the */etc/config/batman-adv* this tim instead of using the uci commands.

Starting with the ***batman-adv.ko***, we will need to edit */etc/config/batman-adv* file. The kernel module only provides the kernel module, but it still has to be configured to enable the routing or bridging. The first step is to identify in */etc/config/batman-adv* that there is a new mesh network *bat0*. That is done with adding batman-adv ***config 'mesh' 'bat0'*** to the file as seen below. I would normally use the *uci* commands, but the default install for batman-adv, and alfred have many unknown options. As I am just starting with the mesh networking, I will start with the defaults. If you don't want all the default options you can use the three uci commands below, or edit the */etc/config/batman-adv* file to look like following the uci commands.

```
uci set batman-adv.bat0=mesh
uci set batman-adv.bat0.interfaces='mesh0'
uci commit
```

Current configuration

```
root@OpenWrt:/etc/config# cat batman-adv

config 'mesh' 'bat0'
option 'aggregated_ogms'
option 'ap_isolation'
option 'bonding'
option 'fragmentation'
option 'gw_bandwidth'
option 'gw_mode'
option 'gw_sel_class'
option 'log_level'
option 'orig_interval'
option 'vis_mode'
option 'bridge_loop_avoidance'
option 'distributed_arp_table'
option 'multicast_mode'
option 'network_coding'
option 'hop_penalty'
option 'isolation_mark'

# yet another batX instance
# config 'mesh' 'bat5'
# option 'interfaces' 'second_mesh'
```

IX. Bridge the Mesh and The Ethernet networks

This completes the batman-adv setup. We move onto the network configuration where we bridge the batman-adv mesh (*bat0*) to the local ethernet (*eth0*). The assumption here is that you will want the connectivity to the outside world, or to some other wired network. We will be bridging ***bat0 and eth0*** this is as simple as:

```
root@OpenWrt:/etc/config# uci set network.lan.ifname='eth0 bat0'
root@OpenWrt:/etc/config# uci commit
```

This modifies the */etc/config/network* file to look like

```
root@OpenWrt:/etc/config# cat network

config interface 'loopback'
option ifname 'lo'
option proto 'static'
option ipaddr '127.0.0.1'
option netmask '255.0.0.0'

config globals 'globals'
option ula_prefix 'fd91:61dd:8444::/48'

config interface 'lan'
option force_link '1'
option type 'bridge'
option proto 'static'
option netmask '255.255.255.0'
option ip6assign '60'
option ipaddr '192.168.1.42'
option gateway '192.168.1.1'
```

```
option dns '192.168.1.1'
option ifname 'eth0 bat0'
```

Those uci commands changed *option ifname* to ***eth0 bat0*** from *option ifname eth0*. These changes presume having a gateway, DNS server, and DHCP server enabling clients to join the network and connect to the outside internet.

X. Adding batman-adv to networking

This section adds the bat0 interface to the */etc/config/network* as an interface and changes the framing size to accomodate the extra batman-adv data (mtu).

```
root@OpenWrt:/etc/config# uci set network.bat0=interface
root@OpenWrt:/etc/config# uci set network.bat0.ifname=bat0
root@OpenWrt:/etc/config# uci set network.bat0.proto=none
root@OpenWrt:/etc/config# uci set network.bat0.mtu=1528
root@OpenWrt:/etc/config# uci commit
root@OpenWrt:/etc/config# cat network
```

Below you can see the new interface called bat0 and with the mtu size for the addtional overhead.

```
config interface 'loopback'
option ifname 'lo'
option proto 'static'
option ipaddr '127.0.0.1'
option netmask '255.0.0.0'

config globals 'globals'
option ula_prefix 'fd91:61dd:8444::/48'

config interface 'lan'
option force_link '1'
option type 'bridge'
option proto 'static'
option netmask '255.255.255.0'
option ip6assign '60'
option ipaddr '192.168.1.42'
option gateway '192.168.1.1'
option dns '192.168.1.1'
option ifname 'eth0 bat0'

config interface 'bat0'
option ifname 'bat0'
option proto 'none'
option mtu '1528'
```

XI. Adding the mesh interface to networking

In order for the system to operate correctly there are three (3) logical interfaces. The *lan* connects to the outside world. The *mesh0* network connects all of the mesh nodes to each other. The *bat0* bridges the two together (this is not quite correct). You can see below the uci commands required to add the mesh interface to */etc/config/network* file.

```
root@OpenWrt:/etc/config# uci set network.mesh0=interface
root@OpenWrt:/etc/config# uci set network.mesh0.proto=batadv
root@OpenWrt:/etc/config# uci set network.mesh0.mtu=1528
root@OpenWrt:/etc/config# uci set network.mesh0.mesh=bat0
root@OpenWrt:/etc/config# uci commit
```

Below you can see the output of the commands.

```
root@OpenWrt:/etc/config# cat network

config interface 'loopback'
```

```
option ifname 'lo'
option proto 'static'
option ipaddr '127.0.0.1'
option netmask '255.0.0.0'

config globals 'globals'
option ula_prefix 'fd91:61dd:8444::/48'

config interface 'lan'
option force_link '1'
option type 'bridge'
option proto 'static'
option netmask '255.255.255.0'
option ip6assign '60'
option ipaddr '192.168.1.42'
option gateway '192.168.1.1'
option dns '192.168.1.1'
option ifname 'eth0 bat0'

config interface 'bat0'
option ifname 'bat0'
option proto 'none'
option mtu '1528'

config interface 'mesh0'
option proto 'batadv'
option mtu '1528'
option mesh 'bat0'

root@OpenWrt:~ # reboot
```

The software network interfaces are now implemented. The connectivity is the ethernet lan network (eth0) connects to the batman-adv network (bat0), the batman-adv (bat0) connects the mesh network (mesh0). Batman-adv provides the bridge between the mesh and the lan. Now we need to configure the actual wireless configuration. After the box finishes rebooting, ssh back to the box for further instructions.

XII. Connecting it all Together

Now that we have edited the three main files: */etc/config/ network, batman-adv, and wireless* . We need to see if our changes have worked. This is doen easily with a single command. *batctl* is how we interact with the batman-adv kernel module. This next command lets us know that the some of the things we have done work up to now.

```
root@OpenWrt: batctl o
[B.A.T.M.A.N. adv 2014.2.0, MainIF/MAC: wlan0/e8:de:27:e0:f5:a0 (bat0 BATMAN_IV)]
  Originator      last-seen (#/255)           Nexthop [outgoingIF]:   Potential nexthops ...
No batman nodes in range ...
```

This basically says that batman-adv is up and running, but there are no other nodes avail. You will need to repeat this procedure for all the nodes, but some things get changed and some things don't.

- Change:
  - IP of the node
  - hostname of the node
- Stays the same
  - bssid of mesh
  - ssid of mesh
  - channel that mesh operates on

Assuming that you have been successful installing the system on several nodes. you can type ***batctl o*** at any of the command prompts from the nodes and get a response that looks like:

```
[B.A.T.M.A.N. adv 2014.2.0, MainIF/MAC: wlan0/e8:de:27:e0:f5:a0 (bat0 BATMAN_IV)]
```

```
  Originator      last-seen (#/255)           Nexthop [outgoingIF]:   Potential nexthops ...
e8:de:27:e0:e3:fc    0.640s   (201) e8:de:27:e0:e3:fc [     wlan0]: e8:de:27:e0:ec:42 (143) e8:de:27:e0:f6:90 (161) e8:de:27:e0:ec:36 (152) e8:de:27:e0:e3:fc (201)
e8:de:27:e0:ec:42    0.620s   (201) e8:de:27:e0:ec:42 [     wlan0]: e8:de:27:e0:e3:fc (143) e8:de:27:e0:ec:36 (149) e8:de:27:e0:f6:90 (152) e8:de:27:e0:ec:42 (201)
e8:de:27:e0:ec:36    0.430s   (206) e8:de:27:e0:ec:36 [     wlan0]: e8:de:27:e0:ec:42 (146) e8:de:27:e0:e3:fc (149) e8:de:27:e0:f6:90 (163) e8:de:27:e0:ec:36 (206)
e8:de:27:e0:f6:90    0.720s   (223) e8:de:27:e0:f6:90 [     wlan0]: e8:de:27:e0:ec:42 (143) e8:de:27:e0:e3:fc (146) e8:de:27:e0:ec:36 (138) e8:de:27:e0:f6:90 (223)
```

Results like this tell you that all the nodes are working correctly and the do form a mesh. In the next section, we will introduce connecting to an outside network.

XIII. Connecting to the outside world

Now that the network is up an running. And you still have your gateway, DNS, and DHCP server still serving out data. Think of the mesh network that you just built as if were a simple switch on your desk, that anybody can jack into over the air. That computer will talk to your gateway and get an IP address in the normal method. Alternatively, they could self-assign an IPv6 or IPv4 address and be on the network. They would have to manually add the gateway and DNS servers to their configuration, but that is pretty easy to do for people trying to get on your network.

A. WARNING: Security

Mesh networking is not secure by design. Think of it the same as plugging into any other public network just like at your home or coffee house. Users can institute VPNs over the network for privacy, and firewalls for security. Assigning a WPA2 password will work for awhile until the password leaks out. There is an old saying it is not a secret if two people know it. This type of network is designed to be public.

Connecting to the network from any number of hosts is actually quite trivial. I have tested OSX (macbook pro) and Ubuntu 14.07. Connect to the mesh network in the same way you would connect to any other network, except that is an open public network. Assuming that you have a gateway, DNS, and DHCP server running the mesh network should be transparent to you.

Please be aware that there is **Zero** security. Your network is now exposed to the world. Anyone can see the SSID, and join the network if their computer supports ad-hoc networking.

B. Example: 3 Node PookieMesh

So this becomes easy to follow, and cheap to duplicate I chose the TPlink MR-3040 as the basis for learning and experimenting. I chose these as they are battery powered, small, cheap (30USD), 802.11n, USB, Ethernet (1GBit), and most importantly supported by OpenWRT out of the box. I had to verify v4 of the hardware worked. OpenWRT wiki updated.

The OpenWRT precompile firmware is available for both the factory and sys upgrade. The factory.bin is designed to be uploaded through the default web interface that ships with the unit. After it is installed, LuCI is now the basic interface. You will find that the wireless is off by default, and it is on 192.168.1.1. Connect with it via Ethernet cable, it should be running DHCP and IPv4/IPv6. Change the static IPv4 to something usable like 192.168.1.42, 52, 62. Change the password for all of them. Make it easy the first time as you will be messing with it many times. Change the name of each of them. I changed mine to PookieNode0, PookieNode1, PookieNode2. Turn the radios on, Turn the radios ON! All of these options can be changed with the default install of LuCI. LuCI is just the name of the web management interface.

C. OpenWRT setup

system: administration: password system : system : change hostname PookieNode0 network: wifi : enable wifi button network:wifi:edit wifi name PookieTest

XIV. Conclusion

The conclusion goes here.

Appendix A
Appendix

A. Definitions

1) Wireless Lan Modes: Most wireless routers can operate as an access point (AP) for clients. Some add other wireless modes that can be used to extend the range, introduce multiple router/access points to the network, or bridge network segments together. Below is a summary of the different modes and their meaning:

AP mode: this is the default, most common mode for all wireless routers, also called Infrastructure mode. Your router acts as an central connection point, which wireless clients can connect to.

Client mode: The radio interface is used to connect the internet-facing side of the router (i.e., the WAN) as a client to a remote accesspoint. NAT or routing are performed between WAN and LAN, like in ”normal” gateway or router mode. Use this mode, e.g., if your internet connection is provided by a remote accesspoint, and you want to connect a subnet of your own to it.

Client Bridged mode: The radio interface is used to connect the LAN side of the router to a remote accesspoint. The LAN and the remote AP will be in the same subnet (This is called a ”bridge” between two network segments). The WAN side of the router is unused and can be disabled. Use this mode, e.g., to make the router act as a ”WLAN adapter” for a device connected to one of its LAN ethernet ports.

Repeater: In general, a repeater simply regenerates a network signal in order to extend the range of the existing network infrastructure. A WLAN repeater does not physically connect by wire to any part of the network. Instead, it receives radio signals (802.11 frames) from an access point, end user device, or another repeater and retransmits the frames. This makes it possible for a repeater located in between an access point and distant user to act as a relay for frames traveling back and forth between the user and the access point.

Repeater bridge: A wireless bridge connects two LAN segments with a wireless link. The two segments are in the same subnet and look like two ethernet switches connected by a cable to all computers on the subnet. Since the computers are on the same subnet, broadcasts reach all machines. DHCP clients in one segment can get their addresses from a DHCP server in the other segment.

Ad-Hoc mode: This is for peer to peer wireless connections. Clients running in Ad-Hoc mode can connect to each other as required without involving central access points.

B. Basic Script to Program Nodes

Basic Script for Programming nodes For those of you impatient you can copy this script onto the nodes and run it on each one. You will need to change the **HOSTNAME**, and the **IP** for each one at a minimum. This particular script has **ZERO** security. It sets up a wide open mesh.

```
#!/bin/sh

### Main radio0 will broadcast one AP with no encryption, another AP with WPA2,
### and both interfaces will be bridged together with eth0 and bat0
### Another VAP in adhoc mode is added to main radio0,
### as well as adhoc networks in radio1 and radio2 if they are present.
### All three adhoc networks are added to bat0 and thus managed by batman-adv

### Node-specific settings
export HOSTNAME="PookieMeshNode1"  # this should be changed for each node
export IP="192.168.1.135" # this should be changed for each node
#export WPA_ESSID="$HOSTNAME.wpa"  # this should be changed for each node
#export WPA_KEY="password"  # this should be the same for all
#export PKA_MAC_HASH="something here"   # FIXME this is were we would start the hashing
#set wireless.radio0.macaddr= $PKA_MAC_HASH # down in the echo statement

### These parameters should be consistent across all nodes
export NETMASK="255.255.255.0"  # lets keep the address space reasonable
export DNS="8.8.4.4"   # google.com -- assume we can hit the internet
export GATEWAY="192.168.1.1"  # Host Network gateway
#export PUBLIC_ESSID="PookieAPTest"   # Name of the Mesh -- experiment with changing this
export MESH0_MODE="adhoc" # FIXME added this as it looks like is missing
export MESH0_BSSID="CA:CA:CA:CA:CA:00" # FIXME should this be the same or different
export MESH0_ESSID="PookieMesh"         # FIXME should this be the same.
export MESH0_CHANNEL="1"  # we have to choose something

# deleted this link
#set wireless.radio0.macaddr=
### Ensure of populating /etc/config/wireless with
```

```
### autodetected wifi-device entries (radioX)
### to get all list_capab and hwmode correct. Otherwise
### OpenWRT might fail to configure the radio properly.
wifi detect >>/etc/config/wireless

### Clear preexisting wifi-iface sections to avoid conflicts or dups
# this looks like it works
# FIXME check to make sure it deleted non-applicable ones
( for i in `seq 0 9` ; do echo "delete wireless.@wifi-iface[]" ; done ) | uci batch -q

### Create /etc/config/batman-adv if it's not there yet.
uci import -m batman-adv </dev/null #FIXME I believe

echo "
set system.@system[0].hostname=$HOSTNAME

set batman-adv.bat0=mesh
set batman-adv.bat0.interfaces='mesh0'

set network.lan.ipaddr=$IP
set network.lan.netmask=$NETMASK
set network.lan.dns='$DNS'
set network.lan.gateway=$GATEWAY
set network.lan.ifname='eth0 bat0'

set network.bat0=interface
set network.bat0.ifname=bat0
set network.bat0.proto=none
set network.bat0.mtu=1528

set network.mesh0=interface
set network.mesh0.proto=batadv
set network.mesh0.mtu=1528
set network.mesh0.mesh=bat0


set wireless.radio0=wifi-device
set wireless.radio0.channel=$MESH0_CHANNEL
set wireless.radio0.disabled=0
set wireless.radio0.phy=phy0


add wireless wifi-iface
set wireless.@wifi-iface[-1].device=radio0
set wireless.@wifi-iface[-1].encryption=none
set wireless.@wifi-iface[-1].network=mesh0
set wireless.@wifi-iface[-1].mode=adhoc
set wireless.@wifi-iface[-1].bssid=$MESH0_BSSID
set wireless.@wifi-iface[-1].ssid='$MESH0_ESSID'
set wireless.@wifi-iface[-1].mcast_rate=11000

commit" \
| uci batch


#add wireless wifi-iface
#set wireless.@wifi-iface[-1].device=radio0
#set wireless.@wifi-iface[-1].encryption=none
```

```
#set wireless.@wifi-iface[-1].network=lan
#set wireless.@wifi-iface[-1].mode=ap
#set wireless.@wifi-iface[-1].ssid='$PUBLIC_ESSID'
```

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.