Remote Management Guide for Smart Radio

Advanced Mesh Router for Private Wireless Networks

Introduction

The Smart Radio runs the Mesh Rider OS. It is a customized version of Openwrt with enhancements useful for applications requiring low-latency command-and-control transmission and high-throughput video - e.g. UAV and robotics.

The purpose of this guide is to aide a user in remotely configuring Smart Radio settings. There are three primary ways to configure the Smart Radio. All of these interfaces can be accessed either locally (over Ethernet/USB) or remotely (over the wireless link).

1. The Web GUI
2. SSH
3. The JSON-RPC API
4. MQTT

Each of these interfaces serves a different purpose. The Web GUI is designed for initial configuration. For example, when you first start using the device, during bench testing.

SSH access is enabled for advanced system configuration and status monitoring. It provides root access to the underlying Linux system and is a very powerful way to access the system. Typically, equipment manufacturers should not allow SSH access to the end users of the radios. SSH can be very fast when using multiplexing.

The JSON-RPC API is designed for integration into customer software. As with SSH access, it potentially provides complete access to the underlying Linux system, however access permissions can also be tailored to the equipment manufacturer's requirements so that end-users cannot access the nuts and bolts of the radios system.

MQTT is an alternative to the JSON-RPC API which is ubiquitous in IoT applications. Only user-defined messages can be sent using MQTT. MQTT can be very fast if encryption is turned off.

A summary of the differences between the command-line APIs is shown in Table 1.

Table 1 – Comparison of Smart Radio APIs

	SSH	JSON-RPC	MQTT
Network Model	P2P, Client-server	P2P, Client-server	Centrally Managed
Primary Usage	Debugging	Software Integration	IoT, simple messaging
Access	Full	User-Defined	User-Defined
Command Set	All	ubus only	User-defined messaging
Security	Required	Required	Optional
Latency	Fast with multiplexing (10s of milliseconds)	Slow due to TLS (~2s)	- Slow with TLS (~2s) - Fast without (10s of milliseconds)

End users will typically never use any of these APIs directly. In fact, they should not even have the password to access the radios. Instead, they use application software such as ground-control-station (GCS) software which uses the JSON-RPC API to talk to the radio and relay information to the user.

The remainder of the main content of this document discusses how to run commands in the CLI. Detailed information on how to use each particular interface is discussed in the following Appendices

1. Appendix A – The Web GUI
2. Appendix B – SSH
3. Appendix C – The JSON RPC API
4. Appendix D – MQTT
5. Appendix E – Common CLI Commands
6. Appendix F – Creating a Bootup Script

The Smart Radio includes a Central Configuration utility and a Link Status Log utility which are discussed below. Read the “Running Commands in the CLI” section for details.

Central Configuration

The Central Configuration utility is designed to quickly modify the operating channel, TX power, and distance setting, and to poll status information from the entire network of radios. After

enabling Central Config in the GUI, it is possible to perform Central Config tasks over the SSH and JSON-RPC APIs. See the Central-Config section under "Using the JSON-RPC API", and MQTT section in Appendix A for details. As each radio uses the Central Config utility to send its own link information to the primary node, the central configuration utility is a good way to get limited network-wide status information.

Link Status Log

The Link Status Log utility is designed to log the radio's link status over time. It keeps much more detailed information that the Central Configuration utility, but each node operates separately and does not share information with other nodes. Aside from downloading the logs, you can also get the latest status from any particular node.

Running Commands in the CLI

If you login to the Smart Radio's Linux Ash shell (similar to Bash) using SSH, you can run Linux commands. Some commonly used commands are summarized in Appendix E. It will also help if you are familiar with UBUS, Doodle Labs' Central Configuration utility, or Doodle Labs' Link Log utility.

UBUS

Calls to the JSON-RPC API go through the Openwrt ubus system [2]. Before going into the JSON-RPC API, you should become familiar with ubus. In order to run ubus directly, first SSH into the radio. You can view a list of available ubus commands using (result abridged)

```
root@smartradio:~# ubus list
central-config
dhcp
dnsmasq
file
iwinfo
…
```

Note that the central-config call is only available after enabling the Central Configuration utility. You can get information about how to use specific ubus calls by running

```
root@smartradio:~# ubus -v list <CALL>
```

For example,

```
root@smartradio:~# ubus -v list iwinfo
```

```
'iwinfo' @68374f72
        "devices":{}
        "info":{"device":"String"}
        "scan":{"device":"String"}
        "assoclist":{"device":"String","mac":"String"}
        "freqlist":{"device":"String"}
        "txpowerlist":{"device":"String"}
        "countrylist":{"device":"String"}
        "survey":{"device":"String"}
        "phyname":{"section":"String"}
```

An example of how to use the `iwinfo` call is shown below. We replaced `"String"` with `"wlan0"` (result abridged).

```
root@smartradio~# ubus call iwinfo assoclist '{"device":"wlan0"}'
{
        "results": [
                {
                        "mac": "00:30:1A:4E:BB:09",
                        "signal": -47,
…
```

We can filter these results using the `jsonfilter` utility. Note in the JSON file above that the results property is an array of values, one for each connected station.

```
root@smartradio:~# ubus call iwinfo assoclist '{"device":"wlan0"}' |
jsonfilter -e '@.results[1].mac' -e '@.results[1].signal'
00:30:1A:4E:BB:01
-62
```

Or if you know the MAC address of the device you want to filter, you can use

```
root@smartradio:~# ubus call iwinfo assoclist '{"device":"wlan0"}' |
jsonfilter -e '@.results[@.mac="00:30:1A:4E:BB:01"].signal'
-62
```

In general, however, we recommend parsing data on your local machine where it should be easier.

Central Config

If you have gained some familiarity with ubus, you can run Central Config commands over ubus. You can use Central Config to either send configuration changes to the entire network or get status information from each node in the network. You will need to enable Central Config in the GUI first. Navigate to `Services → Central Config` in the GUI to enable the service. Fig. 1

shows the Central Config configuration page. Note that one node should be elected as the primary node, and all other nodes need to put the primary node's IP address in the Address bar. The Central Config utility uses TLS PSK for security, and it can be configured in the second tab.

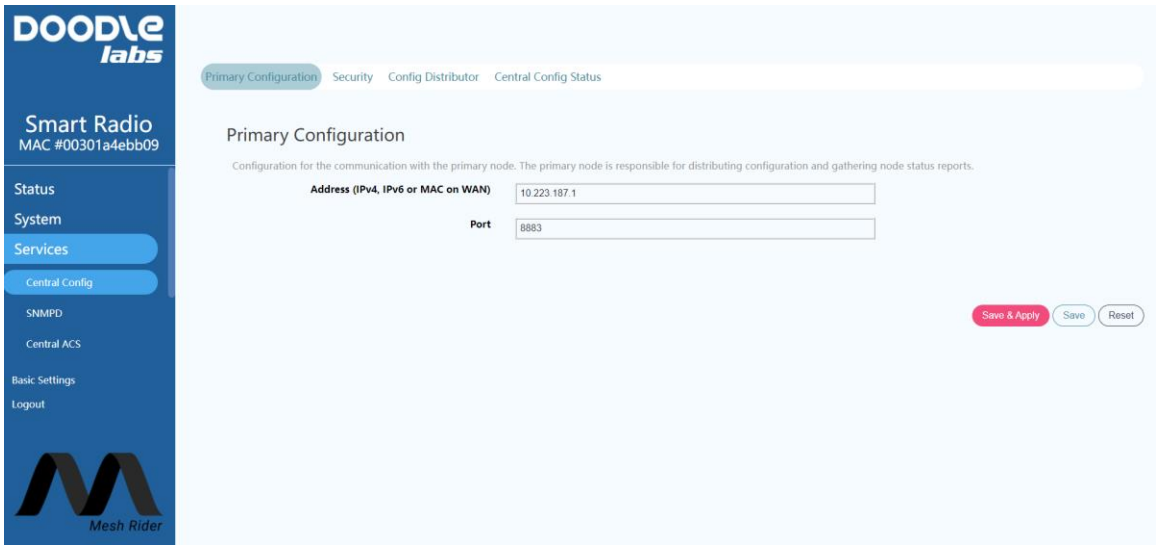

Fig. 1 Central Config configuration page

Currently only three parameters are implemented over Central Config: the operating channel, the distance setting (in meters), and the TX power level (in dBm). Additional options may be added in future. These settings are controlled by the ubus Central Config properties "`channel`", "`distance`", and "`txpower`". For example, to change the operating channel, run

```
root@smartradio:~ # ubus call central-config config
{"dest":"all","delay":0,"config":{"channel":"51"}}'
```

This tells all devices in the network to switch to channel 51. If you run "`iw wlan0 info`" after running the above command, you should see that the radios have moved to the new channel (make sure it is a valid channel first).

- The "`delay`" property can be used to delay the execution of the call (in seconds).
- The property "`dest`" can be either "`all`", "`primary`", or a specific MAC address.
- The property "`config`" is actually a generic property in JSON format. When a node receives a new message, it executes all scripts in the folder "`/usr/lib/doodlelabs/central-config`". It is up to those scripts to parse the json data and perform actions based on the received data.

To get a status update over Central Config, run the command

```
root@smartradio:~ # ubus call central-config config
{"dest":"all","delay":0,"apply":"true","config":{"request_status":"1"}}
'
```

You will not see an output, but each radio defined by "`dest`" will send a status update which will be appended to "`/tmp/status.json`". This file grows each time a new status update is received. We can once again use the `jsonfilter` utility to parse the `/tmp/status.json` file. For example, to get a list of MAC addresses, run

```
root@smartradio:~# jsonfilter -i /tmp/status.json -a -e '@[*].mac' |
sort -u
00:30:1A:4E:AA:01
00:30:1A:4E:AA:02
00:30:1A:4E:AA:09
```

- `-i` : file input
- `-a` : because the file is a stack of several JSON strings, this switch treats the file as an array
- `-e` : filter pattern
- `sort -u` : remove duplicates

To get the latest status update from a particular MAC address, run

```
root@smartradio:~# jsonfilter -i /tmp/status.json -a -e
'@[@.mac="00:30:1A:4E:AA:01"]' | tail -n1
{
  "mac": "00:30:1A:4E:AA:01",
  "hostname": "smartradio-301a4ebb01",
  "model": "RM-2250-2J-X",
  "Interfaces": [
    {
      "wlan0": {
        "mac": "00:30:1A:4E:BB:01",
        "associations": [
          {
            "mac": "00:30:1A:4E:BB:09",
            "signal": -52,
            "inactive": 0,
            "tx_mcs": 65000,
            "rx_mcs": 58500,
            "tx_packets": 368820,
            "rx_packets": 1069144
          },
          {
            "mac": "00:30:1A:4E:BB:02",
```

```
            "signal": -59,
            "inactive": 0,
            "tx_mcs": 65000,
            "rx_mcs": 58500,
            "tx_packets": 368938,
            "rx_packets": 1063279
          }
        ],
        "Batman_originator": [
          {
            "best": "true",
            "orig_address": "00:30:1a:4e:bb:02",
            "last_seen_msecs": 90,
            "tq": 239
          },
          {
            "best": "true",
            "orig_address": "00:30:1a:4e:bb:09",
            "last_seen_msecs": 60,
            "tq": 246
          }
        ]
      }
    }
  ],
  "phy0": {
    "aqm_backlog": 0
  }
}
```


- `@[]` : print only the array element defined in the square braces
- `@.mac="00:30:1A:4E:AA:01"` : filter the array element with this matching MAC address

When first using this API, we recommend copying the file to your local machine, and parsing the data using "`jq`" [3], which will make the output human readable. If we want to get an array of MAC addresses, and corresponding RSSI, we need to run the command

```
root@smartradio:~# jsonfilter -i /tmp/status.json -a -e
'@[@.mac="00:30:1A:4E:AA:01"]' | tail -n1 | jsonfilter -e
'@.Interfaces[0].wlan0.associations.results[*].signal' -e
'@.Interfaces[0].wlan0.associations.results[*].mac'
-63
-54
00:30:1A:4E:BB:02
00:30:1A:4E:BB:09
```

We could also run this command twice, once to get the MAC addresses, and the second time to get the RSSI. However, in most cases, it makes more sense to parse the json file on your local machine rather than in the Smart Radio.

Link Status Log Utility

The Link Log utility was introduced in the October 2022 firmware release. Each device independently maintains a log of the link status information. The Link Log utility can be configured at `Services → Link Status Log`. Fig. 2 shows the configuration page. The logs are accessible in the shell in the folder `/tmp/longtermlog`. Alternatively, you can download the logs from the web GUI. The contents of the Link Log utility are shown here,

```
root@smartradio:/tmp/longtermlog# ls
22-05-05_13-26-21.log  22-05-05_13-51-09.log  22-05-05_14-14-34.log
ipv6list               stationlist            status.json
```

The log files are limited to 500 lines, and the file name is the date when the log started. Aside from long term logs, the Link Log utility keeps the latest status line in the file `/tmp/longtermlog/status.json`. The output of each line is,


```
{
  "date": "22-05-05_16:29:49",
  "ipv6list": [
    {
      "ip6address": "fe80::230:1aff:fe4f:960f",
      "rtt": "1.631"
    }
  ],
  "stationlist": [
    {
      "station": "00:30:1A:4F:96:0E",
      "iwseen": "50",
      "RSSI": "-54,-58,-57",
      "RATE": "MCS15,100.0",
      "packet_drop": "backlog,0,drops,0,collisions,0,packets,134390",
      "peer_rssi": "-55",
      "peer_power": "17",
      "reply_rssi": "-55",
      "reply_power": "21",
      "fixed_txpower": "255",
      "next_hop": "direct",
      "batseen": "0.090"
    }
  ],
```

```
  "wirelessStats": {
    "noise": -95.414276,
    "act_s": 34.1,
    "bus_s": 0.23,
    "RX_kb": 1246,
    "TX_kb": 770,
    "usrrst": 1925,
    "Fatal": 0,
    "TXPath": 0,
    "bbhang": 0,
    "deafhang": 0,
    "backlog": 0
  }
}
```


The output is in JSON format with the following sections. Some options may not be enabled by default in the GUI.

- Time stamp. This is the time stamp for the information set.
- Ipv6 station list. This section shows the Ipv6 address of each connected station, and the round-trip time to that station.
- MAC list. This section shows layer 2 connectivity information to all nodes in the network.
    - iwseen: time in milliseconds since a packet was received by the wireless interface.
    - RSSI: RSSI of the packets received from that station in the format “total, antenna0, antenna1”.
    - RATE: MCS rate and packet success rate for packets sent to that station.
    - packet_drop: detailed information about packets sent to that station
    - peer_rssi: TPC-related utility – rssi from a particular station
    - peer_power: TPC-related utility – power to a particular station
    - reply_rssi: TPC-related utility – rssi recorded by a particular station from the current node
    - reply_power: TPC-related utility – power set by a particular station to the current node
    - fixed_txpower: TPC-related utility – power factor
    - next_hop: In mesh mode, whether the station is directly connected.
    - batseen: time in seconds since the last packet was received by the mesh interface. Similar to “iwseen”
- Wireless Statistics
    - noise: level of the background noise in dBm
    - act_s: active time in seconds since the last time-stamp

- bus_s: the amount of time the wireless medium was in use by any station in seconds. The medium usage duty cycle is bus_s/act_s.
- RX_kb: amount of data received in kilobits since the last time-stamp.
- TX_kb: amount of data transmitted by this node in kilbits since the last time-stamp.
- The remaining fields are driver related and should be diagnosed by Doodle Labs technicians if required.

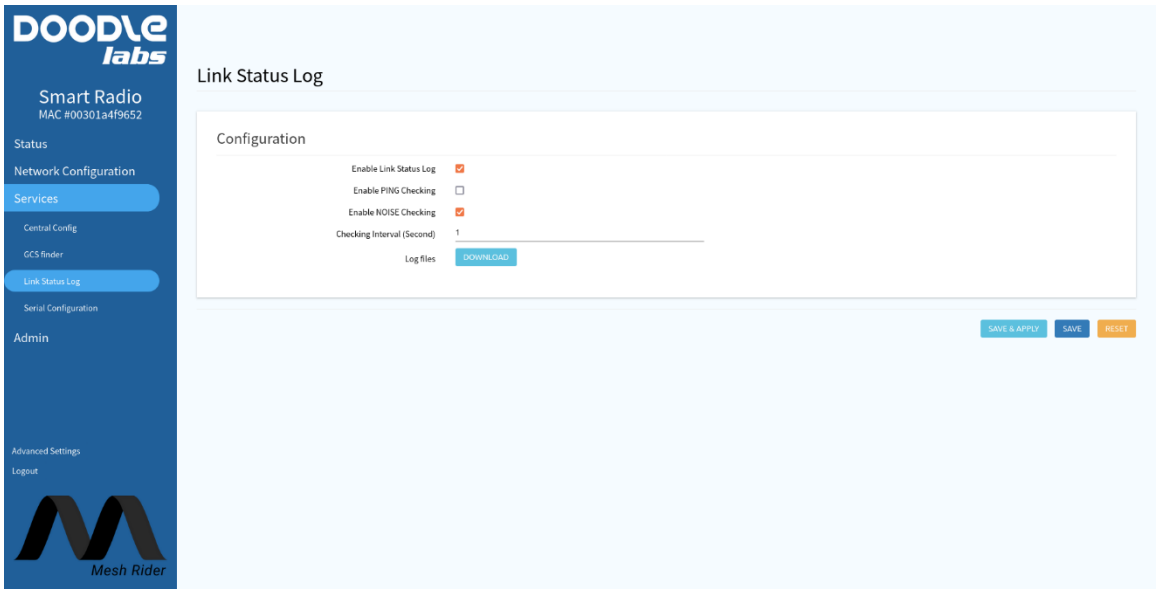

Fig. 2 Link Status Log

Appendix A – The Web GUI

The Web GUI can be accessed in any web browser at `https://<IP ADDRESS>` (port 443). Note that the web browser uses a self-signed certificate. This means that connection to the web browser is encrypted, but not authenticated. The first time you access the Smart Radio from a new browser, you will get a SSL certificate warning. It is okay to ignore the warning and proceed.

Please see the online Configuration Guide for details on using the Web GUI [1].

Appendix B –SSH

SSH or Secure Shell is a way to securely login to the Smart Radio. The easiest way to do so is to open up a command prompt (Windows) or terminal (Linux), and type

```
ssh root@<IP ADDRESS>
```

Where `<IP ADDRESS>` is the IP address of the Smart Radio. There are numerous configuration options that your SSH client supports, such as public key authentication, and quiet output and you are encouraged to research them.

Note that your SSH client keeps a list of known hosts, and after a firmware upgrade, you may need to remove the Smart Radio from the known hosts list. You can do so by running

```
ssh-keygen -R <IP ADDRESS>
```

Sending Remote Commands

You can remotely execute a command via SSH to obtain network information from the node. For example,

```
ssh root@<IP ADDRESS> "iw wlan0 info"
Interface wlan0
      ifindex 13
      wdev 0x7
      addr 00:30:1a:4e:86:46
      type mesh point
      wiphy 0
      channel 12 (915 MHz), width: 20 MHz, center1: 915 MHz
      txpower 32.00 dBm
```

Speeding up the Connection

If your SSH client supports Multiplexing (OpenSSH for example), then it is a good way to improve the connection speed. Multiplexing allows you to send multiple commands over a single SSH connection. Information about the setup can be found here:

https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Multiplexing

As an example, modify your SSH config file (usually `~/.ssh/config`) with the following settings

```
Host *
      IdentitiesOnly yes
```

```
ControlPersist yes
COntrolMaster auto
ControlPath ~/.ssh/%r@%h:%p
```

Create the file if it doesn’t exist.

Appendix C – The JSON-RPC API

The JSON-RPC API is normally preferred when integrating radio access into custom software. In order to enable the JSON-RPC API, navigate to

https://<IP ADDRESS>/cgi-bin/luci/admin/services/rpcd

in your web browser. Fig. 2 shows the JSON-RPC API web configuration page.

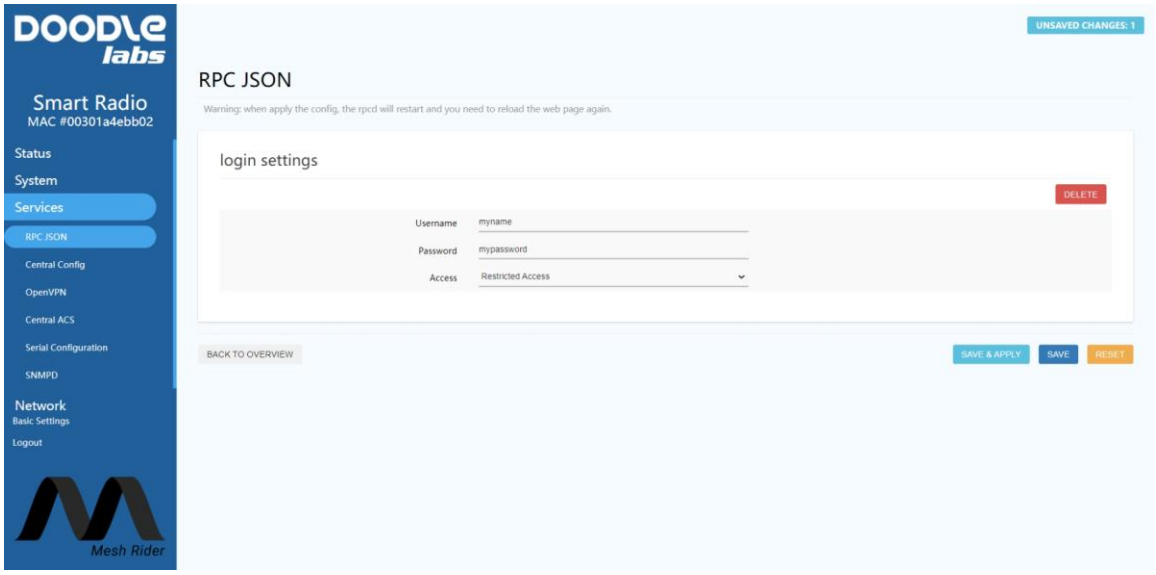

Fig. 2 – JSON RPC API Configuration Page

Click Add to configure the API. Choosing Restricted Access opens up the API for a limited set of commands which we will detail later. You can also choose Full Access which allows unrestricted access to the Linux filesystem, and Custom Access, if you know how to customize the JSON RPC API. We recommend choosing Restricted Access if you are not sure.

Using the JSON-RPC API

To use API, you need to get a session ID and use it for subsequent requests. For example, you can try the following call,

```
USER=myusername
PASS=mypassword
curl -k https://<IP-ADDRESS>/ubus -d '
{
  "jsonrpc": "2.0",
```

```
  "id": 1,
  "method": "call",
  "params": [ "00000000000000000000000000000000", "session", "login", {
"username": '\"$USER\"', "password": '\"$PASS\"'  } ]
}'
```

the `-k` option is required because the Smart Radio doesn't use a third party certificate authority. An example of using JSON-RPC API for file access is shown below. Substitute `<TOKEN>` with the value returned above.

```
TOKEN=$1
curl -k https://<IP-ADDRESS>/ubus -d '
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "call",
  "params": [ '\"$TOKEN\"', "file", "read", { "path":
"/tmp/status.json" } ]
}'
```

We recommend parsing data on your local machine rather than trying to parse it on the Smart Radio.

Using the JSON-RPC API requires knowledge of UBUS. Please read the section Running Commands in the CLI for more information.

Appendix D –MQTT

The Smart Radio has supported MQTT broker and client protocols since the February 2022 firmware release. MQTT uses a publish/subscribe model. Clients can publish messages to a topic, and all clients which are subscribed to that topic will receive the message. All communications are handled by a central broker.

Fig. 1 – MQTT publish/subscribe model

The Smart Radio uses MQTT for it's Central Config utility, so the easiest way to start a broker is to simply set one of the radios as the primary node in the Central Config configuration page. Navigate to `Services → Central Config` in the GUI and use the setup below.

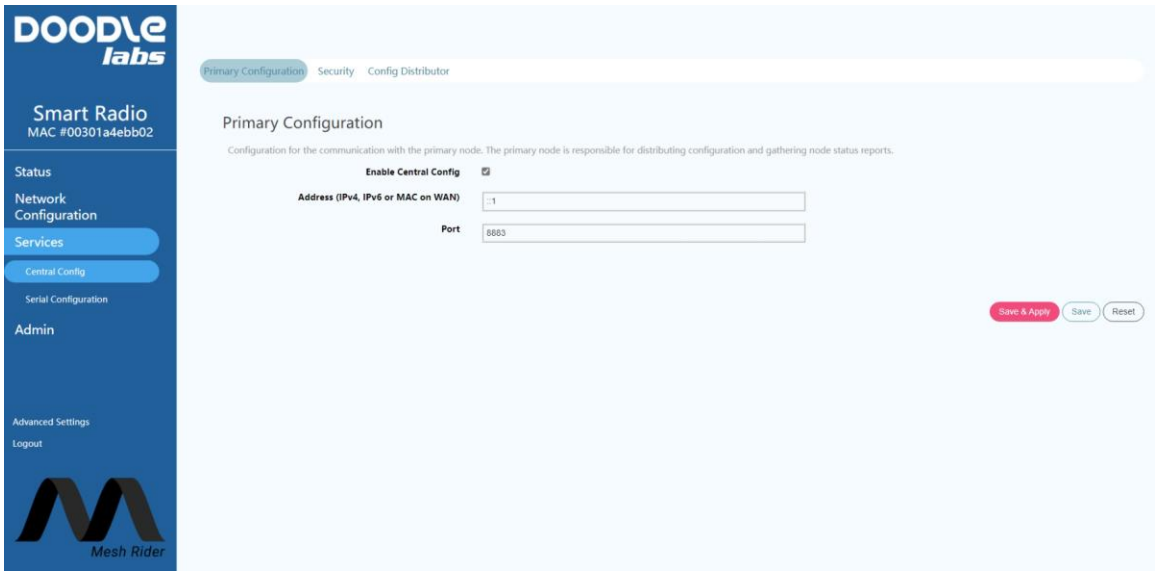

Fig. 2 – Central Config setup

This radio will now run an MQTT broker. You can modify the security settings on the broker in the security tab. You can check that the broker is running by logging into the radio over SSH and running the following command

```
ssh root@<host IP>
```

```
ps w | grep mosquitto | grep -v grep
```

You can test out the following pub/sub commands from the Smart Radio itself.

Subscribe to the topic “mytopic”

```
mosquitto_sub -h <BROKER IP> -p 8883 -t "mytopic" --psk
"0123456789abcdef" --psk-identity "doodlelabs"
```

Publish a message to the topic “mytopic”

```
mosquitto_pub -h <BROKER IP> -p 8883 -t "mytopic" --psk
"0123456789abcdef" --psk-identity "doodlelabs" -m 'Hello'
```

For details on common CLI commands, see Appendix E. If you want to create program that starts automatically on boot, see Appendix F.

Speeding up MQTT

The speed at which MQTT can send commands is limited by the TLS handshaking required for every message sent. You can also run an MQTT broker without TLS security by simply running

```
mosquitto
```

over the CLI. You can also create a start-up script to do this automatically on boot (Appendix F). The insecure MQTT broker listens on port 1883, so you will need to open the firewall on port 1883 for the broker to receive messages. With TLS disabled, the mosquitto_pub/sub commands are the same except the `--psk` and `--psk-identity` arguments are not required.

Appendix E – Common CLI Commands

This section provides commands commonly used in the Smart Radio for configuration and diagnostics.

UCI

The UCI system is used for configuration. Most UCI files are found at `/etc/config/`. This is a slow method of configuration, but changes are saved over a reboot. After committing changes, it is necessary to restart the relevant service (see the section below).

Command	Example Output	Purpose
uci show	Too long	Shows the full UCI configuration including all sections
uci show wireless	wireless.radio0=wifi-device wireless.radio0.type='mac80211' wireless.radio0.hwmode='11g' wireless.radio0.path='platform/qca953x_wmac' wireless.radio0.htmode='HT20' wireless.radio0.fes_disabled='0' wireless.radio0.rxantenna='1 2' wireless.radio0.txantenna='1 2' wireless.radio0.channel='7' wireless.radio0.chanbw='15' wireless.radio0.disabled='0' wireless.radio0.legacy_rates='0' wireless.radio0.distance='4000' wireless.wifi0=wifi-iface wireless.wifi0.device='radio0' wireless.wifi0.mode='mesh' wireless.wifi0.network='mesh_dev' wireless.wifi0.mesh_id='simpleconfig' wireless.wifi0.mesh_fwding='0' wireless.wifi0.mesh_nolearn='1' wireless.wifi0.mesh_ttl='1' wireless.wifi0.mcast_rate='12000' wireless.wifi0.encryption='psk2+ccmp' wireless.wifi0.key='DoodleSmartRadio'	Shows the wireless configuration. You can further filter the output with “uci show wireless.radio0” for example.
uci set wireless.radio.chanbw=5	NONE	Sets the channel bandwidth to 5 MHz
uci set wireless.radio0.txantenna=’1’	NONE	Sets the transmit antenna to Antenna 0 ONLY
uci commit	NONE	Saves changes. You can save individual sections too. For example “uci commit wireless”. After committing changes, you need to restart the service.

Restarting a service

After making configuration changes, restart the relevant service.

Command	Example Output	Purpose
ls /etc/init.d	RUN COMMAND	See a list of services to enable, restart, stop etc.
/etc/init.d/network restart **Available commands: restart/start/stop/enable/disable/reload**	NONE	Restarts the networking. Required after making changes to the network, like IP addressing. This will also restart other networking services like the firewall, dhcp, and the wireless interfaces.
wifi	NONE	Restarts the wireless interfaces
/etc/init.d/firewall restart	NONE	Restarts the firewall. Required after making changes to the firewall, like opening a network port.
reboot	NONE	Reboots the radio

Real-Time Configuration

The commands below work on-the-fly, but do not survive a network restart or reboot.

Command	Example Output	Purpose
iw wlan0 set txpower fixed 2000	NONE	Sets the TX power to 20 dBm. Note that the power is measured in millibels, so divide by 100 to get decibels.
iw wlan0 set txpower auto	NONE	Sets the TX power to auto (highest power)
iw wlan0 set bitrates ht-mcs-2.4 <RATE>	NONE	Fixes the bitrate. <RATE> is the MCS rate between 0 and 15 where 0-7 are single-stream rates, and 8-15 are dual-stream rates.
iw wlan0 set bitrates	NONE	Sets the MCS rate to auto
iw dev wlan0 mesh chswitch <CHANNEL> <# BEACONS>	NONE	Sends a channel switch announcement to all MESH nodes to switch to <CHANNEL> after sending <# BEACONS> beacons. CURRENTLY BUGGED (August 2021) and only the local radio will switch frequency.
hostapd_cli chan_switch <# BEACONS> <FREQ> ht	Selected interface 'wlan0' OK	Sends a channel switch announcement to all WDS Client nodes to switch to <FREQ> after sending <# BEACONS> beacons.

Getting Connection Information

Command	Example Output	Purpose
iw wlan0 station dump **OR** **iw wlan0 station get <MAC>**	Station 00:30:1a:4e:bb:26 (on wlan0) inactive time: 70 ms rx bytes: 2009144 rx packets: 18003 tx bytes: 91052 tx packets: 404 tx retries: 81 tx failed: 0 rx drop misc: 47 signal: -45 [-46, -49] dBm signal avg: -45 [-46, -50] dBm Toffset: 69680156242 us tx bitrate: 58.5 MBit/s MCS 6 rx bitrate: 52.0 MBit/s MCS 11 …	Gets information about all connected stations or an individual station. Expected throughput is not accurate.
iw wlan0 station get <MAC> | grep "inactive time"	inactive time: 70 ms	Shows how long it has been since a station was last seen
iw wlan0 station get <MAC> | grep "signal: "	signal: -45 [-46, -49] dBm	Shows the RSSI for a particular station
iwinfo wlan0 assoclist	00:30:1A:4E:F3:00 -36 dBm / -95 dBm (SNR 59) 90 ms ago RX: 29.2 MBit/s, MCS 4, 15MHz 34646 Pkts. TX: 39.0 MBit/s, MCS 11, 15MHz 510 Pkts. expected throughput: 27.3 MBit/s 00:30:1A:4E:BB:25 -51 dBm / -95 dBm (SNR 44) 100 ms ago RX: 87.7 MBit/s, MCS 14, 15MHz 32927 Pkts. TX: 39.0 MBit/s, MCS 5, 15MHz 796 Pkts. expected throughput: 27.3 MBit/s	Information on all associated stations
iw wlan0 info	Interface wlan0 ifindex 24 wdev 0x3 addr 00:30:1a:4e:bb:09 type mesh point wiphy 0 channel 7 (2442 MHz), width: 15 MHz, center1: 2442 MHz txpower 36.00 dBm …	Get information about the current wireless settings
iw wlan0 survey dump	… Survey data from wlan0 frequency: 2442 MHz [in use] noise: -95 dBm channel active time: 52516424 ms channel busy time: 11001429 ms channel receive time: 8834551 ms channel transmit time: 388874 ms	Get channel usage statistics

	…	
batctl	RUN APPLICATION	See a list of commands for mesh interface information and configuration
batctl o	[B.A.T.M.A.N. adv 2021.0-openwrt-1, MainIF/MAC: wlan0/00:30:1a:4e:bb:09 (bat0/8a:0a:22:0a:2e:58 BATMAN_IV)] Originator last-seen (#/255) Nexthop [outgoingIF] 00:30:1a:4e:f3:00 0.000s (187) 00:30:1a:4e:bb:27 [wlan0] 00:30:1a:4e:f3:00 0.000s (178) 00:30:1a:4e:bb:0a [wlan0] * 00:30:1a:4e:f3:00 0.000s (243) 00:30:1a:4e:f3:00 [wlan0] 00:30:1a:4e:bb:0a 0.000s (184) 00:30:1a:4e:bb:27 [wlan0] 00:30:1a:4e:bb:0a 0.000s (184) 00:30:1a:4e:f3:00 [wlan0] * 00:30:1a:4e:bb:0a 0.000s (243) 00:30:1a:4e:bb:0a [wlan0] * 00:30:1a:4e:bb:27 0.080s (253) 00:30:1a:4e:bb:27 [wlan0] 00:30:1a:4e:bb:27 0.080s (184) 00:30:1a:4e:bb:0a [wlan0] 00:30:1a:4e:bb:27 0.080s (180) 00:30:1a:4e:f3:00 [wlan0]	See information about connected mesh nodes including preferred hop (*), last-seen time, transmit quality (#/255)…

Networking Information

Command	Example Output	Purpose
ifconfig br-wan	br-wan Link encap:Ethernet HWaddr 00:30:1A:4E:AA:09 inet addr:10.223.187.9 Bcast:10.223.255.255 Mask:255.255.0.0 inet6 addr: fe80::230:1aff:fe4e:aa09/64 Scope:Link UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1 RX packets:113310 errors:0 dropped:0 overruns:0 frame:0 TX packets:105496 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000 RX bytes:36813834 (35.1 MiB) TX bytes:28067314 (26.7 MiB)	Show information about the WAN bridge
route -n	Kernel IP routing table Destination Gateway Genmask Flags Metric Ref Use Iface 10.223.0.0 0.0.0.0 255.255.0.0 U 0 0 0 br-wan 192.168.1.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1 192.168.153.0 0.0.0.0 255.255.255.0 U 0 0 0 br-wan	Show the routing table
netstat -tuapn	… tcp 0 0 10.223.187.9:22 10.223.0.90:65297 ESTABLISHED 26182/dropbear	Show socket connection information

	…	
arp	IP address HW type Flags HW address Mask Device 10.223.0.90 0x1 0x2 b0:25:aa:2d:d3:8e * br-wan 10.223.187.6 0x1 0x0 00:00:00:00:00:00 * br-wan	Show the address resolution protocol table.
ip a	… 5: br-wan: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000 link/ether 00:30:1a:4e:aa:09 brd ff:ff:ff:ff:ff:ff inet 10.223.187.9/16 brd 10.223.255.255 scope global br-wan valid_lft forever preferred_lft forever inet 192.168.153.1/24 brd 192.168.153.255 scope global br-wan valid_lft forever preferred_lft forever inet6 fe80::230:1aff:fe4e:aa09/64 scope link valid_lft forever preferred_lft forever …	Show information about IP addresses
fw3 print	… iptables -t filter -A zone_wan_input -p udp -m udp --dport 2000 -m comment -- comment "!fw3: Allow-Socat" -j ACCEPT iptables -t filter -A zone_wan_input -p tcp -m tcp --dport 2000 -m comment -- comment "!fw3: Allow-Socat" -j ACCEPT …	fw3 is a front end to iptables and can be used to configure the firewall.
cat /proc/net/nf_conntrack	… ipv4 2 tcp 6 7439 ESTABLISHED src=10.223.0.90 dst=10.223.187.9 sport=65297 dport=22 packets=767 bytes=54829 src=10.223.187.9 dst=10.223.0.90 sport=22 dport=65297 packets=640 bytes=107146 [ASSURED] mark=0 use=2 …	See exiting network connections
bmon -b **OR** **bmon -b -p wlan0**	RUN APPLICATION	Network usage information

System Information

Command	Example Output	Purpose
dmesg	… [39.776169] batman_adv: bat0: IGMP Querier appeared [39.776180] batman_adv: bat0: MLD Querier appeared [40.781438] IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready [41.215540] batman_adv: bat0: Adding interface: wlan0 [41.215567] batman_adv: bat0: Interface activated: wlan0	See kernel messages
cat /var/log/messages **OR** **logread**	… Sat Feb 27 19:25:19 2021 daemon.warn dnsmasq-dhcp[2133]: no address range available for DHCP request via br-wan …	See logger messages
top	RUN APPLICATION	Check processor load
free	total used free shared buff/cache available Mem: 59436 19244 19492 952 20700 18288	Check memory usage

Appendix F – Creating a Bootup Script

The Smart Radio uses Openwrt's procd system for init scripts [4].

Example

We will create a simple script to echo a message to the system logs every 5 seconds. Save the following listing as `/usr/bin/my_startup_script.sh`

```
#!/bin/sh

while (sleep 5) do
        logger -t "My Message" "Hello"
done
```

You now have to make the script executable. Run

```
chmod +x /usr/bin/my_startup_script.sh
```

You can use the following basic listing for a startup script. Save the file in your Smart Radio as `/etc/init.d/my_init_script.`

```
#!/bin/sh /etc/rc.common

USE_PROCD=1
START=99
PROG="/usr/bin/my_startup_script.sh"

start_service() {
        procd_open_instance
        procd_set_param command $PROG -p $PORT
        procd_set_param respawn 0 5 0
        procd_close_instance
}
```

After creating the file, make it executable, and then enable and start the init script.

```
chmod +x /etc/init.d/my_init_script
/etc/init.d/my_init_script enable
/etc/init.d/my_init_script start
```

You can also follow the system log messages from my_startup_script.sh by running

```
logread -f "My Message"
```

References

[1] Doodle Labs Knowledge Base, https://doodlelabstechsupport.zohodesk.com/portal/en/kb/doodle-labs, Sept 2022
[2] Openwrt UBUS, https://openwrt.org/docs/techref/ubus, Sept 2022
[3] jq, https://stedolan.github.io/jq/, Sept 2022
[4] Openwrt procd, https://openwrt.org/docs/guide-developer/procd-init-script-example, Oct 2022