WISP in a box

Task specifications

version:

- 0.1 20080630, sebastian
- 1.0 20090910 sebastian

Table of Contents

Introduction and context

The WISP in a box project is aiming to

take the best known ingredients from the open source / free software world,
bundle them and make them easy to use,
put them on low cost, low power hardware (which will be solar powered)

to create a
easy-to-use-and-run wireless ISP box

and make this product available to
entrepreneurs, activists, movers of all kinds

in order to help bringing
connectivity to underserviced, underprivileged and overcharged communities in Africa

The background of the project starts at

The London, UK meeting, December 2006,
organized by the Association for Progressive Communications

which brings together about 50 people active in wireless networking on the african continent, to look into past and future of capacity building initiatives.

Janet Haven (OSI) writes:
“Another group looked at software issues: if one were to aggregate the technology needed to run a WISP - from mesh networking software to billing systems that worked in a world without credit cards - what would it look like? Building off the Tactical Technology Collective's popular "in-a-box" idea, everyone around this table agreed to work towards a "WISP-in-a-box".

The idea remains “homeless” until early 2008, when finally the project is initiated by a team based at the Meraka Institute, South Africa.

Feedback from practicioners and entrepreneurs during the work phases March-July 2008 shows that major focus is on the following areas:

1. Network management
2. Billing and Authentication

This document specifies tasks within these two area.

To give examples,

- a web interface to give workflow oriented overview and access to configurations

- traffic shaping via web interface
- web design integration
- provosioning (remote commands and file push) on the mesh

Common to all tasks is the fact that part of the envisioned functionality may already be offered by software components used on the system – where that is the case, the tasks should focus on integration and customization.

Architecture

Hardware

The WISP in a box system consists of two parts:

- gateway server: a x386 compatible server
- access node

The current reference implementation is as follows:

Gateway server:

- ALIX3C3 Board, 500 MHz AMD Geode LX800, 256MB RAM, 2 mini PCI, USB, VGA
- power: ~ 8 W
- (alternative: VIA C7 1.2 GHz (but: 25 W!), upcoming Atom boards, and others)
-

Access node:

- Linksys WRT54GL
- power: ~5 W
- (with several alternatives: Ubiquiti, Accton, custom built from Gateworks/PCEngines boards, ..)

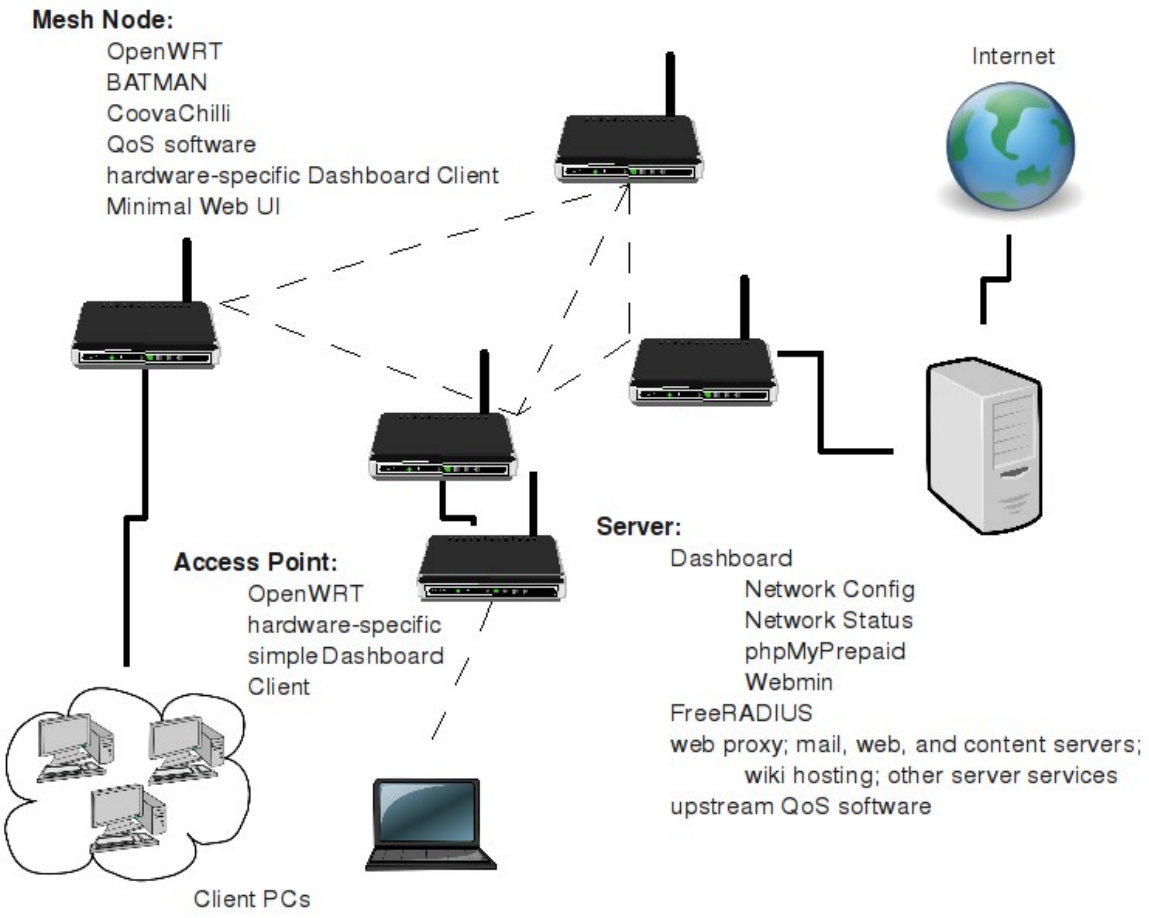

Illustration 1: High level overview of network

Software Architecture

Overview

The following tables and diagrams give an overview of the software architecture of the system.

Gateway server	**Access node**
Gateway to Internet (where applicable)	Access
Services	Authentication
Management	
Storage	

Gateway server			**Access node**		
System / Core	Services	Custom elements	System / Core	Services	Custom elements
OS: Ubuntu 8.04			FW: OpenWRT + CoovaChilli / CoovaAP *	Captive Portal / Authentication	
FreeRadius			OLSR		
MySQL			B.A.T.M.A.N.		
Apache	Nagios				
	phpMyPrepaid				
	Webmin/ISP Config				
		Meshboard			
			* depending on network topology (mesh/non-mesh)		

Mesh and non-mesh operations

It has to be noted that there are challenges around the question whether one expects to run in mesh or non-mesh mode.

Firmware on and interfaces to the first node will be different, depending on mode to run in.

Furthermore, management dashboards on the gateway server wil differ, depending on the topology and firmwares on the wireless network.

The current approach will

assume the first access node to be a mesh node, and eventually have non-mesh nodes (access points) be installed on the LAN side of this mesh node.

Pure non-mesh modes might be addressed in further iterations of the design.

Tasks and context

General guidelines for all web interfaces

Where web GUIs are involved, the guidelines for these are:

- minimal graphics and size
- mark up: validate against HTML 4.01 / Frameset
- CSS 2.0

The following may be assumed to be in place:

Apache2
php5
cgi (perl)
MySQL

Task overview and organisation

Module	**Submodule**	**Task**	**Suggested techniques / connections**	**Comments**
Management Dashboard				
	(general, top-level)	1.1. Workflow dashboard	Webmin, Nagios	
	Meshboard	1.2. Generic remote commands	Generic ssh, scp	
	QoS	1.3. Traffic shaping on server	Netfilter / iptables Mastershaper	
Node	-	1.4 Traffic shaping on nodes	OpenWRT packages	
Node	-	1.5. R.O.B.I.N on Broadcom / chipset independent	UCI	
Management Dashboard	(general, top-level)	1.6. Web integration		

Task 1.1: Workflow dashboard

Description

The workflow dashboard, a web GUI interface, will provide an overview of the mandatory variable settings that need to be in place in order for the server to function fully.
It acts as a reminder list for the network administrator, highlighting empty and conflicting settings.

To the extent possible, this functionality should draw on the Webmin and Nagios modules, and use the edit capabilities of those to allow for the change variables.

Where this is not sufficient, additional modules will have to be written. To this end, we

- Need list of all relevant configuration files and their relevant properties (file location, delimiter character, comment character , ...)
- Store these lists in flat text file or SQL db

Settings are distributed over many files, e.g. /etc/sysconfig/* , /etc/olsr.conf, batman conf, etc.

The list of mandatory settings will have to be kept variable to some extent, and be discussed during the development phase.

However, a starting point is given in the table below, to be extended.

Further inspiration may be taken from:

- http://ff-firmware.sourceforge.net/ (web UI of the Freifunk Firmware, with checklist)
- Open-mesh / OrangeMesh dashboards

Setting	**Location**	**Accesible via**
IP address / WAN (uplink)		Webmin
IP address / LAN, WLAN		Webmin
DNS info		Webmin
NTP info		Webmin?
Wireless LAN / ESSID		
Wireless LAN / BSSID		

Wireless LAN / channel		
Storage space check		Command: df / Webmin?
tba		

Prerequisites / Languages / Environment

Dependencies:

Ubuntu 8.04., apache2, php5, MySql (if needed), command line
Webmin
Nagios

Languages to be used:

php, shell script, html, javascript, (SQL)

Task 1.2: Generic remote command execution

Description

In order to deploy, provision and manage a mesh network, a generic remote command execution interface is needed.

This will be a web GUI that lets the user (the network andministrator)

- push files to all nodes accessible in a mesh
- execute commands on all nodes in a mesh

This interface should allow for

- scheduling – e.g. “reboot command on all nodes at midnight” (optional)
- status check
- logging of all activities

The script will take its starting point in identification of available nodes through

- broadcast ping
- node lists available from other modules, e.g. Nagios

Once the list of available nodes is established,

- remote access is achieved via key based ssh (no password or prompt)
- file push via scp
- remote commands via ssh/rsh

Note 1:
Typical commands to be executed on the nodes include

- *wget* of firmware updates and configuration files, e.g. from a default URL
- *iptables*
- *cron definitions*
- *iwconfig*
- *reboot*
- *custom scripts for testing of settings*

Note 2:
During the specification phase, it has been discussed whether an API (possibly XML based) would be desirable. While this might be the case in the future, at this point, we are aiming for a plain commands based version. Dependency on specific types of nodes, and assumptions about what software will be available on these, must be kept at a minimum (only standard Linux commands assumed). The interface should not depend on e.g. ROBIN or Webmin or such to be installed on

the nodes.

Note 3:
The installation of ssh keys on the nodes is a prerequisite for this module.

Prerequisites / Languages / Environment

Languages: php, shell script, (maybe SQL)

Task 1.3: Traffic shaping on server

Description

An integrated web interface interface is needed for defining QoS and Traffic shaping rules on the server.

To a large etxtent, this will draw on existing GUIs in

- Webmin
- Turtle Firewall
- Mastershaper

The task therefore is an integration task, leading to a easy-to-use web GUI interface with only a set of pre-made rules available to the user.
(An expert mode with full access to custom rules may be added later).

Settings need to allow for shaping by

Mandatory rules –“ must have”:

- service / port, e.g. web, mail, voice
- network – e,g, a prioritized premium network, a standard low service quality network
- user (identified by IP address)

Optional rules – “nice to have”:

- time dependent (e.g. One limit for daytime use, one for nighttime, ...)
- system utilization / congestion (make limits depend on how busy the network is)

An exact set of rules will be agreed on in the start up phase of the project.

Prerequisites / Languages / Environment

Dependencies:

Ubuntu 8.04., apache2, php5, MySql (if needed), command line iptables
Webmin, Mastershaper

Languages:

shell, iptables syntax, conf files, php

Task 1.4: Traffic shaping on nodes

Description

On the mesh nodes, running OpenWRT or a variation of it,
Layer 7 traffic shapingis needed, in order to assure QoS for intra-mesh traffic (i,e, traffic that never reaches the gateway).

A set of rules for iptables and qos-scripts, and a web GUI to acces those remotely as well as locally, is needed.

This will utilize the OpenWRT packages of

- iptables
- iptables-mod-filter
- iptables-mod-nat
- qos-scripts

and might utilize

- shorewall

The exact best choice for this is not known yet, the task is partly a research and identification task.
We suggest looking at X-WRT as an example, given that qos is integrated in this firmware.

Once this module is developed, it has to be reachable through the Remote commands interface descibed above, e.g. In order to push new rules out tho the nodes..

Prerequisites / Languages / Environment

Dependencies:

OpenWRT, white russian or kamikaze, X-WRT

Languages:

shell, iptables syntax, conf files, php, html

Task 1.5.: R.O.B.I.N on Broadcom / chipset independent

Description

Research task.

We need to explore the possibilities of creating a chipset-independent version of R.O.B.I.N, or, if that can not be reached in its entirety (full driver abstraction), a version that will run on Broadcom chipsets.

A rough dessription of the R.O.B.I.N. Architecture and possible ways to a solution have been discussed, quoting from email discussion with Corinna Aichele:

```
“
Robin takes advantage of the UCI (Unified Configuration Interface) of
OpenWRT Kamikaze, and defines its own configuration files (for example
/etc/config/batman, which does a lot more than just starting batmand)

As said before we can modify the UCI configuration for our needs and
check whether open-mesh/robin has a clever way for node monitoring and
administration.

<Quote about UCI from OpenWRT>

UCI which stands for Unified Configuration Interface is a C library which
provides configuration context for user space and system configuration and
management. UCI was adopted with the extent of OpenWrt to other devices
which did not have the NVRAM to store their settings into a separate flash
partition.
Since UCI is a C library, it can be easily integrated into an existing
user-space
application or to develop a configuration storage that is OpenWrt compatible
for your new application.
Further developments for UCI include a web interface that uses UCI as a
configuration file format as well as SNMP plugins to easily change the con-
figuration and take actions on the embedded device.
For instance adding a new configuration file is as simple as creating a new
file in /etc/config/package which should contain the following lines:

config <type> ["<name>"] # Section
option <name> "<value>" # Option


Later on, the system scripts and the UCI library allows you to parse this
configuration context from either an init script or directly an
user-space pro-
gram.
```

</Quote>

FFLUCI makes also use of UCI (hence the name) to provide a web-frontend.
And it is fairly easy to customize a web interface for our needs.“

Task 1.6: Web interface integration

Description

An integrated web interface for the various components described above needs to be created.

The task divides into two subtasks:

1. Creation on a menu and content structure as sketched below
2. Skinning of existing components (like Nagios, Webmin, etc) in order to give a consistent look and feel.

The overall guidelines for all web interfaces are:

- minimal graphics and size
- mark up: validate against HTML 4.01 / Frameset
- CSS 2.0

For subtask 2, the strategy will be to
identify the locations of all relevant css's, includes , etc and to merge or source those into one master css file.
This will mean some change in the source code of the individual modules.

Web mockups should be produced as the first step, however it is crucial that functional logic of the modules and not the graphical representation is governing the development process.

Organisation of web interface

Frame: Menu 1	Frame: Menu 2	Frame: Main

Menu 1	**Menu 2**	**functions**	**Connection to Main ...**
Server Administration			Webmin
Vouchers / Payment	Administration		PhpMyPrepaid, modified
	Statistics / Reporting		(to be written)
Network	Network Status	* List of nodes (IP/ MAC) * uptimes * last seen * routes / hops	Nagios + (to be written)
	Mesh Configuration	* change SSID * Change channel * reboots * firmware push * tba	Generic remote commands module (to be written)
	Statistics		To be written, drawing from FreeRadius, a.o.
	Traffic shaping	* enable premade settings, e.g. Voice priority, close for torrents, etc * custom commands Nodes:	Netfilter/iptables & On-Node shaping (to be written)

		* push QoS rules out to nodes for intra-mesh shaping	

Prerequisites / Languages / Environment

Dependencies:

Apache2

php5

Languages:

html, css, php