Installation von einem DExtra-Reflector auf einem RaspberryPI

In dieser Anleitung möchte ich beschreiben, wie ein **DExtra**-Reflector auf einem sehr beliebten Kleinrechner namens RaspberryPI installiert wird. Prinzipiell kann nach dieser Anleitung der Reflector-Server auch auf einem beliebigen Debian basierten Linux-System (bspw. Ubuntu) installiert werden.
Mein besonderer Dank gilt an dieser Stelle dem **Artöm R3ABM**, der die bekannten Sources überprüft hat und sogar einige Verbesserungen eingebracht hat. Die so entstandene Version kann gleichzeitig mit Clients sprechen, die unterschiedlichen „**DExtra**-Dialekten“ verwenden, was auf anderen Systemen zu vielen Problemen führt.

1 Installation von Rasbian

Als erstes laden wir den aktuellen **Raspbian**-Image von der folgenden Webseite http://www.raspberrypi.org/downloads

Aus Sicherheitsgründen sollen wir gleich den Image auf evtl. Manipulation wie folgt überprüfen:

```
bede@geek-vm:~/Software/RaspberryPi$ sha1sum 2013-09-25-wheezy-raspbian.zip
99e6b5e6b8cfbf66e34437a74022fcf9744ccb1d  2013-09-25-wheezy-raspbian.zip
```

Die berechnete Prüfsumme sollte mit der auf der Webseite angegebenen übereinstimmen. Nun habe ich eine 4GB SD-Card in einen USB-Leser eingeschoben und mit folgenden Linux-Befehlen das entpackte Image vom Rechner auf die Speicherkarte kopiert:

```
bede@geek-vm:~/Software/RaspberryPi$ unzip  2013-09-25-wheezy-raspbian.zip
bede@geek-vm:~/Software/RaspberryPi$ sudo dd if=2013-09-25-wheezy-raspbian.img of=/dev/sdb
5785600+0 Datensätze ein
5785600+0 Datensätze aus
2962227200 Bytes (3,0 GB) kopiert, 1755,16 s, 1,7 MB/s
```

Jetzt können wir die fertig beschriebene SD-Karte in den RaspberryPI einschieben, den Minirechner mit unserem Netzwerk verbinden und einschalten. In der DHCP-Tabelle meines Routers habe ich nachgeschaut, daß der gerade gestartete RaspberryPI die IP-Adresse 192.168.1.24 bekommen hat. Da ich keinen Monitor, Maus und Tastatur an den RaspberryPI angeschlossen haben, wurde auch der „Desktop“ nicht gestartet, was sich sehr schonend auf den Minirechner auswirkt. Um den RaspberryPI weiter konfigurieren zu können, habe ich mich mit ihm mittels einer üblichen SSH-Verbindung verbunden.

```
bede@geek-vm:~/Software/RaspberryPi$ ssh 192.168.1.24 -l pi
The authenticity of host '192.168.1.24 (192.168.1.24)' can't be established.
ECDSA key fingerprint is a3:60:e0:8f:04:6c:a2:5f:cd:e7:b6:51:91:36:26:15.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.24' (ECDSA) to the list of known hosts.
pi@192.168.1.24's password: raspberry ← Das ist das Initialpasswort!
Linux raspberrypi 3.6.11+ #538 PREEMPT Fri Aug 30 20:42:08 BST 2013 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
```

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

NOTICE: the software on this Raspberry Pi has not been fully configured. Please
run 'sudo raspi-config'

pi@raspberrypi ~ $
```

Nun sollte man, wie vom System vorgeschlagen mit „**sudo raspi-config**“ das Konfigurationsprogramm starten. Es empfiehlt sich den ersten Schritt „**Expand Filesystem**“ auszuführen. Daraufhin wird der restliche Platz der verwendeten Speicherkarte erschlossen und für das Grundsystem verfügbar gemacht. Mit der Funktion 2 „**Change User Password**“ kann beim Wunsch das Standardpasswort von dem Nuzter „**pi**“ geändert werden.

2 Kopieren von Quelltextverzeichnis

Zunächst wird das vorbereitete Archiv **dxrfd_Bln_23.10.2013.zip**, welches alle erforderlichen Quelltexte enthält, lokal entpackt. Anschließend kopiert man mit folgendem Befehl das ganze Quelltextverzeichnis vom lokalen Linux-Rechner auf den RaspberryPI:

```
scp -r dxrfd_Bln_23.10.2013 pi@192.168.1.24:/home/pi
```

Jetzt loggen wir uns auf den RaspberryPI mit dem bereits bekannten SSH-Befehl:

```
ssh 192.168.1.24 -l pi
```

Die weitere Einstellung müssen wir nicht als der Benutzer „**pi**“, sondern als „**root**“ durchführen. Deswegen geben wir folgende Befehle nach einander ein:

```
sudo bash
cd
cp -r /home/pi/dxrfd_Bln_23.10.2013 .
mv dxrfd_Bln_23.10.2013 dxrfd
```

3 Einrichten von statischer IP-Adresse

Jetzt haben wir einen lauffähigen RaspberryPI-Rechner. Das erste, was gemacht werden soll (da unser RaspberryPI als ein Server agieren soll), ist die Vergabe dem Minirechner einer statischen IP-Adresse z.B. 192.168.1.210. Dazu soll die Datei **/etc/network/interfaces** entsprechend geändert werden. Um diese Änderung jederzeit rückgängig machen zu können, hat sich unter Linux folgende Vorgehensweise etabliert. Wir benennen die Originaldatei **interfaces** in **interfaces_orignal** um. Kopieren die vorher vorbereitete Datei **interfaces_192.168.1.210** in den Ordner und legen einen symbolischen Link, der auf die gewünschte Datei zeigt.

```
cd /etc/network
mv interfaces interfaces_orignal
mv /root/dxrfd/interfaces_192.168.1.210 .
ln -s interfaces_192.168.1.210 interfaces
```

Jetzt starten wir den Rechner neu, um die gerade durchgeführte Änderung der IP-Adresse zu aktivieren.

Dabei bin ich von der folgenden IP-Konfiguration im lokalen LAN ausgegangen:

IP-Bereich: 192.168.1.0/24
Netmask: 255.255.255.0
GW: 192.168.1.1
Broadcast: 192.168.1.255

Der in diesem LAN aktive **DHCP** Server (meistens eine Funktion im handelsüblichen DSL-Router) soll dabei von dem o.a. Netz nur den Bereich 192.168.1.**20...100** verwalten. Die restlichen Adressen können also manuell vergeben werden.

4 Compilieren und Aktivierung von DXRFD

Zunächst verbinden wir zu dem RaspberryPI, der jetzt nun unter der neuen statischen IP-Adresse erreichbar ist:

```
ssh 192.168.1.210 -l pi
```

Nun müssen wir die Quelltexte compilieren. Das geht mit folgenden Befehlen:

```
sudo bash
cd /root/dxrfd/

root@raspberrypi:~/dxrfd# ./dxrfd.doit
root@raspberrypi:~/dxrfd# ./xrf_lh.doit
```

Nun sollte die Konfiguratioinsdatei **dxrfd.cfg** entsprechend eigener Bedürfnisse angepasst werden. Im wesentlichen sind es die Variablen ADMIN und OWNER.
OWNER=XRF210 bedeutet, daß unser Reflector unter dem Raumkürzel (in UP4DAR) XRF210 läuft.

Nachdem die Konfigurationsdatei angepasst ist, wird das gerade compilierte Datei als ein s.g. Dienst bei der Startmaschinerie des Kernels mit folgenden Befehlen angemeldet.

```
root@raspberrypi:~/dxrfd# chmod +x dxrfd.SVC
root@raspberrypi:~/dxrfd# mv dxrfd.SVC /etc/init.d/dxrfd
root@raspberrypi:~/dxrfd# update-rc.d dxrfd defaults
update-rc.d: using dependency based boot sequencing
insserv: warning: script 'dxrfd' missing LSB tags and overrides
root@raspberrypi:~/dxrfd# service dxrfd start
```

Ab nun an läuft bereits unser **DExtra**-Dienst. Das können wir mit folgenden Befehlen kurz kontrollieren:

```
root@raspberrypi:~/dxrfd# netstat -n -a -p|grep 300
udp        0    0 0.0.0.0:30001   0.0.0.0:*           1949/dxrfd
udp        0    0 0.0.0.0:30002   0.0.0.0:*           1949/dxrfd
root@raspberrypi:~/dxrfd#
```

5 Dashboard für den XRF210

Nun wollen wir natürlich, daß auf unserem RaspberryPI passend zu dem bereits voll funktionsfähigen Reflector **XRF210** ein entsprechendes Dashboard läuft und abrufbar ist. Dafür müssen wir ein Webserver-Dienst installieren. Um die sehr einfach gestaltete Dashboard-Seite anzuzeigen, reicht vollkommen ein sehr schlanker und trotzdem leistungsfähiger Webserver namens **lighttpd**. Diesen installieren wir wie folgt:

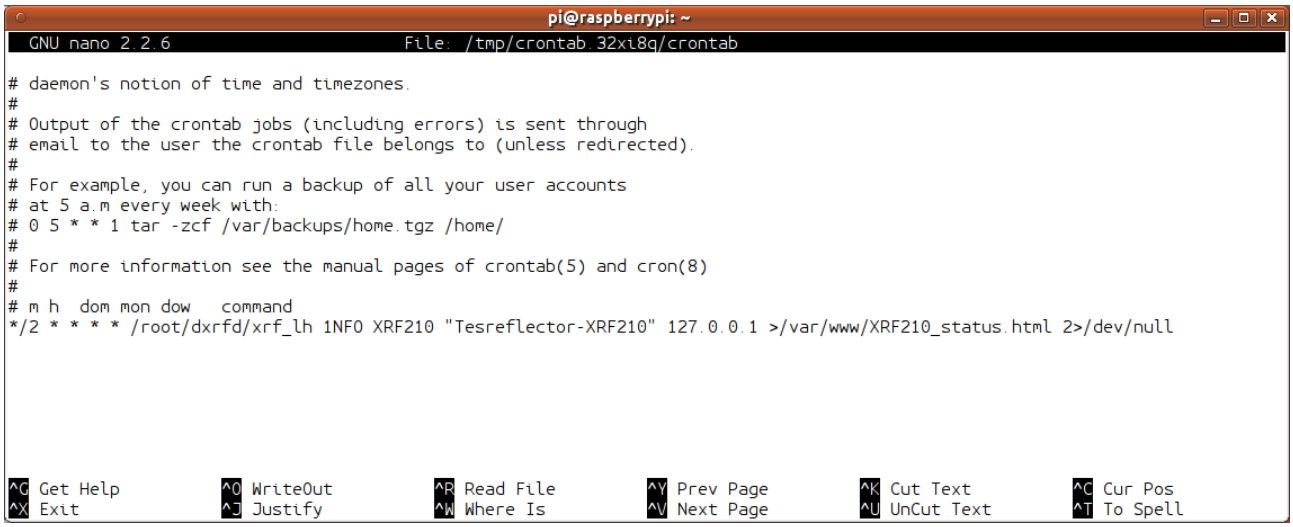
```
root@raspberrypi:~/dxrfd# aptitude install lighttpd
```

Dann richten wir die benötigten Verzeichnisse an und kopieren einige mitgelieferte Dateien:

```
root@raspberrypi:~# cd /var/www/
root@raspberrypi:/var/www# mkdir g2_ircddb
root@raspberrypi:/var/www# cd g2_ircddb
root@raspberrypi:/var/www/g2_ircddb# mv /root/dxrfd/mm_spacer.gif .
root@raspberrypi:/var/www/g2_ircddb# mv /root/dxrfd/mm_training.css .
```

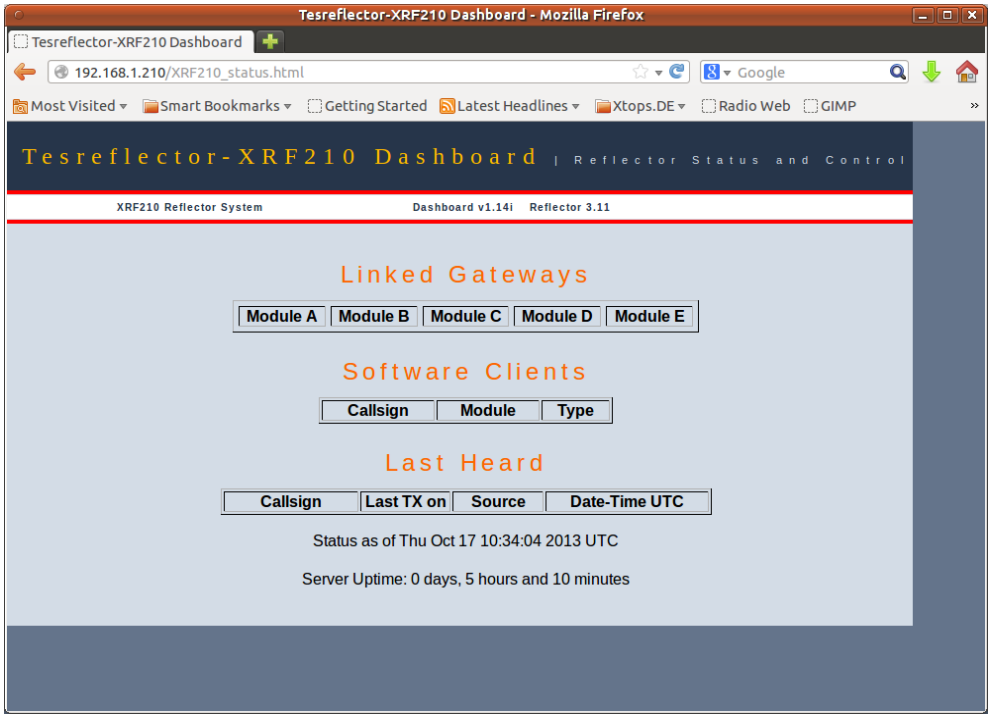
Nun läuft bereits unser Webserver und ist richtig konfiguriert. Jetzt müssen wir dem Kernel von RaspberryPI (genauer dem CRON-Job-Daemon) den Auftrag geben, eine HTML-Seite mit Dashboard-Informationen z.B. alle 2 min durch das mitgelieferte und von uns bereits compilierte Programm **xrf_lh** zu erstellen. Dazu rufen wir den Cron-Job-Editor mit dem Befehl „**`crontab -e`**“ und tragen die folgende Zeile von Hand ein.

*/2 * * * * /root/dxrfd/xrf_lh 1NFO XRF210 "Testreflector-XRF210" 127.0.0.1 >/var/www/XRF210_status.html 2>/dev/null



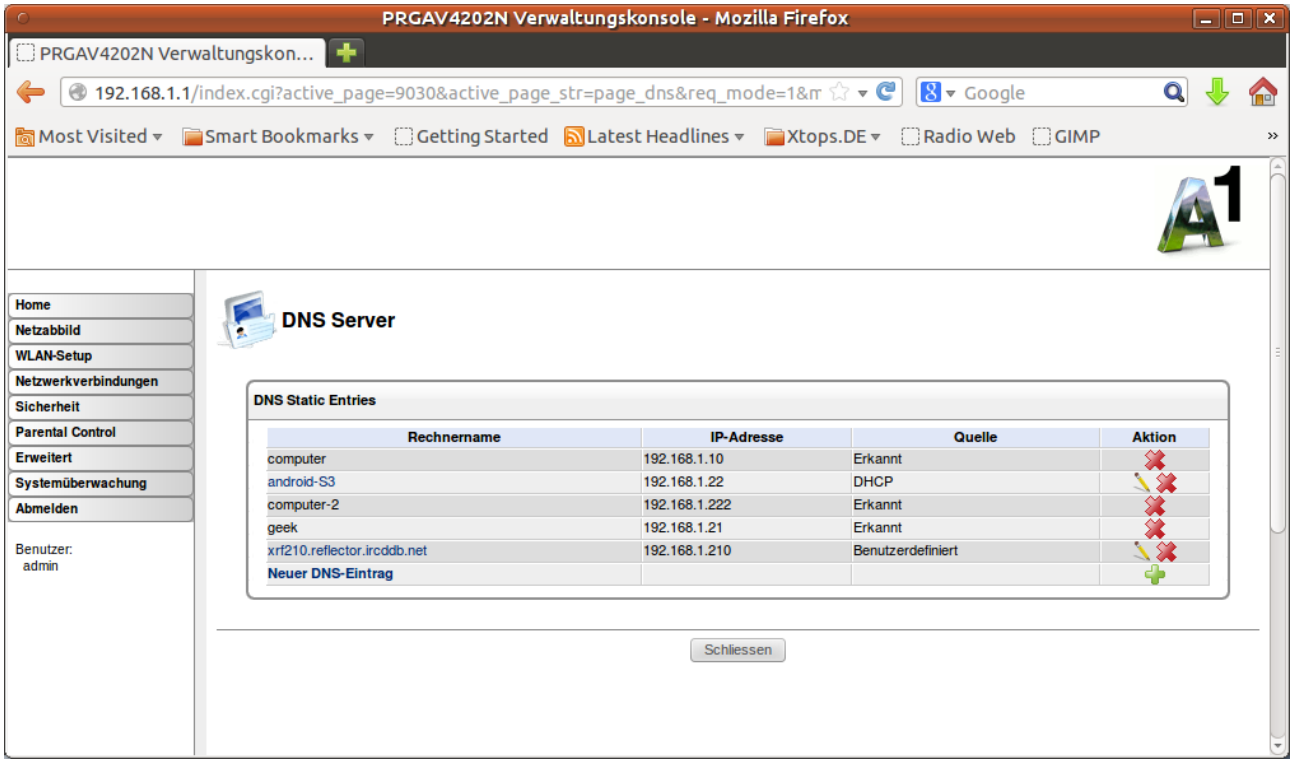
Mit CTRL+X wird der Editor geschlossen. Eventuell muß noch der Speicherung der gerade geänderten Datei zugestimmt werden.

Nun können wir den Dashboard im Browser eines beliebigen Rechners in unserm LAN unter der folgenden Adresse aufrufen:
`http://192.168.1.210/XRF210_status.html`



6 Festlegen der IP-Adresse für XRF210

Nun möchten wir die erste Verbindung von z.B. UP4DAR zu unserem XRF210 aufbauen. Sobald wir in der bekannten Ansicht als Betriebsart „**IP Reflector**“ einstellen und in der nächsten Zeile unser Ziel „**XRF210**“ wird UP4DAR versuchen, die IP-Adresse zu ermitteln, die folgender symbolischen URL-Adresse entspricht: **`xrf210.reflector.ircddb.net`**
Im Normalfall würde man dem IRCDDB-Team eine e-mail schreiben und die eigene öffentliche IP-Adresse eintragen lassen. Möchte man nicht, daß der eigene Testreflector öffentlich bekannt wird, könnte man sich eine andere Lösung einfallen lassen. Beispielsweise erlaubt mein DSL-Router diese s.g. DNS-Anfragen auch in einer lokalen Tabelle anzulegen. Gibt es also ein lokaler Eintrag, so wird dieser genommen und an den anfragenden Rechner im LAN zurückgeliefert, ohne diese Anfrage nach außen ins Internet weiterzuleiten.



Nun verbindet sich mein UP4DAR zu dem XRF210 ohne Probleme!

7 Betreiben eines eigenen DNS-Servers im LAN

Die im letzten Kapitel gezeigte DNS-Eintragung beherrschen bei weitem nicht alle handelsübliche Router, so daß diese Möglichkeit für viele ausscheidet. Deswegen möchte ich hier eine Lösung vorstellen, die auf jeden Fall funktionsfähig ist und uns den Einblick in die DNS-Technologie, so wie sie tatsächlich im Internet eingesetzt wird, ermöglicht.
Die Grundlage bildet der wahrscheinlich mit Abstand am häufigsten im Internet eingesetzte DNS-Server **bind9**. Zur Installation von **bind9** verbinden wir uns wieder mit unserem RaspberryPI wie schon gehabt mit:

```
ssh 192.168.1.210 -l pi
```

Und installieren **bind9** mit folgenden Befehlen:

```
sudo bash

aptitude install dnsutils
aptitude install bind9
```

Da die Konfiguration von dem gerade installierten DNS-Server für den Anfänger etwas schwierig ist, habe ich für unsere Zielsetzung relevante Konfigurationsdateien erstellt bzw. angepasst. Diese müssen nur noch an die richtige Stelle kopiert werden. Dafür gibt man folgende Befehle ein:

```
cd /etc/bind
mv /root/dxrfd/named.conf.local .
mv /root/dxrfd/named.conf.options .
mv /root/dxrfd/db.reflector.ircddb.net .
mv /root/dxrfd/db.192 .
```

und startet anschließend den DNS-Server mit:

```
service bind9 restart
```

Nun läuft auf dem RaspberryPI ein richtiger DNS-Server, der so konfiguriert ist. Stellt man eine beliebige Anfrage an ihn, so holt er sich die Antwort seinerseits bei dem Router (also unter **192.168.1.1**) und leitet diese weiter an den Anfragenden, s.g. *Weiterleitungsbetrieb*. Die bearbeiteten Anfragen werden von dem DNS-Server zwischen gespeichert und im Wiederholungsfall aus dem internen Speicher sofort beantwortet (s.g. *Caching*). Geht eine Anfrage für die Domäne **reflector.ircddb.net** ein, so beantwortet unser DNS-Server diese Anfrage grundsätzlich alleine, da er für diese Domäne als Master konfiguriert ist. Das ist aber genau was wir bezwecken wollen. Nun müssen wir selber dafür sorgen, daß alle IP-Adressen von dieser Domäne immer aktuell sind und können jederzeit eigene IP-Adressen eintragen. Die dafür zuständige Konfigurationsdatei befindet sich unter **/etc/bind/db.reflector.ircddb.net** und sieht so aus:

```
;
; BIND data file for local loopback interface
;
$TTL  604800
@     IN    SOA   ns.reflector.ircddb.net. root.reflector.ircddb.net. (
                  2013102901  ; Serial
                   604800     ; Refresh
                    86400     ; Retry
                  2419200     ; Expire
                   604800 )   ; Negative Cache TTL
;
@     IN    NS    ns.reflector.ircddb.net.
ns    IN    A     192.168.1.1

; list of all other computers
xrf210      IN    A     192.168.1.210

xrf001      IN    A     93.174.138.180
xrf002      IN    A     193.16.217.17
xrf003      IN    A     95.110.157.13
xrf004      IN    A     96.36.58.9
xrf005      IN    A     216.16.240.236
xrf006      IN    A     80.62.20.151
xrf007      IN    A     84.232.6.94
xrf008      IN    A     87.106.84.53
xrf009      IN    A     141.22.15.29
xrf013      IN    A     202.191.108.233
xrf020      IN    A     184.154.35.204
xrf017      IN    A     87.253.159.18
xrf019      IN    A     66.30.81.236
xrf020      IN    A     184.154.35.204
xrf021      IN    A     74.204.50.67
xrf023      IN    A     141.75.245.225
xrf026      IN    A     139.13.100.34
xrf027      IN    A     194.116.29.66
xrf028      IN    A     193.190.240.228
xrf030      IN    A     77.76.146.234
```

Wir sehen, daß diese Datei neben einiger gültigen IP-Adressen auch den Eintrag für unser Testreflector-XRF210 hat (gelb hinterlegt). Dieser Eintrag bedeutet, daß die IP-Adresse **xrf210.reflector.ircddb.net** auf die IP-Adresse **192.168.1.210** aufgelöst wird. Jedes Mal wenn wir diese Datei verändern, müssen wir die Seriennummer (gelb hinterlegt) verändern! Es hat sich eingebürgert dafür eine Seriennummer zu vergeben, die dem aktuellen Datum entspricht und einer z.B. zweistelligen Nummer, die die Änderung an dem entsprechenden Tag repräsentiert. So wird die

Wahrscheinlichkeit für eine falsche Seriennummer fast ausgeschlossen. Nach jeder Veränderung von Konfigurationsdateien soll der DNS-Server neu gestartet werden, was mit dem Befehl erfolgt:

```
service bind9 restart
```

Nun läuft unser lokale DNS-Server auf dem RaspberryPI, aber der Minirechner selber nutzt ihn eigentlich gar nicht! Das können wir in der folgenden Datei nachschauen:

```
pi@raspberrypi ~ $ cat /etc/resolv.conf
domain fritz.box
search fritz.box
nameserver 192.168.1.1
pi@raspberrypi ~ $
```

Der RaspberryPI selber nutzt also den DSL-Router zur Namensauflösung. Deswegen wundert uns nicht, daß die folgende Anfrage nach unserem Testreflector-XRF210 nichts zurückliefert:

```
pi@raspberrypi ~ $ nslookup xrf210.reflector.ircddb.net
Server:     192.168.1.1
Address:    192.168.1.1#53

** server can't find xrf210.reflector.ircddb.net: NXDOMAIN
```

Dagegen die Anfrage nach einem tatsächlich existierenden Server seine gültige IP-Adresse:

```
pi@raspberrypi ~ $ nslookup xrf001.reflector.ircddb.net
Server:     192.168.1.1
Address:    192.168.1.1#53

Non-authoritative answer:
Name: xrf001.reflector.ircddb.net
Address: 93.174.138.180

pi@raspberrypi ~ $
```

Nun ändere ich in der Datei /etc/resolv.conf die IP-Adresse des zu benutzenden Nameservers von 192.168.1.1 auf unser eben installierten und konfigurierten DNS-Server um und gebe die gleiche Anfrage noch mal ein:

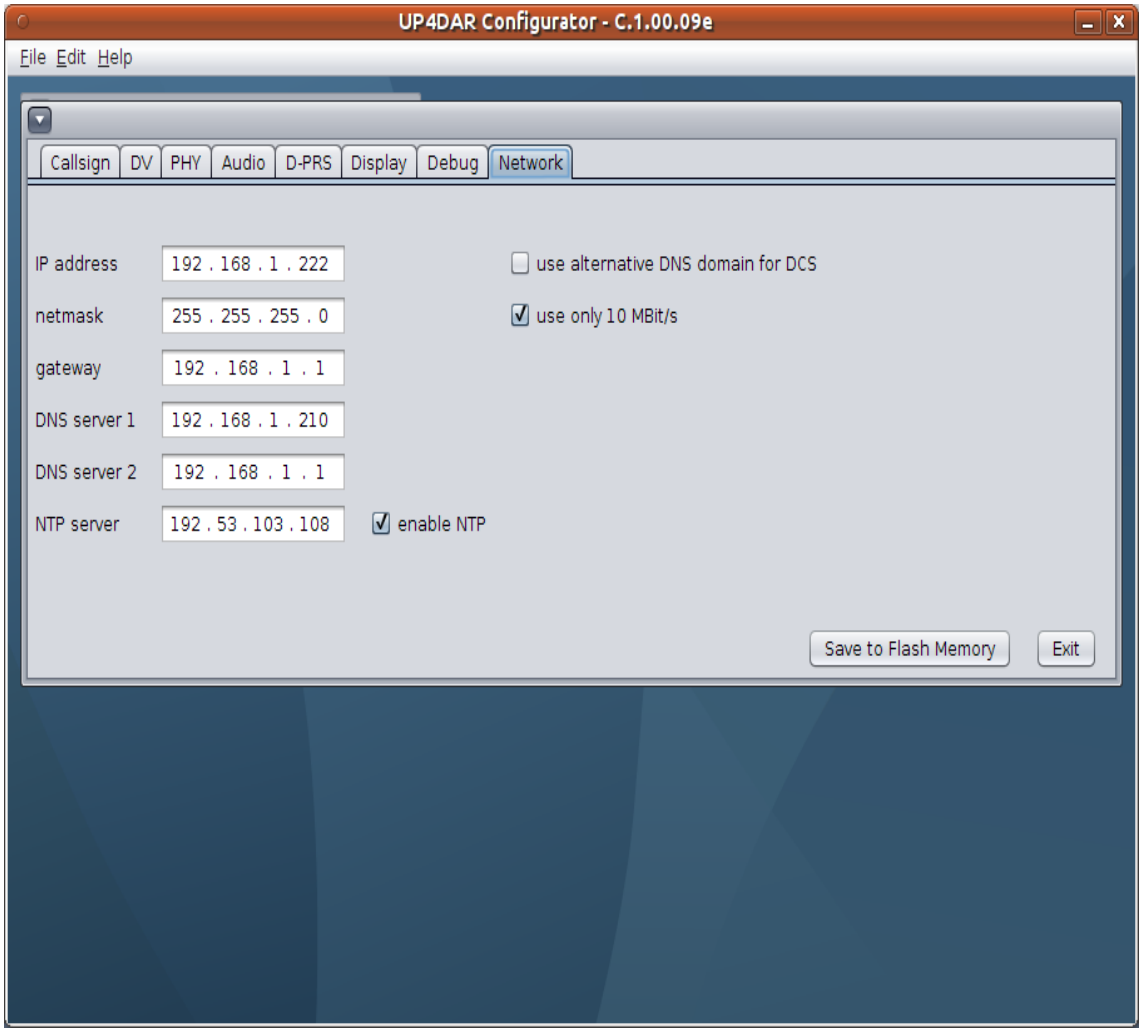
```
pi@raspberrypi ~ $ nslookup xrf210.reflector.ircddb.net
Server:     192.168.1.210
Address:    192.168.1.210#53

Name: xrf210.reflector.ircddb.net
Address: 192.168.1.210

pi@raspberrypi ~ $
```

Jetzt bekommen wir die IP-Adresse, die wir selber vorgegeben haben!
Da unser DNS-Server richtig läuft und getestet ist, ändere ich den zu benutzenden DNS-Server auf dem RaspberryPI zurück zu **192.168.1.1**

Jetzt konfiguriert man auf dem UP4DAR als „**DNS server 1**“ unseren lokalen DNS-Server mit der Adresse **192.168.1.210** und als „**DNS server 2**“ den „richtigen“ von unserem DSL-Router mit der Adresse **192.168.1.1**



Jetzt werden die UP4DAR-Anfragen von dem lokalen DNS-Server aufgelöst, sofern er im Netzwerk läuft. Ist der RaspberryPI ausgeschaltet, so laufen die Anfragen wir gewohnt über unser DSL-Router!

Viel Spaß beim Ausprobieren!