

OpenWRT Presentation Documentation

Ian Reinhart Geiser

Jun 11, 2017

Contents

1	Why wouldn't you do this	2
2	Why would you do this	3
3	About OpenWRT	4
4	Configuring OpenWRT	5
4.1	UCI	5
4.2	LuCI	6
5	Building Your First Image	7
5.1	Prerequisites	7
5.2	Feeds	7
5.3	Configuration	8
5.4	Base Configuration	8
5.5	Managing Customizations	9
6	Extending your image	10
6.1	Custom default settings	10
6.2	Common files to include	10
7	Fun applications!	15
7.1	Debug Switch	15
7.2	Private Access Point	17
8	Going Further	22

Docs: <http://openwrt-presentation.rtf.d.io>

Repo: <http://bitbucket.org/geiseri/openwrt-presentation>

By: Ian Reinhart Geiser

My name is Ian Geiser and I am actually not a developer or in any way associated with the OpenWRT project. I just wanted to learn more about it so I decided to research it for this conference. This presentation is a result of my adventures in learning about OpenWRT. A static version of this document can be found at [Read the Docs](#). I have also included resources and the meta-data in my [bitbucket repository](#).

CHAPTER 1

Why wouldn't you do this

There are plenty of reasons why building your own router is a bad idea. The most important one is that you will not save money. By the time you get the parts it would be cheaper just to buy a mid-range router. The other factor is time. Tweaking builds, reflashing and subsequent debugging can be a real time sink for even the most experienced geek. That also brings up the last reason you wouldn't want to do this. If you are not comfortable with build tools this process can be a real nightmare.

CHAPTER 2

Why would you do this

If those reasons do not scare you off then there are some great opportunities that await you. The coolest thing about OpenWRT is that it can give an introduction to working with and developing for embedded systems. Once you have a working system you also have a flexible playground for exploring the wonderful world of networking. Lastly like all great engineering challenges we do them because we can!

CHAPTER 3

About OpenWRT

OpenWRT is a great example of how a company who complies with the OSS licenses can spawn a community. OpenWRT started as a fork of the GPL components that Linksys released for their WRTG-54g router back in 2003.

The community took this code and extended it with a custom user interface and tools that took the product far beyond what the engineers at Linksys first imagined.

Since the first release back in 2006 the number of devices supported has exploded to over 300 types of consumer and commercial grade routers.

CHAPTER 4

Configuring OpenWRT

OpenWRT has two main configuration options UCI a command line application that is accessible with SSH or the console and LuCI the web interface.

UCI

```
root@OpenWrt:~# uci show wireless.radio0
wireless.radio0=wifi-device
wireless.radio0.type='mac80211'
wireless.radio0.hwmode='11g'
wireless.radio0.path='pci0000:00/0000:00:1c.5/0000:04:00.0'
wireless.radio0.htmode='HT20'
wireless.radio0.country='00'
wireless.radio0.channel='11'
root@OpenWrt:~# uci set wireless.radio0.channel=6
root@OpenWrt:~# uci commit wireless
root@OpenWrt:~#
```

```
1 config 'wifi-device' 'radio0'
2     option 'type' 'mac80211'
3     option 'hwmode' '11g'
4     option 'htmode' 'HT20'
5     option 'country' '00'
6     option 'channel' '11'
```

UCI gives a structured view of the key/value pairs of each configuration file. The tool can read and

write values to a temporary instance of each configuration file and then commit to the physical file when all of the edits are complete. Since the UCI tool is a command line utility it makes it easier to script things for dynamic operations.

LuCI
















OpenWrt


Status ▾System ▾Network ▾Logout

AUTO REFRESH ON

Interfaces

Interface Overview

Network	Status	Actions
<div><div>LAN</div><div>br-lan</div></div>	<div>Uptime: 3d 12h 28m 16s</div> <div>MAC-Address: 00:E0:4C:68:31:71</div> <div>RX: 7.94 MB (44355 Pkts.)</div> <div>TX: 47.42 MB (73213 Pkts.)</div> <div>IPv4: 192.168.1.1/24</div> <div>IPv6: fd3e:bf78:1d8f::1/60</div>	<div> Connect</div> <div> Stop</div> <div> Edit</div> <div> Delete</div>
<div><div>WAN</div><div>Client "GEEKCENTRAL"</div></div>	<div>Uptime: 3d 12h 28m 11s</div> <div>MAC-Address: 6C:71:D9:0D:73:D3</div> <div>RX: 91.82 MB (303833 Pkts.)</div> <div>TX: 8.10 MB (33593 Pkts.)</div> <div>IPv4: 10.0.5.20/24</div>	<div> Connect</div> <div> Stop</div> <div> Edit</div> <div> Delete</div>
<div><div>WAN6</div><div>Client "GEEKCENTRAL"</div></div>	<div>Uptime: 0h 0m 0s</div> <div>MAC-Address: 6C:71:D9:0D:73:D3</div> <div>RX: 91.82 MB (303833 Pkts.)</div> <div>TX: 8.10 MB (33593 Pkts.)</div>	<div> Connect</div> <div> Stop</div> <div> Edit</div> <div> Delete</div>

 Add new interface...

LuCI has a more visual approach where properties of a configuration are edited in a HTML form and then applied once editing is complete. Unlike the UCI tool the HTML interface gives a richer amount of feedback to the user at the expense of automation.

CHAPTER 5

Building Your First Image

Before you build your first image you should be comfortable with the Linux command line as well as tools like git and make. None the less most of the real work will be done in a text editor, a text user interface or a few specific commands.

Prerequisites

To do the actual build there needs to be a few support tools installed on the host operating system. The key ones for your host os can be found on the OpenWRT wiki. To confirm most packages there is a `make prereq` target that can be run to verify correct installation of the tools. The key ones that you will need though to get started are git, a host compiler tool chain and ncurses.

Feeds

```
$ ./scripts/feeds update -a  
$ ./scripts/feeds install -a
```

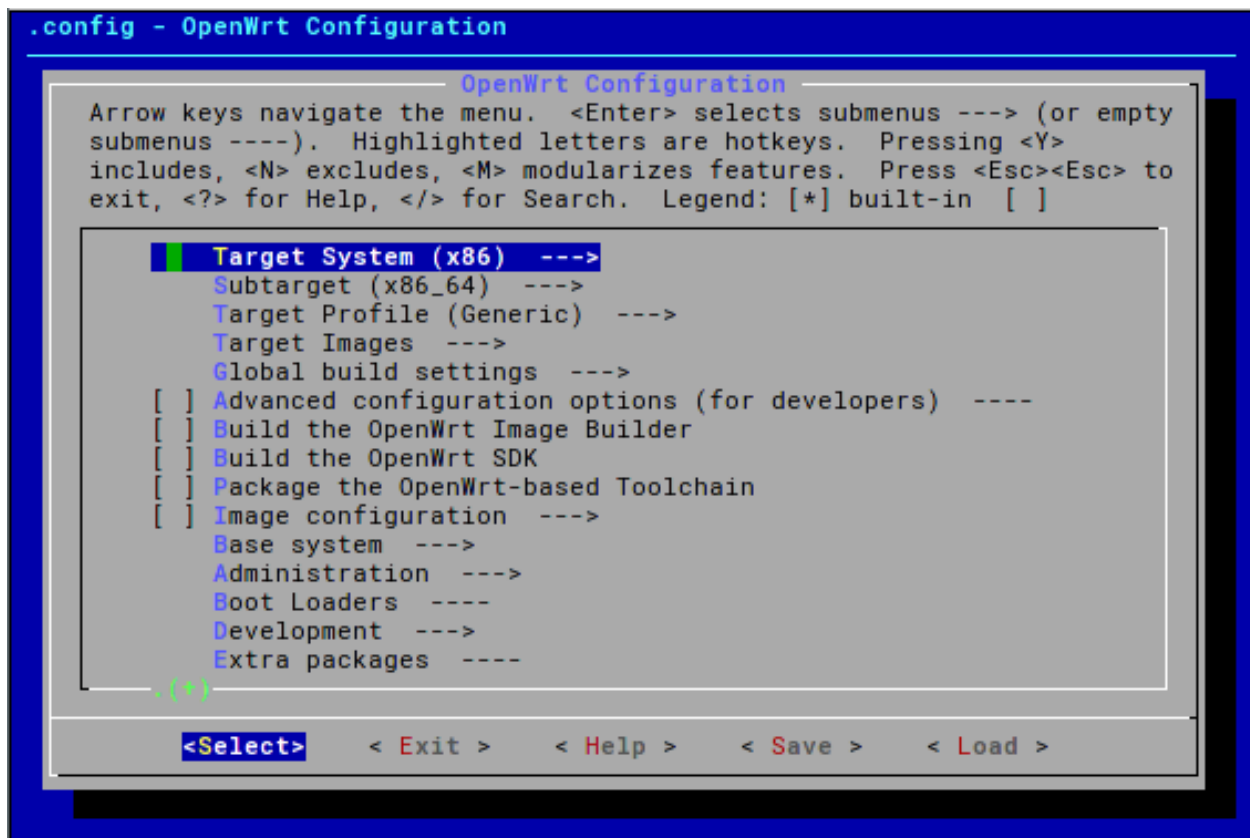
Once the environment has been added the next step is to add some applications to the build. This step is optional but in the end is needed to provide extensions.

The key way of adding applications to your image in OpenWRT is using feeds. Feeds are different repositories that organize the different apps. Depending on how simple or complicated you wish to make your image you can add some or all of the feeds. In this example all of the feeds have been

added. When maintaining your image you should periodically run the update and install process to ensure you have all of the latest updates to applications.

Configuration

The build system of OpenWRT is descended from buildroot2 so it has a nice ncurses configuration menu similar to the kernel's `make menuconfig`. This allows a very easy way to configure custom images with all the creature comforts such as searching and help for each option.



Base Configuration

```
1 CONFIG_TARGET_x86=y
2 CONFIG_TARGET_x86_64=y
3 CONFIG_TARGET_x86_64_Generic=y
4 CONFIG_PACKAGE_kmod-usb-hid=y
5 CONFIG_PACKAGE_kmod-usb2-pci=y
6 CONFIG_PACKAGE_kmod-usb3=y
7 CONFIG_PACKAGE_kmod-ath9k=y
8 CONFIG_ATH9K_SUPPORT_PCOEM=y
```

```
9 CONFIG_PACKAGE_luci-ssl-openssl=y
10 CONFIG_PACKAGE_wpad=y
```

One advantage of using `kconfig` is that you can edit configuration fragments outside of `make menuconfig`. Here we have a simple configuration template to base the projects here off of. This configuration will set the target to be a generic 64bit x86 PC. While OpenWRT supports many different routers for the sake of hackability I chose to use a simple PC with multiple Intel e1000 NICs and an Atheros 9k compatible WiFi card.

The lines 1-3 are setting the platform to be a generic x86-64 board.

Lines 4-6 are adding USB and keyboard support. While this is not 100% necessary it is helpful in cases where you are debugging and SSH may not be accessible.

Lines 7-8 are enabling the Atheros wireless card that I am using in this example. In practice you can put any wireless card in here, but the easiest way to do this is via the `make menuconfig` UI.

Line 9 is enabling the web UI. This is optional but it can be easier to tweak things from the web UI during production operations.

Finally line 10 is enabling the wireless access point feature.

This is only a fragment so you would copy this into the `.config` of your source directory and run `make menuconfig` or `make defconfig`.

Managing Customizations

```
$ ./scripts/diffconfig > ../test.config
$ cp ../other.config .config
$ make defconfig
$ make
```

Once the you have configured the image with `menuconfig` it is good practice to save your configuration. Since OpenWRT is constantly updating you should only save your local changes from the default configuration. OpenWRT provides a tool called `diffconfig` that will do just that. It will compare your changes to the configuration defaults and generate a fragment file. This file can then later be copied into the `.config` and rehydrated by using the `make defconfig` or `make menuconfig` targets.

CHAPTER 6

Extending your image

Once an image has been built it can be installed on the device as is and configure it during run time. Settings are persisted between reboots and optionally between firmware updates. In some cases though it is desirable to have specific settings as defaults on the image. OpenWRT allows for the adding of files in the base image. The advantage of doing this is that these settings will survive a factory reset as well as come up correctly on first boot.

Custom default settings

```
$ mkdir -p $BASE_DIR/files/etc/config  
$ $EDITOR $BASE_DIR/files/etc/config/network
```

To add files is very easy. Under the source checkout a `files` directory would be created. Then the directory structure under that directory will be copied into the final image.

It is recommended that settings are tested on the device first and then copied into the project. Otherwise there could end up with a bad default configuration that may not even allow the image to boot.

Common files to include

Most common configuration files that the OpenWRT configuration system uses are in a single directory. In the applications for this presentation the key ones are system, network, and wireless.

In most cases if these files are not included in the image they are generated with sensible defaults on the first boot. This is the case for configurations as firewall and DHCP settings.

System Configuration File Structure

```
1 config 'system'
2     option '<key>' '<value>'
3
4 config 'timeserver' 'ntp'
5     list 'server' '<server1>'
6     list 'server' '<server2>'
7     option enable_server 0
```

The system configuration file is in the OpenWRT UCI format. This format is acceptable at run time via the `uci` command or the web interface. In the case of this example the file will be embedded in the image at build time.

The main components covered here are the system and the time server sections.

Common System Options

Commonly used options in the system configuration are the hostname and timezone. In the time-server section there are options to set and enable NTP servers.

system.hostname Hostname displayed by the router. In cases that the router is providing DHCP and DNS this name will be used for the gateway.

system.timezone The timezone string that will be used for local date-time displays on the router. This code is specific to OpenWRT and the full list of codes can be found at: https://wiki.openwrt.org/doc/uci/system#time_zones

timeserver.server This is a list of NTP servers to query for the current time. In the case that there are no servers defined then the builtin NTP server is not enabled.

timeserver.enable_server This is an optional setting that will explicitly enable or disable the NTP server on the router.

Network Configuration File Structure

```
1 config 'interface' '<name>'
2     option '<key>' '<value>'
```

The structure of the network configuration file should seem familiar to those who have edited the Debian network `interfaces` file. It has a section where you can define the interface and then a set of options for that interface.

There is another configuration type called `switch` that can be used for embedded devices that support hardware switches. This can be used to set up VLANs specific to parts of the switch if needed.

For the case of our examples we will be focused on discrete network interfaces.

Common Network Interface Options

The main option for the network interface configuration is the protocol. This paper will be focusing on static and DHCP configurations. There is in fact support for 3g modems, PPP, PPPoE and many other protocols.

proto Protocol used to define the interface it is usually static, or DHCP for most cases.

type This allows for the changing of the interface type to `bridge` Normally this value is left undefined.

ifname The name of the physical interface that is configured by this section. In cases that the type defined is `bridge` then this value can have multiple interfaces listed.

enabled This can allow the interface to explicitly be enabled or disabled by default. This is helpful if the interface needs extra configuration at run time before it is ready to be used.

ipaddr The IP address of the interface. Static configuration protocol only.

netmask The netmask given to the interface. Static configuration protocol only.

gateway The default gateway for the interface. Static configuration protocol only.

dns A list of DNS servers that are used for this interface. This is an optional setting for the static configuration protocol.

Wireless Configuration File Structure

```
1 interface 'wifi-device' '<name>'
2     option '<key>' '<value>'
3
4 config wifi-iface
5     option 'device' '<wifi-device.name>'
6     options 'network' '<network name>'
7     option '<key>' '<value>'
```

The wireless configuration file structure is broken into two distinct parts.

The first part is the “wifi-device” description. This is the definition for the radio that will be used. The options are a list of key and value pairs.

The second part is the actual definition of the wireless interface. This would map to the network device such as `wlan0` or, `wwan`, etc. There are two mandatory options that need to be present.

The first one is the `device`. This maps to the name used for the `wifi-device` in the first section. The second option is the `network`. The reason that the sections are separate is because you can have multiple interfaces per radio. The interfaces each map to a `network`. Usually devices are mapped to networks in the `/etc/config/network` configuration file. Since the interface names are not deterministic the `network` is set here and maps back to the network name in the `/etc/config/network` configuration file.

Common Wireless Interface Options

type This is the driver of wireless interface. Usually it is `mac80211` for modern wireless cards. For embedded platforms they may need special drivers such as `ath5k` or `broadcom`. Most of the time this value can be auto-detected.

ifname This is the name of the wireless interface. In most cases this value is automatically determined, but you can set it when you have hard coded settings that need a deterministic wireless card name.

hwmode This is the protocol that the wireless card will operate on. The three acceptable options are `11b`, `11g`, and `11a`. In practice `11g` enabled all 2.4Ghz protocols and `11a` enabled all supported 5Ghz protocols.

disabled This setting toggles the on and off states of the radio. Most cases this can be used to disable the radio in cases that there needs to be further configuration at run time to make it functional.

channel This is the setting that selects the channel that the wireless card will function on. In most cases you can leave this value as `auto` so it will select the minimum available channel.

country This is the country code that will dictate what channels and transmission powers are available to the radio. If this is empty it will use the card's default but for best performance this value should be set.

Common Wireless Configuration Options

device This is the name of the device. If there was a `ifname` defined in the interface options then it would be here. In most cases if there is only one wireless card it would be `radio0`

mode This selects the mode of the wireless interface. In most cases it would be `ap`, but if you want the card to operate as a client `sta` would be used.

ssid This sets the SSID that will be used for the wireless network. In cases of the mode being `ap` this would be the SSID broadcast to clients. In the `sta` mode this would be the SSID to attach to.

encryption This is the encryption type the wireless network will use. In cases of the mode being `ap` this would be the encryption type clients would need to connect with. In the `sta` mode it would be the encryption type the wireless card will connect to the SSID. In most

cases `psk` for WPA personal passkey, or `psk2` to use WPA2 personal passkey. If there is no password then the value of `open` can be used.

key This setting is the WPA key associated with the network. In cases of the mode being `ap` then this would be the passkey clients would use to connect to the SSID. In the `sta` mode this would be the passkey used to connect to the SSID.

network This is the name of the network to associate the wireless network with. Wireless configurations do not have a deterministic interface name so it is important to associate the network here. This would then be excluded from the `network` configuration.

disabled This setting toggles the on and off states of the radio. Most cases this can be used to disable the radio in cases that there needs to be further configuration at run time to make it functional.

hidden This setting is only valid in `ap` mode. When set to `1` it will disable the broadcast of the SSID.

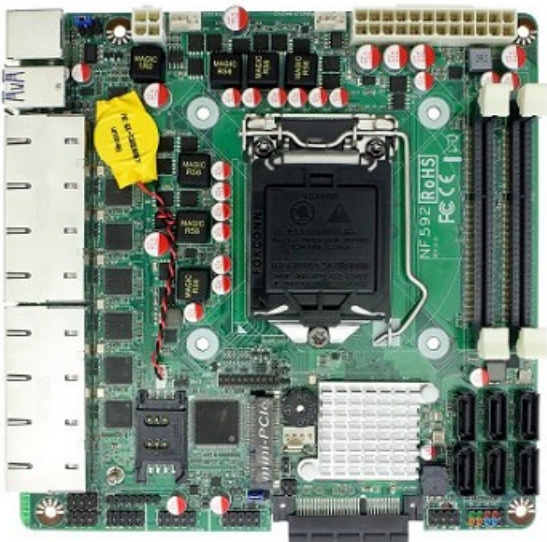
isolate This setting is only valid in `ap` mode. When set to `1` it will make it so wireless clients cannot communicate with each other.

CHAPTER 7

Fun applications!

Now that the basics of building and configuring an image it is time to apply it to some projects. The first application will be a “switch” that can be used for network debugging. The second application is a private access point that will secure an otherwise insecure wireless network.

Debug Switch



One cool thing about OpenWRT is that it not only makes configuring hard network things easy, it

is also extensive enough to include advanced debugging tools. This application is a software based switch. It uses a Jetway x86 based board with 8 built in Ethernet ports. The ports are bridged together so that as far as devices connected to the Ethernet are concerned it is an ordinary switch.

From this platform captures can be performed to visualize network traffic at a packet level. Network load metrics can also be viewed from the LuCI web interface.

Image Configuration

```
1 CONFIG_PACKAGE_luci-app-vnstat=y
2 CONFIG_PACKAGE_tcpdump=y
3 CONFIG_PACKAGE_kmod-igb=y
```

Most of the configuration that is needed for this application is already included in the base configuration template from earlier. The key things to add are the debugging and visualization tools for making the switch useful.

Line 1 adds a web interface for the utility `vnstat`. This tool will help an administrator to see traffic statistics for the switch.

The second line adds the popular packet dumping tool `tcpdump`.

The last line is needed to add support for the 8 port Ethernet controller.

Device Configuration

```
1 config interface lan
2 option type      'bridge'
3 option ifname    'eth0 eth1 eth2 eth3 eth4 eth5 eth6 eth7 eth8'
4 option proto     'dhcp'
```

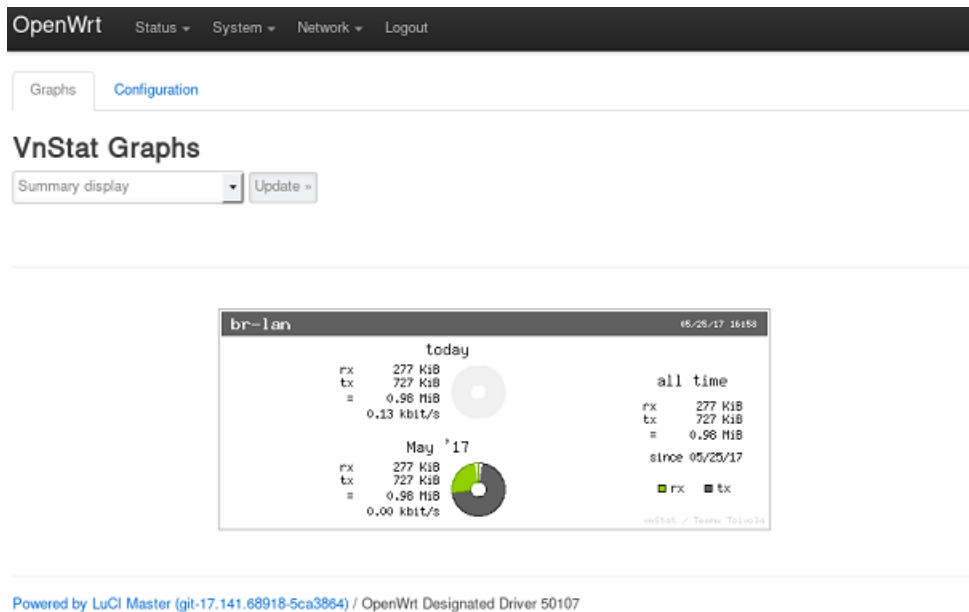
The actual configuration is quite minimal. Only network file needs to be changed. In this case to emulate a switch a bridge is created of all of the physical interfaces. While not as efficient as a real switch the x86 CPU is more than capable of managing the traffic.

Putting it Together

```
$ ssh root@HOST tcpdump -U -s0 -w - 'not port 22' wireshark -k -i -
```

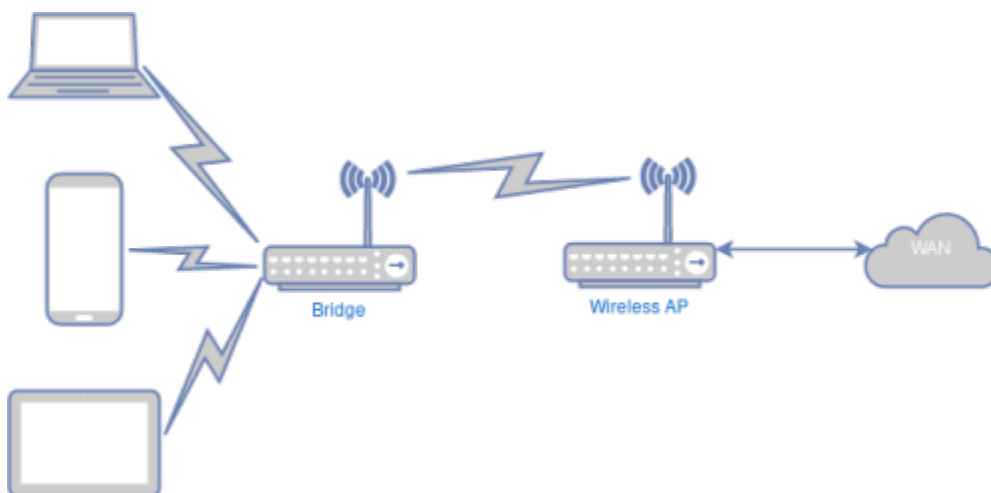
Being able to capture remote traffic at a switch level is a big help when dealing with strange network errors. Normally in a unmanaged switch there is no insight into the entire network's traffic. Since the "switch" here is actually a bridge network interface `tcpdump` can be run directly on that interface. Then the output can be piped over a SSH connection to a client running wireshark. One thing to be aware of is that port 22 traffic should be filtered to keep `tcpdump` from capturing the

packets its sending to the client. Different filters can be applied depending on what the user is looking for and how much bandwidth that should be captured. Once the packets are in wireshark there is a slick user interface that will give insight into the local network.



One other useful feature of this “switch” is that network traffic can be monitored and metrics gathered. A nice tool for OpenWRT is called `vnstat`. This tool will generate graphs of various network metrics over time. This can help identify top consumers of network resources or just generate pretty graphs to look at.

Private Access Point



This will allow the AP to act as secure link to a remote SSH server that is considered safe and expose it as a local SOCKS5 proxy. This provides two features: The first is adding security to

an otherwise insecure wireless network. The second is adding the ability to get around any local firewalls that might be limiting access to specific sites.

Image Configuration

```
1 ...  
2 CONFIG_PACKAGE_openssh-client=y  
3 CONFIG_PACKAGE_sshtunnel=y
```

For this application the basic template `.config` file is extended to add SSH and `sshtunnel`. The `sshtunnel` package is a set of shell script provided by OpenWrt to automate setting up SSH tunnels from UCI. It depends on the full OpenSSH client so that needs to be added to the application configuration.

Radio Configuration

```
1 config wifi-device 'radio0'  
2     option type 'mac80211'  
3     option hwmode '11g'  
4     option htmode 'HT20'  
5     option country 'US'
```

After the configuration is created there are some default settings that should be baked into the image. This will cause the router to come up in enough of a default state that the remainder can be configured at run time.

In the `/etc/config/wireless` configuration file the radio should be defined in a way compatible with the desired devices. In this case line 3 is going to set the wireless mode into legacy support. Most devices support 2.4Ghz so this is the best. There is also line 5 that defines the country code for any wireless regulations that need to be enforced. While this line is optional, it is a nice thing to do.

Access Point Configuration

```
1 config wifi-iface  
2     option device 'radio0'  
3     option network 'lan'  
4     option mode 'ap'  
5     option ssid 'OpenWrt'  
6     option encryption 'psk2'  
7     option key 'password'
```

The next section of the `/etc/config/wireless` is going to be the access point for the station to connect to.

Line 2 is going to be the radio we defined in the section above.

Line 3 is the network that this access point will be attached to. This needs to be done here and not in the `/etc/config/network` configuration file because wireless interfaces are calculated on the fly.

Uplink Configuration

```
1 config wifi-iface
2     option device 'radio0'
3     option mode 'sta'
4     option encryption 'none'
5     option ssid 'hotel'
6     option network 'wan'
```

Once the access point part of the configuration has been done the station section can be added.

Lines 5 and 6 will need to be modified at run time, but the other lines can be configured at build time. You can put some default values in here too. It won't hurt anything. Since the station and access point share the same radio line 2 needs to be the same as the other sections.

To configure the interface as a station to the wireless network line 3 needs to be `sta`.

The last line then needs to be configured to set the network name to be `wan`. This will be needed for the firewall to set up forwarding from the private access point.

LAN Configuration

```
1 config interface 'lan'
2     option proto 'static'
3     option ipaddr '192.168.1.1'
4     option netmask '255.255.255.0'
```

The next file that needs to be configured is the `/etc/config/network` file. Normally this can be left empty and it will come up with sane defaults. In this case the configuration will be provisioned at build time. The `lan` interface will be the gateway so it needs a fixed address.

Line 2 will set the interface as static and line 3 is the IP address of the router.

The last line is needed to set the netmask. One key thing to notice in the configuration is that the `iface` configuration is missing. This is because it was defined in the wireless configuration. It is important that the interface name in line 1 is the same as the `network` option in the `/etc/config/wireless` file.

WAN Configuration

```
1 config interface 'wan'
2     option proto 'dhcp'
```

The last section needed in the `/etc/config/network` file is the `wan` configuration. In this case the DHCP protocol will be used to configure the IP address and gateway. It is important again to make sure the interface name matches the `network` option for the `wifi-iface` in the `/etc/config/wireless` file.

SSH Tunnel Configuration

```
1 config server 'home'
2     option user 'username'
3     option hostname 'ssh.remotehost.com'
4     option IdentityFile '/root/.ssh/id_rsa'
5     option retrydelay '15'
6
7 config tunnelD 'proxy'
8     option server 'home'
9     option localaddress '*'
10    option localport '4055'
```

The last file that needs to be created is the ssh tunnel configuration file, `/etc/config/sshtunnel`. This will automate the connection and configuration of the ssh tunnel. In the configuration file there are two sections. The first section is the server connection and the second section is the proxy configuration.

Line 2 is the username for the connection and line 3 is the remote ssh host that will be connected to. Since this is an automated system service it cannot take an interactive password. For this reason public key authentication is used.

Line 4 is the path to the private key. This key will actually be configured at run time in this example.

The line 5 is important because otherwise the service will timeout and not retry again. This value should be something sensible because it may take a few seconds for the wireless client to make a connection to the WAN.

Line 8 is the server configuration for the proxy. In this case the server name on line 1 is used. This gives the advantage of having multiple remote hosts or multiple exposed proxies.

Lines 9 and 10 are related to what the proxy should listen on. In this case the SOCKS5 proxy will listen on port 4055 of all interfaces.

Putting it Together

Once the image has been created and booted the SSH connection needs to be configured. In theory a static key could be baked into the image and copied to the remote server at build time tough. In this example the key is generated at first boot via a SSH connection, or the local console.

Line 1 will run the `ssh-keygen` command to generate the public and private ssh keys.

Lines 2 through 4 will copy this generated key to the remote ssh host. This also has a side effect of adding the remote host to the `known_hosts` file.

Line 5 is optional, but will confirm that the connection is working before the service is enabled. The last line will then enable the ssh tunnel to be created at boot time.

```
1 $ ssh-keygen
2 $ cat /root/.ssh/id_rsa.pub | \
3   ssh username@ssh.remotehost.com \
4   'mkdir -p .ssh && cat >> .ssh/authorized_keys'
5 $ ssh username@ssh.remotehost.com
6 $ /etc/init.d/sshtunnel enable
```

The last step in configuration is to connect to the access point and then set up the browser proxy to use SOCKS5 on port 4055 of the access point's static IP address. It is important to remember to add an exception for the 192.168.1.1 address so that the LuCI html5 interface is still accessible.

CHAPTER 8

Going Further

OpenWRT includes over 3000 packages that can be added to an image. These packages include everything from captive portals to proxies to VPNs. There are even applications beyond just a router like a DNLA media streamer or a samba file share. While not all of these applications have documentation many of them are documented on the OpenWRT wiki. You can also search github for many other people's contributions.