opennds Documentation

Release 5.1.0

the opennds contributors

Oct 28, 2022

Contents

openNDS is a high performance, small footprint Captive Portal, offering by default a simple splash page restricted Internet connection, yet incorporates an API that allows the creation of sophisticated authentication applications.

It was derived originally from the codebase of the Wifi Guard Dog project.

openNDS is released under the GNU General Public License.

- Original Homepage *down*: http://kokoro.ucsd.edu/nodogsplash
- Wifidog: http://dev.wifidog.org/
- GNU GPL: http://www.gnu.org/copyleft/gpl.html

The following describes what openNDS does, how to get it and run it, and how to customize its behavior for your application.

Contents:

CHAPTER 1

Overview

openNDS (NDS) is a high performance, small footprint Captive Portal, offering by default a simple splash page restricted Internet connection, yet incorporates an API that allows the creation of sophisticated authentication applications.

1.1 Captive Portal Detection (CPD)

All modern mobile devices, most desktop operating systems and most browsers now have a CPD process that automatically issues a port 80 request on connection to a network. NDS detects this and serves a special "**splash**" web page to the connecting client device.

1.2 Provide simple and immediate public Internet access

NDS provides two pre-installed methods.

- **Click to Continue**. A simple static web page with template variables (*default*). This provides basic notification and a simple click/tap to continue button.
- **username/email-address login**. A simple dynamic set of web pages that provide username/email-address login, a welcome page and logs access by client users. (*Installed by default and enabled by un-commenting a line in the configuration file*)

  Customising the page seen by users is a simple matter of editing the respective html or script files.

1.3 Write Your Own Captive Portal.

NDS can be used as the "Engine" behind the most sophisticated Captive Portal systems using the tools provided.

- **Forward Authentication Service (FAS)**. FAS provides pre-authentication user validation in the form of a set of dynamic web pages, typically served by a web service independent of NDS, located remotely on the Internet, on the local area network or on the NDS router.
- **PreAuth**. A special case of FAS that runs locally on the NDS router with dynamic html served by NDS itself. This requires none of the overheads of a full FAS implementation and is ideal for NDS routers with limited RAM and Flash memory.
- **BinAuth**. A method of running a post authentication script or extension program.

CHAPTER 2

Installing openNDS

2.1 Prerequisites

openNDS is designed to run on a device configured as an IPv4 router and will have at least two network interfaces:

A WAN interface (Wide Area Network). This interface must be connected to an Internet feed:

- Either an ISP CPE (Internet Service Provider Customer Premises Equipment)
- Or another router, such as the venue ADSL router.
- It must be configured as a DHCP client, obtaining its IPv4 address and DNS server from the connected network.

A LAN interface (Local Area Network). This interface MUST be configured to:

- Provide the Default IPv4 gateway in a private IPv4 subnet that is different to any private subnets between it and the ISP CPE
- Provide DHCP services to connected clients
- Provide DNS services to connected clients
- Provide Network Address Translation (NAT) for all outgoing traffic directed to the WAN interface.

2.2 Installing on OpenWrt

- Have a router working with OpenWrt. At the time of writing, openNDS has been tested with OpenWrt 18.06.x, 19.7.x and Snapshot.
- It may or may not work on older versions of OpenWrt or on other kinds of Linux-based router firmware.
- Make sure your router is basically working before you try to install openNDS. In particular, make sure your DHCP daemon is serving addresses on the interface that openNDS will manage.

  The default interface is br-lan but can be changed to any LAN interface by editing the /etc/config/opennds file.

- To install openNDS, you may use the OpenWrt Luci web interface or alternatively, ssh to your router and run the command:

  ```
  opkg update
  ```

  followed by

  ```
  opkg install opennds
  ```

- openNDS is enabled by default and will start automatically on reboot or can be started and stopped manually.
- If the interface that you want openNDS to manage is not br-lan, edit /etc/config/opennds and set GatewayInterface.
- To start openNDS, run the following, or just reboot the router:

  ```
  service opennds start
  ```

- To test the installation, connect a client device to the interface on your router that is managed by openNDS (for example, connect to the router's wireless lan).

  Most client device operating systems and browsers support Captive Portal Detection (CPD) and the operating system or browser on that device will attempt to contact a pre defined port 80 web page.

  CPD will trigger openNDS to serve the default splash page where you can click or tap Continue to access the Internet.

  See the Authentication section for details of setting up a proper authentication process.

  If your client device does not display the splash page it most likely does not support CPD.

  You should then manually trigger openNDS by trying to access a port 80 web site (for example, google.com:80 is a good choice).

- To stop openNDS:

  ```
  service opennds stop
  ```

- To uninstall openNDS:

  ```
  opkg remove opennds
  ```

2.3 Generic Linux

openNDS can be compiled for most distributions of Linux

openNDS **requires the libmicrohttpd (MHD) library**. The version must be greater than 0.9.51, but preferably version 0.9.69 or higher.

If your distribution of Linux has a package of version less then 0.9.69, you can set the openNDS config option *use_outdated_mhd* to 1. This will force openNDS to use it.

> Older versions of MHD convert & and + characters to spaces when present in form data.
>
> This can make a PreAuth or BinAuth impossible to use for a client if form data contains either of these characters eg. in username or password.
>
> MHD versions earlier than 0.9.69 are detected.
>
> If the option *use_outdated_mhd* is set to 0 (default), NDS will terminate if MHD is earlier than 0.9.69
>
> If this option is set to 1, NDS will start but log an error.

You can also compile libmicrohttpd yourself to get the latest version.

To compile libmicrohttpd and openNDS, see the chapter “How to Compile and install openNDS”.

CHAPTER 3

How openNDS (NDS) works

openNDS is a Captive Portal Engine. Any Captive Portal, including NDS, will have two main components:

- Something that does the capturing, and
- Something to provide a Portal for client users to log in.

openNDS MUST run on a device configured as an IPv4 router.

A wireless router will typically be running OpenWrt or some other Linux distribution.

A router, by definition, will have two or more interfaces, at least one to connect to the wide area network (WAN) or Internet feed, and at least one connecting to the local area network (LAN).

Each LAN interface must also act as the Default IP Gateway for its LAN, ideally with the interface serving IP addresses to client devices using DHCP.

Multiple LAN interfaces can be combined into a single bridge interface. For example, ethernet, 2.4Ghz and 5Ghz networks are typically combined into a single bridge interface. Logical interface names will be assigned such as eth0, wlan0, wlan1 etc. with the combined bridge interface named as br-lan.

NDS will manage one or more of them of them. This will typically be br-lan, the bridge to both the wireless and wired LAN, but could be, for example, wlan0 if you wanted NDS to work just on the wireless interface.

3.1 Summary of Operation

By default, NDS blocks everything, but intercepts port 80 requests.

An initial port 80 request will be generated on a client device, usually automatically by the client device's built in Captive Portal Detection (CPD), or possibly by the user manually browsing to an http web page.

This request will of course **be routed by the client device to the Default Gateway** of the local network. The Default Gateway will, as we have seen, be the router interface that NDS is managing.

3.1.1 The Thing That Does the Capturing (NDS)

As soon as this initial port 80 request is received on the default gateway interface, NDS will "Capture" it, make a note of the client device identity, allocate a unique token for the client device, then redirect the client browser to the Portal component of NDS.

3.1.2 The Thing That Provides the Portal (Splash, FAS or PreAuth)

The client browser is redirected to the Portal component. This is a web service that is configured to know how to communicate with the core engine of NDS.

This is commonly known as the Splash Page.

NDS has its own web server built in and this can be used to serve the Portal "Splash" pages to the client browser, or a separate web server can be used.

NDS comes with two standard Splash Page options pre-installed.

One provides a trivial Click to Continue splash page with template variables and the other provides a Client User form requiring Name and Email address to be entered.

Both of these can be customised or a complete specialised Portal can be written by the installer (See FAS, PreAuth).

FAS, or Forward Authentication Service may use the web server embedded in NDS, a separate web server installed on the NDS router, a web server residing on the local network or an Internet hosted web server.

The user of the client device will always be expected to complete some actions on the splash page.

Once the user on the client device has successfully completed the splash page actions, that page then links directly back to NDS.

For security, NDS expects to receive the same valid token it allocated when the client issued its initial port 80 request. If the token received is valid, NDS then "authenticates" the client device, allowing access to the Internet.

Post authentication processing extensions may be added to NDS (See BinAuth). Once NDS has received a valid token it calls a BinAuth script.

If the BinAuth script returns positively (ie return code 0), NDS then "authenticates" the client device, allowing access to the Internet.

Where FAS is used, secure modes are provided (levels 1 and 2), where the client token and other required variables are kept securely hidden from the Client, ensuring verification cannot be bypassed.

Note: FAS and Binauth can be enabled together. This can give great flexibility, with FAS providing remote verification and Binauth providing local post authentication processing closely linked to NDS.

3.2 Captive Portal Detection (CPD)

All modern mobile devices, most desktop operating systems and most browsers now have a CPD process that automatically issues a port 80 request on connection to a network. NDS detects this and serves a special "splash" web page to the connecting client device.

The port 80 html request made by the client CPD can be one of many vendor specific URLs.

Typical CPD URLs used are, for example:

- *http://captive.apple.com/hotspot-detect.html*
- *http://connectivitycheck.gstatic.com/generate_204*
- *http://connectivitycheck.platform.hicloud.com/generate_204*
- *http://www.samsung.com/*
- *http://detectportal.firefox.com/success.txt*
- Plus many more

It is important to remember that CPD is designed primarily for mobile devices to automatically detect the presence of a portal and to trigger the login page, without having to resort to breaking SSL/TLS security by requiring the portal to redirect port 443 for example.

Just about all current CPD implementations work very well but some compromises are necessary depending on the application.

The vast majority of devices attaching to a typical Captive Portal are mobile devices. CPD works well giving the initial login page.

For a typical guest wifi, eg a coffee shop, bar, club, hotel etc., a device connects, the Internet is accessed for a while, then the user takes the device out of range.

When taken out of range, a typical mobile device begins periodically polling the wireless spectrum for SSIDs that it knows about to try to obtain a connection again, subject to timeouts to preserve battery life.

Most Captive Portals have a session duration limit (NDS included).

If a previously logged in device returns to within the coverage of the portal, the previously used SSID is recognised and CPD is triggered and tests for an Internet connection in the normal way. Within the session duration limit of the portal, the Internet connection will be established, if the session has expired, the splash page will be displayed again.

Early mobile device implementations of CPD used to poll their detection URL at regular intervals, typically around 30 to 300 seconds. This would trigger the Portal splash page quite quickly if the device stayed in range and the session limit had been reached.

However it was very quickly realised that this polling kept the WiFi on the device enabled continuously having a very negative effect on battery life, so this polling whilst connected was either increased to a very long interval or removed all together (depending on vendor) to preserve battery charge. As most mobile devices come and go into and out of range, this is not an issue.

A common issue raised is:

My devices show the splash page when they first connect, but when the authorization expires, they just announce there is no internet connection. I have to make them "forget" the wireless network to see the splash page again. Is this how is it supposed to work?

The workaround is as described in the issue, or even just manually disconnecting or turning WiFi off and on will simulate a "going out of range", initialising an immediate trigger of the CPD. One or any combination of these workarounds should work, again depending on the particular vendor's implementation of CPD.

In contrast, most laptop/desktop operating systems, and browser versions for these still implement CPD polling whilst online as battery considerations are not so important.

For example, Gnome desktop has its own built in CPD browser with a default interval of 300 seconds. Firefox also defaults to something like 300 seconds. Windows 10 is similar.

This IS how it is supposed to work, but does involve some compromises.

The best solution is to set the session timeout to a value greater than the expected length of time a client device is likely to be present. Experience shows a limit of 24 hours covers most situations eg bars, clubs, coffee shops, motels etc. If for example an hotel has guests regularly staying for a few days, then increase the session timeout as required.

Staff at the venue could have their devices added to the Trusted List if appropriate, but experience shows, it is better not to do this as they very soon learn what to do and can help guests who encounter the issue. (Anything that reduces support calls is good!)

3.3 Network Zone Detection (Where is the Client Connected?)

Client devices can be connected to one of a number of local WiFi SSIDs, connected directly or indirectly by ethernet, or connected via a wireless mesh network. Each connection type available is considered as a Network Zone.

NDS detects which zone each client is connected to. This information can be used to dynamically customise the login for each zone.

For example a coffee shop might have two SSIDs configured:

- Staff (Secure SSID ie with access code)
- Customers (open SSID with login form)

In this example SSID "Staff" is configured on interface wlan0, and considered as Zone "Private".

However, SSID "Customers" is configured on virtual interface wlan0-1, and considered as Zone "Public".

NDS detects which zone is being used by a client and a relevant login page can be served.

3.4 Packet filtering

openNDS considers four kinds of packets coming into the router over the managed interface. Each packet is one of these kinds:

1. **Blocked**, if the MAC mechanism is block, and the source MAC address of the packet matches one listed in the BlockedMACList; or if the MAC mechanism is allow, and source MAC address of the packet does not match one listed in the AllowedMACList or the TrustedMACList. These packets are dropped.
2. **Trusted**, if the source MAC address of the packet matches one listed in the TrustedMACList. By default, these packets are accepted and routed to all destination addresses and ports. If desired, this behavior can be customized by FirewallRuleSet trusted-users and FirewallRuleSet trusted-users-to-router lists in the opennds.conf configuration file, or by the EmptyRuleSetPolicy trusted-users EmptyRuleSetPolicy trusted-users-to-router directives.
3. **Authenticated**, if the packet's IP and MAC source addresses have gone through the openNDS authentication process and has not yet expired. These packets are accepted and routed to a limited set of addresses and ports (see FirewallRuleSet authenticated-users and FirewallRuleSet users-to-router in the opennds.conf configuration file).
4. **Preauthenticated**. Any other packet. These packets are accepted and routed to a limited set of addresses and ports (see FirewallRuleSet preauthenticated-users and FirewallRuleSet users-to-router in the opennds.conf configuration file). Any other packet is dropped, except that a packet for destination port 80 at any address is redirected to port 2050 on the router, where openNDS's built in libhttpd-based web server is listening. This begins the 'authentication' process. The server will serve a splash page back to the source IP address of the packet. The user clicking the appropriate link on the splash page will complete the process, causing future packets from this IP/MAC address to be marked as Authenticated until the inactive or forced timeout is reached, and its packets revert to being Preauthenticated.

openNDS implements these actions by inserting rules in the router's iptables mangle PREROUTING chain to mark packets, and by inserting rules in the nat PREROUTING, filter INPUT and filter FORWARD chains which match on those marks.

Because it inserts its rules at the beginning of existing chains, openNDS should be insensitive to most typical existing firewall configurations.

3.5 Data volume and Rate Quotas

openNDS (NDS) has built in *Data Volume* and *Data Rate* quota support.

Data volume and data rate quotas can be set globally in the config file.

The global values can be overridden on a client by client basis as required.

3.6 Traffic Shaping

openNDS (NDS) supports Traffic Shaping (Bandwidth Limiting) using the SQM - Smart Queue Management (sqm-scripts) package, available for OpenWrt and generic Linux.

CHAPTER 4

The Splash Page

As you will see mentioned in the "How openNDS (NDS) Works" section, an initial port 80 request is generated on a client device, either by the user manually browsing to an http web page, or, more usually, automatically by the client device's built in Captive Portal Detection (CPD).

This request is intercepted by NDS and an html Splash Page is served to the user of the client device to enable them to authenticate and obtain Internet access.

4.1 Types of Splash Page

This Splash page can be one of the following:

- **A Static Web Page served by NDS:**

  A page generated from the basic splash.html file installed with NDS and includes Template Variables (as listed in the splash.html file). *This is the default configuration of a fresh installation of NDS.*

  A script or executable file can optionally be called by NDS for post authentication processing (see **BinAuth**).

  An example of the use of BinAuth is to check the Username and Password entered by a user into an authentication form supplied by the splash page.

- **A Dynamic Web Page served by NDS**

  A script or executable file is called by NDS immediately (without serving splash.html). The called script or executable will generate html code for NDS to serve in place of splash.html. (see **PreAuth**).

  This enables a dialogue with the client user, for dissemination of information, user response and authentication.

  This is implemented using **FAS**, but *without the resource utilisation of a separate web server*, particularly useful for legacy devices with limited flash and RAM capacity.

- **A Dynamic Web Page served by an independent web server** on the same device as NDS, on the same Local Area Network as NDS, or on External Web Hosting Services.

A script or executable file is called by NDS immediately (without serving splash.html). The called script or executable will generate html code to be served by an independent Web Server. (see FAS).

This not only enables a dialogue with the client user, for dissemination of information, user response and authentication but also full flexibility in design and implementation of the captive portal functionality from a self contained system through to, for example, a fully integrated multi site system with a common database.

4.2 The Two Installed Basic Splash Pages

By default, two fully functional but basic "Splash" pages are installed. Simple config options allow you to choose which one to use.

- The Simple "Click to Continue" splash page. (Default)
- The "Username/Email-address" Login script.

See the chapter on PreAuth for details on how to switch between these splash page types.

In many instances, one or other of these simple methods will be all that is required, but the power of FAS, PreAuth and BinAuth can be used to create very sophisticated Captive Portal Systems.

4.3 Displaying Remote Content

FASand PreAuth can be used to display remote content on the client user login screen. This is ideal for serving information, banner advertising etc.

An example is described in the **Displaying Remote Banner Images** section of the PreAuth chapter.

CHAPTER 5

Forwarding Authentication Service (FAS)

5.1 Overview

openNDS (NDS) has the ability to forward requests to a third party authentication service (FAS). This is enabled via simple configuration options.

These options are:

1. **fasport**. This enables Forwarding Authentication Service (FAS). Redirection is changed from splash.html to a FAS. The value is the IP port number of the FAS.
2. **fasremoteip**. If set, this is the remote ip address of the FAS, if not set it will take the value of the NDS gateway address.
3. **fasremotefqdn** If set, this is the remote fully qualified domain name (FQDN) of the FAS
4. **faspath**. This is the path from the FAS Web Root (not the file system root) to the FAS login page.
5. **fas_secure_enable**. This can have four values, "0", "1", "2" or "3" providing different levels of security.
6. **faskey** Used in combination with fas_secure_enable level 1, 2 and 3, this is a key phrase for NDS to encrypt data sent to FAS.

Note: FAS (and Preauth/FAS) enables pre authentication processing. NDS authentication is the process that NDS uses to allow a client device to access the Internet through the Firewall. In contrast, Forward Authentication is a process of "Credential Verification", after which FAS, if the verification process is successful, passes the client token to NDS for access to the Internet to be granted.

5.2 Using FAS

Note: All addresses (with the exception of fasremoteip) are relative to the *client* device, even if the FAS is located remotely.

When FAS is enabled, NDS automatically configures firewall access to the FAS service.

The FAS service must serve a splash page of its own to replace the NDS splash.html. For fas_secure_enable "0", "1", and "2" this is enforced as http. For fas_secure_enable level "3", it is enforced as https.

Typically, the FAS service will be written in PHP or any other language that can provide dynamic web content.

FAS can then provide an action form for the client, typically requesting login, or self account creation for login.

The FAS can be on the same device as NDS, on the same local area network as NDS, or on an Internet hosted web server.

5.3 Security

If FAS Secure is enabled (Levels 1 (default), 2 and 3), the client authentication token is kept secret until FAS verification is complete.

> **If set to "0"** The FAS is enforced by NDS to use **http** protocol. The client token is sent to the FAS in clear text in the query string of the redirect along with authaction and redir.
>
> **If set to "1"** The FAS is enforced by NDS to use **http** protocol. When the sha256sum command is available AND faskey is set, the client token will be hashed and sent to the FAS identified as "hid" in the query string. The gatewayaddress is also sent on the query string, allowing the FAS to construct the authaction parameter. FAS must return the sha256sum of the concatenation of the original hid and faskey to be used by NDS for client authentication. This is returned in the normal way in the query string identified as "tok". NDS will automatically detect whether hid mode is active or the raw token is being returned.
>
> Should sha256sum not be available or faskey is not set, then it is the responsibility of the FAS to request the token from ndsctl.
>
> **If set to "2"** The FAS is enforced by NDS to use **http** protocol.
>
> clientip, clientmac, gatewayname, client token, gatewayaddress, authdir, originurl and clientif are encrypted using faskey and passed to FAS in the query string.
>
> The query string will also contain a randomly generated initialization vector to be used by the FAS for decryption.
>
> The cipher used is "AES-256-CBC".
>
> The "php-cli" package and the "php-openssl" module must both be installed for fas_secure level 2.
>
> openNDS does not depend on this package and module, but will exit gracefully if this package and module are not installed when this level is set.
>
> The FAS must use the query string passed initialisation vector and the pre shared fas_key to decrypt the query string. An example FAS level 2 php script (fas-aes.php) is preinstalled in the /etc/opennds directory and also supplied in the source code.
>
> **If set to "3"** The FAS is enforced by NDS to use **https** protocol. Level 3 is the same as level 2 except the use of https protocol is enforced for FAS. In addition, the "authmon" daemon is loaded. This allows the external FAS, after client verification, to effectively traverse inbound firewalls and address translation to achieve NDS authentication without generating browser security warnings or errors. An example FAS level 3 php script (fas-aes-https.php) is preinstalled in the /etc/opennds directory and also supplied in the source code.

Option faskey must be set if fas secure is set to levels 2 and 3 but is optional for level 1.

Option faskey is used to encrypt the data sent by NDS to FAS. It can be any combination of A-Z, a-z and 0-9, up to 16 characters with no white space.

This is used to create a sha256 digest that is in turn used to encrypt the data using the aes-256-cbc cypher.

A random initialisation vector is generated for every encryption and sent to FAS with the encrypted data.

Option faskey must be pre-shared with FAS.

5.4 Example FAS Query strings

Level 0 (fas_secure_enabled = 0), NDS sends the token and other information to FAS as clear text.

http://fasremoteip:fasport/faspath?authaction=http://gatewayaddress:gatewayport/opennds_auth/?clientip=[clientip]&gatewayn

> Although the simplest to set up, a knowledgeable user could bypass FAS, so running fas_secure_enabled at level 1 or 2 is recommended.

Level 1 (fas_secure_enabled = 1), NDS sends only information required to identify, the instance of NDS, the client and the client's originally requested URL. The client token is never exposed.

If faskey is set, NDS sends a digest of the random client token:

http://fasremotefqdn:fasport/faspath?hid=[hash_id]&gatewayname=[gatewayname]&clientip=[clientip]&redir=[requested-url]

> The FAS must return the hash of the concatenated hid value and the value of faskey identified in the query string as "tok". NDS will automatically detect this.

If faskey is not set the following is sent:

http://fasremotefqdn:fasport/faspath?gatewayname=[gatewayname]&clientip=[clientip]&redir=[requested-url]

> It is the responsibility of FAS to obtain the unique client token allocated by NDS as well as constructing the return URL to NDS.
>
> The return url will be constructed by FAS from predetermined knowledge of the configuration of NDS using gatewayname as an identifier.
>
> The client's unique access token will be obtained from NDS by the FAS making a call to the get_client_token library utility:
>
> ```
> /usr/lib/opennds/./get_client_token $clientip
> ```
>
> A json parser could be used to extract all the client variables supplied by ndsctl, an example can be found in the default PreAuth Login script in /usr/lib/nogogsplash/login.sh.

Levels 2 and 3 (fas_secure_enabled = 2 and fas_secure_enabled = 3), NDS sends encrypted information to FAS.

http://fasremotefqdn:fasport/faspath?fas=[aes-256-cbc data]&iv=[random initialisation vector] (level 2)

https://fasremotefqdn:fasport/faspath?fas=[aes-256-cbc data]&iv=[random initialisation vector] (level 3)

> It is the responsibility of FAS to decrypt the aes-256-cbc data it receives, using the pre shared faskey and the random initialisation vector.

The decrypted string received by FAS will be of the form: [varname1]=[var1], [varname2]=[var2], ….. etc. (the separator being comma-space).

eg *clientip=192.168.8.23, clientmac=04:15:52:6a:e4:ad, tok=770bfe05, originurl=…..*

Variables sent by NDS in the encrypted string in NDS v4.0.0 and above are as follows:

clientip clientmac gatewayname tok gatewayaddress authdir originurl clientif

Where: tok is the client token

gatewayaddress is authentication address of NDS ie [nds_ip]:[nds_port]

authdir is the NDS virtual authentication directory

clientif is the interface string identifying the interface the client is connected to in the form of: [local interface] [meshnode mac] [local mesh interface]

Future versions of NDS may send additional variables and the order of the variables in the decrypted string may also vary, so it is the responsiblity of FAS to parse the decrypted string for the variables it requires.

5.5 Network Zones - Determining the Interface the Client is Connected To

The Network coverage of a Captive Portal can take many forms, from a single SSID through to an extensive mesh network.

Using FAS, it is quite simple to dynamically adapt the Client Login page depending on the Network Zone a client is connected to. NDS can determine the local interface or 802.11s mesh network node a client is using. A simple lookup table can then be included in a custom FAS, relating interfaces or mesh nodes to sensibly named coverage zones.

A very simple example would be a captive portal set up with a wireless network for "Staff", another for "Guests" and office machines connected via ethernet.

- Ethernet connected office machines would gain access by simply clicking "Continue".
- Staff mobiles connect to the Staff WiFi using a standard access code then clicking "Continue".
- Guests connect to the open Guest Wifi and are required to enter details such as Name, email address etc.

NDS is aware of the interface or mesh node a client is using.

For a FAS using *fas_secure_enabled = 2*, an additional variable, clientif, is sent to the FAS in the encrypted query string (local or remote FAS).

For all other levels of fas_secure_enabled, PreAuth and BinAuth, the library utility "get_client_interface" is required to be used by the relevant script (local FAS only).

Working examples can be found in the included scripts:

- fas-aes.php
- login.sh
- demo-preauth.sh
- demo-preauth-remote-image.sh

For details of the clientif variable and how to use get_client_interface, see the section **Library Utilities**.

5.6 After Successful Verification by FAS

If the client is successfully verified by the FAS, FAS will return the unique token, or its hashed equivalent to NDS to finally allow the client access to the Internet.

5.7 Post FAS processing

Once the client has been authenticated by the FAS, NDS must then be informed to allow the client to have access to the Internet.

The method of achieving this depends upon the fas_secure_enabled level.

5.7.1 Authentication Method for fas_secure_enabled levels 0,1 and 2

Once FAS has verified the client credentials, authentication is achieved by accessing NDS at a special virtual URL.

This virtual URL is of the form:

http://[nds_ip]:[nds_port]/[authdir]/?tok=[token]&redir=[landing_page_url]

This is most commonly achieved using an html form of method GET. The parameter redir can be the client's originally requested URL sent by NDS, or more usefully, the URL of a suitable landing page.

Be aware that many client CPD processes will **automatically close** the landing page as soon as Internet access is detected.

5.7.2 Authentication Method for fas_secure_enabled level 3 (Authmon Daemon)

When fas_secure_enabled level 3 is used (https protocol), post verification authentication is achieved by the openNDS Authmon daemon.

Authmon is started by openNDS to facilitate NAT traversal and allow a remote https FAS to communicate with the local openNDS.

FAS will deposit client authentication variables for the Authmon daemon to use for the authentication process. These variables are as follows:

- clientip: The ip address of the client to be authenticated
- sessionlength: length of session - minutes
- uploadrate: maximum allowed upload data rate - kbits/sec
- downloadrate: maximum allowed download data rate - kbits/sec
- uploadquota: allowed upload data quota - kBytes
- downloadquota: allowed download data quota - kBytes
- custom: A custom data string that will be sent to BinAuth

Details can be found in the example script fas-aes-https.php

5.8 BinAuth Post FAS Processing

As BinAuth can be enabled at the same time as FAS, a BinAuth script may be used for custom post FAS processing. (see BinAuth).

The example BinAuth script, binauth_log.sh, is designed to locally log details of each client authentication and receives client data including the token, ipaddress and macaddress. In addition it receives the custom data string sent from FAS.

5.9 Manual Access of NDS Virtual URL

If the user of an already authenticated client device manually accesses the NDS Virtual URL, they will be redirected back to FAS with the "status" query string.

This will be of the form:

http://fasremoteip:fasport/faspath?clientip=[clientip]&gatewayname=[gatewayname]&status=authenticated

FAS should then serve a suitable error page informing the client user that they are already logged in.

5.10 Running FAS on your openNDS router

FAS has been tested using uhttpd, lighttpd, ngnix, apache and libmicrohttpd.

Running on OpenWrt with uhttpd/PHP:

A FAS service may run quite well on uhttpd (the web server that serves Luci) on an OpenWrt supported device with 8MB flash and 32MB ram but shortage of ram will be an issue if more than two or three clients log in at the same time.

For this reason a device with a **minimum** of 8MB flash and 64MB ram is recommended.

A device with 16MB flash or greater and 128MB ram or greater is recommended as a target for serious development.

Although port 80 is the default for uhttpd, it is reserved for Captive Portal Detection so cannot be used for FAS. uhttpd can however be configured to operate on more than one port.

We will use port 2080 in this example.

Install the module php7-cgi. Further modules may be required depending on your requirements.

To enable FAS with php in uhttpd you must add the lines:

```
list listen_http 0.0.0.0:2080
list interpreter ".php=/usr/bin/php-cgi"
```

to the /etc/config/uhttpd file in the config uhttpd 'main' or first section.

The two important NDS options to set will be:

1. fasport. We will use port 2080 for uhttpd
2. faspath. Set to, for example, /myfas/fas.php, your FAS files being placed in /www/myfas/

5.11 Using a Shared Hosting Server for a Remote FAS

A typical Internet hosted **shared** server will be set up to serve multiple domain names.

To access yours, it is important to configure the two options:

fasremoteip = the **ip address** of the remote server

AND

fasremotefqdn = the **Fully Qualified Domain name** of the remote server

5.12 Using the FAS Example Scripts (fas-hid, fas-aes.php and fas-aes-https.php)

These three, fully functional, example FAS scripts are included in the package install and can be found in the /etc/opennds folder. To function, they need to be copied to the web root or a folder in the web root of your FAS http/php server.

5.12.1 fas-hid.php

You can run the FAS example script, fas-hid.php, locally on the same device that is running NDS, or remotely on an Internet based FAS server.

The use of http protocol is enforced. fas-hid is specifically targeted at local systems with insufficient resources to run PHP services, yet facilitate remote FAS support without exposing the client token or requiring the remote FAS to somehow access the local ndsctl.

If run locally on the NDS device, a minimum of 64MB of ram may be sufficient, but 128MB or more is recommended.

If run on a remote FAS server, a minimum of 32MB of ram on the local device may be sufficient, but 64MB or more is recommended.

5.12.2 fas-aes.php

You can run the FAS example script, fas-aes.php, locally on the same device that is running NDS (A minimum of 64MB of ram may be sufficient, but 128MB is recommended), or remotely on an Internet based FAS server. The use of http protocol is enforced.

5.12.3 fas-aes-https.php

You can run the FAS example script, fas-aes-https.php, remotely on an Internet based https FAS server. The use of https protocol is enforced.

On the openNDS device, a minimum of 64MB of ram may be sufficient, but 128MB is recommended.

5.12.4 Example Script File fas-aes.php

Http protocol is enforced.

Assuming you have installed your web server of choice, configured it for port 2080 and added PHP support using the package php7-cgi, you can do the following.

(Under other operating systems you may need to edit the opennds.conf file in /etc/opennds instead, but the process is very similar.)

- Install the packages php7-cli and php7-mod-openssl
- Create a folder for the FAS script eg: /[server-web-root]/nds/ on the Internet FAS server
- Place the file fas-aes.php in /[server-web-root]/nds/

  (You can find it in the /etc/opennds directory.)
- Edit the file /etc/config/opennds

  adding the lines:

  ```
  option fasport '2080'
  option faspath '/nds/fas-aes.php'
  option fas_secure_enabled '2'
  option faskey '1234567890'
  ```
- Restart NDS using the command `service opennds restart`

5.12.5 Example Script File fas-aes-https.php

Https protocol is enforced.

Assuming you have access to an Internet based https web server you can do the following.

(Under other operating systems you may need to edit the opennds.conf file in /etc/opennds instead, but the process is very similar.)

- Install the packages php7-cli and php7-mod-openssl on your NDS router
- Create a folder for the FAS script eg: /[server-web-root]/nds/ on the Internet FAS server
- Place the file fas-aes.php in /[server-web-root]/nds/

  (You can find it in the /etc/opennds directory.)
- Edit the file /etc/config/opennds

  adding the lines:

  `option fasport '443'` (or the actual port in use if different)

  `option faspath '/nds/fas-aes-https.php'`

  `option fas_secure_enabled '3'`

  `option faskey '1234567890'`

  `option fasremoteip '46.32.240.41'` (change this to the actual ip address of the remote server)

  `option fasremotefqdn 'blue-wave.net'` (change this to the actual FQDN of the remote server)
- Restart NDS using the command `service opennds restart`

5.12.6 Example Script File fas-hid.php

Http protocol is enforced.

fas-hid.php can be configured to be run locally or remotely in the same basic way as fas-aes.

However it is targeted for use on devices with limited resources as it does not require openNDS to have locally installed php-cli modules.

It uses fas_secure_enabled level 1, but sends a digest of the client token to the remote FAS. The digest is created using faskey, so the client token is not exposed.

The fas-hid.php script then uses this digest along with the pre shared faskey to request authentication by openNDS, thus mitigating any requirement for remotely accessing ndsctl that otherwise would be required.

Assuming you have access to an Internet based http web server you can do the following.

(Under other operating systems you may need to edit the opennds.conf file in /etc/opennds instead, but the process is very similar.)

- Create a folder for the FAS script eg: /[server-web-root]/nds/ on the Internet FAS server
- Place the file fas-hid.php in /[server-web-root]/nds/

  (You can find it in the /etc/opennds directory.)
- Edit the file /etc/config/opennds

  adding the lines:

  `option fasport '80'` (or the actual port in use if different)

  `option faspath '/nds/fas-hid.php'`

  `option fas_secure_enabled '1'`

  `option faskey '1234567890'`

  `option fasremoteip '46.32.240.41'` (change this to the actual ip address of the remote server)

  `option fasremotefqdn 'blue-wave.net'` (change this to the actual FQDN of the remote server)
- Restart NDS using the command `service opennds restart`

5.13 Changing faskey

The value of option faskey should of course be changed, but must also be pre-shared with FAS by editing the example or your own script to match the new value.

CHAPTER 6

PreAuth Option

6.1 Overview

PreAuth is an implementation of FAS *without the resource utilisation of a separate web server*, particularly useful for legacy devices with limited flash and RAM capacity.

PreAuth is a pre-authentication process that enables NDS to directly serve dynamic web content generated by a script or executable program.

Note: From version 3.3.1 onwards, a PreAuth login script is pre-installed. This generates a page asking for username and email address. Logins are recorded in a log file. It is enabled by un-commenting just 3 lines in the config file. **From version 4.0.2 onwards** it is enabled by a single line in the config file that overrides any other FAS configuration. **From version 4.3.0 onwards** it is enabled by setting config option login_option_enabled to "1"

PreAuth is enabled by configuring NDS FAS to point to a virtual URL in the NDS web root instead of an independent FAS server. The location of the PreAuth script or program is provided in the config file.

The PreAuth script can be a shell script or any other script type that an interpreter is available for (for example, PHP-cli, Python etc.).

A PreAuth program could be, for example, a compiled program written in C or any other language that has a compiler available for the platform.

The PreAuth script or program will parse the url encoded command line (query string) passed to it and output html depending on the contents of the query string it receives from NDS. In turn, NDS will serve this html to the client device that is attempting to access the Internet.

6.2 Selecting Pre-Installed Username / Email Login Script (v4.3.0 onwards)

The default preauth login script is installed as part of the openNDS package providing username/emailaddress login as an alternative to the basic splash page.

It is enabled by setting in config:

option login_option_enabled = '1'

No additional FAS or PreAuth config settings are required.

6.3 Using PreAuth version 4.0.2 onwards

From version 4.0.2 onwards, PreAuth is enabled with a single configuration option:

- **preauth**. This the path to the PreAuth script or executable.

This option overrides any other FAS configuration and takes the form of the path to the PreAuth script. The path to the preinstalled login script is included in option preauth in the default config files, for example in OpenWrt:

#option preauth '/usr/lib/opennds/login.sh'

The "#" symbol means the line is commented. To activate, remove the "#". save and restart opennds.

6.4 Using PreAuth version 3.3.1 to version 4.0.1

From version 3.3.1 to version 4.0.1, PreAuth is set up using the standard NDS configuration for FAS (See the **Forwarding Authentication Service (FAS)** section of this documentation).

In addition a single PreAuth configuration option is required to inform NDS of the location of the PreAuth script or program.

In summary, the following configuration options should be set:

1. **fasport**. This enables FAS and *must* be set to the same value as the gateway port.
2. **faspath**. This *must* be set to the PreAuth virtual url, "/opennds_preauth/" by default.
3. **preauth**. This the path to the PreAuth script.

The remaining FAS configuration options must be left unset at the default values.

ie:

1. **fasremoteip**. Not set (defaults to the gateway ip address).
2. **fas_secure_enable**. Not set (defaults to enabled).

Note: From version 3.3.1 onwards, the example PreAuth script is preinstalled.

6.5 Enabling the Preinstalled Login Script (v3.3.1 to 4.0.1)

On Openwrt, edit (to uncomment) following lines in the /etc/config/opennds file:

#option fasport '2050'

#option faspath '/opennds_preauth/'

#option preauth '/usr/lib/opennds/login.sh'

To read:

option fasport '2050'

option faspath '/opennds_preauth/'

option preauth '/usr/lib/opennds/login.sh'

6.6 Enabling the Preinstalled Login Script (v4.0.2 onwards)

On Openwrt, edit (to uncomment) following line in the /etc/config/opennds file:

#option preauth '/usr/lib/opennds/login.sh'

To read:

option preauth '/usr/lib/opennds/login.sh'

For other operating systems edit the equivalent lines in the /etc/opennds/opennds.conf file

After making the change, save the file and restart the router.

6.7 What Does the Example Login Script Do?

This example shell script generates html output for NDS to serve as a dynamic splash page.

The example asks the client user to enter their name and email address. On entering this information the client user then clicks or taps "Continue".

The script then generates html code to send to NDS to serve a second "Thankyou" page and creates a log entry (/tmp/ndslog.log), recording the client authentication details.

On tapping "Continue" for the second time, the client user is given access to the Internet.

This is a simple example of a script to demonstrate how to use PreAuth as a built in FAS. The script could of course ask for any response from the client and conduct its own authentication procedures - entirely at the discretion of the person setting up their own captive portal functionality.

6.8 PreAuth with Remote Images

An additional example PreAuth script, demo-preauth-remote-image.sh, is available in the source code:

https://github.com/opennds/opennds/archive/master.zip

and extracting from the folder:

"forward_authentication_service/PreAuth/"

This is an enhancement of the preinstalled login.sh, giving an example of how to display images pulled in from remote web servers, both http and https.

The example displays the opennds avatar image dynamically retreived from Github.

6.9 Writing A Preauth Script

A Preauth script can be written as a shell script or any other language that the system has an interpreter for. It could also be a complied program.

NDS calls the preauth script with a command line equivalent to an html query string but with ", " (comma space) in place of "&" (ampersand).

Full details are included in the example script demo-preauth.sh available by downloading the opennds zip file from

https://github.com/opennds/opennds/

and extracting from the folder

"forward_authentication_service/PreAuth/"

6.10 Defining and Using Variables

The query string is sent to us from NDS in a urlencoded form, so we must decode it here so we can parse it. In a shell script we would use the code:

```
query=$(printf "${query_enc//%/\\x}")
```

In the example script we want to ask the client user for their username and email address.

We could ask for anything we like and add our own variables to the html forms we generate.

If we want to show a sequence of forms or information pages we can do this easily.

To return to the script and show additional pages, the form action must be set to:

```
<form action=\"/opennds_preauth/\" method=\"get\">
```

Note: In a shell script, quotes (") must be escaped with the

```
"\"
```

character.

Any variables we need to preserve and pass back to ourselves or NDS must be added to the form as hidden:

```
<input type=\"hidden\" name=......
```

Such variables will appear in the query string when NDS re-calls this script.

We can then parse for them again.

When the logic of this script decides we should allow the client to access the Internet we inform NDS with a final page displaying a continue button with the form action set to:

```
"<form action=\"/opennds_auth/\" method=\"get\">"
```

We must also send NDS the client token as a hidden variable, but first we must obtain the token from ndsctl using a suitable command such as:

```
tok="$(ndsctl json $clientip | grep token | cut -c 10- | cut -c -8)"
```

In a similar manner we can obtain any client or NDS information that ndsctl provides.

The query string NDS sends to us will always be of the following form (with a "comma space" separator):

```
?clientip=[clientipaddress], gatewayname=[gatewayname],  redir=[originalurl],␣
↪var4=[data], var5=[data], var6......
```

The first three variables will be clientip, gatewayname and redir

We have chosen to name redir as $requested here as it is actually the originally requested url.

There is one exception to this. If the client presses "back" on their browser NDS detects this and tells us by returning status=authenticated instead of redir=[originalurl]

If we detect this we show a page telling the client they are already logged in.

Additional variables returned by NDS will be those we define here and send to NDS via an html form method=get

See the example script which uses $username and $emailaddr

There is no limit to the number of variables we can define dynamically as long as the query string does not exceed 2048 bytes.

The query string will be truncated if it does exceed this length.

6.11 Displaying Remote Banner Images

A modified version of the Username/Email-address login script is available that demonstrates how to display remotely hosted images on its login pages.

This additional example PreAuth script, demo-preauth-remote-image.sh, is available in the source code:

https://github.com/opennds/opennds/archive/master.zip

and extracting from the folder:

"forward_authentication_service/PreAuth/"

This is an enhancement of the preinstalled login.sh, giving an example of how to display images pulled in from remote web servers, both http and https.

The example displays the openNDS avatar image dynamically retrieved from Github.

CHAPTER 7

BinAuth Option

7.1 Overview

BinAuth provides a method of running a post authentication script or extension program. BinAuth is ALWAYS local to NDS and as such will have access to all the resources of the local system.

BinAuth works with, but does not require FAS and in a simple system can be used to provide site-wide username/password access.

BinAuth is available when FAS is used at all levels of fas_secure_enabled (0, 1, 2 and 3).

With FAS, a custom variable is forwarded to BinAuth This can contain an embedded payload of custom data defined by the FAS. As FAS is typically remote from the NDS router, this provides a link to the local system.

BinAuth has the means to set session timeout, data rate and data volume quotas on a client by client basis.

BinAuth is called by NDS at the following times:

- After the client CPD browser makes an authentication request to NDS
- After the client device is granted Internet access by NDS
- After the client is deauthenticated by request
- After the client idle timeout interval has expired
- After the client session timeout interval has expired
- After a data upload or download quota has been exceeded
- After the client is authenticated by ndsctl command
- After the client is deauthenticated by ndsctl command
- After NDS has received a shutdown command

7.2 BinAuth Command Line Arguments

When OpenNDS calls the configured BinAuth script, it sends a set of command line arguments depending on the reason for the call.

7.2.1 BinAuth Command Methods

The first argument, arg[1], is always the "method".

The method will be set to one of the following values:

- "**auth_client**" This is a request for authentication by the client.
- "**client_auth**" This is an acknowledgement of successful authentication by NDS.
- "**client_deauth**" This is an acknowledgement that the client has been deauthenticated by NDS.
- "**idle_deauth**" - NDS has deauthenticated the client because the idle timeout duration has been exceeded.
- "**timeout_deauth**" - NDS has deauthenticated the client because the session length duration has been exceeded.
- "**downquota_deauth**" - NDS has deauthenticated the client because the client's download quota has been exceeded
- "**upquota_deauth**" - NDS has deauthenticated the client because the client's upload quota has been exceeded
- "**ndsctl_auth**" - NDS has authorised the client because of an ndsctl command (most commonly sent by the NDS AuthMon daemon).
- "**ndsctl_deauth**" - NDS has deauthenticated the client because of an ndsctl command.
- "**shutdown_deauth**" - NDS has deauthenticated the client because it received a shutdown command.

Additional arguments depend on the method type:

7.2.2 Method auth_client

The first argument is auth_client and the following arguments are set to:

- arg[2] = client_mac
- arg[3] = username
- arg[4] = password
- arg[5] = url-escaped redir variable (the URL originally requested by the client.
- arg[6] = url-escaped client user agent string
- arg[7] = client_ip
- arg[8] = client_token
- arg[9] = url-escaped custom variable string

7.2.3 Method ndsctl_auth

The first argument is ndsctl_auth and the following arguments are set to:

- arg[2] = client_mac
- arg[3] = bytes_incoming (set to 0, reserved for future use)
- arg[4] = bytes_outgoing (set to 0, reserved for future use)
- arg[5] = session_start - the session start time
- arg[6] = session_end - the session end time
- arg[7] = client_token
- arg[8] = url-escaped custom variable string

7.2.4 All Other Methods

When the first argument is other than auth_client or ndsctl_auth, the following arguments are set to:

- arg[2] = client_mac
- arg[3] = bytes_incoming (total incoming bytes for client)
- arg[4] = bytes_outgoing (total incoming bytes for client)
- arg[5] = session_start - the session start time
- arg[6] = session_end - the session end time
- arg[7] = client_token

7.3 Example BinAuth Scripts

Two example BinAuth scripts are included in the source files available for download at: https://github.com/opennds/opennds/releases

Both of them are preinstalled and ready to be enabled in the config file.

In addition, the files can be extracted from the downloaded release archive file and reside in the folder:

*/opennds-[*version*]/forward_authentication_service/binauth*

7.4 Example 1 - Sitewide Username/Password

This example is a script designed to be used with or without FAS and provides site wide Username/Password login for groups of users, in this case “staff”, “guest” and “member” with corresponding sets of credentials. If used without FAS, a special html splash page must be installed, otherwise FAS must forward the required username and password variables.

7.5 Manual Installation (Example 1)

The binauth_sitewide example is pre-installed. However, a manual installation is described here by way of example to aid developers in understanding the procedure required for installing their own scripts. The binauth_sitewide script actually has three components, the binauth script itself, an associated html file and a user database file.

- binauth_sitewide.sh
- splash_sitewide.html
- userlist.dat

The file binauth_sitewide.sh should be copied to a suitable location on the NDS router, eg */usr/lib/opennds/*

The file splash_sitewide.html should be copied to */etc/opennds/htdocs/*

The file userlist.dat should be copied to */etc/opennds/*

Assuming FAS is not being used, NDS is then configured by setting the BinAuth and SplashPage options in the config file (/etc/config/opennds on Openwrt, or /etc/opennds/opennds.conf on other operating systems.

On OpenWrt this is most easily accomplished by issuing the following commands:

uci set opennds.@opennds[0].splashpage='splash_sitewide.html'

uci set opennds.@opennds[0].binauth='/usr/lib/opennds/binauth_sitewide.sh'

uci commit opennds

The script file must be executable and is flagged as such in the source archive. If necessary set using the command:

chmod u+x /usr/lib/opennds/binauth_sitewide.sh

This script is then activated with the command:

service opennds restart

7.6 Example 2 - Local NDS Access Log

This example is a script designed to be used with or without FAS and provides local NDS logging. FAS is often remote from the NDS router and this script provides a simple method of interacting directly with the local NDS. FAS can send custom data to Binauth as a payload in the custom variable parameter that is relayed to BinAuth by NDS.

The log file is stored by default in the /tmp/ndslog/ directory. This works for many operating systems including OpenWrt.

The location however must be changed on some operating systems, such as Debian and its variants (eg Raspbian). Here a default location of /run/ndslog/ works well.

The log location is simply changed by editing variables at the beginning of the script file.

Free space checking is done and if the log file becomes too large, logging ceases and an error is sent to syslog.

Log files do not persist through a reboot so it would be sensible to change the location of the log file to a USB stick for example.

7.7 Using Example 2

The binauth_log example is pre-installed.

This script has a single component, the shell script.

- binauth_log.sh

The file binauth_log.sh is preinstalled in the /usr/lib/opennds directory.

This is enabled by setting the BinAuth option in the config file (/etc/config/opennds on Openwrt, or /etc/opennds/opennds.conf on other operating systems.

This script is then activated with the command:

service opennds restart

CHAPTER 8

Library Utilities

8.1 Overview

A number of library utilities are included. These may be used by NDS itself, FAS, Preauth and BinAuth. These may in the future, be enhanced, have additional functionality added.

By default, library utilities will be installed in the folder

`/usr/lib/opennds/`

8.2 List of Library Utilities

8.2.1 get_client_token.sh

This utility allows the unique token of a client to be determined from the client ip address.

It can be used in BinAuth, PreAuth and local FAS scripts.

Usage: get_client_token.sh [clientip]

Returns: [client token]

Where: [client token] is the unique client token string.

8.2.2 get_client_interface.sh

This utility allows the interface a client is using to be determined from the client mac address.

It is used by NDS when fas secure level 2 is set. Its output is sent to FAS in the encrypted query string as the variable "clientif"

Usage: get_client_interface.sh [clientmac]

Returns: [local_interface] [meshnode_mac] [local_mesh_interface]

Where:

[local_interface] is the local interface the client is using.

[meshnode_mac] is the mac address of the 802.11s meshnode the client is using (null if mesh not present).

[local_mesh_interface] is the local 802.11s interface the client is using (null if mesh not present).

8.2.3 unescape.sh

This utility allows an input string to be unescaped. It currently only supports url-decoding.

It can be used by NDS as the unescape callback for libmicrohttpd.

To enable, set the unescape_callback_enabled option to "1"

To disable, set the unescape_callback_enabled option to "0"

The default is disabled (use internal MHD unescape)

eg In the OpenWrt configuration file

```
option unescape_callback_enabled '0'
```

Usage: unescape.sh [-option] [escapedstring]

Returns: [unescapedstring]

Where:

[-option] is unescape type, currently -url only

CHAPTER 9

Data Quotas and Traffic Shaping

9.1 Data volume and Rate Quotas

openNDS (NDS) has built in *Data Volume* and *Data Rate* quota support.

Data volume and data rate quotas can be set globally in the config file.

The global values can be overridden on a client by client basis as required.

9.1.1 Data Volume Quota

If a client exceeds the global data volume quota, or the individual client quota, that client will be forced out by deauthentication. To continue, the client must re-authenticate.

Configuring Data Volume Quotas

Global volume quotas are configured in the config file.

Example UCI configuration options:

```
# If the client data quota exceeds the value set here, the client will be forced out
# Values are in kB
# If set to 0, there is no limit
# Integer values only
#
option uploadquota '0'
option downloadquota '0'
```

Note: upload means to the Internet, download means from the Internet

Quotas for individual clients will override configured global values and are set either by BinAuth or the Authmon Daemon (fas_secure_enable level 3) - see example BinAuth script (binauth_log.sh) and example FAS script (fas-aes-https.php).

9.1.2 Data Rate Quota

A rate quota is a moving average data rate.

The average is calculated over a configured time window.

If a client exceeds the global data rate quota, or the individual client quota, that client will be blocked for a configured interval before being allowed to continue automatically.

As the Data Rate Quota is a moving average, clients are able to transfer data at the maximum rate the router can achieve for short bursts.

Data Rate Quotas are ideal for allowing opening of web pages and emails etc in the fastest way possible, yet preventing an individual client from monopolizing all the available bandwidth by streaming or transferring large files.

Configuring Data Rate Quotas

Global Data Rate quotas are configured in the config file.

```
# Note: upload means to the Internet, download means from the Internet
# Defaults 0
# Integer values only
#
# If the client average data rate exceeds the value set here, the client will be␣
↪blocked
# Values are in kb/s
# If set to 0, there is no limit
#
# Quotas and rates can also be set by FAS via Authmon Daemon, by BinAuth, and by␣
↪ndsctl auth.
# Values set by these methods, will be override values set in this config file.
#
option uploadrate '0'
option downloadrate '0'
################################################################################
↪#####
# The client data rate is calculated using a moving average.
# This allows clients to burst at maximum possible rate, only blocking if the moving␣
↪average
# exceeds the specified upload or download rate.
# The moving average window size is equal to ratecheckwindow times checkinterval␣
↪(seconds)
# Default 2
option ratecheckwindow '2'
```

Note: upload means to the Internet, download means from the Internet

Data Rate Quotas for individual clients will override configured global values and are set either by BinAuth or the Authmon Daemon (fas_secure_enable level 3) - see example BinAuth script (binauth_log.sh) and example FAS script (fas-aes-https.php).

9.2 Traffic Shaping

openNDS (NDS) supports Traffic Shaping (Bandwidth Limiting) using the SQM - Smart Queue Management (sqm-scripts) package, available for OpenWrt and generic Linux.

https://github.com/tohojo/sqm-scripts

SQM does efficient bandwidth control, independently for both upload and download, on an IP connection basis. This ideal for enforcing a fair usage policy on a typical Captive Portal implementation.

In addition the Queue management SQM provides, results in significantly improved WiFi performance, particularly on the modern low cost WiFi routers available on the market today.

Finally, SQM controls quality of service (QOS), allowing priority for real time protocols such a VOIP.

Overall, SQM can enhance significantly the experience of clients using your Captive Portal, whilst ensuring a single client is unlikely to dominate the available Internet service at the expense of others.

9.3 Installing SQM

The generic Linux scripts can be downloaded from the link above.

On OpenWrt, SQM can be installed from the LuCi interface or by the following CLI commands on your router:

opkg update

opkg install sqm-scripts

Note: The standard and default SQM installation expects monitoring of the interface connecting to the WAN. What we need is for SQM to monitor the interface NDS is bound to. This of course will be a LAN interface. The default configuration will limit bandwidth from the WAN connection to services on the Internet. Our configuration will limit client bandwidth TO NDS, thus enabling a true fair usage policy.

To prevent confusion it is important to understand that SQM defines "Upload" as traffic "Out" of the interface SQM is monitoring and "Download" as traffic "In" to the SQM interface.

In the default SQM configuration, Upload will mean what is normally accepted, ie traffic to the Internet and Download will mean traffic from the Internet.

In our case however the terms will be reversed!

The default SQM configuration file on OpenWrt is:

```
config queue
    option enabled '0'
    option interface 'eth1'
    option download '85000'
    option upload '10000'
    option qdisc 'fq_codel'
    option script 'simple.qos'
    option qdisc_advanced '0'
    option ingress_ecn 'ECN'
    option egress_ecn 'ECN'
    option qdisc_really_really_advanced '0'
    option itarget 'auto'
    option etarget 'auto'
    option linklayer 'none'
```

For simple rate limiting, we are interested in setting the desired interface and the download/upload rates.

We may also want to optimize for the type of Internet feed and change the qdisc.

A typical Internet feed could range from a high speed fiber optic connection through fast VDSL to a fairly poor ADSL connection and configured rates should be carefully chosen when setting up your Captive Portal.

A typical Captive Portal however will be providing free Internet access to customers and guests at a business or venue, using their mobile devices.

A good compromise for a business or venue might be a download rate from the Internet of ~3000 Kb/s and an upload rate to the Internet of ~1000 Kb/s will be adequate, allowing for example, a client to stream a YouTube video, yet have minimal effect on other clients browsing the Internet or downloading their emails. Obviously the values for upload and download rates for best overall performance depend on many factors and are best determined by trial and error.

If we assume we have NDS bound to interface br-lan and we have a VDSL connection, a good working setup for SQM will be as follows:

- *Rate to* Internet 1000 Kb/s (but note this is from the perspective of the interface SQM is monitoring, so this means DOWNLOAD from the client).
- *Rate from* Internet 3000 Kb/s (also note this is from the perspective of the interface SQM is monitoring, so is means UPLOAD to the client).
- *VDSL* connection (usually an ethernet like connection)
- *NDS* bound to br-lan

We will configure this by issuing the following commands:

Note the reversed "upload" and "download" values.

```
uci set sqm.@queue[0].interface='br-lan'

uci set sqm.@queue[0].download='1000'

uci set sqm.@queue[0].upload='3000'

uci set sqm.@queue[0].linklayer='ethernet'

uci set sqm.@queue[0].overhead='22'

uci set sqm.@queue[0].qdisc='cake'

uci set sqm.@queue[0].script='piece_of_cake.qos'

uci set sqm.@queue[0].enabled='1'

uci commit sqm

service sqm restart
```

Replace the linklayer and overhead values to match your Internet feed.

The following table lists LinkLayer types and Overhead for common feed types:

Connection Type	LinkLayer	Overhead
Fibre/Cable	Ethernet	18
VDSL2	Ethernet	22
Ethernet	Ethernet	38
ADSL/DSL	ATM	44

Some broadband providers use variations on the values shown here, contacting them for details sometimes helps but often the request will be "off script" for a typical helpdesk. These table values should give good results regardless. Trial and error and the use of a good speed tester is often the only way forward. A good speed tester web site is http://dslreports.com/speedtest

Further details about SQM can be found at the following links:

https://openwrt.org/docs/guide-user/network/traffic-shaping/sqm

https://openwrt.org/docs/guide-user/network/traffic-shaping/sqm-details

CHAPTER 10

Using ndsctl

A openNDS install includes ndsctl, a separate application which provides some control over a running openNDS process by communicating with it over a unix socket. Some command line options:

- To print to stdout some information about your openNDS process:

```
/usr/bin/ndsctl status
```

- To print to stdout the list of clients in human readable format:

```
/usr/bin/ndsctl clients
```

- To print to stdout the list of clients and trusted devices in json format:

```
/usr/bin/ndsctl json
```

- To print to stdout the details of a particular client in json format (This is particularly useful if called from a FAS or Binauth script.):

```
/usr/bin/ndsctl json [mac|ip|token]
```

- To block a MAC address, when the MAC mechanism is block:

```
/usr/bin/ndsctl block MAC
```

- To unblock a MAC address, when the MAC mechanism is block:

```
/usr/bin/ndsctl unblock MAC
```

- To allow a MAC address, when the MAC mechanism is allow:

```
/usr/bin/ndsctl allow MAC
```

- To unallow a MAC address, when the MAC mechanism is allow:

```
/usr/bin/ndsctl unallow MAC
```

- To deauthenticate a currently authenticated user given their IP or MAC address:

```
/usr/bin/ndsctl deauth IP|MAC
```

- To set the verbosity of logged messages to n:

```
/usr/bin/ndsctl debuglevel n
```

- debuglevel 0 : Silent (only LOG_ERR and LOG_EMERG messages will be seen, otherwise there will be no logging.)
- debuglevel 1 : LOG_ERR, LOG_EMERG, LOG_WARNING and LOG_NOTICE (this is the default level).
- debuglevel 2 : debuglevel 1 + LOG_INFO
- debuglevel 3 : debuglevel 2 + LOG_DEBUG

All other levels are undefined and will result in debug level 3 being set.

For more options, run ndsctl -h. (Note that if you want the effect of ndsctl commands to to persist across openNDS restarts, you have to edit the configuration file.)

CHAPTER 11

Customising openNDS

After initial installation, openNDS (NDS) should be working in its most basic mode and client Captive Portal Detection (CPD) should pop up the default splash page.

Before attempting to customise NDS you should ensure it is working in this basic mode before you start.

NDS reads its configuration file when it starts up but the location of this file varies depending on the operating system.

As NDS is a package that requires hardware configured as an IP router, perhaps the most common installation is using OpenWrt. However NDS can be compiled to run on most Linux distributions, the most common being Debian or one of its popular variants (eg Raspbian).

If NDS is working in the default, post installation mode, then you will have met the NDS dependencies and can now move on to your own customisation.

11.1 Rules for Customised Splash Pages

It should be noted when designing a custom splash page that for security reasons many client device CPD implementations:

- Immediately close the browser when the client has authenticated.
- Prohibit the use of href links.
- Prohibit downloading of external files (including .css and .js, even if they are allowed in NDS firewall settings).
- Prohibit the execution of javascript.

11.2 The Configuration File

In OpenWrt, or operating systems supporting UCI (such as LEDE) the configuration is kept in the file:

```
/etc/config/opennds
```

In other operating systems the configuration is kept in the file:

```
/etc/opennds/opennds.conf
```

Both of these files contain a full list of options and can be edited directly. A restart of NDS is required for any changes to take effect.

In the case of OpenWrt though, once you are confident in your configuration requirements you can use UCI to read and set any of the configuration options using simple commands, making this very convenient if making changes from scripts, such as those you may write to use with Binauth and FAS.

For example, to list the full configuration, at the command line type:

```
uci show opennds
```

To display the Gateway Name, type:

```
uci get opennds.@opennds[0].gatewayname
```

To set the Gateway Name to a new value, type:

```
uci set opennds.@opennds[0].gatewayname='my new gateway'
```

To add a new firewall rule allowing access to another service running on port 8888 on the router, type:

```
uci add_list opennds.@opennds[0].users_to_router='allow
tcp port 8888'
```

Finally you must tell UCI to commit your changes to the configuration file:

```
uci commit opennds
```

11.3 The Default Click and Go Splash Page

Enabled by setting option login_option_enabled = "0" (default) The default default splash page can be found at:

```
/etc/opennds/htdocs/splash.html
```

When the splash page is served, the following variables in the page are replaced by their values:

- *$gatewayname* The value of GatewayName as set in opennds.conf.
- *$authtarget* A URL which encodes a unique token and the URL of the user's original web request. If opennds receives a request at this URL, it completes the authentication process for the client and replies to the request with a "302 Found" to the encoded originally requested URL.

  It should be noted however that, depending on vendor, the client's built in CPD may not respond to simple html links.

  An href link example that my prove to be problematical:

  ```
  <a href="$authtarget">Enter</a>
  ```

  (You should instead use a GET-method HTML form to send this information to the opennds server; see below.)

- *$tok*, *$redir*, *$authaction*, and *$denyaction* are available and should be used to write the splash page to use a GET-method HTML form instead of using $authtarget as the value of an href attribute to communicate with the opennds server.

$authaction and *$denyaction* are virtual urls used to inform NDS that a client should be authenticated or deauthenticated and are of the form:

http://gatewayaddress:gatewayport/openndsauth/

and

http://gatewayaddress:gatewayport/openndsdeny/

A simple example of a GET-method form:

```
<form method='GET' action='$authaction'>
  <input type='hidden' name='tok' value='$tok'>
  <input type='hidden' name='redir' value='$redir'>
  <input type='submit' value='Click Here to Enter'>
</form>
```

- *$clientip*, *$clientmac* and *$gatewaymac* The respective addresses of the client or gateway. This might be useful in cases where the data needs to be forwarded to some other place by the splash page itself.
- *$nclients* and *$maxclients* User stats. Useful when you need to display something like "n of m users online" on the splash site.
- *$uptime* The time opennds has been running.

  A list of all available variables are included in the splash.html file.

  If the user accesses the virtual url *$authaction* when already authenticated, a status page is shown:

  `/etc/opennds/htdocs/status.html`

  In the status.html file, the same variables as in the splash.html site can be used.

It should be noted when designing a custom splash page that for security reasons many client device CPD implementations:

- Immediately close the browser when the client has authenticated.
- Prohibit the use of href links.
- Prohibit downloading of external files (including .css and .js, even if they are allowed in NDS firewall settings).
- Prohibit the execution of javascript.

Also, note that any images you reference should reside in the subdirectory /etc/opennds/htdocs/images/.

11.4 Dynamic Splash Pages

11.4.1 Pre-Installed User Login Dynamic Splash Page

The pre-installed dynamic splash page is enabled by setting option login_option_enabled = "1".

It generates a login page asking for username and email address. User logins are recorded in the log file /tmp/ndslog.log Details of how the script works are contained in comments in the script itself.

11.4.2 Custom Dynamic Splash Pages

Custom designed dynamically generated splash pages are supported using FAS and PreAuth (such as the included alternative username/email login script).

For details see the FAS and PreAuth chapters.

CHAPTER 12

Frequently Asked Questions

12.1 What's the difference between v0.9, v1, v2, v3, v4 and v5?

v0.9 and v1 are the same codebase with the same feature set. If the documentation says something about v1, this is usually also valid for v0.9.

v2 was developed before version v1 was released. In v2 the http code was replaced by libmicrohttpd and the template engine was rewritten. Many features became defunct because of this procedure.

v3 cleans up the source code and adds three major new features,

- **FAS**

  A forwarding authentication service. FAS supports development of "Credential Verification" running on any dynamic web serving platform, on the same device as openNDS, on another device on the local network, or on an Internet hosted web server.

- **PreAuth**

  An implementation of FAS running on the same device as openNDS and using openNDS's own web server to generate dynamic web pages. Any scripting language or even a compiled application program can be used. This has the advantage of not requiring the resources of a separate web server.

- **BinAuth**

  Enabling an external script to be called for simple username/password authentication as well as doing post authentication processing such as setting session durations. This is similar to the old binvoucher feature, but more flexible.

In addition, in v3, the ClientTimeout setting was split into PreauthIdleTimeout and AuthIdleTimeout and for the ClientForceTimeout setting, SessionTimeout is now used instead.

v4 continues to add enhancements towards improving NDS as a Captive Portal Engine that can be used in the development of custom solutions.

Three major new features are introduced.

- **FAS FQDN**

  Enabling simple configuration for a FAS running on a remote shared web hosting server.

- **FAS secure level 1**

  From v4.3.0 onwards, FAS secure level 1 supports token hashing. This enhances security and mitigates issues accessing ndsctl remotely to obtain the client token. This is particularly useful on legacy router devices with small flash and ram capacity.

- **FAS secure level 2**

  Enabling aes256cbc encryption of NDS data transferred to remote FAS, thus preventing knowledgable client users from bypassing verification. Access to the FAS server using **http** protocol is enforced.

v5 represents the first version of openNDS after forking from NoDogSplash. This split of codebase allows openNDS to continue development as a powerful Captive Portal engine with API to enable sophisticated applications to be developed.

> *NoDogSplash will continue as an optimised, simplistic Captive Portal solution for use on IoT or legacy devices with severe restrictions on available resources.*

Version 5 introduces many enhancements including major additional functionality:

- **FAS secure level 3**

  Enabling *https* access to a remote, Internet based FAS server, ensuring the client device does not receive any security warnings or errors. Access to the FAS server using **https** protocol is enforced.

  Level 3 otherwise functions in the same way as level 2 with aes256cbc encryption of NDS data.

- **Data volume and Rate Quotas**

  Enabling built in *Data Volume* and *Data Rate* quota support. Data volume and data rate quotas can be set globally in the config file. The global values can be overridden on a client by client basis as required.

- **Introduction of library scripts**

  Numerous library scripts are introduced to simplify development of applications.

12.2 Can I update from v0.9 to v1?

Updating to v1.0.0 and v1.0.1, this is a very smooth update with full compatibility.

Updating to 1.0.2 requires iptables v1.4.21 or above.

12.3 Can I update from v0.9/v1 to v2.0.0?

You can, if:

- You don't use BinVoucher
- You have iptables v1.4.21 or above

12.4 Can I update from v0.9/v1/v2 to v3.0.0?

You can, if:

- You don't use BinVoucher
- You have iptables v1.4.21 or above
- You use the new options contained in the version 3 configuration file

12.5 Can I update from v0.9/v1/v2/v3 to v4?

You can, if:

- You don't use BinVoucher
- You have iptables v1.4.21 or above
- You use the new options contained in the version 4 configuration file

12.6 Can I update from v0.9/v1/v2/v3/v4 to v5?

No. Nodogsplash must be uninstalled before installing openNDS.

- Previous implementations of FAS/PreAuth and BinAuth should function without modification.
- v4 NoDogSplash config files will be compatible but must be renamed.
- MHD (libmicrohttpd) versions earlier than 0.9.69 are detected, a warning is given and openNDS terminates. A new config option is provided to force openNDS to use an earlier version and can be enabled at the discretion and risk of the installer.

12.7 How do I manage client data usage?

openNDS (NDS) has built in *Data Volume* and *Data Rate* quota support.

Data volume and data rate quotas can be set globally in the config file.

The global values can be overridden on a client by client basis as required.

12.8 Can I use Traffic Shaping with openNDS?

SQM Scripts (Smart Queue Management), is fully compatible with openNDS and if configured to operate on the openNDS interface (br-lan by default) will provide efficient IP connection based traffic control to ensure fair usage of available bandwidth.

This can be installed as a package on OpenWrt. For other distributions of Linux it is available at: https://github.com/tohojo/sqm-scripts

12.9 Is an *https splash page* supported?

Yes. FAS Secure Level 3 enforces https protocol for the splash login page on an external FAS server.

12.10 Is *https capture* supported?

No. Because all connections would have a critical certificate failure.

> HTTPS web sites are now more or less a standard and to maintain security and user confidence it is essential that captive portals **DO NOT** attempt to capture port 443.
>
> All modern client devices have the built in, industry standard, *Captive Portal Detection (CPD) service*. This is responsible for triggering the captive portal splash/login page.

12.11 What is CPD / Captive Portal Detection?

CPD (Captive Portal Detection) has evolved as an enhancement to the network manager component included with major Operating Systems (Linux, Android, iOS/macOS, Windows).

> Using a pre-defined port 80 web page (which one gets used depends on the vendor) the network manager will detect the presence of a captive portal hotspot and notify the user. In addition, most major browsers now support CPD.

CHAPTER 13

How to Compile openNDS

13.1 Linux/Unix - Compile in Place on Target Hardware

Make sure the development suite for your Linux distribution is installed.

The libmicrohttpd library (MHD) is a dependency of openNDS so compiling and installing this is a prerequisite.

First, create a working directory and "cd" into it.

Next, Download and un-tar the libmicrohttpd source files.

You can find a version number for MHD at https://ftp.gnu.org/gnu/libmicrohttpd/

```
wget https://ftp.gnu.org/gnu/libmicrohttpd/libmicrohttpd-0.9.70.tar.gz
tar  -xf libmicrohttpd-0.9.70.tar.gz
cd libmicrohttpd-0.9.70
```

where "0.9.70" is the MHD version number we are using in this example.

Now configure and compile:

```
./configure
make
sudo make install
sudo ldconfig
cd ..
```

Then proceed to download the opennds source files.

You can find a release version number for openNDS at https://github.com/openNDS/openNDS/releases

```
wget https://codeload.github.com/opennds/opennds/tar.gz/v5.1.0
tar -xf v5.1.0
cd openNDS-5.1.0
make
sudo make install
systemctl enable opennds
```

Where “5.1.0” is the openNDS version we are using in this example.

openNDS should now start automatically at boot time.

It can be manually started, restarted, stopped or disabled with the following commands:

```
systemctl start opennds

systemctl restart opennds

systemctl stop opennds

systemctl disable opennds
```

The status of openNDS can be checked with the following command:

```
ndsctl status
```

On most Linux distributions you can read the system message log with the command:

```
sudo /usr/bin/cat /var/log/messages
```

13.2 OpenWrt Package

The OpenWrt package feed supports cross-compiled openNDS packages for all OpenWrt targets. See the “Installing openNDS” section of this documentation.

13.2.1 Cross Compiling for OpenWrt

You can cross-compile openNDS from source and create your own installable package using the package definition from the feeds package.

```
git clone git://git.openwrt.org/trunk/openwrt.git
cd openwrt
./scripts/feeds update
./scripts/feeds install
./scripts/feeds install opennds
```

Select the appropriate “Target System” and “Target Profile” in the menuconfig menu and build the image:

```
make defconfig
make menuconfig
make
```

CHAPTER 14

Debugging openNDS

14.1 Syslog Logging

openNDS supports four levels of debugging to syslog.

- debuglevel 0 : Silent (only LOG_ERR and LOG_EMERG messages will be seen, otherwise there will be no logging.)
- debuglevel 1 : LOG_ERR, LOG_EMERG, LOG_WARNING and LOG_NOTICE (this is the default level).
- debuglevel 2 : debuglevel 1 + LOG_INFO
- debuglevel 3 : debuglevel 2 + LOG_DEBUG

All other levels are undefined and will result in debug level 3 being set.

To see maximally verbose debugging output from openNDS, set log level to 3.

On OpenWrt, you can use the following commands:

```
uci set opennds.@opennds[0].debuglevel='3'
uci commit opennds
service opennds restart
```

Debug messages are logged to syslog. You can view messages with the logread command.

The default level of logging is 1, and is more appropriate for routine use.

Logging level can also be set using ndsctl.

14.2 Firewall Cleanup

When stopped, openNDS deletes its iptables rules, attempting to leave the router's firewall in its original state. If not (for example, if openNDS crashes instead of exiting cleanly) subsequently starting and stopping openNDS should remove its rules.

On OpenWrt, restarting the firewall will overwrite openNDS's iptables rules, so when the firewall is restarted it will automatically restart openNDS if it is running.

14.3 Packet Marking

openNDS operates by marking packets. Many packages, such as mwan3 and SQM scripts, also mark packets.

By default, openNDS marks its packets in such a way that conflicts are unlikely to occur but the masks used by openNDS can be changed if necessary in the configuration file.

14.4 IPtables Conflicts

Potential conflicts may be investigated by looking at your overall iptables setup. To list all the rules in all the chains, run

```
iptables -L
```

For extensive suggestions on debugging iptables, see for example, Oskar Andreasson's tutorial at:

https://www.frozentux.net/iptables-tutorial/iptables-tutorial.html

CHAPTER 15

TODO List

Not all features are finished or working as properly or as efficiently as they should. Other features have not been thought of yet!

Features should be aimed at providing tools to allow NDS to be used as flexible Captive Portal engine, rather than building in specific solutions.

Here is a list of things that need to be improved:

- While (un-) block/trust/allow via the ndsctl tool take effect, the state object of the client in NDS is not affected.

  Both systems still need to be connected (in src/auth.c).

- Include blocked and trusted clients in the client list - so that they can be managed.
- Extend Status processing to display a page when a user's authentication is rejected, e.g. because the user exceeded a quota or is blocked etc.
- Implement Traffic control on a user by user basis. This functionality was originally available but has been broken for many years.
- The code in src/http_microhttpd.c has evolved from previous versions and possibly has some missed edge cases. It would benefit from a rewrite to improve maintainability as well as performance.
- ip version 6 is not currently supported by NDS. It is not essential or advantageous to have in the short term but should be added at some time in the future.
- Automatic Offline mode. Either for forced offline use, or automatic detection of a failed Internet feed could be implemented. Some thought and discussion has been put into this and it is quite possible to achieve.

CHAPTER 16

Indices and tables

- genindex
- search