VDR auf dem DockStar unter Arch Linux ARM

Olaf Bauer

11. Juni 2015

Inhaltsverzeichnis

1 Einleitung

1.1 Hintergrund

Der Seagate FreeAgent DockStar ist ein Netzwerkadapter (NAS), der USB-Speichermedien im lokalen Netzwerk und - über den kostenpflichtigen Pogoplug-Service - auch per Internet zugänglich macht. Das Gerät verfügt über einen 1,2 GHz ARM Prozessor, 128 MB RAM, 1x GBit LAN und 4 USB 2.0 Schnittstellen, von denen eine zum Anschluss kompatibler Festplatten (FreeAgent Go) als Mini-USB-Stecker ausgeführt ist. Das Betriebssystem, ein angepasstes Linux - im folgenden Pogoplug Linux genannt -, befindet sich auf dem internen 256 MB Flash Speicher.

Eine Modifikation des Bootloaders ermöglicht es am DockStar auch jede andere für die ARMv5-Architektur verfügbare Linux-Distributionen zu betreiben. Dadurch wird der DockStar unabhängig vom Pogoplug-Dienst und der Verwendungszweck lässt sich vielfältig erweitern. Konsolenzugriff erhält man im lokalen Netzwerk - und nach Einrichtung einer Portweiterleitung am Router auch via Internet - über SSH. Windows-Nutzer greifen zu diesem Zweck gern auf PuTTY zurück, als lohnende Alternative sei an dieser Stelle das Cygwin-Projekt genannt, das neben OpenSSH z.B. auch rsync anbietet.

Mitte 2010 war der DockStar für ca. 20-30 Euro erhältlich, heute werden bei eBay hin und wieder noch Gebrauchtgeräte angeboten. Ich würde aber stattdessen auf einen Pogoplug E02 bieten, der 256 MB RAM mitbringt, meistens günstiger und auch in schwarz erhältlich ist. Achtung: Nicht selten unterscheidet sich die Modellnummer auf der Verpackung von derjenigen auf der Unterseite des Pogoplug. Nur die letztgenannte ist verlässlich, dies sollte ggf. mit dem Verkäufer zuvor geklärt werden.

1.2 Überblick

Den Leser erwartet hier eine relativ detaillierte, mit der Zeit gewachsene Dokumentation der Einrichtung des DockStar als DVB-Rekorder. Die meisten Beschreibungen sind auch für den Pogoplug E02 gültig, der bei mir mittlerweile hauptsächlich zum Einsatz kommt. Als Betriebssystem nutze ich Arch Linux ARM (kurz: ALARM), das aus PlugApps Linux (vormals Plugbox Linux) und ArchMobile hervorgegangen ist und auf dem seit längerem für die i686- und x86_64-Architekturen verfügbaren Arch Linux basiert. Zur Aufnahme von TV-Sendungen dient das als Daemon im Hintergrund laufende Programm VDR. An den DockStar/Pogoplug angeschlossen sind

- ein USB-Speicherstick, auf dem Arch Linux ARM installiert ist
- ein vom Linux-Kernel unterstützter USB-DVB-Stick (Hauppauge HVR-930C[1])
- eine USB-Festplatte, auf der sich das primäre Aufnahmeverzeichnis des VDR befindet.

Ein weiteres, sekundäres Aufnahmeverzeichnis auf dem USB-Speicherstick ermöglicht einen zeitweilig lautlosen Betrieb ohne Zugriff auf die Festplatte. Beide Verzeichnisse werden mit dem

[1] Achtung: Der HVR-930C wird nicht mehr produziert. Die aktuell im Handel erhältlichen Versionen werden vom Linux-Kernel nicht unterstützt. Sie sind an der Zusatzbezeichnung “HD” auf der Verpackung zu erkennen. Diese Bezeichnung ist übrigens irreführend: Auch der HVR-930C versteht sich auf den Empfang von in HD (MPEG-4/H.264) ausgestrahlten (unverschlüsselten) Sendungen. Vermutlich wurde der älteren Version des Sticks lediglich eine Windows-Software beigelegt, die H.264 nicht zu dekodieren vermag, unter Linux stellt dies jedenfalls kein Problem dar.

Overlay-Dateisystem Aufs zusammengeführt und per NFS im lokalen Netzwerk freigegeben. Die VDR-Oberfläche ist dort per Remote Frontend zugänglich. Zum Anlegen von Aufnahmetimern, auch via Internet, dient ein Webinterface. Bei Bedarf zeigt ein externes Display Programminformationen und Menü des VDR an. Live-TV kann via HTTP Stream an geeignete Clients übertragen werden. Auch eine Integration in XBMC/Kodi ist möglich, ich verwende dafür einen RaspberryPi mit OpenELEC.

Die von mir erstellten Binär-Pakete für den VDR und einige Plugins bzw. Addons stelle ich in einem Repository zur Verfügung. Es kann für alle Geräte auf Basis der ARMv5-Plattform unter ALARM genutzt werden. Auf anderen Architekturen (ARMv6/7) müsstet ihr die jeweils verlinkten Quellpakete selbst kompilieren oder das VDR4Arch Repository nutzen.

2 Vorbereitung des USB-Sticks

Die offizielle Installationsanleitung beschreibt die Formatierung eines USB-Sticks mit *Ext3*, das Aufspielen das Betriebssystem-Images und die Installation des neuen, von Arch Linux ARM bereitgestellten Bootloaders *U-Boot* am DockStar unter Pogoplug Linux. Von dieser Vorgehensweise weiche ich in einigen Punkten ab:

- Ich verwende das ältere U-Boot Environment von Jeff Doozan, da das neue Environment ein Booten des originalen Pogoplug Linux nicht vorsieht.
- Als Dateisystem für die Rootpartition setze ich *Ext4* ein.
- Es wird eine separate Bootpartition mit *Ext2* angelegt, weil das verwendete U-Boot Environment das Kernelimage von einer *Ext4*-Partition nicht starten kann[2].
- Die Installation wird an einem PC unter Linux durchgeführt, da der Pogoplug-Kernel keine Unterstützung für *Ext4* mitbringt.

Wer auf seinem PC ein anderes Betriebssystem einsetzt, nutzt für die in diesem Abschnitt durchgeführten Schritte einfach eine aktuelle Linux-Live-CD

2.1 Formatierung

Die Formatierung eines Datenträgers birgt stets die Gefahr eines Datenverlustes durch irrtümliche Angabe eines falschen Devices. Es ist deshalb wichtig das Device des USB-Sticks zweifelsfrei zu identifizieren, etwa anhand der Ausgabe von `dmesg`. Im folgenden wird stellvertretend `/dev/sdx` verwendet. Mit `fdisk` lege ich auf dem Stick drei primäre Partitionen an. Man beachte, dass es Vorteile haben kann, stattdessen `gdisk` zu verwenden, dazu mehr im Forum und im Abschnitt 8.2.

```
# fdisk /dev/sdx
```

Neben Bootpartition (32 MB) und Rootpartition (1200 MB) nimmt eine dritte (Home-)Partition den Rest des 16 GB-Sticks ein. Sie wird das sekundäre VDR-Aufnahmeverzeichnis beherbergen. Eine Swap-Partition sollte man auf Flash-Speichermedien nicht anlegen, 128 MB RAM sind für den beschriebenen Verwendungszweck ohnehin mehr als ausreichend. Hier das Ergebnis der Partitionierung:

```
Disk /dev/sdx: 16.0 GB, 16001269760 bytes
64 Köpfe, 32 Sektoren/Spur, 15260 Zylinder, zusammen 31252480 Sektoren
Einheiten = Sektoren von 1 × 512 = 512 Bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Gerät  boot.     Anfang        Ende     Blöcke   Id  System
/dev/sdx1            2048       67583       32768   83  Linux
/dev/sdx2           67584     2525183     1228800   83  Linux
/dev/sdx3         2525184    31252479    14363648   83  Linux
```

[2] Wird *Ext3* für die Rootpartition eingesetzt, ist keine separate Bootpartition erforderlich. Das neue U-Boot Environment (Schritt 7 und 8 in der offiziellen Installationsanleitung) unterstützt auch das Booten von einer mit *Ext4* formatierten Partition.

Beim Anlegen der Dateisysteme erhält jede Partition zur einfacheren Identifizierung zugleich ein Label. Um keinen Speicherplatz zu verschwenden, wird auf der home-Partition der für root reservierte Anteil an Blöcken (5% per Voreinstellung) auf 1% reduziert.

```
# mkfs.ext2 -L boot /dev/sdx1
# mkfs.ext4 -L root /dev/sdx2
# mkfs.ext4 -L home -m 1 /dev/sdx3
```

Um Schreibzugriffe zu minimieren, kann man das Journal der *ext4*-Partitionen mit der Option `-O ^has_journal` deaktivieren, riskiert damit bei Stromunterbrechung aber ein inkonsistentes Dateisystem.

2.2 Installation

Der Stick wird nun gemountet und das Legacy Image von Arch Linux ARM darauf entpackt. Man achte sorgsam darauf, dieses im richtigen Verzeichnis zu tun, um keine Dateien des laufenden Systems zu überschreiben. Das Image enthält Kernel 3.1.10 und ist dadurch auch mit älteren Versionen des Bootloaders U-Boot kompatibel. Der im alternativ angebotenen Kirkwood Image integrierte, aktuelle Kernel kann bei Bedarf nachträglich installiert werden.

```
# mount /dev/sdx2 /mnt/
# cd /mnt/
# mkdir boot
# mount /dev/sdx1 boot/
# tar -xf /path/to/ArchLinuxARM-armv5te-latest.tar.gz
```

Da der Bootloader *U-Boot* das Kernel-Image auf der ersten Partition im Verzeichnis `/boot` erwartet (so wie es ohne separate Bootpartition wäre), legen wir einfach einen gleichnamigen Link auf das aktuelle Verzeichnis an. Das ist etwas einfacher als die Umgebungsvariablen des Bootloaders anzupassen (Abschnitt 3.2).

```
# cd boot
# ln -s . boot
```

2.3 Vorkonfiguration

Es werden noch einige Konfigurationsdateien auf dem Stick - also unterhalb von `/mnt` - editiert. Zumindest `/etc/fstab` sollte angepasst werden. Dort sind die Partitionen des USB-Sticks einzutragen, andernfalls wird nur die Rootpartition gemountet. Da sich die Zuordnung der Gerätedateien nach einen Reboot ändern kann, empfiehlt sich die Verwendung der oben vergebenen Label. Alternativ können UUIDs (laut `blkid`) genutzt werden, für die Rootpartition auch `/dev/root`. Die Option `noatime` unterbindet das Aktualisieren der Zeitstempel beim Lesezugriff auf Dateien und Verzeichnisse.

```
#/etc/fstab
LABEL=root  /       ext4  noatime  0 0
LABEL=boot  /boot   ext2  noatime  0 0
LABEL=home  /home   ext4  noatime  0 0
```

Die Netzwerkkonfiguration erfolgt unter Arch Linux ARM derzeit mit `netctl`. Per Voreinstellung werden IP und weitere Parameter von einem DHCP-Server (üblichweise dem Hardware-Router) abgerufen. Das zugehörige Profil ist in `/etc/netctl/eth0` hinterlegt[3].

Hinweis: Seit August 2014 ist in den ALARM-Tarballs `systemd-networkd` vorinstalliert. Es ersetzt die Funktionalität von `netctl` und wird über `/etc/systemd/network/eth0.network` konfiguriert.

In kleinen Netzwerken ziehe ich eine statische Konfiguration vor. Die letzte Zeile sorgt für die einmalige Zeiteinstellung per NTP (der DockStar besitzt keine Hardwareuhr) und sollte nicht entfernt werden. Die Option `ExcludeAuto=no` ist seit Version 1.10 von `netctl` erforderlich.

```
#/etc/netctl/eth0
Interface=eth0
Connection=ethernet
IP=static
Address=('192.168.1.5/24')
Gateway='192.168.1.1'
DNS=('192.168.1.1')
DNSDomain="localdomain"
ExcludeAuto=no

ExecUpPost='/usr/bin/ntpd -gq || true'
```

Sind alle Vorbereitungen abgeschlossen, kann der USB Stick ausgehängt werden.

```
# cd / && umount /mnt/boot/ /mnt
```

[3]Das Profil wird über den `ifplug` Daemon aufgerufen, falls ein Patchkabel angeschlossen ist. Es ist nicht notwendig den `netctl` Daemon zu aktivieren.

3 Anpassung des Bootloaders

Um Arch Linux ARM vom USB-Stick starten zu können, muss der Bootloader *U-Boot* aktualisiert und angepasst werden. Dazu wird zunächst das auf dem internen Flash-Speicher befindliche Pogoplug-Linux gebootet. Wer ein ungewolltes Firmwareupdate befürchtet, das u.U. den SSH-Zugang deaktiviert, trennt zuvor einstweilen den Router physisch vom Internet.

3.1 Update

Nach Herstellen der Stromzufuhr können einige Minuten vergehen, bis der DockStar vom Router eine IP Adresse anfordert und der Zugang per SSH (Login: `root`, Passwort: `stxadmin`) möglich ist. Als erstes beendet man dann den Pogoplug Client:

```
$ killall hbwd
```

Danach kann die Internetverbindung ggf. wieder hergestellt werden. Anschließend wird das Installationsscript von Jeff Doozan heruntergeladen und ausgeführt:

Achtung: Das in den folgenden Schritten installierte U-Boot Environment ermöglicht weiterhin auch das auf dem NAND befindliche Pogoplug Linux zu booten. Das neue, von Arch Linux ARM gepflegte Environment erlaubt dies nicht. Das neue Environment wird wie in Schritt 7 und 8 der Installationsanleitung beschrieben mit dem Script `dockstar.sh` (bzw. `pogo_e02.sh` für den Pogoplug E02) installiert. Auf bestehenden ALARM-Installationen erfolgt die Umstellung - sofern gewünscht - durch Installation des Paketes `uboot-dockstar` (bzw. `uboot-pogo_e02`)

```
$ cd /tmp
$ wget http://jeff.doozan.com/uboot/install_uboot_mtd0.sh
$ chmod +x install_uboot_mtd0.sh
$ ./install_uboot_mtd0.sh
```

Bevor das Script eine neue Version von *U-Boot* installiert, deaktiviert es den Pogoplug Client permanent durch Auskommentieren einer Zeile im Startscript `/etc/init.d/rcS`:

```
#Uncomment the line below to enable the pogoplug service
#/etc/init.d/hbmgr.sh start
```

3.2 Konfiguration

Hinweis: Die folgenden Einstellungen beziehen sich auf das U-Boot Environment von Jeff Doozan. Das von Arch Linux ARM gepflegte Environment verwendet z.T. andere Umgebungsvariablen, die vorzugsweise über eine auf dem USB-Stick anzulegende Datei `/boot/uEnv.txt` (re)konfiguriert werden.

Das U-Boot Environment besteht aus Variablen und Befehlen, die auf der U-Boot-Kommandozeile[4] mit `sentenv` gesetzt und mit `printenv` angezeigt werden können. Um dasselbe aus dem laufenden Betriebssystem heraus zu bewerkstelligen zu können, installiert das Script im Verzeichnis `/usr/sbin/`

[4] Zur interaktiven U-Boot-Kommandozeile gelangt man über eine serielle Konsole oder über die Netzkonsole.

die Programme `fw_setenv` und `fw_printenv`[5]. Sie müssen ggf. mit vollständigem Pfad aufgerufen werden, man kann aber auch vorübergehend die Shellvariable `PATH` erweitern:

```
$ PATH=$PATH:/usr/sbin/
$ fw_printenv
```

Wie der Ausgabe zu entnehmen ist, generiert der Befehl `usb_set_bootargs` die Kernel-Bootparameter. Die dort verwendete Variable für die Rootpartition `$usb_root` wird über `usb_scan` ermittelt. Folgende Anpassungen habe ich vorgenommen, um sicherzustellen, dass hier nicht die erste, sondern die zweite Partition des USB-Sticks zugeordnet wird:

```
$ fw_setenv usb_scan_1 'usb=0:1 dev=sda2'
$ fw_setenv usb_scan_2 'usb=1:1 dev=sdb2'
$ fw_setenv usb_scan_3 'usb=2:1 dev=sdc2'
$ fw_setenv usb_scan_4 'usb=3:1 dev=sdd2'
```

Ist das Device der Rootpartition bekannt (z.B. bei fester Anordnung von USB-Datenträgern), kann stattdessen auch das Script `usb_set_bootargs` direkt editiert werden, siehe Abschnitt 8. Vorläufig behelfen wir uns aber wie beschrieben. Weiterhin müssen noch die Angaben betreffs des root-Dateisystems und der `arcNumber`[6] (zur Ansteuerung der LEDs) angepasst werden:

```
$ fw_setenv usb_rootfstype ext4
$ fw_setenv arcNumber 2998
```

Empfehlenswert ist auch das Aktivieren der Netzkonsole (Abschnitt 3.3). Schließlich wird der vorbereitete USB-Stick angeschlossen und der DockStar neu gebootet:

```
$ /sbin/reboot
```

Wie man ggf. den Bootmeldungen entnehmen kann, scheitert der Start von Arch Linux ARM u. U. zunächst. In diesem Fall wird spätestens nach einigen Minuten erneut das weiterhin auf dem internen Flash-Speicher residierende Pogoplug Linux geladen. Nach einem weiteren Neustart sollte es dann gelingen. Wurde dieselbe IP vergeben bzw. statisch eingestellt, ist vor der nächsten Anmeldung (Login: `root`, Passwort: `root`) der öffentliche Schlüssel von Pogoplug Linux aus der Liste der bekannten Hosts zu entfernen (IP-Adresse anpassen).

```
$ ssh-keygen -R 192.168.1.5
```

Je nach verwendetem USB-Stick führt jede künftige Unterbrechnung der Stromzufuhr u.U. dazu, dass zunächst wieder Pogoplug Linux gestartet wird. Um einen Reboot per SSH herbeiführen zu können, muss nun ggf. erneut der Schlüssel gelöscht werden. Eine Möglichkeit diesen Zyklus zu unterbinden und zuverlässig auch nach einem Kaltstart Arch Linux ARM zu booten findet sich im Forum von Jeff Doozan.

```
$ fw_setenv bootcmd 'usb start; usb stop; usb start; run force_rescue_bootcmd; run ↩
    ↪ ubifs_bootcmd; run usb_bootcmd; usb stop; run rescue_bootcmd; run ↩
    ↪ pogo_bootcmd; reset'
```

[5] `fw_printenv` ist ein symbolischer Link auf `fw_setenv`. Innerhalb von Arch Linux ARM stellt das Paket `uboot-tools` diese Befehle bereit, sie können dann ohne Pfad bzw. `PATH`-Anpassung aufgerufen werden.

[6] Auf dem Pogoplug E02 setzt man stattdessen die `machid` auf den Wert `dd6`.

Nun sollte Pogoplug Linux als Fallback nur noch bei nicht angeschlossenem USB-Stick booten.[7]

3.3 Netzkonsole

Mithilfe der Netzkonsole können Meldungen des Bootloaders auf einem anderen Rechner (Server) im Netzwerk angezeigt werden. Die notwendigen Einstellungen beschreibt Jeff Doozan in seinem Forum. Im folgenden Beispiel ist 192.168.1.5 die IP des DockStar, *U-Boot* sendet die Nachrichten an Port 6666 des angegebenen Servers:[8]

```
$ fw_setenv serverip 192.168.1.7
$ fw_setenv ipaddr 192.168.1.5
$ fw_setenv if_netconsole 'ping $serverip'
$ fw_setenv start_netconsole 'setenv ncip $serverip; setenv bootdelay 10; setenv ⟵
    ↪ stdin nc; setenv stdout nc; setenv stderr nc; version;'
$ fw_setenv preboot 'run if_netconsole start_netconsole'
```

Auf demselben Weg lassen sich auch Kernelmeldungen übertragen, die benötigten Kernelparameter können in der Variablen `usb_custom_params` hinterlegt werden[9]. Auch hier wird Port 6666 verwendet. Das ist unproblematisch, da die Kernelmeldungen ohnehin zeitlich nach den Bootmeldungen folgen. Der letzte Parameter ist die MAC-Adresse der Netzwerkkarte des Servers.

```
$ fw_setenv usb_custom_params 'loglevel=7 ⟵
    ↪ netconsole=6666@192.168.1.5/eth0,6666@192.168.1.7/00:13:32:20:r9:a5'
```

Auf dem Server startet man `netcat` wie folgt. Es besteht auch die Möglichkeit mit *U-Boot* zu interagieren: Nach Beginn des Countdowns gelangt man durch mehrmaliges Drücken der Tastenkombination *Strg+j* an den Bootprompt.

```
$ netcat -l -u -p 6666
U-Boot 2011.12 (Feb 12 2012 - 21:33:07)
Seagate FreeAgent DockStar
GNU ld (Sourcery G++ Lite 2009q3-67) 2.19.51.20090709
Hit any key to stop autoboot: 10
 9


 0
u-boot>> help
```

Bleibt das Gerät nach dem Booten im Netzwerk unsichtbar, kann man versuchen mit einer direkt angeschlossenen USB-Tastatur über die Tastenkombination *Strg+Alt+Entf* einen Reboot herbeizuführen. Via `netcat` lässt sich der Bootvorgang dann wie beschrieben aufhalten, um den USB-Stick

[7] Mainline U-Boot bietet diese Funktionalität nicht, bei fehlendem USB-Stick wird der Bootvorgang abgebrochen. Wie man den von Jeff Doozans U-Boot gewohnten Fallback-Mechanismus auf Mainline U-Boot übertragen könnte, habe ich im Forum skizziert.

[8] Unter Mainline U-Boot steht der Befehl `preboot` gegenwärtig nicht zur Verfügung.

[9] Unter Mainline U-Boot ist für diesen Zweck die Variable `optargs` vorgesehen:

```
#/boot/uEnv.txt
optargs=loglevel=7 netconsole=6666@192.168.1.5/eth0,6666@192.168.1.7/00:13:32:20:r9:a5
```

entnehmen und am Linux-PC untersuchen und reparieren zu können[10]. Eingabe von `boot` am *U-Boot* Prompt setzt den Bootvorgang anschließend fort.

[10] So lässt sich, falls vorhanden, das Journal inspizieren:

```
# fsck /dev/sdx2
# mount /dev/sdx2 /mnt
# journalctl -D /mnt/var/log/journal/
```

Dabei könnte z.B. zutage treten, dass die Einbindung ins Netzwerk per DHCP fehlschlägt:

```
Jan 01 01:00:33 alarm dhcpcd[124]: timed out
Jan 01 01:00:33 alarm dhcpcd[124]: exited
Jan 01 01:00:33 alarm ifplugd[96]: client: DHCP IP lease attempt failed on interface 'eth0'
Jan 01 01:00:33 alarm ifplugd[96]: client: Failed to bring the network up for profile 'eth0'
```

Hier wäre dann zumindest vorubergehend eine statische IP-Adresse hilfreich, siehe Abschnitt 2.3.

4 Einrichtung von Arch Linux ARM

Wurde Arch Linux ARM gebootet und Zugang per SSH gewährt, sind noch einige Konfigurationsdateien anzupassen, hier in Kürze einige zuerst von mir vorgenommenen Einstellungen. Die `/etc/fstab` wurde bereits im Rahmen der Vorkonfiguration Abschnitt 2.3 editiert. Bei älteren Installationen notwendige Schritte zur Umstellung auf *systemd* habe ich im Abschnitt 7 zusammengefasst.

- Anpassen der Lokalisierung (de_DE.UTF-8 UTF-8)
- Einstellen der Zeitzone
- Aktivierung des NTP Daemons (um die Zeit im Dauerbetrieb synchron zu halten)
- Update - man beachte dazu Abschnitt 4.2
- Benutzerverwaltung: Anlegen eines Useraccounts, neues Passwort für root.
- SSH absichern: Login für root verbieten, evtl. Passwort-Login deaktivieren.
- Aktivierung der Magic SysRQ Keys.
- Editieren der inputrc, etc.
- Neustart[11] mit *`systemctl reboot`*.

Soll die oben erwähnte Tastenkombination *Strg+Alt+Entf* den DockStar nicht neu booten, sondern lediglich herunterfahren (entsprechend *`systemctl poweroff`*) verlinkt man einfach das zugehörige Target neu.

```
# ln -sf /usr/lib/systemd/system/poweroff.target /etc/systemd/system/ctrl-alt-del.target
```

Je nach Alter des Installationsmediums wird man bei dem erste Update ggf. mit einer Vielzahl an notwendigen manuellen Eingriffen konfrontiert. Die Vorgehensweise ist normalerweise bei den Upstream News dokumentiert, Stichwort *manual intervention required*. Auch nachdem das System komplett eingerichtet ist, sollte man es regelmäßig sichern, aktualisieren und warten, diesbezüglich finden sich im Wiki eine Reihe von wichtigen Hinweisen.

4.1 Virtuelle Datenträger

Verzeichnisse, deren Inhalt nach einem Reboot ohnehin gelöscht bzw. neu erstellt wird, können zur Schonung des USB-Sticks in den RAM verlagert werden. Dazu wird üblicherweise das Dateisystem `tmpfs` eingesetzt, das unter Arch Linux für die Verzeichnisse `/tmp` und `/run` Verwendung findet. Die ehemaligen Verzeichnisse `/var/run` und `/var/lock` sind seit einiger Zeit Links auf `/run` respektive `/run/lock` und somit ebenfalls im RAM verortet.

Das Einbinden des temporären Verzeichnisses `/tmp` erfolgt über eine *Systemd* mount unit `/usr/lib/systemd/system/tmp.mount`. Ein Eintrag in `/etc/fstab` ist optional, etwa um die Maximalgröße (Voreinstellung: 50% des RAM) zu reduzieren. Auch für das Verzeichnis `/var/log` nutze ich `tmpfs`, was sich insbesondere bei Einsatz eines Syslog Daemons wie `syslog-ng` empfiehlt.

```
#/etc/fstab (continued)
tmp      /tmp       tmpfs nodev,nosuid,mode=1777,size=16M  0 0
varlog   /var/log   tmpfs nodev,nosuid,mode=0755,size=8M  0 0
```

[11] Ein Reboot ist normalerweise nur dann notwendig, wenn im Rahmen eines Updates ein neuer Kernel installiert wurde.

Auf aktuellen Systemen erfolgt die Verwaltung des Systemlogs durch `systemd-journald`, folglich gilt `syslog-ng` als obsolet und ist nicht mehr vorinstalliert. Bei älteren Installationen wird man den Dienst normalerweise deaktivieren wollen, um Redundanz zu vermeiden.

```
# systemctl stop syslog-ng.service
# systemctl disable syslog-ng.service
```

Einsehen lässt sich das Journal mit `journalctl`, gespeichert wird es per Voreinstellung unter `/var/log/journal/`. Zur Verlagerung in den RAM ist es nicht notwendig `/var/log/` als `tmpfs` zu mounten, `systemd-journald` stellt dafür eigene Optionen bereit.

```
#/etc/systemd/journald.conf
[Journal]
Storage=volatile
RuntimeMaxUse=8M
```

Das Journal wird nun unter `/run/log/journal/` angelegt, auf `/var/log/` schreiben nurmehr Applikationen, die ihre Meldungen direkt dort ablegen, wie z.B. `pacman`.

4.2 Externe Festplatte

4.2.1 Verzeichnisse einbinden

Die eingangs erwähnte USB-Festplatte ist bereits mit *Ext4* formatiert[12] und bleibt permanent angeschlossen. Sie wird deshalb wie der USB-Stick fest über `/etc/fstab` eingebunden. Neben ihrer eigentlichen Aufgabe als primäres Aufnahmeverzeichnis für den VDR (Abschnitt 6.4) nutze ich die Festplatte zur Erweiterung des Speicherplatzes für das Betriebssystem. Dazu werden Verzeichnisse auf der Festplatte mit der `mount`-Option `--bind` zusätzlich an geeigneter Stelle im Verzeichnisbaum eingebunden. Damit dennoch ein längerer Standby möglich bleibt, berücksichtige ich nur Verzeichnisse, auf die selten zugegriffen wird: Der Paket-Cache, das ABS-Wurzelverzeichnis (Abschnitt 4.3) sowie das `/srv` Verzeichnis. Letzteres beherbergt z.B. von HTTP- und (T)FTP-Server bereitgestellte Inhalte.

```
#/etc/fstab (continued)
LABEL=usbhdd   /hdd                  ext4 defaults 0 0
/hdd/pkg       /var/cache/pacman/pkg none bind     0 0
/hdd/abs       /var/abs              none bind     0 0
/hdd/srv       /srv                  none bind     0 0
```

Alternativ wären natürlich auch symbolische Links oder ein Anpassung der Pfade in `/etc/pacman.conf` und `/etc/abs.conf` möglich. Die Paketdatenbank `/var/lib/pacman/sync` verbleibt auf dem Stick, andernfalls würde jeder Aufruf von `pacman` die Festplatte aus dem Standby holen. Die angegebenen Verzeichnisse müssen, falls nicht bereits vorhanden, vor dem Mounten natürlich angelegt werden.

[12] Externe Festplatten sind ab Werk zumeist mit *NTFS* formatiert. Schreiboperationen unter Linux sind mit ntfs-3g zwar möglich, gehen aber mit außerordentlich hoher CPU-Last einher. Der Versuch auf einem Pogoplug E02 eine *NTFS*-Partition als VDR-Aufnahmeverzeichnis zu verwenden endete mit regelmäßigem Überlauf des Puffers und folglich unbrauchbaren Aufnahmen. Mit einer *Ext4*-Partition ist dagegen auch ein Abspielen von laufenden Aufnahmen völlig unproblematisch.

4.2.2 Standby Timeout einstellen

Die von mir verwendete Festplatte geht nach 30 Minuten ohne Zugriff automatisch in den Standby, andere mit `hdparm -S` eingestellte Werte blieben ohne Wirkung. Um einen früheren Timeout herbeizuführen, wird also ein Programm wie `hd-idle` benötigt. Bei der Konfiguration nutzt man vorzugsweise die ID der Festplatte.

```
#/etc/conf.d/hd-idle
START_HD_IDLE=true
HD_IDLE_OPTS="-i 0 -a /dev/disk/by-id/ata-xxxxxxx -i 600 -l /var/log/hd-idle.log"
```

Mit *`systemctl enable hd-idle.service`* fügt man das Programm den beim Booten automatisch gestarteten Diensten hinzu. Bedarfsweise gelingt mit *`hd-idle -t`* oder *`hdparm -y`* auch ein unmittelbarer Spindown.

4.3 Paketverwaltung

Mit der Paketverwaltung des Systems sollte man, wie bei jeder Linux-Distribution, vertraut sein. Bei Arch Linux ist auch das Erstellen von Software-Paketen relativ leicht. In den folgenden Abschnitten mache ich davon mehrfach gebrauch ohne auf das Vorgehen im Detail einzugehen, deshalb hier ein kurzer Überblick mit obligatorischen Links zum Wiki.

4.3.1 Repositories und Quellpakete

Zweierlei Arten von Paketen muss man unterscheiden:

- Im archlinuxarm-Repository verfügbare Pakete enthalten ausführbare Programme. Die Pakete werden mit `pacman` von einem Mirrorserver heruntergeladen und installiert. Abhängigkeiten werden dabei automatisch aufgelöst. Seit Anfang 2014 kann die Authentizität der Pakete anhand ihrer Signaturen überprüft werden.
- Quellpakete enthalten keine ausführbaren Programme, sondern (mindestens) eine Scriptdatei namens PKGBUILD, welche u.a. die notwendigen Schritte zum Hinunterladen und Kompilieren des Quellcodes beinhaltet. Das Programm `makepkg` führt diese Anweisungen aus und komprimiert das Kompilat zu einem Paket, das dann mit `pacman -U` installiert werden kann. Abhängigkeiten werden dabei nicht automatisch aufgelöst.

Quellpakete haben die Dateiendung `.src.tar.gz`, können aber auch ungepackt in einer Verzeichnistruktur vorliegen. Sie sind dann von Interesse, wenn Pakete aus dem archlinuxarm-Repository angepasst werden sollen oder dort nicht vorhanden sind. Quellpakete kann man von Hand erstellen - Prototypen von PKGBUILDs befinden sich nach Installation von `abs` unter `/usr/share/pacman/` -, meist wird man aber auf vorhandene Pakete zurückgreifen:

- Das Arch Build System (ABS) enthält sämtliche Quellen der für Arch Linux (i686/x86_64) verfügbaren Pakete
- Das Arch User Repository (AUR) enthält inoffizielle Quellpakete für Arch Linux (i686/x86_64)

- Das Git-Repository von Arch Linux ARM enthält die Quellen spezieller ARM-Pakete und solche, die nicht ohne Modifikation aus dem ABS übernommen werden konnten.

Einige Pakete aus dem AUR sind bei ALARM als Binärpakete verfügbar, können also direkt mit `pacman` installiert werden. Die Quellen befinden sich zusätzlich im Git-Repository.

4.3.2 Quellpakete kompilieren

Als Git- und Paketbauverzeichnis bietet sich wiederum die Festplatte an. Vor dem Kompilieren der ABS- und AUR-Pakete ist zumindest das `arch`-Array im PKGBUILD zu editieren, man kann es aber auch einfach mit der `makepkg`-Option `-A` ignorieren. Mitunter sind jedoch weitere Anpassungen an die ARM-Architektur erforderlich.

Da keine Swap-Partition angelegt wurde, macht sich insbesondere im letzten Schritt des Paketbaus, dem Komprimieren des Paketes, mitunter der begrenzte RAM bemerkbar[13]:

```
xz: (stdin): Nicht genügend Hauptspeicher verfügbar
bsdtar: Write error
```

Zumeist genügt es dann Programme mit erhöhtem Bedarf an RAM (wie z.B. `vdradmind`, Abschnitt 6.3.1) vorübergehend zu beenden. Anschließend kann die Zusammenstellung und Komprimierung des betroffenen Paketes mit `makepkg -Rf` erneut veranlasst werden. Eine zusätzliche Maßnahme wäre die Verwendung einer Kompressionsart, die weniger Arbeitsspeicher beansprucht:

```
#/etc/makepkg.conf
PKGEXT='.pkg.tar.gz'
```

Wer die Möglichkeit dazu besitzt, verlagert die Rechenlast per Cross-Compiling auf einen oder mehrere leistungsfähige i686/x86_64-Rechner im Netzwerk[14]. Hier ist zu beachten, dass *distcc* einige Dateien unter `/tmp` ablegt, die ursprüngliche Beschränkung auf 8 MB erwies sich in diesem Zusammenhang als zu knapp bemessen, vgl. Abschnitt 4.1. Alle hier genannten Pakete habe ich auf dem DockStar aber auch ohne Cross-Compiling erstellen können, problematisch war dies nur im laufenden Betrieb.

[13] Wird bereits der Kompiliervorgang mangels RAM unterbrochen, sollte man - nach etwaigen Maßnahmen den RAM betreffend (Abschnitt 4.4) - `makepkg` fortan mit der Option `-e` aufrufen. Andernfalls wird der Quellcode jedesmal erneut entpackt und somit bereits kompilierte Teile der Software gelöscht. Zuvor kann man auch einen etwaigen `./configure`-Schritt im PKGBUILD vorübergehend auskommentieren. Patches dürfen nicht erneut angewendet werden, sind also - sofern sie nicht separat innerhalb der neuen `prepare()`-Funktion zu Anwendung kommen - auszukommentieren, sinnvollerweise ebenso alle weiteren Schritte, die ein Löschen bereits erstellter Binaries nach sich zöge, wie z.B. die standardmäßige Neuerstellung des `build`-Verzeichnisses bei Paketen, deren Quellen aus Versionskontrollsystemen geladen werden. Letzteres lässt sich durch die seit Version 4.1 von `pacman` mögliche Angabe der VCS-Quelle im `source()`-Array bewerkstelligen.

[14] In Version 3.1-9 wurde unter Arch Linux `User=nobody` im `distccd.service` eingefügt. Die Option `--user` wird dadurch ignoriert. Für Arch Linux x86_64 steht ein Quellpaket `distccd-alarm` zur Verfügung, das auf die vorkompilierten `crosstool-ng` toolchains zurückgreift. Es erstellt separate Pakete für ARMv5/6/7-Toolchains mit jeweils eigenen Konfigurations- und Servicedateien für `distccd`.

```
$ git clone git://github.com/WarheadsSE/PKGs
$ cd PKGs/distccd-alarm/
$ makepkg
$ sudo pacman -U distccd-alarm-armv5-4.9.2-1-x86_64.pkg.tar.xz
```

Nach Anpassen von `/etc/conf.d/distccd-armv5` kann der `distccd-armv5.service` gestartet werden.

4.4 Komprimierter RAM

Die Fehlermeldung bei erschöpftem RAM ist selten so eindeutig wie im vorherigen Abschnitt, oft werden Programme, deren Bedarf an RAM nicht gestillt werden kann, einfach beendet. Die Ursache findet man dann mit `dmesg`, hier z.B. bei einer abgebrochenen Datensicherung mit `rsync`:

```
[ 3901.497857] Out of memory: Kill process 1262 (rsync) score 272 or sacrifice child
[ 3901.497872] Killed process 1263 (rsync) total-vm:74980kB, anon-rss:24844kB, ↩
    ↪ file-rss:0kB
```

Zwar könnte man in dieser Situation vorübergehend ein swapfile, vorzugsweise auf der USB-Festplatte, einrichten. Alternativ - oder zusätzlich - gibt es aber auch die Möglichkeit, den verfügbaren RAM besser auszunutzen: Dazu wird mithilfe des Kernelmoduls *zram* eine virtuelle Swap-Partition `/dev/zram0` angelegt und eingebunden. Die in diesen Swap ausgelagerten Daten werden komprimiert und im RAM gespeichert. Die Swap-Partition beansprucht dabei jederzeit nur die den komprimierten Daten entsprechende Menge das RAMs. Im Endeffekt werden also umso mehr Daten im RAM komprimiert, je höher der tatsächliche Bedarf an RAM ist. Der Preis dafür ist eine erhöhte CPU-Belastung.

Das Anlegen und Einbinden des virtuellen Swaps übernimmt das im AUR verfügbare Paket zramswap Pro CPU wird ein zram-Device erzeugt - beim DockStar also eines -, die zusammen maximal eine 20% des RAMs entsprechende Menge an unkomprimierten Daten aufnimmt. Dieser Wert lässt sich über eine zusätzliche Servicedatei beeinflussen:

```
#/etc/systemd/system/zramswap.service.d/size.conf
[Service]
Environment=”ZRAM_SIZE=50”
```

Mit *`systemctl daemon-reload`* wird die Änderung übernommen, *`systemctl enable zramswap.service`* fügt `zramswap` den beim Booten automatisch gestarteten Diensten hinzu.

4.5 Ansteuerung der LEDs

Die Steuerung der LEDs über das `/sys`-Verzeichnis erfordert root-Rechte, soll möglichst aber auch anderen Nutzern offenstehen. Der direkte Einsatz von sudo ist in diesem Zusammenhang etwas umständlich. Ich habe deshalb ein einfaches C-Programm geschrieben, das sich auf herkömmliche Weise mit `sudo` ausführen lässt. Auch kann es einfach innerhalb eines Scriptes genutzt werden, das seinerseits mit `sudo` aufgerufen wurde (Abschnitt 6.4). Das Programm wird mit `gcc`, der im gleichnamigen Paket enthalten ist, kompiliert:

```
$ gcc -o led led.c
```

Es manipuliert jeweils nur eine der beiden LEDs und ermittelt diese anhand des Namens, unter dem das Programm aufgerufen wurde. Deshalb werden zwei Dateien erstellt, die auf denselben Inode zeigen (Hardlink):

```
# cp led /usr/local/bin/led_green
# ln /usr/local/bin/led_green /usr/local/bin/led_orange
```

Ohne Parameter werden verfügbare Zustände und aktueller Status angezeigt. Als erster Parameter kann der gewünschte Status angegeben werden. Der Status `timer` benötigt zusätzlich die Parameter `delay_on` und `delay_off`, hier ein Beispiel:

```
# led_green
none nand-disk timer ide-disk heartbeat [default-on]
# led_green timer 1000 1000
```

Um die LEDs beim Herunterfahren zum Erlischen zu bringen habe ich folgenden Service implementiert.

```
#/etc/systemd/system/leds-off.service
[Unit]
Description=Switch off LEDs before poweroff
DefaultDependencies=no
Before=poweroff.target
After=umount.target

[Service]
Type=oneshot
ExecStart=/usr/local/bin/led_green none ; /usr/local/bin/led_orange none

[Install]
WantedBy=poweroff.target
```

4.6 Kompilierung des Kernels

Zur Zeit gibt es für mich keine Veranlassung einen angepassten Kernel zu erstellen: Das Aufs-Dateisystem wurde unlängst in den ALARM-Kernel integriert und seit der Berücksichtigung einer größeren Anzahl von DVB-Treibern in Version 3.1.10-11 läuft auch mein DVB-Stick von Freecom out of the box. Anhand eines anderen Sticks beschreibe ich in Abschnitt 5, wie man auch ohne Kernelkompilation an die neuesten DVB-Treiber gelangt.

4.6.1 Konfiguration

Wer eine Anpassung der Kernelkonfiguration vornehmen will, kann sich an den Artikel Kernel Compilation im Wiki halten. Das zu modifizierende Kernelpaket befindet sich jedoch nicht im ABS, sondern im Git-Repository, siehe Abschnitt 4.3, nachfolgend das zuletzt von mir verwendete, überarbeitete Paket. Anders als im ALARM-Kernel werden Patches und Quelldateien für das Aufs-Dateisystem direkt aus dem Aufs-Git-Repository heruntergeladen.

- linux-custom.tar.gz (*discontinued*)

Um die Kompilierzeit zu reduzieren, verwende ich im PKGBUILD das `make`-Target `localmodconfig`. Es deaktiviert in der Kernelkonfiguration sämtliche Module, die momentan nicht geladen sind. Als Basis dient die Konfiguration des laufenden Kernels aus `/proc/config.gz`. Man sollte also vor Aufruf

von `makepkg` sämtliche Hardware, die man nutzen möchte, anschließen (ggf. nacheinander) und in Betrieb nehmen, sowie Programme und Daemons starten, die auf Kernelmodule angewiesen sind (z.B. der NFS Server). Im Anschluss werden neu hinzugekommene Kerneloptionen einzeln abgefragt. Man kann hier die Vorgaben zunächst bestätigen und hat im letzten, durch `make menuconfig` ausgelösten Schritt Gelegenheit in einer textbasierten Oberfläche die Konfigurationen zu überarbeiten. Hilfe bei der Auswahl benötigter Optionen bietet das Buch Linux Kernel in a Nutshell insbesondere Kapitel 7. Ein daraus abgeleitetes praktisches Beispiel folgt auch in Abschnitt 5.1.

4.6.2 Installation

Ist der Kernel konfiguriert, bestätigt man das Speichern der `.config`-Datei. Nach dem Kompilieren entstehen die Pakete `linux-custom` und `linux-custom-headers`. Sie lassen sich parallel zu `linux` und `linux-headers` installieren, das Kernelimage befindet sich unter `/boot/uImage-custom`. Um dieses Image zu booten, kann man behelfsweise `/boot/uImage` damit ersetzen:

```
# mv /boot/uImage /boot/uImage-alarm
# cp /boot/uImage-custom /boot/uImage
```

Sollte nach einem Reboot der eigene Kernel nicht starten, stellt man das ursprüngliche Image wieder her (den Stick dazu an einen anderen PC anschließen) und kann so wieder den offiziellen Kernel nutzen. Ein unbeabsichtigtes Überschreiben von `/boot/uImage` durch ein Update - `pacman` geht ja davon aus, das diese Datei zum Paket `linux` gehört - lässt sich mit der Option `NoUpgrade` vermeiden, die neue Datei wird dann mit der Endung `.pacnew` versehen. Alternativ kann man natürlich auch die Kernelpakete vom Update ausschließen:

```
#/etc/pacman.conf
NoUpgrade = boot/uImage
#IgnorePkg = linux linux-headers
```

5 Inbetriebnahme des DVB-Sticks

Wie oben erwähnt ist mittlerweile eine größere Anzahl von DVB-Treibern im ALARM-Kernel enthalten. Ob der DVB-Stick erkannt wird, lässt sich der Ausgabe von `dmesg` entnehmen.

```
usb 1-1.4: new high speed USB device number 7 using orion-ehci
dvb-usb: found a 'WideView WT-220U PenType Receiver (Typhoon/Freecom)' in cold ⟵
    ↪ state, will try to load a firmware
dvb-usb: did not find the firmware file. (dvb-usb-wt220u-02.fw) Please see ⟵
    ↪ linux/Documentation/dvb/ for more details on firmware-problems. (-2)
```

In diesem Fall kann man gleich mit Abschnitt 5.2 fortfahren. Ist dagegen kein passender Treiber verfügbar, meldet `dmesg` lediglich ein unbekanntes USB-Gerät

```
usb 1-1.4: new high speed USB device number 7 using orion-ehci
```

Zur genaueren Identifizierung dienen Hersteller- und Produkt-ID laut `lsusb` (Ausgabe hier beschränkt auf das neue Gerät an Bus 1, Device 7):

```
# lsusb -s 1:7
Bus 001 Device 007: ID 2040:1605 Hauppauge
```

Es handelt sich hier um den Hauppauge WinTV HVR 930C, der im folgenden als konkretes Beispiel dienen soll. Er wird zwar seit Kernel 3.3 unterstützt[15], Arch Linux ARM verwendete aufgrund von Problemen mit *U-Boot* bis Anfang 2014 jedoch per Voreinstellung Version 3.1 des Kernels. Erfreulicherweise können die aktuellen (und künftigen) Kerneltreiber für TV-/Videogeräte aber aus dem v4l-dvb Repository von linuxtv.org installiert werden.

Hinweis: Mit dem alternativen - im Kirkwood Image auch vorinstallierten - Kernelpaket `linux-kirkwood` ist unlängst auch ein aktueller Kernel verfügbar. Damit erübrigt sich bei diesem Stick eine Installation von `v4l-dvb`. Der Einsatz eines Kernels jenseits von Version 3.1 setzt bei älteren Installationen ggf. ein vorheriges Update des Bootloaders voraus. Auch `linux-kirkwood` kann mittels `v4l-dvb` um neuere oder noch nicht in den Kernel integrierte Treiber bereichert werden.

Im AUR befindet sich ein Paket, das sämtliche v4l-dvb Treiber installiert. Um die Auswahl auf die tatsächlich benötigten Treiber beschränken zu können, habe ich das PKGBUILD angepasst.

- v4l-dvb-git.tar.gz

Vor dem Bau des Paketes ist als root einmalig folgender Befehl abzusetzen:

```
# make -C /usr/src/linux-$(uname -r)/ menuconfig
```

Das erscheinende Menü beendet man sogleich über den Button `<Exit>`. Nach Aufruf von `makepkg` (als normaler User) wird man dann mit einem auf die Multimediatreiber reduzierten Menü konfrontiert. Sind die zu aktivierenden Optionen nicht ersichtlich, findet man im folgenden Abschnitt einige Hilfestellungen. Wer den Aufwand scheut und eine längere Kompilierzeit hinnimmt, verwendet einfach das unmodifizierte Paket aus dem AUR.

[15] Das gilt nicht für das Nachfolgemodell HVR-930C-HD.

Hinweis: Nutzt man am DockStar weitere Videogeräte (z.B. eine Webcam), sollten im Zweifelsfall auch deren Module über die `v4l-dvb`-Treiber aktualisiert werden, da es andernfalls zu Versionskonflikten der Kernelmodule kommen kann.

Das Paket `v4l-dvb-git` muss nach jedem Kernelupdate erneut kompiliert und installiert werden. Dies kann bereits vor dem Booten des neuen Kernels geschehen, im PKGBUILD muss dazu lediglich die Variable `_kernver` explizit auf die neue Version gesetzt werden.

```
#_kernver="$(uname -r)"     # build for running kernel
_kernver="3.1.10-34-ARCH" # build for different kernel
```

5.1 Ermittlung des Treibers

Ist das benötigte Modul nicht bekannt, sucht man im Quellcode nach jenen Dateien, die unabhängig von Groß-/Kleinschreibung Hersteller- und Produkt-ID innerhalb einer Zeile enthalten.

```
$ cd src/media_build/
$ grep -R -i "2040.*1605"
linux/drivers/media/usb/em28xx/em28xx-cards.c:  { USB_DEVICE(0x2040, 0x1605),
```

Dem Pfad zur Quelldatei ist unschwer der Name des zugehörigen Moduls entnehmen, in diesem Beispiel `em28xx`. Gibt es keine Fundstelle, beschränkt man sich zunächst auf die Hersteller-ID und sucht in den gefundenen Dateien dann nach der Produkt-ID[16]. Anschließend durchforschen wir gezielt die `Makefiles` nach dem Modulnamen (Unterstrich ggf. durch Minuszeichen ersetzen) wobei uns nur Zeilen interessieren, in denen die Verbindung zu einer CONFIG-Variablen hergestellt wird:

```
$ find -type f -name Makefile | xargs grep "CONFIG.*em28xx"
./linux/drivers/media/usb/em28xx/Makefile:obj-$(CONFIG_VIDEO_EM28XX) += em28xx.o
./linux/drivers/media/usb/em28xx/Makefile:obj-$(CONFIG_VIDEO_EM28XX_ALSA) += ↩
    ↪ em28xx-alsa.o
```

[16] Hier ein Beispiel für meinen Freecom-Stick (14aa:0222):

```
$ grep -R -l -i 14aa | xargs grep -l 0222
linux/drivers/media/dvb-core/dvb-usb-ids.h
```

In der gefundenen Datei werden die IDs offenbar zunächst durch symbolische Namen ersetzt, wobei die Produkt-ID doppelt vergeben ist.

```
$ grep -i -e 14aa -e 0222 linux/drivers/media/dvb-core/dvb-usb-ids.h
#define USB_VID_WIDEVIEW                        0x14aa
#define USB_PID_WT220U_COLD                             0x0222
#define USB_PID_PCTV_450E                               0x0222
```

Also wiederholen wir die vorherige Suche, nun aber mit diesen Namen.

```
$ grep -R -l USB_VID_WIDEVIEW | xargs grep -l -e USB_PID_WT220U_COLD -e USB_PID_PCTV_450E
linux/drivers/media/dvb-core/dvb-usb-ids.h
linux/drivers/media/usb/dvb-usb/dtt200u.c
```

In den Makefiles findet man über den Dateinamen (ohne Endung) dann die CONFIG-Variable.

```
$ find -type f -name Makefile | xargs grep "CONFIG.*dtt200u"
./linux/drivers/media/usb/dvb-usb/Makefile:obj-$(CONFIG_DVB_USB_DTT200U) += dvb-usb-dtt200u.o
```

```
./linux/drivers/media/usb/em28xx/Makefile:obj-$(CONFIG_VIDEO_EM28XX_DVB) += em28xx-dvb.o
./linux/drivers/media/usb/em28xx/Makefile:obj-$(CONFIG_VIDEO_EM28XX_RC) += em28xx-rc.o
./linux/drivers/media/usb/Makefile:obj-$(CONFIG_VIDEO_EM28XX) += em28xx/
```

Wir müssen also für sorgen, dass die Variable CONFIG_VIDEO_EM28XX_DVB gesetzt wird. Im Konfigurationsdialog gelangen wir durch Eingabe eines Schrägstrichs (/) in eine Suchmaske und geben dort die Variable ein. Das Ergebnis liefert den gesuchten Prompt und beschreibt, wie man dorthin gelangt:

```
Symbol: VIDEO_EM28XX_DVB [=n]
Type  : tristate
Prompt: DVB/ATSC Support for em28xx based TV cards
  Defined at ./Kconfig:2000
  Depends on: MEDIA_SUPPORT [=n] && MEDIA_USB_SUPPORT [=n] && (MEDIA_CAMERA_SUPPORT ↩
      ↪ [=n] || MEDIA_ANALOG_TV_SUPPORT [=n] || MEDIA_DIGITAL_TV_SUPPORT [=n]) && ↩
      ↪ VIDEO_EM28XX [=n] && DVB_CORE [=n]
  Location:
    -> Multimedia support (MEDIA_SUPPORT [=n])
      -> Media USB Adapters (MEDIA_USB_SUPPORT [=n])
        -> Empia EM28xx USB video capture support (VIDEO_EM28XX [=n])
```

Folgt man dem angegebenen Pfad, sind einige Untermenüs nicht unmittelbar zugänglich. Ursächlich sind weitere nach *Depends on* aufgelistetete Abhängigkeiten. Die hier zusätzlich benötigten Optionen *Analog TV support* und *Digital TV support* sollten aber leicht zu identifizieren sein.

```
<M> Multimedia support  --->
  [*]   Analog TV support
  [*]   Digital TV support
  [*]   Remote Controller support
  <M>   Compile Remote Controller keymap moduless (NEW)
  [*]   Media USB Adapters  --->
    <M>   Empia EM28xx USB devices support
    <M>     Empia EM28xx analog TV, video capture and/or webcam support
    <M>     Empia EM28xx ALSA audio module
    <M>     DVB/ATSC Support for em28xx based TV cards
    <M>     EM28XX Remote Controller support (NEW)
  [*]   Autoselect ancillary drivers (tuners, sensors, i2c, frontends) (NEW)
```

Die Unterstützung für die Fernbedienung (vorletzte Zeile) wird nur dann angeboten, wenn im übergeordneten Menü *Remote Controller support* ausgewählt wurde. Via <Exit> beendet man alle Untermenüs und bestätigt mit <Yes> das Speichern der config-Datei. Ist der sich anschließende Kompiliervorgang erfolgreich, wird das Paket wie üblich mit pacman installiert.

Vor dem Anschließen des DVB-Sticks sollte man aufgrund der o.g. Modulversionsproblematik bereits verwendete TV-/Videogeräte und die zugehörigen Module ggf. entfernen[17]. Natürlich kann der DockStar stattdessen auch einfach neu gebootet werden.

[17] Andernfalls finden sich via dmesg z.B. solche Fehlermeldungen:

```
usb 1-1.4: USB disconnect, device number 8
usb 1-1.4: new high speed USB device number 9 using orion-ehci
v4l2_common: disagrees about version of symbol v4l2_subdev_init
```

5.2 Firmware

Wenn der Kernel den DVB-Stick unterstützt oder wie oben beschrieben ein passender Treiber nachinstalliert wurde, sollte `dmesg` nach dem (erneuten) Anschließen ausführliche Informationen zu dem Stick liefern. Dabei achte man auf Hinweise betreffs fehlender Firmwaredateien:

```
drxk: Could not load firmware file dvb-usb-hauppauge-hvr930c-drxk.fw.
drxk: Copy dvb-usb-hauppauge-hvr930c-drxk.fw to your hotplug directory!
```

Eine Internetsuche nach dem Dateinamen führt dann entweder direkt zu einem Download der Firmware oder wie in diesem Fall zu einer Anleitung wie diese aus dem Windows-Treiber zu extrahieren ist, siehe Abschnitt 1) im o.g. Link. Die Datei legt man dann im Verzeichnis `/usr/lib/firmware/` ab. Der HVR-930C benötigt eine weitere Firmwaredatei, angefordert wird diese laut `dmesg` aber erst beim Senderscan (s.u.).

```
xc5000: waiting for firmware upload (dvb-fe-xc5000-1.6.114.fw)...
xc5000: firmware read 12401 bytes.
xc5000: firmware uploading...
xc5000: firmware upload complete...
```

Diese Datei ist offensichtlich schon vorhanden, sie gehört zum Paket `linux-firmware`, das als Abhängigkeit des Kernelpakets bereits installiert sein sollte.

Hinweis: Ende 2014 gelang es mir nach einem Reboot unter Kernel 3.1.10 nicht mehr, den HVR-930C mit den `v4l-dvb` Treibern in Betrieb zu nehmen, die vorhandene Firmware `dvb-usb-hauppauge-hvr930c-dr` konnte nicht geladen werden (siehe Fehlermeldung oben). Deshalb bin ich nun auf `linux-kirkwood` umgestiegen, der Stick arbeitet damit jedoch nicht zuverlässig, gelegentlich ist ein Ent- und Neuladen der DVB-Treiber erforderlich, siehe Abschnitt 6.1.

5.3 Sendersuche

Hiesige TV-Sender lassen sich mit dvbv5-scan aus dem Paket `linuxtv-dvb-apps` ermitteln. Das Programm benötigt eine Datei mit den lokal gültigen Frequenzen (*initial-tuning-data-file*). Dazu kann man das Paket `dtv-scan-tables-git` aus dem AUR installieren oder die passende Datei einfach direkt aus dem Repository herunterladen.

Um einen Sendersuchlauf mit dem älteren Programm `scan` durchführen zu können, muss die benötigte Frequenztabelle zuvor mit `dvb-format-convert` aus dem Paket `v4l-utils` in das DVBv3-Format umgewandelt werden. Die folgenden Beispiele beziehen sich auf Kabelfernsehen (DVB-C)[18].

```
v4l2_common: Unknown symbol v4l2_subdev_init (err -22)
v4l2_common: disagrees about version of symbol v4l2_device_register_subdev
v4l2_common: Unknown symbol v4l2_device_register_subdev (err -22)
v4l2_common: disagrees about version of symbol v4l2_ctrl_fill
v4l2_common: Unknown symbol v4l2_ctrl_fill (err -22)
```

[18] Durch den Wegfall der Grundverschlüsselung sind seit 2013 erfreulicherweise auch private Sender zumindest in SD unproblematisch zu empfangen. Die Zahl der in HD ausgestrahlten öffentlich-rechtlichen Sender ist Anfang 2014 auf ca. 15 gestiegen.

```
# dvb-format-convert -I dvbv5 -O channel ↩
    ↪ /usr/share/dvb/dvb-c/de-Kabel_Deutschland-Hannover initial-tuning-data-file
# scan -x 0 -t 1 initial-tuning-data-file > channels.conf
```

Eine Alternative ist w_scan, das ohne Frequenzdateien auskommt, im Vergleich zu `scan` aber ein Vielfaches der Zeit benötigt.

```
# w_scan -fc -cDE -X -E0 -R0 -O0 > channels.conf
```

Die `linuxtv-dvb-apps` bieten eine primitive Möglichkeit zur Aufnahme von Sendungen, dazu wird zunächst mit `czap` (bzw. `tzap` oder `szap`) im Hintergrund (z.B. via `screen`) ein Sender eingestellt

```
# czap -c channels.conf -r ”Das Erste”
```

Der Sendername muss exakt dem Format in der `channels.conf` entsprechen. Den Datenstrom leitet man dann einfach in eine Datei (Abbruch mit *Strg+C*).

```
# cat /dev/dvb/adapter0/dvr0 > recording.ts
```

Komfortabler geht es mit dem VDR, mehr dazu in Abschnitt 6.

5.4 Fernbedienung

Der im HVR-930C integrierte IR-Empfänger wird in `/proc/bus/input/devices` als Eingabegerät (HID) aufgeführt (Ausgabe gekürzt)[19]:

```
I: Bus=0003 Vendor=2040 Product=1605 Version=0001
N: Name=”em28xx IR (em28xx #0)”
H: Handlers=kbd event0
```

Folglich stehen die Chancen gut, dass sich die Fernbedienung mit dem `devinput`-Treiber von LIRC in Betrieb nehmen lässt. Das benötigte Paket `lirc-utils` deckt eine Vielzahl von Fernbedienungen ab und setzt deshalb weitere Pakete voraus, die eigentlich nicht benötigt werden. Zur Kompilierzeit lässt sich LIRC aber durchaus auf einen einzelnen Treiber beschränken. Ich habe das PKGBUILD aus dem ABS entsprechend angepasst.

- lirc-utils-devinput.tar.gz

[19] Fehlt ein solcher Eintrag wird eventuell das im Abschnitt 5.1 ausgewählte Modul `em28xx-rc` nicht automatisch geladen. Das holen wir einfach mit `modprobe em28xx-rc` nach und überprüfen ggf. das Resultat mit `dmesg`:

```
Registered IR keymap rc-hauppauge
input: em28xx IR (em28xx #0) as /devices/platform/orion-ehci.0/usb1/1-1/1-1.4/rc/rc0/input0
rc0: em28xx IR (em28xx #0) as /devices/platform/orion-ehci.0/usb1/1-1/1-1.4/rc/rc0
Em28xx: Initialized (Em28xx Input Extension) extension
```

Um sich das manuelle Laden des Moduls künftig zu ersparen, kann man es in eine Datei `/etc/modules-load.d/em28xx-rc.conf` eintragen. Oder man sorgt dafür, dass es immer dann geladen wird, wenn `em28xx` angefordert wird.

```
#/etc/modprobe.d/em28xx.conf
install em28xx /sbin/modprobe --ignore-install em28xx $CMDLINE_OPTS ; /sbin/modprobe em28xx-rc
```

Speziell für diesen Zweck gibt es neuerdings auch das Kommando `softdep`, siehe `man modprobe.d`.

Die passende Konfigurationsdatei `/etc/lirc/lircd.conf` ist in diesem Paket bereits integriert. Wer `lirc-utils` nutzt, muss sie hingegen noch an die richtige Stelle kopieren:

```
# cp /usr/share/lirc/remotes/devinput/lircd.conf.devinput /etc/lirc/lircd.conf
```

Zur Konfiguration wird ferner das Device des IR-Empfängers benötigt, laut `/proc/bus/input/devices` ist dies `/dev/input/event0`, langfristig wird man aber einen von `udev` erzeugten, eindeutigen Link auf das Device nutzen wollen:

```
#/etc/udev/rules.d/99-lirc.rules
SUBSYSTEM==”input”, ATTRS{idVendor}==”2040”, ATTRS{idProduct}==”1605”, ↩
    ↪ SYMLINK=”input/ir”
```

Mit *`udevadm trigger --subsystem-match=input`* wird `udev` veranlasst, die Datei nachträglich auszuwerten und den Symlink zu erzeugen. Damit Device und Treiber vom LIRC-Daemon berücksichtigt werden, muss die Servicedatei angepasst werden - eine separate Konfigurationsdatei gibt es zur Zeit nicht, siehe Bugreport.

```
#/etc/systemd/system/lirc.service.d/device.conf
[Service]
ExecStart=
ExecStart=/usr/bin/lircd --driver=devinput --device=/dev/input/ir
```

Ein Aufruf von *`systemctl daemon-reload`* macht die Änderung wirksam. Bei Verwendung von `lirc-utils-devinput` sind `devinput` als Treiber und `/dev/input/event0` als Device voreingestellt. so dass sich diese Anpassung ggf. erübrigt.

Nach Start des LIRC-Daemons mit *`systemctl start lirc.service`* findet man mit `irw` die Tastencodes heraus[20]. In einer lircrc-Datei können sie bestimmten Aktionen zugeordnet werden[21]. Betreffs der Steuerung des VDR siehe Abschnitt 6.

[20] Bleibt das Betätigen von Tasten auf der Fernbedienung wirkungslos, wurde evtl. bei der Konfiguration von `v4l-dvb` die Option *Compile Remote Controller keymap modules* nicht berücksichtigt. Die keycode/scancode-Tabelle muss dann mit `ir-keytable` aus dem Paket `v4l-utils` manuell zugeordent werden.

```
# ir-keytable
Found /sys/class/rc/rc0/ (/dev/input/event0) with:
        Driver em28xx, table rc-empty
        Supported protocols: NEC RC-5 RC-6
        Enabled protocols:
        Name: em28xx IR (em2884 #0)
        bus: 3, vendor/product: 2040:1605, version: 0x0001
        Repeat delay = 500 ms, repeat period = 125 ms
# ir-keytable -w /usr/lib/udev/rc_keymaps/hauppauge
Read hauppauge table
Wrote 172 keycode(s) to driver
Protocols changed to RC-5
```

[21] Beispielsweise könnte der Power-Knopf genutzt werden, um via `irexec` einen anderen Rechner per Wake-on-LAN zu booten:

```
#/etc/lirc/lircrc
begin
button = KEY_POWER
prog   = irexec
repeat = 0
```

```
# irw
0000000080010074 00 KEY_POWER devinput
```

Die Codes lassen sich auch - unabhängig von LIRC - mit `evtest` ermitteln. Im Gegensatz zu `irw` listet es zunächst sämtliche Codes auf, die der IR-Empfänger erzeugen kann. Beim HVR-930C sind dies mehr als die mitgelieferte Fernbedienung auszulösen vermag.

```
config = wol 00:11:22:33:44:55
end
```

6 Einrichtung des VDR

Aktuelle Quellpakete für den VDR und eine Vielzahl an Plugins findet man

- im AUR
- im git-Repository des VDR4Arch Projektes, das neuerdings auch Binärpakete für ARM bereitstellt.

Diese sind jedoch relativ komplex und ziehen z.T. weit reichende Paketabhängigkeiten nach sich, um Funktionen zu realisieren, die auf dem DockStar nicht ohne weiteres genutzt werden können. Ich verwende deshalb eigene, auf den gegebenen Einsatzzweck abgestimmte Pakete, deren Quellen ich hier wie in den vorangegangenen Abschnitten verlinke. Zusätzlich stelle ich aber auch die von mir kompilierten Binärpakete in einem Repository zur Verfügung. Wer es nutzen will, fügt es einfach der Konfigurationsdatei von `pacman` hinzu.

```
#/etc/pacman.conf
[vdr]
Server = http://linux.bplaced.net/repo/arm
```

Nach einer Synchronisierung der Datenbank mit *`pacman -Sy`* listet *`pacman -Sl vdr`* die enthaltenen Pakete auf. Diese installiert und aktualisiert man dann wie üblich, Abhängigkeiten werden natürlich aufgelöst. Es wird allerdings zusätzlich ein Font benötigt.

```
# pacman -S noad vdr-live vdr-epgsearch vdr-streamdev-server ttf-bitstream-vera
```

Wurde die Überprüfung der Paketsignaturen eingerichtet, muss auch mein öffentlicher Schlüssel in den *pacman*-Schlüsselbund aufgenommen und lokal signiert werden. Die erforderlichen Schritte sind im Wiki beschrieben. Ihr könnt aber auch das Paket vdr-keyring herunterladen und mit `pacman -U` installieren. In jedem Fall sollte anschließend mit `pacman-key -f` der Fingerabdruck abgeglichen werden:

- Key ID: B5B41E7B
- Fingerprint: 72B3 6FCF 991F FC8D CD08 5EE1 D010 F471 B5B4 1E7B
- Owner: Olaf Bauer

Obwohl unter Arch Linux derzeit unüblich habe ich mittlerweile auch die Paketdatenbank `vdr.db` signiert. Andernfalls liefert der Server, auf dem die Pakete gehostet sind, eine Fehlerseite aus, die als Signatur fehlinterpretiert wird:

```
Fehler: GPGME error: Keine Daten
Fehler: Konnte vdr nicht aktualisieren (Ungültige oder beschädigte Datenbank ↩
    ↪ (PGP-Signatur))
Fehler: Datenbank 'vdr' ist nicht gültig (Ungültige oder beschädigte Datenbank ↩
    ↪ (PGP-Signatur))
```

Falls diese Fehlermeldung auftritt, muss die irrtümlich als `vdr.db.sig` gespeicherte HTML-Seite wieder entfernt werden:

```
# rm /var/lib/pacman/sync/vdr.db.sig
# pacman -Syy
```

Eine andere Möglichkeit besteht darin die Datenbanksignaturprüfung in `/etc/pacman.conf` global oder nur für das vdr-Repository mit `SigLevel = Required DatabaseNever` auszuschalten.

6.1 Konfiguration

Fürs erste genügt es, lediglich das Paket `vdr` ohne weitere Plugins zu installieren. Auch die normalerweise integrierten Demo-Plugins werden nicht benötigt und sind deshalb in dem Paket nicht enthalten.

- vdr.tar.gz

Von den Voreinstellungen abweichende Optionen (siehe `man vdr`) können in der Konfigurationsdatei `/etc/conf.d/vdr` eingetragen werden. Um VDR mit eingeschränkten Nutzerrechten[22] zu betreiben und die regelmäßige Abspeicherung der EPG-Daten zu unterbinden[23], habe ich folgende Optionen hinzugefügt:

[22] Generell sollten Dienste nur dann mit Rootrechten ausgestattet sein, wenn dies unbedingt erforderlich ist. Für einen dedizierten Account lassen sich darüberhinaus Disk Quota einrichten. Diese nutze ich, um via `warnquota` per E-Mail informiert zu werden, wenn VDR-Aufnahmen die Homepartition zu füllen drohen (Soft-Limit).

[23] Per Voreinstellung speichert VDR die EPG-Daten alle 30 Minuten in `/video/epg.data`. Diese Datei wird von VDR selbst nicht benötigt und ihre regelmäßige Aktualisierung würde den USB-Stick nur unnötig belasten bzw. die Festplatte regelmäßig wecken. Fehlt die Datei, stehen allerdings die EPG-Daten nach einem Neustart des VDR (der z.B. dann erforderlich ist, wenn man ein Plugin ausprobieren möchte) nicht sofort wieder zur Verfügung. Als Kompromiss kann man die Datei in den RAM verlegen (siehe Abschnitt 4.1):

```
#/etc/conf.d/vdr (revised)
VDROPTIONS='... -E /tmp ...'
```

Um die Datei `/tmp/epg.data` auch über einen Reboot hinaus zu bewahren, verwende ich einen custom service.

```
#/etc/systemd/system/epg.service
[Unit]
Description=Save and restore /tmp/epg.data
After=local-fs.target
Before=vdr.service
IgnoreOnIsolate=yes

[Service]
Type=oneshot
ExecStart=/usr/bin/cp -au /home/vdr/epg.data /tmp
ExecStop=/usr/bin/cp -au /tmp/epg.data /home/vdr
TimeoutSec=0
RemainAfterExit=yes
StandardOutput=syslog

[Install]
WantedBy=multi-user.target
```

Das Backup von `epg.data` muss zunächst einmalig manuell angefertigt werden, andernfalls schlägt der ExecStart-Befehl fehl und der Service beendet sich vorzeitig.

```
# systemctl stop vdr
# cp -au /tmp/epg.data /home/vdr/
```

```
#/etc/conf.d/vdr
VDROPTIONS='... -E- -u vdr'
```

Nötigenfalls sind in dieser Datei auch die Funktionen zum (Ent-)Laden der DVB-Treiber zu implementieren[24].

Soll VDR über eine am DockStar angeschlossene Fernbedienung gesteuert werden, fügt man noch die Option `--lirc` hinzu, erstellt im Verzeichnis `/etc/vdr/` eine Datei remote.conf und passt darin die Keycodes (zweite Spalte) an, vgl. Abschnitt 5.4.

Der User `vdr` muss noch angelegt werden. Als Mitglied der Gruppe `video` erhält er Zugriff auf die DVB-Devices unter `/dev/dvb/adapter0/`. Des weiteren benötigt er Schreibrecht im Konfigurations- und Cacheverzeichnis.

```
# useradd -m -G video -s /usr/bin/nologin vdr
# chown -R vdr.vdr /etc/vdr /var/cache/vdr/ /usr/share/vdr
```

VDR legt Aufnahmen per Voreinstellung unter `/video` ab. Vorerst hängen wir dort das Homeverzeichnis des Users `vdr` ein. Dieser besitzt dort bereits die benötigten Schreibrechte. Zusätzlich erhalten andere Nutzer Lesezugriff.

```
# chmod a+rx /home/vdr/
# mkdir /video && mount -o bind /home/vdr/ /video
```

```
# systemctl start epg
# systemctl start vdr
# systemctl enable epg.service
```

Mit `systemctl status epg` überzeugt man sich nun, dass der epg-Service aktiv ist und folglich beim Shutdown den ExecStop-Befehl ausführen wird.

[24] Für den HVR-930C habe ich die Funktionen wie folgt implementiert

```
#/etc/conf.d/vdr (continued)
# Detect whether the DVB driver is already loaded
# and return 0 if it *is* loaded, 1 if not:
function DriverLoaded()
{
  grep -qse em28xx_dvb /proc/modules
}

# Load all DVB driver modules needed for your hardware:
function LoadDriver()
{
  modprobe em28xx_dvb
}

# Unload all DVB driver modules loaded in LoadDriver():
function UnloadDriver()
{
  modprobe -r em28xx_dvb em28xx_alsa em28xx_rc rc_hauppauge rc_core xc5000 drxk dvb_core em28xx_v4l ↩
      ↪ em28xx
}
```

Erforderlich wurde dies erst nach dem Umstieg auf Kernel 3.18 aus dem Paket `linux-kirkwood`. Es kam gelegentlich zu leeren Aufnahmen, VDR vermeldet dann im Journal "ERROR: video data stream broken" und beendet sich mit negativem Status, der ein Neustart von VDR erwirkt. Der nun damit einhergehende Reload der DVB-Treiber hat als Notbehelf das Problem einstweilen beseitigt.

Soll das Homeverzeichnis dauerhaft für Aufnahmen dienen, erstellt man einen entsprechenden Eintrag in der `fstab`.

```
#/etc/fstab (continued)
/home/vdr  /video  none  bind     0 0
```

Je nach Region und DVB-Übertragungsweg muss die mitgelieferte DVB-S Kanalliste noch ersetzt werden, z.B. mit den `linuxtv-dvb-apps`

```
# scan -x 0 -t 1 -o vdr initial-tuning-data-file > /etc/vdr/channels.conf
```

oder mit `w_scan`, siehe Abschnitt 5.3.

```
# w_scan -fc -cDE -E0 -R0 -O0 > /etc/vdr/channels.conf
```

Anschließend wird VDR gestartet.

```
# systemctl start vdr.service
```

Im Problemfall nützliche Logmeldungen des VDR erscheinen im Journal[25]. Läuft alles zur Zufriedenheit, wird man VDR den beim Booten automatisch gestarteten Diensten hinzufügen wollen.

```
# systemctl enable vdr.service
```

Zur ersten Kontaktaufnahme kann man `svdrpsend` einsetzen.

6.2 Plugins

Bei der Installation von Plugins werden unter `/etc/vdr/plugins` weitere Konfigurationsdateien angelegt. Die Zugriffsrechte müssen dann erneut angepasst werden, auch nach einem Update bereits installierter Plugins:

```
# chown -R vdr.vdr /etc/vdr
```

Außerdem müssen die Plugins explizit aktiviert werden, ggf. mit eigenen Optionen:

```
#/etc/conf.d/vdr (continued)
VDRPLUGINS='-Plive -Pepgsearch -Pstreamdev-server -P"xineliboutput --primary ↩
    ↪ --local=none --remote=37890" -Pxvdr -Pvnsiserver -P"graphlcd -d ax206dpf"'
```

[25] Ist kein Fontpaket installiert, beendet sich VDR mit einem Speicherzugriffsfehler:

```
# journalctl -f -u vdr
runvdr[560]: vdr: no fonts available - OSD will not show any text!
runvdr[560]: /usr/bin/runvdr: line 39:   574 Segmentation fault      /usr/bin/vdr -w 60 -E- -u vdr
```

Hier wurde die systemweite Locale nicht aktiviert:

```
# journalctl -b -e -u vdr
vdr[224]: [224] found 0 locales in /usr/share/locale
vdr[224]: [224] no locale for language code 'deu,ger'
```

6.2.1 Live

Das Live Interactive VDR Environment bietet eine komfortable Möglichkeit zur Interaktion mit dem VDR per Webbrowser. Die Webserverfunktion wird über den Webapplicationserver `tntnet` realisiert, der seinerseits die `cxxtools` voraussetzt.

- cxxtools.tar.gz
- tntnet.tar.gz
- vdr-live-git.tar.gz

Mit der Option `-p` kann ein anderer Port (Voreinstellung: 8008) festgelegt werden. Nach dem Einloggen (Login: `admin`, Passwort: `live`) wird man unter "Einstellungen" zunächst die Zugangsdaten ändern wollen. Soll die Weboberfläche per Internet erreichbar sein und ist kein VPN - beispielsweise OpenVPN - eingerichtet, empfiehlt es sich die Datenübertragung per HTTPS (Port 8443) zu verschlüsseln. Das benötigte Zertifikat kann man wie in der `README`-Datei beschrieben behelfsweise mit dem eigenen Schlüssel signieren.

```
# cd /etc/vdr/plugins/live/
# openssl req -new -x509 -keyout live-key.pem -out live.pem -days 365 -nodes
```

Als "Common Name" ist der Domain-Name (FQDN) einzutragen, normalerweise also die DDNS-Adresse des Routers. Auf diesem ist außerdem die Portweiterleitung entsprechend zu konfigurieren. Eine Alternative zum Live-Plugin ist VDRAdmin-AM (Abschnitt 6.3.1).

6.2.2 Epgsearch

Ist das Epgsearch-Plugin installiert, können über VDRAdmin-AM oder das Live-Plugin Suchtimer erstellt werden. Aufnahmen werden dann automatisch programmiert, sobald ein passender Eintrag im EPG erscheint.

- vdr-epgsearch-git.tar.gz

Das Plugin informiert zudem per E-Mail über Timerkonflikte. Die notwendigen Einstellungen kann man via Xineliboutput-Frontend über das OSD des VDR vornehmen (*Einstellungen*|*Plugins*|*epgsearch*|*Email-Benachrichtigung*), die Eingabe von Daten ist darin allerdings etwas mühsam. Einfacher ist es, VDR vorübergehend zu beenden und die Konfigurationsdatei direkt zu editieren.

```
#/etc/vdr/setup.conf
epgsearch.MailAddress = user@example.com
epgsearch.MailAddressTo = user@example.com
epgsearch.MailNotificationConflicts = 1
epgsearch.MailNotificationSearchtimers = 0
epgsearch.MailViaScript = 0
```

Die hier verwendete Mail-Methode *sendmail* setzt einen entsprechend eingerichteten MTA voraus. Ich habe zu diesem Zweck *Postfix* als Nullclient konfiguriert. Auf diesem Wege werden auch Systemnachrichten wie z.B. Quota-Warnungen und die Konsolenausgabe von cron-jobs zugestellt. Bei

der alternativen Mail-Methode `sendEmail.pl` kommt stattdessen das gleichnamige Perl-Script zum Einsatz[26].

6.2.3 Streamdev

Das Streamdev-Plugin stellt Live-TV als HTTP-Stream bereit, der z.B mit VLC oder MPlayer auf anderen Rechnern (Clients) im Netzwerk abgespielt werden kann. Auch Smartphones/Tablets können den Stream darstellen, auf Android-Geräten empfehlen sich dafür VPlayer oder MX Player, die komfortabel aus AndroVDR oder VDR Manager heraus gestartet werden können[27].

Auf dem DockStar wird nur das Streamdev-Server Plugin benötigt, das Client Plugin ist für andere Instanzen des VDR vorgesehen, die den Server als zusätzliche Empfangsquelle einbinden können.

- vdr-streamdev-server-git.tar.gz

Die IP-Adressen der Clients, die Zugriff auf den DockStar haben sollen, sind dort in der Datei `streamdevhosts.conf` im Unterverzeichnis des Plugins einzutragen. Dabei kann, wie in folgendem Beispiel, auch einfach das zugehörige Subnetz angegeben werden:

```
#/etc/vdr/plugins/streamdev-server/streamdevhosts.conf
192.168.1.0/24
```

Nach Aktivierung des Plugins findet man unter der IP-Adresse des DockStar auf Port 3000 ein spartanisches Interface mit URLs zu den verfügbaren Sendern. VDRAdmin-AM zeigt bei aktiviertem Plugin lediglich Links (m3u-Playlists) zu den Streams an, mit dem Live-Plugin gelingt die Darstellung von Live-TV im Web-Browser leichter, ein entsprechendes Browser-Plugin wie z.B. das VLC Plugin natürlich vorausgesetzt.

[26] Die aktuelle Version 1.56 von `sendEmail.pl` ist im Paket `vdr-epgsearch-git` integriert, es ist also nicht notwendig `sendemail` aus dem AUR zu installieren. Den in einem Kommentar erwähnten Patch im Hinblick auf `perl-io-socket-ssl` habe ich berücksichtigt. Dieses optionale Paket sorgt für einen E-Mail-Versand per TLS, sofern der Mailserver es unterstützt. Folgende Daten sind in der VDR-Konfigurationsdatei zu hinterlegen und anzupassen:

```
#/etc/vdr/setup.conf
epgsearch.MailAddress = user@example.com
epgsearch.MailAddressTo = user@example.com
epgsearch.MailAuthPass = ”password”
epgsearch.MailAuthUser = login
epgsearch.MailNotificationConflicts = 1
epgsearch.MailNotificationSearchtimers = 0
epgsearch.MailServer = smtp.example.com
epgsearch.MailUseAuth = 1
epgsearch.MailViaScript = 1
```

Das Passwort muss, sofern es Sonderzeichen enthält, in Anführungszeichen eingeschlossen werden. Da es im Klartext vorliegt, sollten nur `root` und ggf. der Nutzer `vdr` Leserecht besitzen. Allerdings werden die Zugriffsrechte von `setup.conf` beim Beenden von VDR auf 0644 zurückgesetzt.

[27] VDR Manager setzt ein gleichnamiges Plugin voraus

- vdr-vdrmanager-git.tar.gz

AndroVDR kommuniziert mit dem VDR dagegen per SVDRP, den Zugriff regelt `/etc/vdr/svdrphosts.conf`.

6.2.4 Xineliboutput

Das Xineliboutput-Plugin generiert - ähnlich wie das Streamdev-Plugin - einen HTTP-Stream, erlaubt bei Verwendung spezieller Frontends wie `vdr-sxfe` aber auch eine Darstellung und Steuerung der kompletten Oberfläche des VDR samt OSD wie sie am TV-Ausgang einer DVB-Karte mit MPEG-Decoder erscheinen würde, so dass beispielsweise auch Einstellungen vorgenommen und Aufnahmen abgespielt werden können. VDRAdmin-AM unterstützt Xineliboutput seit Version 3.6.9.

- vdr-xineliboutput-git.tar.gz

Die Frontends sind normalerweise Bestandteile des Plugins. Da ein lokales Frontend auf dem DockStar mangels Monitorausgang nicht sinnvoll ist, habe ich dieses im Paket deaktiviert, die `xine-lib` wird daher nicht benötigt. Frontends (und folglich `xine-lib`) müssen nur auf den Clients installiert sein. Dazu lässt sich *xineliboutput* ggf. auch ohne VDR kompilieren:

- vdr-xineliboutput-frontends-git.tar.gz

Die Zugriffskontrolle erfolgt ähnlich wie beim Streamdev-Plugin.

```
#/etc/vdr/plugins/xineliboutput/allowed_hosts.conf
192.168.1.0/24
```

Um den VDR auf den Clients per Tastatur oder Fernbedienung[28] steuern zu können, muss auf dem DockStar die Datei `/etc/vdr/remote.conf` angelegt bzw. erweitert werden. Für das OSD wird ferner ein Font benötigt, z.B. `ttf-bitstream-vera`. Auf einem Client startet man das Frontend beispielsweise mit

```
$ vdr-sxfe -A alsa -f --post ⟷
    ↪ tvtime:method=Linear,cheap_mode=1,pulldown=0,use_progressive_frame_flag=1 ⟷
    ↪ xvdr+tcp://192.168.1.5:37890
```

6.2.5 XVDR/VNSI

Wer den VDR in das XBMC/Kodi Media Center integrieren will, benötigt eines der folgenden Plugins. Zeitversetztes Fernsehen (Timeshift) ist derzeit nur mit XVDR möglich. Beim Abspielen von Aufnahmen berücksichtigt XVDR außerdem Schnittmarken (vgl. Abschnitt 6.3.2). Die Wiedergabe beginnt dann bei der ersten Schnittmarke und endet bei der letzten, dazwischen liegende Bereiche (Werbeunterbrechungen) werden automatisch übersprungen[29].

[28] Zur Steuerung des VDR über einen am Client angeschlossenen IR-Empfänger sind in remote.conf die auf dem Client(!) ausgelösten Keycodes einzutragen. LIRC muss auf dem Client eingerichtet und gestartet sein. Außerdem ist dem Aufruf von `vdr-sxfe` die Option `--lirc` hinzuzufügen.

[29] Die Anzahl der Schnittmarken muss stets geradzahlig sein, d.h. zu jeder Start-Marke muss es eine End-Marke geben, andernfalls erfolgt im XBMC keine Wiedergabe. Das automatische Springen von einer End-Marke zur nächsten Start-Marke funktioniert leider nicht immer zuverlässig, mitunter findet (nachvollziehbar) ein Rücksprung zur vorherigen Start-Marke statt. Beim Vorhandensein von Marken wird in Kodi - genauso wie bei Verwendung von edit decision lists - die Abspielzeit falsch angezeigt und ein manuelles Navigieren führt zu teilweise fehlerhaften Reaktionen.

- vdr-xvdr-git.tar.gz
- vdr-vnsiserver-git.tar.gz

Das Paket `vdr-vnsi-git` habe ich nach `vdr-vnsiserver-git` umbenannt, nach dem Update muss in `/etc/conf.d/vdr/` die Option `-Pvnsiserver5` durch `-Pvnsiserver` ersetzt werden. Die Zugriffskontrolle erfolgt über `/etc/vdr/plugins/xvdr/allowed_hosts.conf` bzw. `/etc/vdr/plugins/vnsiserver/allowed_h`
Auf dem XBMC-Client ist das `xbmc-addon-xvdr` bzw. `xbmc-pvr-addons` zu installieren. Näheres zur Konfiguration von XBMC/Kodi findet sich im Wiki

6.2.6 Graphlcd

Mit dem Graphlcd-Plugin lassen sich EPG und Menü des VDR auf einem graphischen Display darstellen. Ich nutze dafür diesen beliebten digitalen Bilderrahmen (DPF) und folgende Pakete.

- dpf-graphlcd-base-git.tar.gz
- vdr-graphlcd-git.tar.gz

Eine in `graphlcd-base` enthaltene `udev`-Regel sorgt dafür, dass Mitglieder der Gruppe `uucp` Schreibrecht auf die Devicedateien des DPF erhalten, `vdr` ist folglich dieser Gruppe hinzuzufügen.

```
# usermod -a -G uucp vdr
```

Der Bilderrahmen muss vor dem Einsatz gehackt und in den Developer-Modus gebracht werden. Letzerer aktiviert sich 10 Minuten nach dem Anschließen auch selbständig, das Display kann also problemlos vorübergehend getrennt werden (z.B. per schaltbarem USB-Hub), das graphlcd-plugin greift dann beizeiten wieder darauf zu, ohne dass der VDR neu gestartet werden muss. Das Display kann auch unabhängig vom VDR via `lcd4linux` bzw. `lcdproc` als Statusdisplay oder Konsole dienen, Einzelheiten siehe z.B. hier.

6.3 Zusatzprogramme

6.3.1 VDRAdmin-AM

Das in Perl geschriebene Programm VDRAdmin-AM bietet ähnliche Funktionen wie das Live-Plugin (Abschnitt 6.2.1), ist aber im Vergleich deutlich langsamer und belegt mehr Arbeitsspeicher. Anders als beim Live-Plugin können Vorlagen für Suchtimer angelegt werden.

- vdradmin-am-git.tar.gz

Das im AUR verfügbare Paket habe ich wie folgt überarbeitet:

- Die aktuellen Programmquellen werden aus dem VDR-Developer Git-Repository geladen
- Das `install.sh` Script wird nicht für die Installation verwendet (also kein Patch nötig)
- Keine Unterverzeichnisse für PID- und Logdatei (wegen `tmpfs`, Abschnitt 4.1)
- Konfiguration: SVDRP Port 6419, Aufnahmen unter `/video`

Nach dem Start des Daemons mit *systemctl start vdradmind.service* ist die Weboberfläche im lokalen Netz unter der IP-Adresse des DockStar erreichbar (Port:8001, Login: vdradmin, Passwort: vdradmin). Im Menüpunkt *Konfiguration* kann man die wichtigsten Einstellungen vornehmen, zumindest das Passwort sollte geändert werden. Innerhalb eines angegebenen lokalen Netzes, z.B. 192.168.1.0/24, erlaubt vdradmind einen Zugang ohne Passwort.

Die Einstellungen werden in /etc/vdradmin/vdradmind.conf gespeichert. Man muss die Datei evtl. auch manuell editieren, z.B. um den in SERVERPORT definierten Port (Voreinstellung: 8001), auf den der Daemon lauscht, zu ändern. Für etwaige Kommandozeilenoptionen ist eine Datei im conf.d-Verzeichnis vorgesehen.

```
#/etc/conf.d/vdradmind
VDRADMIND_OPTS="--ssl --log 4 -L /var/log/vdradmind.log"
```

Die in diesem Beispiel aktivierte Beschränkung auf HTTPS-Verbindungen empfiehlt sich, wenn VDRAdmin-AM übers Internet erreichbar sein soll. Dazu muss das Paket perl-io-socket-ssl installiert, ein Verzeichnis /etc/vdradmin/certs angelegt und darin Zertifikat und Schlüssel hinterlegt werden. Letzteres ist in der Manpage von vdradmind anhand eines Beispiels beschrieben.

6.3.2 Noad

Um zur Überbrückung von Werbung automatisch Schnittmarken setzen zu lassen, hat sich das Programm noad bewährt[30], ein Paket stelle ich im AUR zur Verfügung. Zur Dekodierung greift noad auf ffmpeg zurück, ersatzweise wird libmpeg2 verwendet. Letzteres versteht sich jedoch nicht auf HD-Aufnahmen[31]. Da die werbefinanzierten Sender ohnehin nur in SD unverschlüsselt zu empfangen sind, habe ich ffmpeg aus der Liste der Abhängigkeiten (depends) herausgenommen und die configure-Option --without-ffmpeg hinzugefügt. Die sonst unnötige Installation von ffmpeg würde samt weiterer Abhängigkeiten über 120 MB beanspruchen. Durch den Verzicht auf ffmpeg steht auch die Option --asd (audio silence detection) nicht zur Verfügung.

- noad.tar.gz

Damit das Programm schon während der Aufnahme seiner Tätigkeit nachkommen kann, lässt man es am besten direkt vom VDR starten. Dieser kennt für solche Zwecke die Option -r. Ein darüber angegebenes Script wird vor Aufnahmebeginn und nach der Aufnahme mit entsprechendem Parameter (before/after) aufgerufen.

[30] Eine Alternative ist markad. Es besteht aus einem Plugin und einem separaten Programm, das sich auch unabhängig von vdr einsetzen lässt, normalerweise aber über das Plugin gesteuert und per Voreinstellung zu Beginn jeder Aufnahme gestartet wird.

- vdr-markad-git.tar.gz

Aufgrund der Abhängigkeit von ffmpeg habe ich markad nicht als Binärpaket in mein Repository aufgenommen.

[31] Im Online-Modus arbeitet noad in der Regel zuverlässig. Bei manuellem Aufruf identifiziert er SD-Aufnahmen mitunter fälschlicherweise als HDTV. Steht als Dekoder nur libmpeg2 zur Verfügung, verweigert noad dann unberechtigterweise seinen Dienst.

```
looks like HDTV, can't handle this file with libmpeg2, give up
```

```
#/etc/conf.d/vdr (continued)
OPTIONS=" ... -r /usr/local/bin/vdrcmd.sh"
```

Der erste Parameter wird direkt weitergereicht, der zweite spezifiziert das Aufnahmeverzeichnis, in dem `noad` auch die Datei `marks` anlegt, in der die Schnittmarken eingetragen werden. Neben Datum und Uhrzeit enthält das Aufnahmeverzeichnis auch die Kanalnummer. Diese Information kann genutzt werden, um `noad` sinnvollerweise nur bei Aufnahmen von privaten Sendern einzusetzen.

```
#/usr/local/bin/vdrcmd.sh
#!/bin/bash
case "$1" in
  before)
    CH=${2%-*.rec}
    CH=${CH##*.}
    case $CH in
      5|6|7|8|11|12|19|20|21|27|35|36|37|38) # adapt to your needs
        echo "executing noad for channel $CH"
        noad before --online=2 --comments "$2"
        ;;
     *)
        echo "not executing noad for channel $CH"
        ;;
     esac
    ;;
  edited)
    [ -f $2/marks ] && rm -v $2/marks
esac
```

So aufgerufen, beginnt Noad seine Tätigkeit 60 Sekunden nach Aufnahmebeginn mit einem `nice`-Wert von 19. Mit dem Xineliboutput-Frontend sollten die Schnittmarken überprüft und ggf. korrigiert werden, näheres im Benutzerhandbuch.

Im Verlauf des (optionalen) Schneidprozesses legt VDR im Verzeichnis der geschnittenen Aufnahme mitunter eine (sinnlose) `marks`-Datei mit einer ungeradzahligen Anzahl von Schnittmarken an, was dazu führt, dass diese in Kodi mit dem XVDR-Addon nicht abgespielt wird. Die vorletzte Zeile im obigen Script sorgt dafür, dass diese Datei ggf. direkt nach dem Schneiden wieder entfernt wird.

6.4 Weitere Aufnahmeverzeichnisse

Das Aufs-Dateisystem ermöglicht das gleichzeitige Einbinden mehrerer Verzeichnisse unter einem einzigen Mountpunkt. So lässt sich neben dem auf dem USB-Stick befindlichen Verzeichnis `/home/vdr` zusätzlich ein Verzeichnis auf der externen Festplatte als Aufnahmeverzeichnis nutzen. Im Gegensatz zu LVM oder mhddfs erlaubt diese Vorgehensweise in gewissen Grenzen eine Steuerung dahingehend, auf welchem Datenträger die einzelnen Aufnahmen jeweils tatsächlich abgelegt werden.

Aufs ist im Kernel von Arch Linux ARM bereits integriert, es werden nur die `aufs-utils` benötigt. Das Paket lässt sich derzeit nur auf Umwegen kompilieren, auch die im Bugreport beschriebenen

Lösungen führen nicht mehr zum Ziel. Ich habe deshalb die benötigte Datei `aufs_type.h` durch Aufruf von `make headers_install` im Quellverzeichnis von Kernel 3.1.10 erstellt und in das Paket integriert.

- aufs-util-git.tar.gz

Bei nur einem Verzeichnis entspricht des Mounten mit Aufs im Ergebnis dem oben verwendeten `mount --bind`:

```
# umount /video
# mount -v -t aufs -o br:/home/vdr/ none /video/
none on /video type aufs (rw,relatime,si=eb4b663e)
```

Ein Verzeichnis auf der Festplatte wird nun einfach als weitere Schicht darüber gelegt[32].

```
# mkdir /hdd/video
# chown vdr.vdr /hdd/video
# mount -o remount,prepend:/hdd/video,xino=/hdd/video/.aufs.xino none /video/
```

Unter `/video` erscheinen jetzt die Inhalte von `/hdd/video` und `/home/vdr` gemeinsam, wobei ersteres als oberste Schicht (Branch) gleichlautende Dateien und Verzeichnisse des letzteren ggf. überdeckt. Die aktuelle Zusammensetzung des Aufs-Verbundes kann man aus dem `sys`-Verzeichnis auslesen:

```
# cat /sys/fs/aufs/si_eb4b663e/br[01]
/hdd/video=rw
/home/vdr=rw
```

Neue (Aufnahme)-Verzeichnisse werden stets auf der obersten beschreibbaren Schicht `br0`, hier nun also unter `/hdd/video`, angelegt[33]. Beide Branches sind schreibbar eingebunden, damit z.B. auch bestehende Aufnahmen auf dem USB-Stick per Webfrontend gelöscht werden können.

Beim gleichzeitigen Mounten mehrerer Verzeichnisse ist per Voreinstellung nur die oberste Schicht beschreibbar, der den obigen Befehlen entsprechende `fstab` Eintrag erfordert deshalb die Angabe der `rw`-Option. Den ursprünglichen Eintrag aus Abschnitt 6.1 entfernen wir wieder bzw. kommentieren ihn aus. Bei Verwendung von *systemd* (Abschnitt 7) nutzt man für *aufs* anstelle der `fstab` besser eine mount-unit[34].

[32] Die Verlagerung der von Aufs intern zur Inode-Verwaltung genutzen `xino`-Datei muss jeweils explizit angegeben werden. Andernfalls würde diese Datei je nach Anfangsstatus permanent auf einem der beiden Datenträger verbleiben und folglich den USB-Stick auch dann mit Schreibvorgängen belasten, wenn die Festplatte verfügbar ist, bzw. die Festplatte auch dann in Rotation versetzen, wenn für Aufnahmen lediglich der USB-Stick vorgesehen ist. Die eigentlich überflüssige Angabe `none` soll einer Verwechslung mit dem letzten Eintrag in `/etc/fstab` (Abschnitt 6.5) vorbeugen.

[33] Genauer gesagt werden - gemäß der Default Policy von Aufs bei mehreren beschreibbaren Branches - neue Dateien und Verzeichnisse stets in dem höchsten Branch, in dem sich das zugehörige Unterverzeichnis befindet, geschrieben: Existiert also auf `br1` bereits eine dem Titel der anzufertigenden Aufnahme entsprechendes Verzeichnis, unter `br0` jedoch nicht, so wird diese Aufnahme auf `br1` gespeichert. Dieser Fall kann leicht bei Wiederholungen und Serien eintreten.

[34] *Systemd* versucht `/video` noch vor dem Einbinden der involvierten Datenträger zu mounten. Da dies zwangsläufig fehlschlägt, wird der Bootvorgang angehalten. Ich habe deshalb den Eintrag in `/etc/fstab` auskommentiert, eine entsprechende mount-unit angelegt

```
#/etc/systemd/system/video.mount
[Unit]
Description=VDR recording directory
```

```
#/etc/fstab (revised)
# /home/vdr  /video  none  bind     0 0
none        /video  aufs  br:/hdd/video=rw:/home/vdr=rw  0 0
```

Sollen auf der Festplatte vorerst keine weiteren Aufnahmen gespeichert werden, damit diese in absehbarer Zeit in den Standby geht und verbleibt, kann man das Verzeichnis `/hdd/video` aus dem Aufs-Verbund wieder entfernen. Letzteres ist jedoch nicht möglich, wenn momentan (via Aufs) auf das Verzeichnis zugegriffen wird, z.B. weil gerade eine Aufnahme angefertigt wird. In diesem Fall kann man die Branches aber vertauschen. Eine entsprechende Option kennt aufs leider (noch?) nicht, man muss stattdessen den unteren Branch entfernen und als oberen Branch wieder einbinden.

```
# mount -o remount,del:/home/vdr,prepend:/home/vdr=rw,xino=/home/vdr/.aufs.xino none ↩
    ↪ /video
```

Die ggf. laufende Aufnahme wird nun weiterhin unter `/hdd/video` gespeichert, die nächsten hingegen unter `/home/vdr` angefertigt. Überschneiden sich die Aufnahmen zeitlich (gleiches Bouquet oder Twin-Receiver), wird also auf beide Datenträger gleichzeitig geschrieben. Ist die Aufnahme auf der Festplatte abgeschlossen, kann diese schließlich aus dem Verbund genommen werden:

```
# mount -o remount,del:/hdd/video/,xino=/home/vdr/.aufs.xino none /video
```

Die ursprüngliche Reihenfolge wird dann später wieder durch Hinzufügen von `/hdd/video` als oberstem Branch (`prepend`) erreicht. Für die nötigen Schritte nutze ich ein Script aufs_switch.sh, das auch die LED dem Status entsprechend anpasst[35]. Letzteres erfolgt unter Zuhilfenahmen des C-Programms aus Abschnitt 4.5. Der optional zu übergebende Parameter des Scriptes impliziert die gewünschte Anzahl und Rangfolge der Branches:

Bei erfolgreichen Statuswechsel liefert das Script den Rückgabewert 0, andernfalls 1, den aktuellen Status gibt es auf `/dev/stdout` aus. Das Script wird per `cron` aufgerufen, hier meine derzeitigen (mit `crontab -e`) vorgenommenen Einträge:

```
@reboot    /usr/local/bin/aufs_switch.sh
```

```
After=hdd.mount home.mount

[Mount]
What=none
Where=/video
Type=aufs
Options=br:/hdd/video=rw:/home/vdr=rw
```

und als zusätzliche Bedingung für `vdr.service` formuliert.

```
#/etc/systemd/system/vdr.service
.include /usr/lib/systemd/system/vdr.service

[Unit]
Requires=video.mount
After=video.mount
```

[35] Zudem stellt das Script sicher, dass die Aufnahmen tatsächlich auf `br0` angefertigt werden. Dazu legt es vor einer Rotation der Verzeichnisse auf dem unteren bzw. nicht eingebundenen Branch zunächst sämtliche Aufnahmeverzeichnisse des oberen Branches als leere Verzeichnisse an. Leere Verzeichnisse aus vorherigen Aktionen sowie Whiteout-Dateien werden entfernt. Letztere entstehen beim Löschen von (Aufnahme-)Verzeichnissen, die in beiden Branches existieren.

```
0 21 * * * /usr/local/bin/aufs_switch.sh -1 || /usr/local/bin/aufs_switch.sh -2
0 0  * * * /usr/local/bin/aufs_switch.sh -1
0 9  * * * /usr/local/bin/aufs_switch.sh 2
```

Die erste Zeile sorgt lediglich für eine Anpassung der LED. Lässt sich `/hdd/video` um 21 Uhr nicht aushängen (üblicherweise wegen einer laufenden Aufnahme), werden die Branches zunächst getauscht (-2). Um Mitternacht wird der beabsichtigte Zustand (-1) ggf. erneut in die Wege geleitet, zusätzlich aber auch schon direkt nach der Aufnahme: Dies lässt sich durch Erweiterung des in Abschnitt 6.3.2 angelegten Scriptes realisieren.

```
#/usr/local/bin/vdrcmd.sh (revised)
#!/bin/bash
case ”$1” in
  before)
    noad before --online=2 --comments ”$2”
  ;;
  after)
    h=$(date +%k)
    if [ $h -ge 21 -o $h -lt 9 ]; then
      if [ -e $2/noad.pid ]; then
        screen -dm bash -c ”inotifywait -e delete $2/noad.pid ; $0 $@”
      else
        sudo /usr/local/bin/aufs_switch.sh -1
      fi
    fi
  ;;
esac
```

Das Script wird mit den Rechten des Users *vdr* ausgeführt, der Aufruf von `aufs_switch.sh` erfordert aber Rootrechte und erfolgt daher via *sudo*. Der zugehörige Eintrag in `/etc/sudoers` wird mit `visudo` angelegt:

```
vdr ALL=NOPASSWD: /usr/local/bin/aufs_switch.sh
```

Neben der Uhrzeit überprüft das Script vor Aufruf von `aufs_switch.sh`, ob im Aufnahmeverzeichnis eine Datei `noad.pid` existiert. Ist dies der Fall, greift `noad` noch auf den Branch zu, ein Aushängen würde also fehlschlagen. Das Script startet dann einen Hintergrundprozesses, der das Script mit denselben Parametern erneut aufruft, sobald die pid-Datei entfernt wurde. Zu diesem Zweck müssen die Pakete `screen` und `inotify-tools` installiert sein.[36]

6.5 Netzwerkfreigabe

Die Aufnahmen aus beiden Aufnahmeverzeichnisse sollen im lokalen Netzwerk zur Weiterverarbeitung (ggf. demultiplexen, Schnitt) verfügbar sein. Die Einrichtung des NFSv4-Servers ist im Wiki

[36] Im Zusammenhang mit Aufs gibt es noch ein ungelöstes Problem: Weist man VDR an, eine Aufnahme zu löschen, markiert er diese zunächst nur als gelöscht, um sie erst später, wenn keine Aufnahme läuft, tatsächlich zu entfernen. Ist zu diesem Zeitpunkt der Branch, auf dem die Aufnahme liegt, nicht eingebunden, schlägt der Löschvorgang fehl und wird nicht wiederholt, die Aufnahme beansprucht also weiterhin unbemerkt Speicherplatz und muss manuell gelöscht werden.

ausführlich erklärt. Bei entsprechend konfiguriertem Kernel bzw. Aufs-Modulpaket besteht auch die Möglichkeit den Aufs-Mountpunkt `/video` insgesamt zur Verfügung stellen. Im aktuellen ARMv5-Kernel (Paket `linux-kirkwood`) ist die dafür erforderliche Option `CONFIG_AUFS_EXPORT=y` gesetzt, nicht jedoch im Kernel 3.1.10 (Paket `linux`).

Die Freigaben werden an Verzeichnisse unter /export/ gebunden

```
#/etc/fstab (continued)
/hdd/video     /export/video0         none bind     0 0
/home/vdr      /export/video1         none bind     0 0
# only with CONFIG_AUFS_EXPORT=y
# /video         /export/video         none bind     0 0
```

und per `/etc/exports` konfiguriert. Da das Aufs-Dateisystem keine UUID besitzt, muss ggf. zur Identifizierung eine id>0 zugeordnet werden.

```
#/etc/exports
/export           192.168.1.0/24(rw,no_subtree_check,async,no_root_squash,fsid=0)
/export/video0    192.168.1.0/24(rw,no_subtree_check,async,no_root_squash,nohide)
/export/video1    192.168.1.0/24(rw,no_subtree_check,async,no_root_squash,nohide)
# only with CONFIG_AUFS_EXPORT=y
# /export/video     ↩
   ↪ 192.168.1.0/24(rw,no_subtree_check,async,no_root_squash,nohide,fsid=1)
```

7 Umstieg auf Systemd

Nachdem Arch Linux auf `initscripts/sysvinit` basierende Systeme offiziell nicht mehr unterstützt, habe ich mich entschlossen, *systemd* auch unter ALARM einzusetzen, vorerst jedoch mit der Option ggf. kurzfristig zu `initscripts/sysvinit` zurückkehren zu können.

Hinweis: Seit November 2012 ist *systemd* in den ALARM-Images voreingestellt, eine Umstellung wie nachfolgend beispielhaft gezeigt, ist nur bei älteren Installationen erforderlich. Wer allerdings tatsächlich noch ein SysVinit-basiertes System verwendet, dürfte dabei mittlerweile vor einer Herausforderung stehen und sollte stattdessen eine Re-Installation erwägen.

7.1 Netzwerkkonfiguration

Mit `netcfg` erhält man eine unter `systemd` sowie `initscripts/sysvinit` gleichermaßen funktionierende Netzwerkkonfiguration. Beim Erstellen von Profilen kann man sich an den im Paket enthaltenen Beispielen orientieren.

```
# cp /etc/network.d/examples/ethernet-static /etc/network.d/lan
```

Das hier gewählte Profil für eine statische IP-Adresse ist den lokalen Gegebenheiten anzupassen.

```
#/etc/network.d/lan (edit)
ADDR='192.168.1.5'
GATEWAY='192.168.1.1'
DNS=('192.168.1.1')
```

Schließlich wird das zu verwendende Profil festgelegt.

```
#/etc/conf.d/netcfg
NETWORKS=(lan)
```

7.2 Hostname, Tastaturbelegung, Spracheinstellung

Zur Vorbereitung auf Systemd sind weitere Einstellungen von `/etc/rc.conf` auf die neuen Konfigurationsdateien auszulagern:

```
#/etc/hostname
alarm

#/etc/vconsole.conf
KEYMAP=de-latin1-nodeadkeys
FONT=
FONT_MAP=

#/etc/locale.conf
LANG=de_DE.UTF-8
LC_COLLATE=C
```

Die entsprechenden Einträge in `/etc/rc.conf` habe ich auskommentiert, so dass nur noch die DAEMONS-Zeile übrig blieb.

```
#/etc/rc.conf
DAEMONS=(syslog-ng net-profiles openntpd sshd crond lircd irexecd vdr rpcbind ↩
    ↪ nfs-common nfs-server)
```

Um die Verwendbarkeit der Einstellungen unter Beibehaltung von `initscripts/sysvinit` zu testen, folgte ein Reboot.

7.3 Aktivierung der Services

Das Paket `systemd` sollte als Abhängigkeit von `initscripts` bereits installiert sein. Die DAEMONS in `/etc/rc.conf` werden zwar bis auf weiteres auch von *systemd* berücksichtigt, zur besseren Kontrolle sollte man die zu startenden Services aber expliziert aktivieren und schließlich auch die DAEMONS-Zeile in `/etc/rc.conf` komplett auskommentieren. Die von mir bereitgestellten Pakete `vdr` und `vdradmin-am-git` enthalten in den aktuellen Versionen keine rc-Scripts mehr.

```
# systemctl enable netcfg.service openntpd.service sshd.service cronie.service ↩
    ↪ lirc.service irexec.service vdr.service rpc-idmapd.service rpc-mountd.service
```

Man beachte die in einzelnen Fällen vom rc-Script abweichende Namensgebung der Services. Der syslog-Dienst wird nicht benötigt, Logmeldungen können im Journal von systemd eingesehen werden. Die `/etc/fstab` sollte bezüglich der Systemd-Kompatibilität überprüft werden, das betrifft insbesondere, falls man es verwendet, das `aufs`-Dateisystem, siehe Forum.

7.4 Booten mit Systemd

Um nun beim nächsten Booten `systemd` tatsächlich zu verwenden, wird im Forum von Arch Linux ARM oft geraten, das Paket `sysvinit` kurzerhand durch `systemd-sysvcompat` zu ersetzen. Dabei wird das per default vom Kernel aufgerufene `/sbin/init` durch einen symbolischen Link auf `/usr/lib/systemd/systemd` ersetzt. Man kann diesen Schritt aber auch vertagen und `systemd` zunächst testweise per Kernelparameter aktivieren:

```
# fw_setenv usb_custom_params 'init=/usr/lib/systemd/systemd'
```

Im Problemfall lässt sich der Parameter mit `fw_setenv usb_custom_params ''` wieder zurücksetzen. Falls nötig bootet man dazu ohne Speichermedium und führt obigen Befehl unter der Original-Firmware des DockStar durch. Die von `sysvinit` benötigten DAEMONS in `/etc/rc.conf` stellt man nach Mounten des USB-Sticks an einem Linux-PC wieder her.

Gibt es dagegen keine Probleme, kann schließlich `systemd-sysvcompat` installiert werden. Auch dann wird der o.g. Kernelparameter `init` nicht mehr benötigt.

8 Alternative Bootloaderkonfiguration

Neben dem Erkennen und Laden des Kernelimages ist *U-Boot* dafür verantwortlich dem Kernel das richtige Device der Rootpartition zu übergeben. Die in Abschnitt 3.2 beschriebene Methode schlägt u.U. fehl, wenn sich Anzahl und/oder Reihenfolge der von *U-Boot* bzw. Kernel erkannten Speichermedien unterscheiden[37]. Solche Vermittlungsprobleme lassen sich ausschließen, indem der Kernel veranlasst wird, die Rootpartition anhand eindeutiger Merkmale eigenständig zu erkennen. Diese sind:

- UUID oder Label des Dateisystems auf der Rootpartition. Um diese beim Booten zu erkennen ist der Kernel auf eine initial ramdisk angewiesen.
- PARTUUID oder PARTLABEL der Rootpartition. Diese stehen nur dann zur Verfügung wenn der USB-Stick über eine GUID-Partitionstabelle (GPT) verfügt. Bei Angabe der PARTUUID ist eine initial ramdisk nicht erforderlich.

Beide Möglichkeiten werden nachfolgend beschrieben. Um die Bootloaderkonfiguration auch von Arch Linux ARM aus durchführen zu können, wird das Paket `uboot-tools` benötigt. In `/etc/fw_env.config` ist anschließend die Raute vor dem Eintrag für den DockStar zu entfernen.

```
#/etc/fw_env.config
# segate dockstar:
/dev/mtd0               0xc0000         0x20000         0x20000
```

8.1 Initial Ramdisk

Normalerweise wird eine initrd benötigt, um die zum Mounten der Rootpartition benötigten Treiber für Datenträger und Dateisysteme zur Verfügung zu stellen. Diese sind bei Arch Linux ARM jedoch fest in den Kernel integriert, weshalb das Kernelpaket keine initrd bereitstellt. Sie muss folglich nachträglich erstellt werden.

Unter Arch Linux (i686/x86_64) wird anstelle von initrd der Nachfolger initramfs genutzt, das mit mkinitcpio konfiguriert wird. Es werden nur zwei Hooks benötigt:

```
#/etc/mkinitcpio.conf
HOOKS=”base udev”
```

Das initramfs für den laufenden Kernel (andernfalls Option -k nutzen) wird wie folgt generiert:

```
# mkinitcpio -v -g /tmp/initramfs-linux.img
```

Es kann unter Arch Linux ARM nicht direkt genutzt werden, mit `mkimage` (aus dem Paket `uboot-tools`) lässt sich daraus aber die eigentliche initial ramdisk erstellen:

[37] Nachdem ich jüngst eine der speziell für den Betrieb am oberen USB-Anschluss der DockStar vorgesehenen 2,5” Festplatten “FreeAgent Go” erworben hatte, musste ich mittels Netzkonsole festellen, dass diese von *U-Boot* nicht zuverlässig erkannt wurde: War beispielsweise neben der Festplatte nur der USB-Stick angeschlossen, wurde dieser von *U-Boot* als erstes/einziges Device erkannt und gemäß Abschnitt 3.2 folglich `sda2` als Rootpartition übermittelt. Aus Kernelsicht stellte `sda` jedoch die Festplatte dar, so dass der Bootvorgang nicht fortgesetzt werden konnte.

```
# mkimage -A arm -O linux -T ramdisk -C gzip -a 0x00000000 -e 0x00000000 -n ↩
    ↪ initramfs -d /tmp/initramfs-linux.img /boot/uInitrd
```

Die Datei `/boot/uInitrd` wird von *U-Boot* ohne weitere Konfiguration erkannt und eingebunden. Sodann sorgt `udev` für das Anlegen eines dem Label der Rootpartition (hier: `root`) entsprechenden Links unter `/dev/disk/by-label/`, der als Kernelparameter genutzt werden kann. Außerdem setze ich `rootwait` anstelle von `rootdelay` ein:

```
# fw_setenv usb_set_bootargs 'setenv bootargs console=$console ↩
    ↪ root=/dev/disk/by-label/root rootwait rootfstype=ext4 $mtdparts ↩
    ↪ $usb_custom_params'
```

Alternativ kann auch wie in einem Blog beschrieben `usb_init` angepasst werden. Anstelle des Labels sollte sich auch die UUID über den Link unter `/dev/disk/by-uuid/` nutzen lassen, ich habe dies jedoch nicht ausprobiert. Problematisch ist dagegen die Verwendung der Schreibweisen `root=LABEL=...` und `root=UUID=...`. Beide haben nicht zuverlässig funktioniert, näheres im Forum.

8.2 GPT mit hybridem MBR

Speichermedien mit GPT können vom Linux Kernel ab Version 2.6.37 genutzt werden. Die notwendige Option `CONFIG_EFI_PARTITION=y` ist im ARCH-Kernel aktiviert und sollte natürlich ggf. auch bei einem Custom-Kernel ausgewählt sein.

Zum Anlegen einer GPT dient das Programm `gdisk`. Wurde der Stick bereits mit `fdisk` formatiert, kann `gdisk` die GPT unter Verwendung der im MBR vorhandenen Partitionstabelle auch nachträglich anlegen. Die notwendigen Schritte habe ich an meinem Linux-PC durchgeführt. Wie schon in Abschnitt 2.1 verwende ich wieder `sdx` stellvertretend als Device des USB-Sticks (von dem ich zuvor sicherheitshalber ein Backup angelegt habe).

```
# gdisk /dev/sdx
```

Warnhinweise sollte man beachten und das Programm ggf. mit `'q'` abbrechen. Insgesamt gab es zwei Hürden zu nehmen, die ich hier genauer beschreibe:

8.2.1 Verkleinern der letzten Partition

Aus Redundanzgründen beansprucht die GPT auch einen Bereich am Ende des Speichermediums, der jedoch bereits Bestandteil der dritten Partition ist. Folglich vermeldet `gdisk` zunächst:

```
Warning! Secondary partition table overlaps the last partition by
33 blocks!
You will need to delete this partition or resize it in another utility.
```

Es war also erforderlich zunächst die letzte (dritte) Partition um 33 Blöcke (gemeint sind Sektoren á 512 Byte) zu verkürzen. Dazu muss zunächst das auf der Partition befindliche Dateisystem mit `resize2fs` verkleinert werden. Dabei ist die Blockgröße des Dateisystems zu berücksichtigen:

```
# dumpe2fs /dev/sdx3|grep "^Block"
dumpe2fs 1.41.14 (22-Dec-2010)
```

```
Block count:              3590912
Block size:               4096
Blocks per group:         32768
```

Die Anzahl der Dateisystemblöcke muss folglich um 33*512/4096, also mindestens 5 reduziert werden. Es schadet aber nichts, das Dateisystem zunächst stärker als nötig zu verkleinern (s.u.). In jedem Fall muss es vorher mit `e2fsck` einer Prüfung unterzogen werden.

```
# e2fsck -f /dev/sdx3
# resize2fs /dev/sdx3 3590907
# fdisk /dev/sdx
```

Das Verkleinern der Partition erfolgt mit `fdisk` einfach durch Löschen und erneutes Anlegen (d-3-n-p-3). Der vorgeschlagene Startsektor kann übernommen werden, der letzte Sektor ist jedoch um 33 auf 31252446 zu verringern, vgl. Abschnitt 2.1.

Optional lässt sich das Dateisystem mit `resize2fs` dann wieder automatisch auf die durch die neue Partitionsgröße vorgegebene maximale Größe erweitern. Bei der beschriebenen Vorgehensweise erübrigt sich dies:

```
# resize2fs /dev/sdx3
resize2fs 1.41.14 (22-Dec-2010)
Das Dateisystem ist schon 3590907 Blöcke groß. Nichts zu tun!
```

Zuletzt genügt ein erneuter Aufruf von `gdisk /dev/sdx` zur Umwandlung des MBR in eine GPT. Sollten keine Warnungen erscheinen, beendet man dazu einfach mit `'w'`.

8.2.2 Erstellung eines hybriden MBRs

Der Versuch den so manipulierten Stick nun testweise wieder am DockStar zu booten schlug fehl. Vermittels `netcat` stellte sich heraus, dass *U-Boot* die Partitionen das Sticks nicht mehr zu lesen vermochte:

```
Loading file "/boot/uImage" from usb device 1:1 (usbdb1)
Failed to mount ext2 filesystem...
** Bad ext2 partition or disk - usb 1:1 **
```

Abhilfe sollte laut eines Forumsbeitrags das Erstellen eines hybriden MBR bringen[38], der nur die Bootpartition als erste Partition enhält. Hier die genaue Vorgehensweise mit `gdisk` unter Zuhilfenahmen der GPT Dokumentation.

```
Command (? for help): r

Recovery/transformation command (? for help): h

WARNING! Hybrid MBRs are flaky and dangerous! If you decide not to use one,
just hit the Enter key at the below prompt and your MBR partition table will
be untouched.
```

[38] Mit Mainline U-Boot erübrigt sich die Erstellung eines hybriden MBR.

```
Type from one to three GPT partition numbers, separated by spaces, to be
added to the hybrid MBR, in sequence: 1
Place EFI GPT (0xEE) partition first in MBR (good for GRUB)? (Y/N): n

Creating entry for GPT partition #1 (MBR partition #1)
Enter an MBR hex code (default 83):
Set the bootable flag? (Y/N): y

Unused partition space(s) found. Use one to protect more partitions? (Y/N): n

Recovery/transformation command (? for help): w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING
PARTITIONS!!

Do you want to proceed? (Y/N): y
OK; writing new GUID partition table (GPT).
The operation has completed successfully.
```

Erfreulicherweise erkannte *U-Boot* den Stick nun wie bisher. Um den Vorteil der GPT zu nutzen brauchen wir noch die "Partition unique GUID" der Rootpartition. Diese bringt - unter Arch Linux ARM - ein erneuter Aufruf von `gdisk` nach Eingabe von `'i'` und Partitionsnummer zu Vorschein:

```
Command (? for help): i
Partition number (1-3): 2
Partition GUID code: 8EC76FBA-2272-45AF-A844-94ED164A1B86 (Unknown)
Partition unique GUID: 43152BE8-EA6D-47EE-A014-59C52C383DE0
First sector: 67584 (at 33.0 MiB)
Last sector: 2525183 (at 1.2 GiB)
Partition size: 2457600 sectors (1.2 GiB)
Attribute flags: 0000000000000000
Partition name: Linux filesystem
```

Mit dieser Information kann die Bootvariable wie folgt angepasst werden:

```
# fw_setenv usb_set_bootargs 'setenv bootargs console=$console ↩
    ↪ root=PARTUUID=43152BE8-EA6D-47EE-A014-59C52C383DE0 rootwait rootfstype=ext4 ↩
    ↪ $mtdparts $usb_custom_params'
```

Vor dem Reboot sollte man eine etwaig vorhandene `/boot/uInitrd` wieder löschen oder umbenennen. Andernfalls wird der Bootvorgang nach Laden des Kernels nicht fortgesetzt.

Die PARTUUID kann mit `gdisk` auch auf einen bestimmten Wert eingestellt werden (x-c-2). So lässt sich etwa ein Backup-USB-Stick mit identischer PARTUUID herstellen, von dem ohne erneute Anpassung des Bootmanagers ersatzweise gebootet werden kann.

9 Kontakt

Olaf Bauer <obauer (at) freenet (dot) de>

bbs.archlinux.de

archlinuxarm.org/forum