Network boot in GRUB2

GRUB mini-summit 2020

Michał Żygowski

3MDEB

Agenda

- Introduction
- Network boot - what for?
- Network boot in GRUB2
- Other implementations
- Performance issues
- Debugging
- Comparing with iPXE
- Conclusions
- Other issues
- Q&A

Introduction

Michał Żygowski
Firmware Engineer

- @*miczyg*
- michal.zygowski@3mdeb.com
- linkedin.com/in/miczyg
- facebook.com/miczyg1395

- Braswell SoC, PC Engines and Protectli maintainer in coreboot
- interested in:
  - advanced hardware and firmware features
  - coreboot
  - security solutions

Network boot - what for?

Many times in daily work or even at home we may face situation when we need system recovery.

How often it happens that we do not have a free storage device at hand to burn a live image?

In such situation the network boot option may be a time-saver or even a life-saver

Network boot - what for?

- No free storage device at hand to burn an live image on OR
- All storage medias are filled with important data, moving the around would consume much time
- The live/installer images still weigh a lot, downloading an ISO and then burning it on a USB stick for example, take another couple of minutes of our precious time
- Solution? Boot a kernel (and initrd) directly with your favorite bootloader and save efforts of bothering with physical medias
- Potentially can create one-touch installation and provisioning of certain systems

Network boot in GRUB2

So how does it work in GRUB2?

Two main types of implementations of GRUB2:

1. i386-pc (for legacy boot mode)
2. EFI: i386 or x64

What is common?

- GRUB2 does not have any NIC drivers on its own
- GRUB2 relies on the interfaces provided by the firmware

What is different?

Network boot in GRUB2

What is different?

GRUB2 for i386-pc:

- the most limited network boot implementation
- relies on the PXE/UNDI API to talk with NIC
- PXE/UNDI can be delivered only with a PXE environment
- GRUB2 must be launched by PXE in order to have network working!

GRUB2 for EFI:

- standardized by UEFI **Simple Network Protocol**
- available almost on every UEFI implementation nowadays
- requires the UEFI network stack and NIC to be enabled in the setup menu
- even though you enable the required options it is not guaranteed it will work (buggy UEFI implementations)

Other implementations

- iPXE (https://ipxe.org/) - network boot dedicated project, flexible PXE software for many architectures and software stacks
- proprietary PXE, e.g Intel drivers (https://downloadcenter.intel.com/download/29137?v=t)
- webboot for LinuxBoot (https://github.com/u-root/webboot) - written in Go multipurpose network boot environment
- Etherboot/gPXE (http://etherboot.org/wiki/) - another PXE compliant implementation and good replacement for proprietary PXE ROMs

Performance issues

> definitely, there was a problem with tftp, but then I tried GRUB 2.04 using its tftp and http support. Performance of GRUB TFTP and HTTP modules is terrible, I will get to address that with maintainer, maybe on GRUB and 3mdeb minisummit 2020.
>
> -- *Piotr Król, founder of 3mdeb*

https://github.com/xcp-ng/host-installer/issues/18#issuecomment-605654088

Thank you Piotr. Now let's check this out...

Performance issues

Let's craft a simple standalone image of GRUB for EFI to get as close to Piotr's environment as possible. Let's build a GRUB2:

```
./linguas.sh #optional
./bootstrap
./autogen.sh
./configure --prefix=$HOME/local  --target=x86_64 --with-platform=efi
make && make install
$HOME/local/bin/grub-mkstandalone -o grubx64.efi -O x86_64-efi\
    /boot/grub/grub.cfg=./grub.cfg
```

What I did here is a magic trick to create a fully featured standalone EFI binary with an embedded config file. So I don't have to worry to type commands I want...

Performance issues

Quick look at my grub.cfg file:

```
serial --speed=115200 --word=8 --parity=no --stop=1 --unit=0
terminal_input --append serial
terminal_output --append serial

set debug=efinet,loader,linux,http

echo "Installing modules"
insmod efinet
insmod http
insmod net

net_dhcp efinet0

echo "Loading Linux"
linux (http,archive.ubuntu.com)/ubuntu/dists/focal-updates/(...)/amd64/linux initrd=initrd.gz
echo "Loading initrd"
initrd (http,archive.ubuntu.com)/ubuntu/dists/focal-updates/(...)/amd64/initrd.gz
echo "Ready to boot"
```

Performance issues

Let's summarize:

- Linux kernel 11MB
- initrd 49MB
- **Total 60MB**

https://asciinema.org/a/370064

- Initial ramdisk is not even downloaded because of destination unreachable error
- Downloading a 11 MB Linux kernel takes about 47 seconds (~234kB/s)

!!!TOTAL TIME REQUIRED: 4.25 minutes!!!

There is certainly something wrong with it. After some debugging it occurred that GRUB is not using full MTU in packet transmissions: https://asciinema.org/a/369935

- initially when HTTP requests are sent full MTU is used
- when the download is starting the **MTU usage oscillates around 50% or lower**
- when debug is enabled the packets are received **each 20ms**
- sometimes the MTU usage is higher and receive intervals are lower, it is not consistent
- quick calculation: `1s/20ms*1500(MTU)*50%=37,5kB/s` so `60MB/37,5kBs=1600s=~26,5min`

Let's calculate the packet receive frequency based on asciinema:
`60MB/256s=234kB/s 1s/234kB*1500(MTU)*50%=3.2ms`

Is this possible? Does it look reasonable?

Comparing with iPXE

Let's compare it with iPXE: https://asciinema.org/a/370071

It only takes 17 seconds to download 60MB... (3.53MB/s)

With debug:

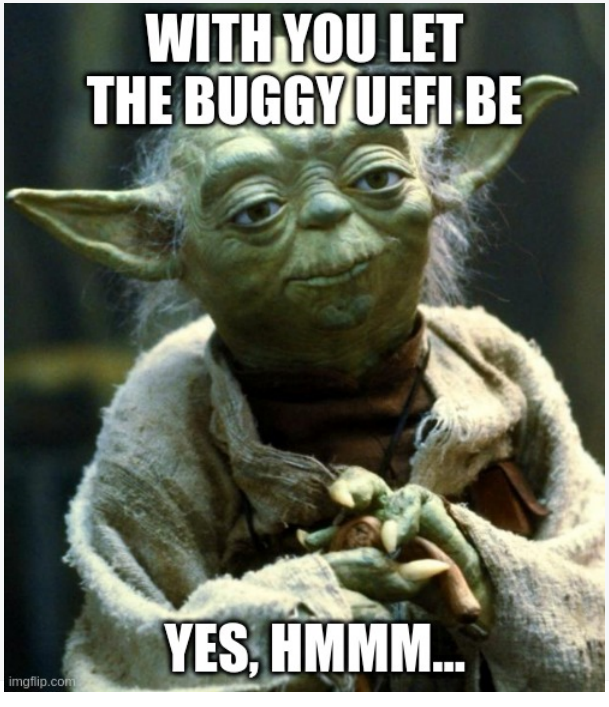
- many various drivers even for EFI: SNP, SNPONLY, EFI_SNP, EFI NII, NII; can't get hold of what is really used
- typically the iPXE uses either native driver or NII(Network Interface Identification Protocol)/UNDI
- it occurred that iPXE uses the native Intel NIC driver, I had to force it to use SNP
- well I did not manage to compare it apple to apple (because SNP is broken on the platform I have been testing, Intel NICs; although I had more luck with Realtek's some time ago)

Comparing with iPXE

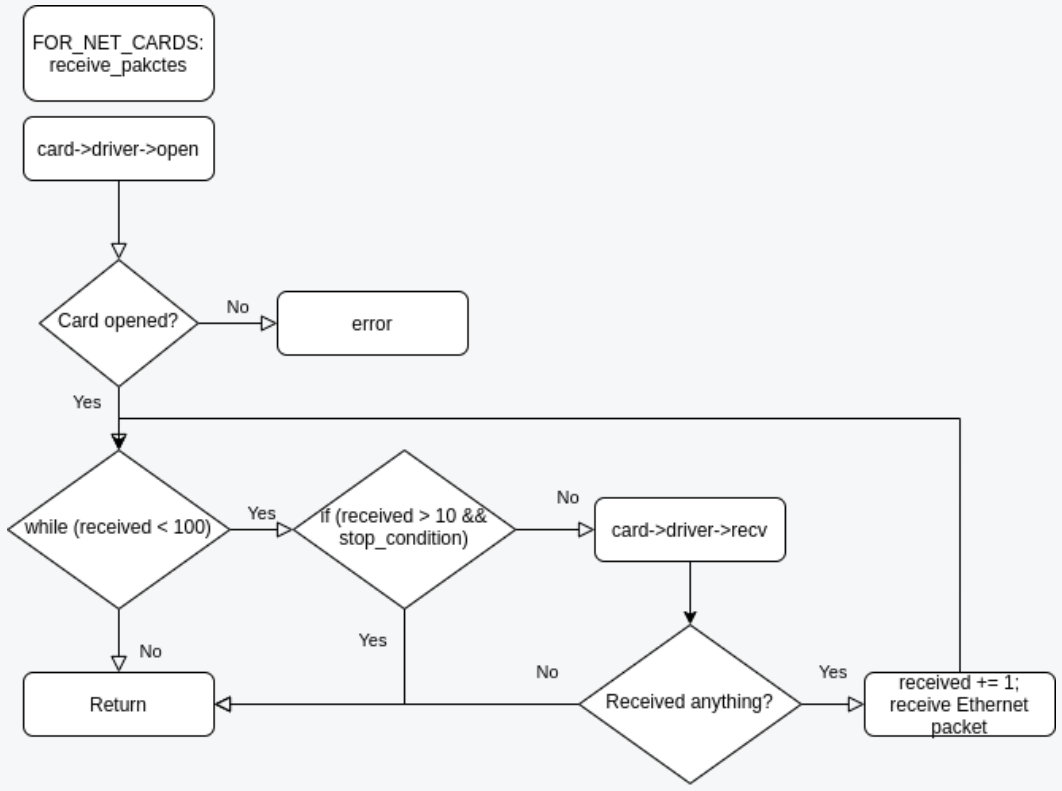
```
SNP net0 could not set station address before initialising:\
    Error 0x7?f594087 (http://ipxe.org/7f594087)
SNP net0 retrying initialisation (retry 1)
(...)
SNP net0 retrying initialisation (retry 9)
SNP net0 could not set receive filters Oxif (have 0x00):\
    Error 0x7f594082 (http://ipxe.org/7f594082)
SNP net0 could not set receive filters 0x07 (have 0x00):\
    Error Ox7f594082 (http: //ipxe.org/7f594082)
SNP net0 st 2 type 1 hdr 14 pkt 1500 rxflt 0x15/0x1f nvram 512 acc 4 mcast 0/8
SNP net0 hw 00:25:90:b9:e3:c2 addr 00:25:90:b9:e3:c2 beast ff:ff:ff:ff:ff:ff
SNP net0 media not present
```

Comparing with iPXE

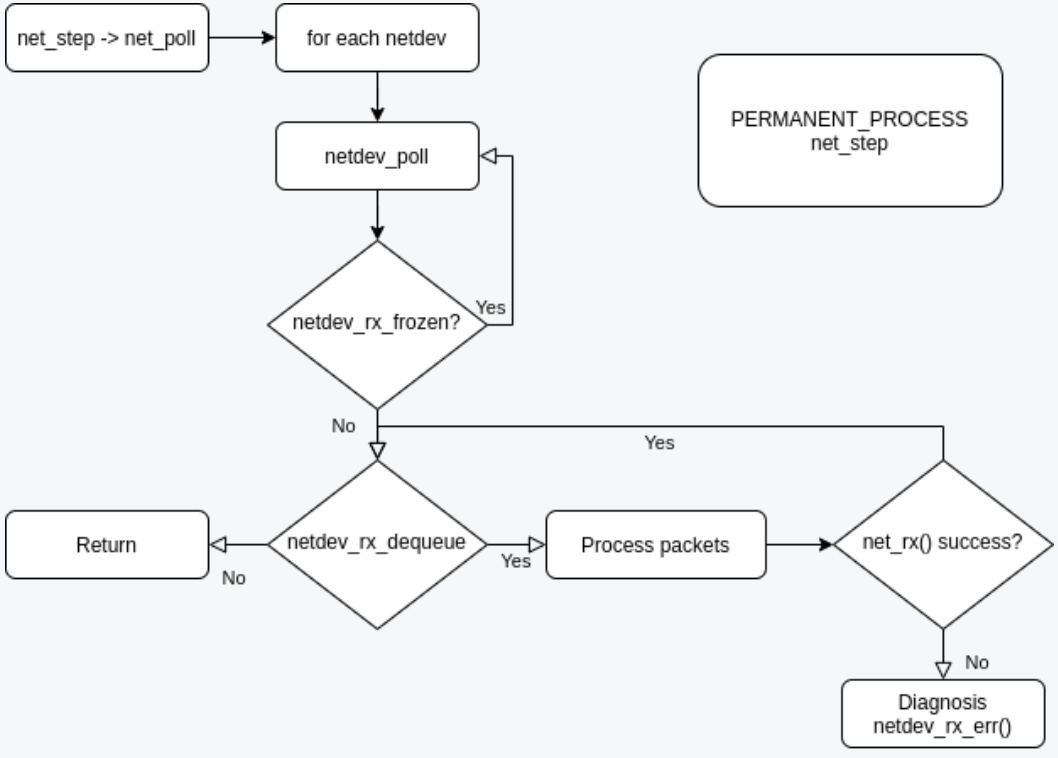
"let the buggy UEFI implementations be with you" because creating something that just works is sometimes too difficult



Comparing with iPXE



Comparing with iPXE



Conclusions

- iPXE is superior in terms of network drivers and its performance (well GRUB2 has other advantages and also other main tasks)
- GRUB has **less than 10%** of iPXE network performance (250kB/s vs 3.53MB/s)
- with a subtle changes the GRUB2 network stack could reach the iPXE level

- https://git.savannah.gnu.org/cgit/grub.git/tree/grub-core/net/net.c line 1490

```
/* Maybe should be better have a fixed number of packets for each card
   and just mark them as used and not used.  */
```

It suggests there is some field for improvement.

- the API and polling in iPXE seems to be better optimized software design (facts speak for themselves) and the software overhead is lower than in GRUB

Other issues

- Python3 HTTP server seems to crash when used as a source for boot images and kernels with GRUB EFI. Credits to Piotr Król for this finding
- Haven't managed to find time to check/debug it, but it should be pretty simple.

Q&A