ArDroneXT - Ar.Drone 2 eXTension for swarming and service hosting

Vincent Autefage, Serge Chaumette

ArDroneXT

Ar.Drone 2 eXTension for swarming and service hosting

Operating System Setup Guide - Release 3

Vincent Autefage
vincent.autefage@labri.fr

Serge Chaumette
serge.chaumette@labri.fr

LaBRI - Univ.Bordeaux

This background image is the property of Parrot

Contents

Foreword

This report explains how to upgrade an `Ar.Drone 2` [1] for swarming and services hosting. In other words, it gives the technical information required to easily create a swarm of Ar.Drone 2 sharing a Wi-Fi network. Moreover, it describes the process to install new services and applications on the drone.

To achieve this process, it is necessary to perform several core modifications inside the Ar.Drone 2 operating system.

The authors decline all responsibility in case of any damage or any problem on any device used in the process, and on the usage of the resulting systems.

1 Requirements

1.1 Hardware requirements

To put in practice the operations described in this document, several pieces of hardware are required:

- an `Ar.Drone 2` [1];
- a `Linux` base station [2];
- a `USB flash drive` with at least 8 GB of memory;
- a `IEEE 802.11 chip` with *g/n* Wi-Fi and *ad-hoc* mode support [3].

The specific items used throughout this report are:

- an `Ar.Drone 2`;
- a computer with a `Debian Wheezy` Linux distribution [4];
- a `Kingston DataTraveler G3` USB key with 8GB of memory [5];
- an `Asus WL-167G` Wi-Fi key [6].

1.2 Software requirements

Several third-party software packages need to be installed:

- a partition manager like `Parted` [7] that will be used to create a suitable partitions table;
- the partition formatting tools contained in `E2fsprogs` [8] and `Dosfstools` [9];
- the `Debootstrap` tool [10] that will enable to create a Linux root file system;
- a local DHCP/DNS server/relay like `Dnsmasq` [11];
- a `telnet client` that will be used to communicate with the drones;
- the `wireless-tools` software suite [12] to manage the Wi-Fi settings.

2 Ar.Drone Base System to ArDroneXT

In order to enable an Ar.Drone 2 to be part of a swarm, it is necessary to modify its operating system. To prevent any damage or alteration of the original drone operating system, we will setup the new one on a dedicated USB flash drive. We call `ArDroneBS`, the original Ar.Drone 2 operating system, and `ArDroneXT` the version that will be installed on the USB flah drive.

2.1 Ar.Drone 2 operating system

The original operating system of the Ar.Drone 2 is a *Parrot home-made Linux* [2] running on an ARM EABI architecture. This Linux lays on a writable disk partition and it is thus possible to modify it. Nevertheless, modifying the root file system directly could be very damageable in case of mistakes. We will thus install another customized Linux system on a USB flash drive which will be plugged into the drone. The process described below requires *root* privileges on the computer Linux distribution.

2.2 USB flash drive partitions setup

We first create three partitions on the USB flash drive:

- one for the new operating system;
- one for the ARM EBAI cross-compiler (which makes development easier);
- one for users files, scripts, photos and videos.

We setup these partitions as follows:

1. `ARDSYS`: *ext2* file system with 5GB;
2. `ARDDEV`: *ext2* file system with 1GB;
3. `ARDUSR`: *fat32* file system with 2GB.

Usually, USB flash drives are *fat32* formatted. However the two first partitions cannot be setup with this kind of file system since it is not compliant with the symbolic links used by Linux (ARDSYS partition) and the ARM cross-compiler (ARDDEV partition). Besides, the drone can only take pictures and record videos on a *fat32* file system. Therefore, the third partition (ARDUSR) has to be setup as a *fat32* partition.

Please note that the second partition (ARDDEV) is optional and designed for cross-compilation from a x86-64 system to the ARM EABI architecture. We choose to install it directly on the USB flash drive in order to get a device ready to use for both end users and developers. Also, note that several Linux distributions provide such a compiler directly in their packages repository.

In the following commands the USB flash drive is identified as *sdb*. We first check the original content of the USB flash drive.

```
linux$ parted /dev/sdb print
Disk /dev/sdb: 7747MB
Partition Table: msdos
Number  Start   End     Size    Type     File system
 1      1049kB  7747MB  7746MB  primary  fat32
```

Unless enough free and un-partitioned space is available on the USB flash drive, it is necessary to erase all previous data before creating the new partitions.

```
linux$ parted /dev/sdb rm 1
linux$ parted /dev/sdb print
...
Number  Start   End     Size    Type     File system
```

Then, we create the partitions table.

```
linux$ parted /dev/sdb mkpart primary 1 5663
linux$ parted /dev/sdb mkpart primary 5663 6737
linux$ parted /dev/sdb mkpart primary 6737 7747
```

Finally, we format the partitions and label them.

```
linux$ mkfs.ext2 -L "ARDSYS" /dev/sdb1
linux$ mkfs.ext2 -L "ARDDEV" /dev/sdb2
linux$ mkdosfs -F32 -n "ARDUSR" /dev/sdb3
```

The USB flash drive is now ready.

```
linux$ parted /dev/sdb print
...
Number  Start   End     Size    Type     File system
 1      1049kB  5663MB  5662MB  primary  ext2
 2      5663MB  6737MB  1074MB  primary  ext2
 3      6737MB  7747MB  1010MB  primary  fat32
```

After creating these 3 partitions, we have to mount them (in case the system did not perform this task automatically).

```
linux$ mkdir /mnt/ARDSYS
linux$ mkdir /mnt/ARDDEV
linux$ mkdir /mnt/ARDUSR
linux$ mount -t ext2 /dev/sdb1 /mnt/ARDSYS
linux$ mount -t ext2 /dev/sdb2 /mnt/ARDDEV
linux$ mount -t vfat /dev/sdb3 /mnt/ARDUSR
```

2.3 Installing the ARM EABI compiler (ARDDEV partition)

In order to compile new programs for the drone, we have to get a cross-compiler which runs on a x86-64 architecture and produces an ARM EABI binary code. We can find a light one on the `Mentor`

`Graphics` website [13]. This one is used by the Debian community members to compile their own ARM EABI Linux distribution [14]. After downloading it, we extract the archive in the ARDDEV partition.

```
linux$ tar -C /mnt/ARDDEV -xvjf \
  arm-2012.09-64.arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2
```

Then, we test the compiler as follows.

```
linux$ export PATH=$PATH:/mnt/ARDDEV/arm-2012/bin/
linux$ cd /tmp
linux$ echo 'int main(){return 0;}' > nothing.c
linux$ arm-none-linux-gnueabi-gcc nothing.c -o nothing.elf
linux$ file nothing.elf
nothing.elf: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV)
dynamically linked (uses shared libs)
for GNU/Linux 2.6.16, not stripped
```

The ARDDEV partition is now ready.

2.4 Installing the operating system (ARDSYS partition)

2.4.1 Setting up the root file system

First of all, we have to create a Linux root file system compliant with the ARM EABI processor architecture. We have chosen to install a stable *Debian* [4] system which provides an ARM compliant architecture called *armel*. We use the *Debootstrap* [10] tool to create the root file system.

```
linux$ cd /mnt/ARDSYS
linux$ debootstrap --arch=armel --foreign squeeze /mnt/ARDSYS \
      http://ftp.debian.org/debian/
```

2.4.2 Connecting to the drone

The last part of the Deboostrap process has to be done from inside the drone. We thus plug the USB flash drive into the drone and turn the drone on.

Then, we have to connect our Linux station to the drone. The drone provides a Wi-Fi access point with no encryption and a DHCP server. Its network interface uses the IP address 192.168.1.1, therefore we can use the 192.168.1.2 for the Wi-Fi chip of the Linux station (or another one in the sub-network 192.168.1.0/24) or even call the DHCP server of the drone to automatically get an IP address. We have also to turn off the power management mode of our Wi-Fi chip in order to increase the quality of the connectivity.

Note that the hardware address (MAC), the channel number and the ESSID of the network can vary. In addition, we have to turn off the network daemon manager on our Linux computer (e.g. *NetworkManager* [15] on Debian) in order to manually configure our device as explained above. Most of Linux distributions have a network daemon program in order to manage network connections (e.g.

Ethernet and Wi-Fi). Please refer to your Linux distribution documentation in order to know how to turn the relative network daemon off.

The corresponding commands are listed hereafter.

```
linux$ ifconfig wlan0 down
linux$ ifconfig wlan0 mode managed
linux$ ifconfig wlan0 up
linux$ iwlist wlan0 scan
wlan0     Scan completed :
...
          Cell xx - Address: 90:03:B0:40:99:AB
                    Channel:1
                    Frequency:x GHz (Channel x)
                    Quality=xx/xx  Signal level=x dBm
                    Encryption key:off
                    ESSID:"ardrone_000"
                    Bit Rates:1 Mb/s; 2 Mb/s; 5,5Mb/s;
                              6 Mb/s; 9 Mb/s; 11 Mb/s;
                              12 Mb/s; 18 Mb/s; 24Mb/s;
                              36 Mb/s; 48 Mb/s; 54 Mb/s
                    Mode:Master
...
linux$ iwconfig wlan0 power off
linux$ iwconfig wlan0 channel 1
linux$ iwconfig wlan0 essid ardrone_000 ap 90:03:B0:40:99:AB
linux$ iwconfig wlan0
wlan0  IEEE 802.11bg  ESSID:"ardrone_000"
       Mode:Managed   Frequency:x GHz  Access Point: 90:03:B0:40:99:AB
...
linux$ dhclient wlan0
# or
linux$ ifconfig wlan0 192.168.1.2

linux$ ifconfig wlan0
wlan0  Link encap:Ethernet   HWaddr 00:11:f0:7c:fe:77
       inet adr:192.168.1.2  Bcast:192.168.1.255      Masque:255.255.255.0
...
```

We can now check the connectivity from the base station to the drone with a ping.

```
linux$ ping -c1 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_req=1 ttl=64 time=2.07 ms

--- 192.168.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.076/2.076/2.076/0.000 ms
```

Then, we have to get a root shell on the drone; we connect by using the telnet server which it runs natively.

```
linux$ telnet 192.168.1.1
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape Character is '^]'.


BusyBox v1.14.0 () built-in shell (ash)
Enter 'help' for a list of built-in commands.

ardrone$
```

2.4.3 Finishing the ARDSYS debootstrap process

From the drone root shell, we have to switch to our ARDSYS partition system in order to finish the debootstrap process. The USB flash drive is identified by the drone as *sda*.

```
ardrone$ mount /dev/sda1 /mnt
ardrone$ mount --bind /dev /mnt/dev
ardrone$ mount --bind /proc /mnt/proc
ardrone$ mount --bind /sys /mnt/sys
ardrone$ mount --bind /dev/pts /mnt/dev/pts
ardrone$ chroot /mnt
```

We finish the debootstrap process from within the ARDSYS partition.

```
ardsys$ cd debootstrap
ardsys$ ./debootstrap --second-stage
ardsys$ exit
```

We also copy the `BusyBox` [16] program of the original Linux to our new one. BusyBox provides light versions of several standard Linux tools.

```
ardrone$ cp /bin/busybox /mnt/bin/
```

The new Linux operating system is now fully installed and ready to run.

2.5 ARDSYS auto-boot

We are now going to modify the core init scripts of the drone and ARDSYS partition in order to switch automatically from the drone operating system to ARDSYS if the USB flash drive is plugged at drone startup time. The startup process is summarized in Figure 1.

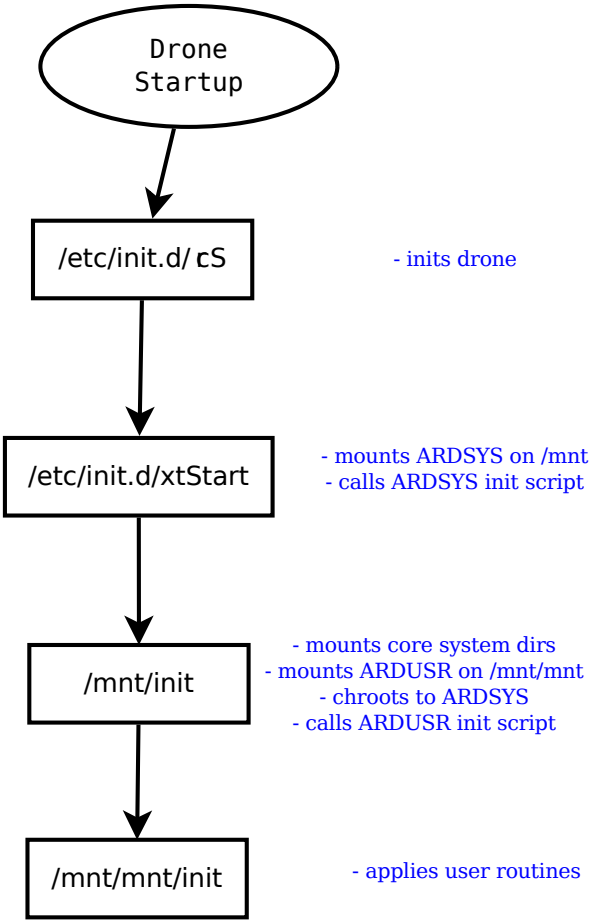

Figure 1: ArDroneXT startup

We create an init script which mounts the ARDSYS partition (*xtStart*); then we insert a call to it in the main init process of the drone (*rcS*). We also save a backup version of the drone init script.

```
ardrone$ touch /etc/init.d/xtStart
ardrone$ chmod 777 /etc/init.d/xtStart
ardrone$ cp /etc/init.d/rcS /etc/init.d/rcS.back
ardrone$ echo '/etc/init.d/xtStart&' >> /etc/init.d/rcS
```

The content of the file */etc/init.d/xtStart* has to be as follows.

```
#!/bin/sh

sleep 4;
if [ -e /dev/sda ]
then
  mount /dev/sda1 /mnt;
  /mnt/init;
fi
```

Then, we create the init script which mounts devices into the ARDSYS partition and chroots to it. Finally, the script calls the init script of the ARDUSR partition. This last script will be developed in the next section.

```
ardrone$ touch /mnt/init
ardrone$ chmod 777 /mnt/init
```

The content of the file */mnt/init* has to be as follows.

```
#!/bin/sh

sleep 3;
IRD=$(dirname $(readlink -f $0));
mount -t tmpfs -o size=8M tmpfs $IRD/tmp;
mount --bind /dev $IRD/dev;
mount --bind /proc $IRD/proc;
mount --bind /sys $IRD/sys;
mount --bind /dev/pts $IRD/dev/pts;
mount /dev/sda3 $IRD/mnt;
chroot $IRD /mnt/init;
```

Then, we test the init script process by rebooting the drone and checking if ARDSYS directories are well mounted after drone startup.

```
ardrone$ reboot
```

Once the drone has started, we reconnect the Linux base station to the drone as explained in paragraph 2.4.2. Then, we get a telnet shell on the drone.

```
linux$ telnet 192.168.1.1
```

We can check that all the directories are properly mounted.

```
ardrone$ mount
/dev/sda1 on /mnt type ext2
tmpfs on /mnt/tmp type tmpfs
dev on /mnt/dev type tmpfs
proc on /mnt/proc type proc
sys on /mnt/sys type sysfs
devpts on /mnt/dev/pts type devpts
/dev/sda3 on /mnt/mnt type vfat
```

The ARDSYS auto-boot process is now complete.

3 ArDroneXT customization

3.1 ARDSYS configuration

There are some modifications that can be applied to the ARDSYS partition.

Upgrading packages repository

```
# file /etc/apt/sources.list
deb http://ftp.fr.debian.org/debian/ squeeze main contrib non-free
deb-src http://ftp.fr.debian.org/debian/ squeeze main contrib non-free
deb http://security.debian.org/ squeeze/updates main contrib non-free
deb-src http://security.debian.org/ squeeze/updates main contrib non-free
```

Changing host name

```
# file /etc/hostname
ArDroneXT
```

Customizing telnet connection message

```
# file /etc/issue
|-----------------------|
|-My Wonderful ArDrone-|
|-----------------------|
```

Defining shell init

```
# file /root/.bashrc
# configure as you wish
```

We also need to create a script in order to directly launch a root shell. It will be used in the next section. This script is called *sroot* and has to be installed in the */bin* of the ARDSYS partition.

```
ardrone$ touch /mnt/bin/sroot
ardrone$ chmod 777 /mnt/bin/sroot
```

The content of the file */mnt/bin/sroot* has to be as follows.

```
#!/bin/sh

if [ $# -eq 0 ]
then
  su root -c /bin/bash;
else
  su root -c "$*";
fi
```

We also need to create scripts in order to halt and reboot the drone. Indeed, the operating system in the ARDSYS partition (i.e. ArDroneXT) cannot perform these tasks directly, their use is reserved to the core drone operating system. Calling *halt* or *reboot* from the ARDSYS partition will not have any effect. Therefore, we use the local telnet server to send instructions to the original base system of the drone. The script will be called *sreturn* and will be placed in the */bin* of the ARDSYS partition.

```
ardrone$ touch /mnt/bin/sreturn
ardrone$ chmod 777 /mnt/bin/sreturn
```

The content of the file */mnt/bin/sreturn* has to be as follows.

```
#!/bin/sh

if [ $# -eq 0 ]
then
  /bin/busybox telnet 127.0.0.1;
else
  echo "$*;exit" |netcat 127.0.0.1 23;
fi
```

3.2 ARDUSR init script

We now have to create a user init script in the ARDUSR partition that will be launched by ArDroneXT just after executing the chroot. This script will enable to launch services at the drone startup time. It is suitable for instance to:

1. set the drone hostname;
2. launch a telnet server inside the ARDSYS partition (port 2323 for instance);
3. enable the drone to communicate with internet to perform some packages installation.

The telnet server will make it possible to run a root shell directly on ArDroneXT without passing by the original drone base system (i.e. ArDroneBS).

Here is an example of such a script which is installed in *ARDUSR/init*.

```
#!/bin/bash

TELNET_PORT=2323;
GATEWAY_IP=192.168.1.2;

hostname $(cat /etc/hostname);
/bin/busybox telnetd -p ${TELNET_PORT} -l '/bin/sroot';
route add default gw ${GATEWAY_IP};
echo "nameserver␣${GATEWAY_IP}" > /etc/resolv.conf;
```

3.3 ARDSYS upgrade

Now we plug the USB flash drive and start the drone. After enabling the Wi-Fi connection, we have to contact our new telnet server.

```
$ telnet 192.168.1.1 2323

|-------------------------|
|-My Wonderful Ar Drone-|
|-------------------------|

ArDroneXT$
```

We have to enable the drone to communicate with internet in order to install new packages. To achieve this task, we need (on the Linux base station):

- to enable IP forwarding;
- to start a DNS server relay;
- to set a NAT masquerading.

The require commands are listed bellow.

```
linux$ echo 1 > /proc/sys/net/ipv4/ip_forward
linux$ dnsmasq
linux$ iptables -t nat -A POSTROUTING --source 192.168.1.0/24 \
      -o eth0 -j MASQUERADE;
```

Now the drone can communicate with internet.

```
ArDroneXT$ ping -c1 www.labri.fr
PING www3.labri.fr (147.210.8.59) 56(84) bytes of data.
64 bytes from www3.labri.fr (147.210.8.59): icmp_req=1 ttl=61 time=3.84 ms

--- www3.labri.fr ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 3.845/3.845/3.845/0.000 ms
```

We can now perform a Debian update/upgrade.

```
ArDroneXT$ apt-get update
ArDroneXT$ apt-get upgrade
ArDroneXT$ apt-get dist-upgrade
ArDroneXT$ apt-get autoclean
ArDroneXT$ apt-get autoremove
```

We can also install new packages. For instance, we can install `Python` [17].

```
ArDroneXT$ apt-get install python
```

It is of course possible to install more packages if needed.

4 Ad-hoc networking

In this section we modify the ARDUSR init script in order to force the drone to join a specific *ad-hoc* network. This feature enables to create a swarm of several Ar.Drone 2 which shares the same Wi-Fi network.

We first have to install `wireless-tools` [12] in order to configure the Wi-Fi connection.

```
ArDroneXT$ apt-get install wireless-tools
```

4.1 ARDUSR init script upgrade

We now upgrade the ARDUSR init script in order to force the drone to join a pre-existing *ad-hoc* network with a predefined IP address. We also need to enable the broadcast ICMP reply (i.e. ping reply) on the drone in order to check ad-hoc network peers status. By default, the drone ignores ICMP requests sent in broadcast. The resulting script is detailed hereafter.

```
#!/bin/bash

IP=192.168.1.1;
TELNET_PORT=2323;
GATEWAY_IP=192.168.1.254;
ESSID=ArDroneXT;

hostname $(cat /etc/hostname);
ifconfig ath0 down;
iwconfig ath0 mode ad-hoc;
iwconfig ath0 essid ${ESSID};
iwconfig ath0 commit;
ifconfig ath0 up;
ifconfig ath0 $IP;

echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts;
/bin/busybox telnetd -p ${TELNET_PORT} -l '/bin/sroot';
route add default gw ${GATEWAY_IP};
echo "nameserver ${GATEWAY_IP}" > /etc/resolv.conf;
```

We can create a DHCP server on our Linux base station and force the drone to use it in order to get its network configuration automatically at startup. However we will first have to bypass the original drone DHCP server. The resulting script is given bellow.

```
#!/bin/bash

TELNET_PORT=2323;
ESSID=ArDroneXT;

hostname $(cat /etc/hostname);

iptables -t filter -A OUTPUT -p udp --sport 67 -j DROP;
iptables -t filter -A INPUT -p udp --dport 67 -j DROP;

ifconfig ath0 down;
iwconfig ath0 mode ad-hoc;
iwconfig ath0 essid ${ESSID};
iwconfig ath0 commit;
ifconfig ath0 up;
dhclient ath0;

echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts;
/bin/busybox telnetd -p ${TELNET_PORT} -l '/bin/sroot';
```

We can setup the DHCP server on the Linux base station with *dnsmasq* [11]. In the following example, our Linux base station has a wireless interface called *wlan0* which IP address is set to 192.168.1.254.

```
linux$  /usr/sbin/dnsmasq --interface=wlan0 \
                          --strict-order \
                          --expand-hosts \
                          --domain="ArDroneXT" \
                          --dhcp-range=192.168.1.10,192.168.1.100,2h \
                          --dhcp-option="option:router,192.168.1.254"
```

4.2 Ad-Hoc network creation

Then, we just have to turn the drone on and to configure the Wi-Fi chip of the base station to join the ad-hoc network. Please note that it is recommended that the Linux base station initiates the *ad-hoc* network to prevent any drone join problem. Indeed, we have noticed that the drone sometimes encounters some problems to init a new ad-hoc network. This problem stills unresolved for the moment.

```
linux$ ifconfig wlan0 down
linux$ iwconfig wlan0 mode ad-hoc
linux$ iwconfig wlan0 essid ArDroneXT
linux$ ifconfig wlan0 up
linux$ ifconfig wlan0 192.168.1.254
```

If we choose the DHCP configuration to set the drone IP address automatically, we have to launch the *dnsmasq* tool as explained in the previous section.

Finally, we can test if several drones are part of the swarm by using a broadcast ping. Each drone that is part of the *ad-hoc* network should reply and thus be visible. In the following example, 2 drones are part of the network (192.168.1.10 and 192.168.1.11), and the Linux base station has the IP address 192.168.1.254.

```
linux$ ping -b 192.168.1.0
PING 192.168.1.0 (192.168.1.0) 56(84) bytes of data.
64 bytes from 192.168.1.254: icmp_req=1 ttl=64 time=0.022 ms
64 bytes from 192.168.1.10: icmp_req=1 ttl=64 time=1.06 ms
64 bytes from 192.168.1.11: icmp_req=1 ttl=64 time=1.17 ms
64 bytes from 192.168.1.254: icmp_req=2 ttl=64 time=0.020 ms
64 bytes from 192.168.1.10: icmp_req=2 ttl=64 time=1.02 ms
64 bytes from 192.168.1.11: icmp_req=2m ttl=64 time=1.15 ms
...
```

5 Swarm of several Ar.Drone 2

In order to configure other drones, we have, for each one:

- to clone the USB flash drive;
- to create the */etc/init.d/xtStart* script in the drone root file system as explained in section 2.5;
- to upgrade the */etc/init.d/rcS* script in the drone root file system as explained in section 2.5;
- optionally:
  - to change the drone IP in *ARDUSR/init* if it is set in this file;
  - to customize the hostname of the drone.

To clone a USB flash drive we can use the **dd** [18] software. In the following example, the original USB flash drive is *sdb* and the new one is *sdc*.

```
linux$ dd if=/dev/sdb of=/dev/sdc bs=4096 conv=notrunc,noerror,sync
```

6 Supporting ARM EABI (using ARDSYS partition without drone)

This section enables to run programs directly inside the ARDSYS partition without using a drone.

6.1 The cross-execution problem

As explained above, the architecture of programs in ARDSYS is ARM EABI compliant. There is a strong probability that our Linux base station lays on a different processor architecture like x86_64 for instance. Consequently, it is theoretically impossible to launch ARDSYS programs directly on our Linux base station.

However, it is possible to use an emulator in order to solve the problem. A well known solution is QEMU [19]. It can emulate a lot of architectures including ARM EABI. After installing QEMU on our Linux base station, we will then be able to run ARM EABI programs directly.

For example, we create a simple ARM EABI executable thanks to our ARDDEV partition as follows.

```
linux$ export PATH=$PATH:/mnt/ARDDEV/arm-2012/bin/
linux$ cd /tmp
linux$ echo "#include <stdio.h>" > hello.c
linux$ echo 'int main(){printf("It works\n"); return 0;}' >> hello.c
linux$ arm-none-linux-gnueabi-gcc hello.c -o hello.elf
linux$ file hello.elf
hello.elf: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV)
dynamically linked (uses shared libs)
linux$ ./hello.elf
./hello.elf: cannot execute binary file: invalid architecture for exec()
```

As we can see, we cannot run the *hello.elf* executable because of its ARM EABI architecture. We use QEMU to bypass the problem.

```
linux$ qemu-arm ./hello.elf
/lib/ld-linux.so.3: No such file or directory
```

The error we get is due to the fact that the *hello.elf* program uses shared libraries which are not installed on our Linux base station. We thus have to compile our program in static mode to solve this issue.

```
linux$ arm-none-linux-gnueabi-gcc hello.c --static -o hello.elf
linux$ file hello.elf
hello.elf: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV)
statically linked
linux$ qemu-arm ./hello.elf
It works
```

6.2 Applying cross-execution to ARDSYS

It is possible to adapt the previous solution to the whole ARDSYS partition in order to support the ARM EABI architecture directly on our Linux base station. The trick is to indicate to our system that it has to use QEMU for all ARM EABI programs. Most Linux distributions enable this binding when QEMU is installed. If your Linux distribution does not perform this action, please refer to Linux kernel module `binfmt_misc` [20].

Then, we have just to perform a *chroot* inside ARDSYS. Please note that a *chroot* implies a file system isolation, thus we have to copy the QEMU executable inside the ARDSYS partition. We also have to use the static version of QEMU as explained in section 6.1.

```
linux$ file $(which qemu-arm)
qemu-arm: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux)
dynamically linked (uses shared libs)
linux$ file $(which qemu-arm-static)
qemu-arm-static: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux)
statically linked
```

We copy the static version of the executable inside the ARDSYS partition.

```
linux$ cp $(which qemu-arm-static) /mnt/ARDSYS/usr/bin/
```

Now we can mount sub-directories and *chroot* in the same way we would have done if using a drone.

```
linux$ mount --bind /dev /mnt/ARDSYS/dev
linux$ mount --bind /proc /mnt/ARDSYS/proc
linux$ mount --bind /sys /mnt/ARDSYS/sys
linux$ mount --bind /dev/pts /mnt/ARDSYS/dev/pts
linux$ chroot /mnt/ARDSYS
ArDroneXT$
```

Then we can run any ARM EABI application. Please not that we first have to call the DHCP server because the network configuration (DNS server, default gateway, etc.) is not inherited when *chrooting*.

```
ArDroneXT$ dhclient eth0
# run any commands
```

References

[1] Parrot. *ArDrone 2.0*. http://ardrone2.parrot.com.

[2] Linux Kernel. *Linux*. http://www.kernel.org.

[3] IEEE. *802.11*. http://www.ieee802.org/11/.

[4] Debian. *Debian*. http://www.debian.org.

[5] Kingston. *DataTraveler G3*. http://www.kingston.com/fr/usb/personal_business/.

[6] Asus. *WL-167G [Ralink RT2571]*. http://fr.asus.com/Networks/Wireless_Adapters/WL167g.

[7] Gnu. *Parted*. http://www.gnu.org/software/parted/.

[8] Theodore Ts'o. *E2fsprogs*. http://e2fsprogs.sourceforge.net.

[9] Daniel Baumann. *Dosfstools*. http://www.daniel-baumann.ch/software/dosfstools.

[10] Debian. *Debootstrap*. http://wiki.debian.org/Debootstrap.

[11] Simon Kelley. *Dnsmasq*. http://www.thekelleys.org.uk/dnsmasq/doc.html.

[12] HP. *Linux Wireless Tools*. http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html.

[13] Mentor Graphics. *GCC/G++ ARM EABI*. http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/.

[14] Debian. *Debian ARM EABI Port*. http://wiki.debian.org/ArmEabiPort.

[15] Gnome. *NetworkManager*. http://projects.gnome.org/NetworkManager.

[16] Bruce Perens. *BusyBox*. http://www.busybox.net.

[17] Python Software Foundation. *Python*. http://www.python.org.

[18] Free Software Foundation. *Coreutils*. http://www.gnu.org/software/coreutils/.

[19] Fabrice Bellard. *QEMU*. http://qemu.org.

[20] Linux Kernel. *Binary Formats Kernel Support*. https://www.kernel.org/doc/Documentation/binfmt_misc.txt.