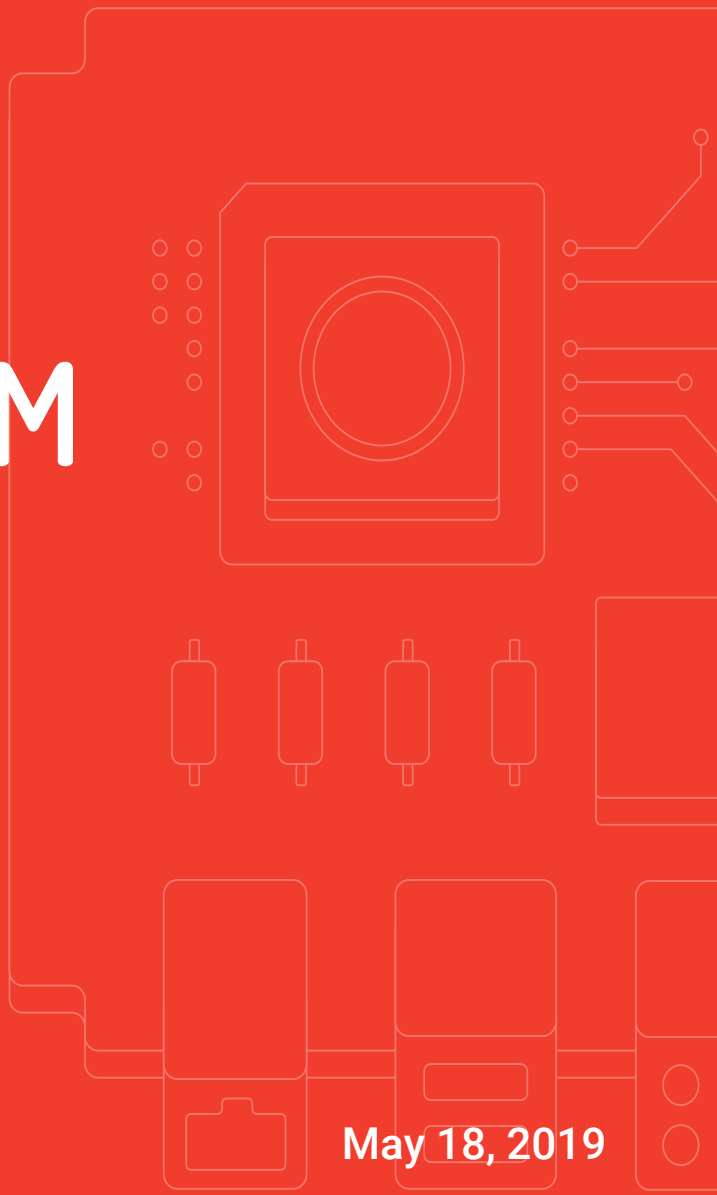
Zagreb, NKOSL

Gentoo Linux on ARM platforms

Davor Popović · Mak Krnic

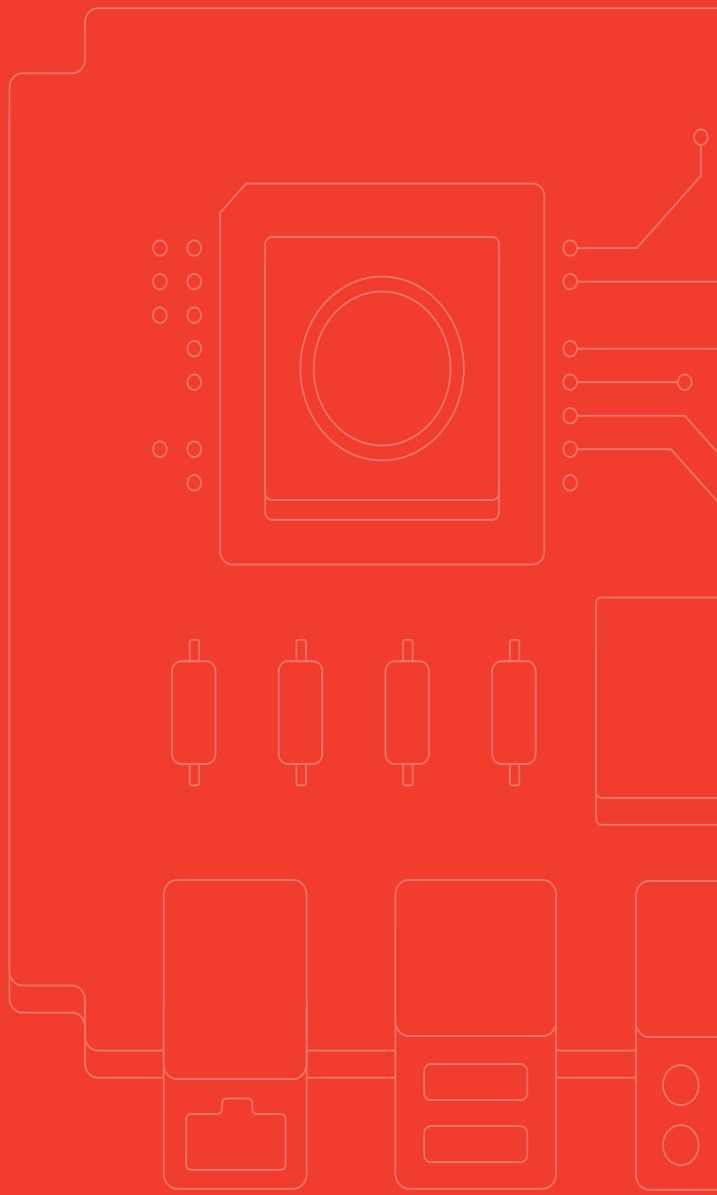


May 18, 2019

About us

- Davor Popović, Mak Krnic
- Delivering solutions based on Linux, OpenWrt and Yocto
  - Focused on software in network edge and CPEs
- Continuous (commercial) participation in Open Source projects

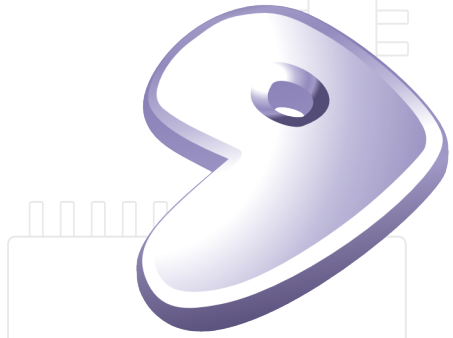
Gentoo Linux



sartura

About Gentoo Linux

- Source-based Linux distribution
- Almost nothing out of the box
  - User-defined base system
  - Manual kernel compilation
    - genkernel tool
  - portage/emerge package manager



gentoo linux™

Portage

- *The heart of Gentoo*
- The only package requiring dependencies
- Support for both binary and source-based packages

/etc/portage/*

- System-wide configuration
- `make.conf`
  - `CHOST`, `CBUILD`, `ACCEPT_KEYWORDS`, `USE`, `ROOT`, `MAKEOPTS`, `PORTDIR`
- `repos.conf`
  - Source repositories configuration

Portage repositories and profiles

- Repositories
  - Default repository at `/usr/portage`
  - Custom repositories (`local/git/...`)
    - Patched or unsupported packages
    - Custom profiles
- Profiles
  - By default located in `/usr/portage/profiles`
  - In any of the repositories

Cross-compiling

- Process of creating binaries for use on different platform than the one it is being compiled on
- cross-toolchain
  - gcc
  - binutils – ld, ar, objcopy, objdump, ...
  - libc – glibc, uClibc, musl, ...

- crossdev - gentoo native cross-toolchain generator
- crosstool-ng – alternative cross-toolchain generator
- Other toolchains (Buildroot, openembedded)
- https://toolchains.bootlin.com/

```c
// hello.c
#include<stdio.h>

#ifdef __aarch64__
#define ARCHITECTURE "arm64"
#elif __amd64__
#define ARCHITECTURE "x86_64"
#else
#define ARCHITECTURE "unknown"
#endif

int main() {
  printf("Hello, world, from " ARCHITECTURE "\n");

  return 0;
}
```

```
gcc ./hello.c -o hello.x86_64
aarch64-unknown-linux-gnu-gcc ./hello.c -o hello.aarch64
```

```
hello.x86_64: ELF 64-bit LSB pie executable, x86-64, version 1 (
    SYSV), dynamically linked, interpreter /lib64/ld-linux-x86
    -64.so.2, for GNU/Linux 3.2.0, not stripped
file hello.aarch64
hello.aarch64: ELF 64-bit LSB pie executable, ARM aarch64,
    version 1 (SYSV), dynamically linked, interpreter /lib/ld-
    linux-aarch64.so.1, for GNU/Linux 3.7.0, not stripped
```

```
./hello.x86_64
Hello, world, from x86_64

./hello.aarch64
zsh: exec format error: ./hello.aarch64
```

crossdev

- Gentoo native cross toolchain generator
- Integration with system
  - gcc, binutils, etc. managed by portage
- Automatically using build host's portage configuration

cross-emerge

- Wrapper scripts for emerge
- Using host's (cross) toolchain
  - Producing binaries for the target architecture
- Usable with any toolchain
- Similar outcome as chroot
- Installs full OS structure to new root
- Suitable for creating rootfs

Installing the base file system

- `cross-emerge [-av] @world`
  - Profile dependent
  - Init system, shell, utils
- Edit `/etc/shadow`
  - `root:*:17140:0:::::`

Cross-compiling issues

- Some packages can't be cross-compiled for all architectures or are built incorrectly
- QEMU
  - A generic and open source machine emulator and virtualizer
- Create ***almost*** complete rootfs and then finish building it in QEMU
- Downside: **slow**

Kernel

- Not built using (cross) emerge
- Obtaining the source manually from kernel.org
- Cross-compiling (env vars)
  - `ARCH, CROSS_COMPILE, INSTALL_MOD_PATH, INSTALL_PATH`
- Device tree (.dts and .dtb)

Assembling it all

- Assumptions
  - Rootfs (built with cross-emerge) at `/opt/rootfs`
  - Kernel source at `/opt/rootfs/usr/src/linux`
  - Target dir at `/opt/stage4`
- Copy base files

```
1 | cd /opt/rootfs
2 | rsync -ruta * --exclude 'usr/src/*' --exclude 'tmp/*' --
  |     exclude 'usr/portage/*'
3 | /opt/stage4/
```

- Copy the kernel and device tree

```
1  mkdir /opt/stage4/boot
2  cd usr/src/linux/
3  cp arch/arm64/boot/Image /opt/stage4/boot/Image
4  cp arch/arm64/boot/dts/marvell/armada-8040-mcbin-singleshot.
       dtb /opt/gentoo/rootfs/boot/armada-8040-mcbin-singleshot
       .dtb
```

Flashing the image

- SD card

```
cp -a /opt/stage4/* /mnt/sdcard
```

- Flash ROM

```
1  fallocate -l 128M ./rootfs.ext4
2  mkfs.ext4 ./rootfs.ext4
3  mount ./rootfs.ext4 /mnt/new-rootfs
4  cp -a /opt/stage4/* /mnt/new-rootfs/
5  umount /mnt/new-rootfs
6  dd if=./rootfs.ext4 /dev/sdX
```

Bootloader

- U-Boot

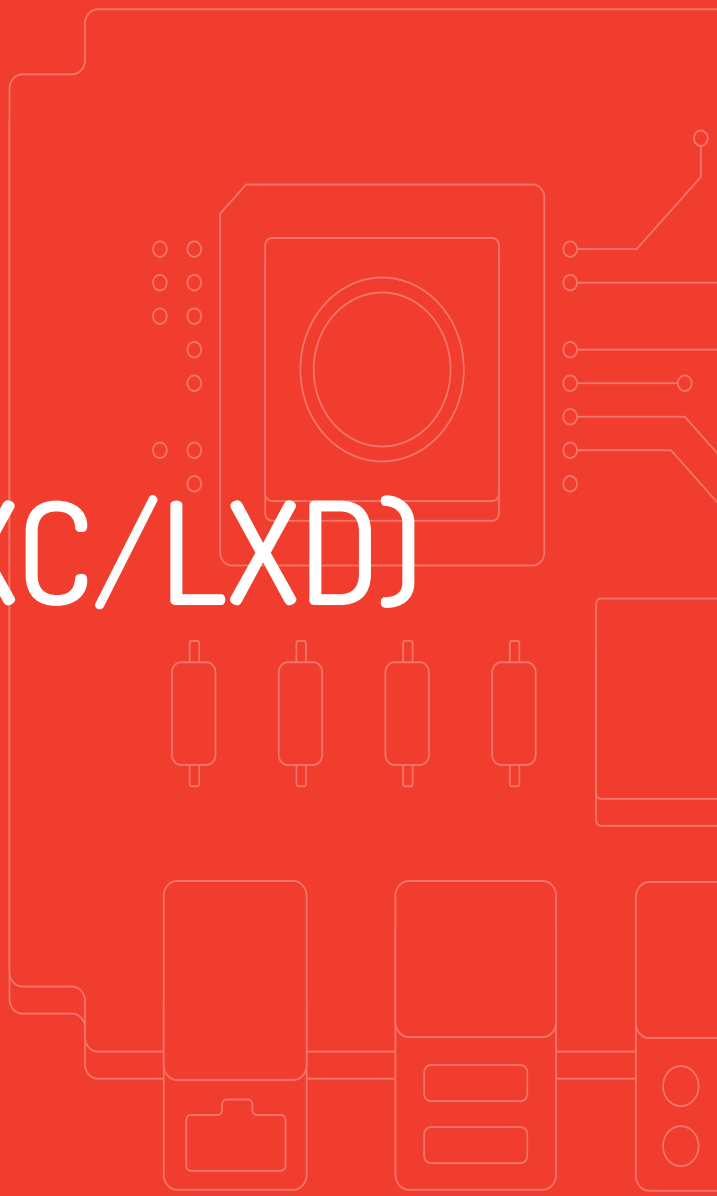
```
1  setenv bootargs 'console=ttyS0,115200 root=/dev/mmcblk1p1
       init=/lib/systemd/systemd rw rootwait'
2  setenv bootcmd 'mmc dev 1; ext4load mmc 1:1 0x5000000 /boot/
       Image;ext4load mmc 1:1 0x4f00000 /boot/armada-8040-mcbin
       -singleshot.dtb; booti 0x5000000 - 0x4f00000'
3  saveenv
4  boot
```

First boot

- Login as root, without password
- Configure the system
  - Root password
  - Network
  - Packages
  - ...

Questions

Linux Containers (LXC/LXD)

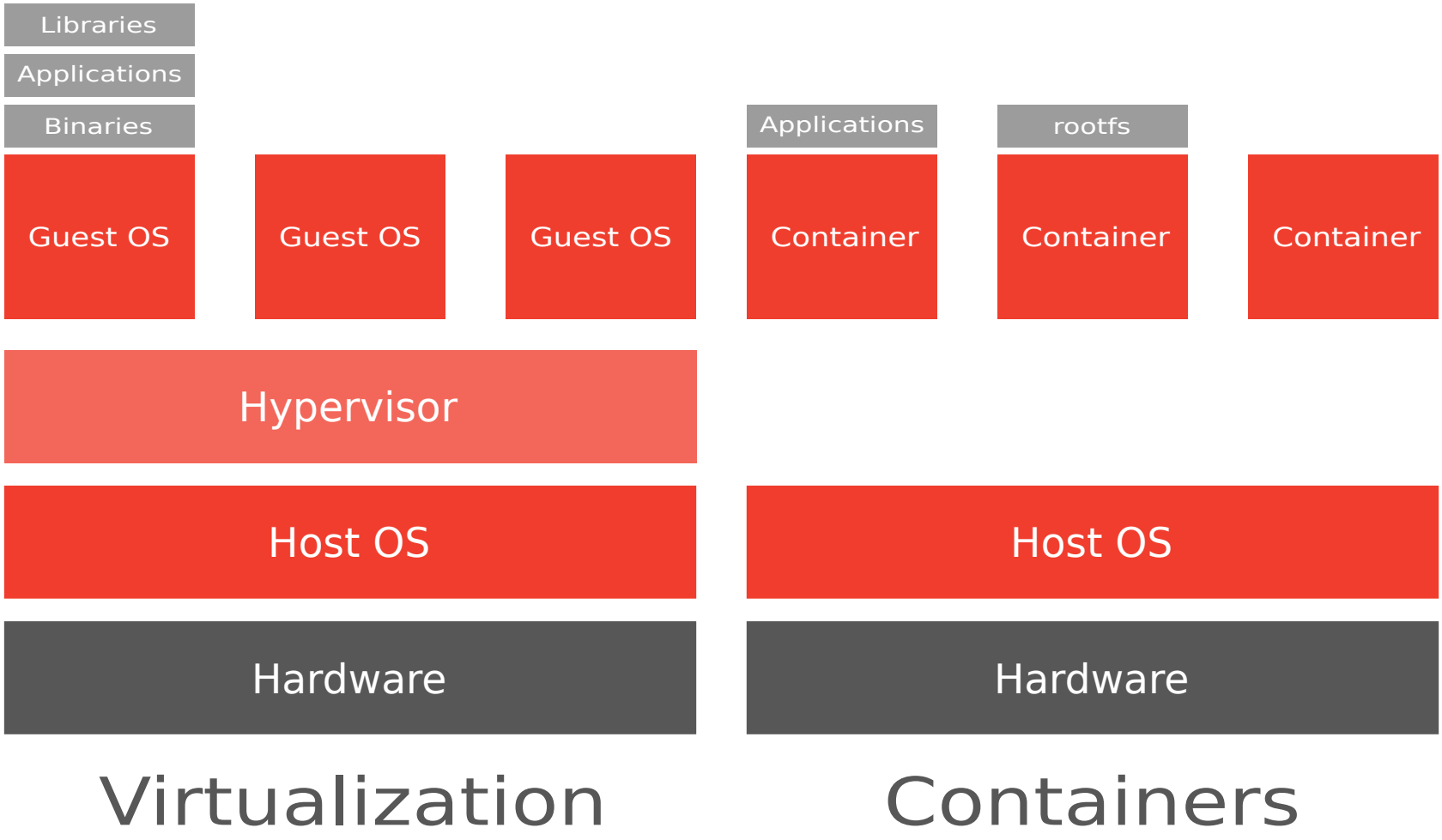


sartura

Virtualization Concepts

Two virtualization concepts:

- Hardware (full/para) virtualization:
  - Virtual machines - emulating complete hardware
- Operating system level virtualization:
  - Containers - kernel feature for simultaneously running more than one
    user-space instance


Libraries
Applications
Binaries
Guest OS
Guest OS
Guest OS
Hypervisor
Host OS
Hardware
Virtualization
Applications
rootfs
Container
Container
Container
Host OS
Hardware
Containers

What is LXC?

- Operating system level virtualization on GNU/Linux
- In-between ***chroot*** and complete virtual machine
- Can be used without hardware support for virtualization on SoC
  - Excellent for virtualization on embedded devices
- Easily configured as full featured file system or minimized as single app

LXC Features

- Namespaces
  - Lightweight process virtualization
  - A single or multiple processes have a different view on the system
  - Current support for: `ipc`, `uts`, `mount`, `pid`, `network` and `user`
  - In the past – support for running only privileged (root) containers
  - User namespaces – allow running unprivileged containers

- Apparmor and SELinux Profiles
  - Linux application security system
  - Switching to defined profiles/contexts before a container actually starts
- Seccomp policies
  - Allow filtering system calls
- Capabilities
  - Setting up which capabilities to keep/drop before starting containers
- Cgroups
  - Used for setting resource quotas (CPU, memory, I/O limits...)
  - Used for setting character or block devices accessible from container on the host

What is LXD?

- Container manager
  - Useful when running many containers
- Concept
  - Daemon + REST API
  - Accessible locally or over network
  - Command line tools communicate this way

- Secure by design (unprivileged containers, resource restrictions and much more)
- Scalable (from containers on your laptop to thousand of compute nodes)
- Intuitive (simple, clear API and crisp command line experience)
- Image-based (with a wide variety of Linux distributions published daily)
- Support for Cross-host container and image transfer (including live migration with CRIU)

- Advanced resource control (CPU, memory, network I/O, block I/O, disk usage and kernel resources)
- Device passthrough (USB, GPU, unix character and block devices, NICs, disks and paths)
- Network management (bridge creation and configuration, cross-host tunnels, ...)
- Storage management (support for multiple storage backends, storage pools and storage volumes)

Working with LXD

- Prerequisites:
  - Initialized deamon (LXD)
  - Rootfs and metadata
  - Container image
  - Container profile

LXD init

- Configuring the LXD daemon

```
lxd init
```

Prepare rootfs

```
1 | mkdir -p ~/gentoo
2 | cd ~/gentoo
3 | mkdir gentoo-rootfs
4 | cp ~/gentoo-rootfs.tar.gz gentoo-rootfs/
```

- In the same folder, create metadata for container
  - Metadata describes basic information about the container

Minimal metadata template file

```
vim metadata.yaml
```

```
1  architecture: "aarch64"
2  creation_date: 1554382805      # mandatory and must be valid.
       Each container must have unique. Take this value: date +%s
3  properties:
4          architecture: "aarch64"
5          description: "Example of Gentoo virtual router"
6          os: "Gentoo Linux"
7          release: "0.1"
8          variant: "Custom"
```

Import rootfs and metadata as image

- Compress both image and metadata

```
tar cf gentoo-matadata.tar metadata.yaml
```

- Import these two into the container image

```
lxc image import gentoo-metadata.tar.gz gentoo-rootfs.tar.gz --
     alias GentooContainer
```

- Check if everything is ok

```
lxc image list
```

Prepare container profile

- Create YAML file to define the container profile

```
vim Gentoo-profile.yaml
```

```
config: {}
description: Gentoo LXD profile
devices:
  eth0:
    name: eth-wan
    nictype: macvlan
    parent: eth2
    type: nic
  eth1:
    name: eth-lan
    nictype: macvlan
    parent: eth1
    type: nic
  root:
    path: /
    pool: lxd-pool
    type: disk
name: default
```

Attach container profile

```
lxc profile create Gentoo-profile
```

```
lxc profile edit Gentoo-profile < Gentoo-profile.yaml
```

- To attach a profile to the container, first create the container from the previously imported image

```
lxc init GentooContainer Gentoo
```

- Now attach the profile

```
lxc profile apply Gentoo Gentoo-profile
```

Verifying the process

- Checking images

```
lxc image list
```

- Checking containers

```
lxc ls
```

- Checking available profiles

```
lxc profile list
```

- Checking and modifying a specific profile

```
lxc profile show Gentoo-profile
lxc profile edit Gentoo-profile
```

Starting the container

```
lxc start Gentoo
```

- What now?
  - Execute any command in the container
  - Access shell (for attaching into the container)

```
lxc exec Gentoo -- /bin/bash
```

  - From this shell we can do everything as regular Linux users
  - Any other program can be run in the same way

```
lxc exec Gentoo - /bin/ping 8.8.8.8 -c2
```

Setting up router functionality inside the container

- What is a router?
  - Forwards data packets between computer networks
  - One network is on LAN other is on WAN side – two different data lines
  - Basic NAT

Configuring the network

- Container profile defines 2 interfaces

```
1   eth0:
2     name: eth-wan
3     nictype: macvlan
4     parent: eth2
5     type: nic
6   eth1:
7     name: eth-lan
8     nictype: macvlan
9     parent: tap1
10    type: nic
```

- Checking if they exist in the container

```
ip link
```

- They exist but are not configured

- Who takes care of the network?
  - `systemd` or more precisely `systemd-networkd`
- Each interface will require different behavior
  - LAN should act as DHCP server
  - WAN should act as DHCP client
- Configuring is done in `/etc/systemd/network` which is empty by default

- Create two interface files used by `systemd` to handle interfaces

  - `10-WAN.network`

```
[Match]
Name=eth-wan

[Network]
DHCP=ipv4

[DHCP]
RouteMetric=10
```

  - `11-LAN.network`

```
[Match]
Name=eth-lan

[Network]
Address=192.168.2.1
DHCPServer=yes

[DHCPServer]
DNS=8.8.8.8
EmitDNS=yes
```

- Restart `systemd-networkd`

```
systemctl restart systemd-networkd
```

- Check that everything works with `ip link`

- Plug cable in eth2 and check that IP is offered on the `eth-wan` in the container
- Plug cable in eth0 and other end into PC. Check if PC is offered an IP from the board.
- Test
  - From PC ping `192.168.2.1`
  - From container ping `192.168.2.X` (PC)
  - From container ping some external ip (`8.8.8.8`)
  - From PC ping some external ip (`8.8.8.8`)

Configure NAT

- NAT – Network Adress Translation
- Configured with iptables
  - Program used for configuring Linux kernel firewall
  - Consists out of tables and chains
  - Network packets flow from one table/chain to another
  - Manipulating packet flows allows setting different firewall rules

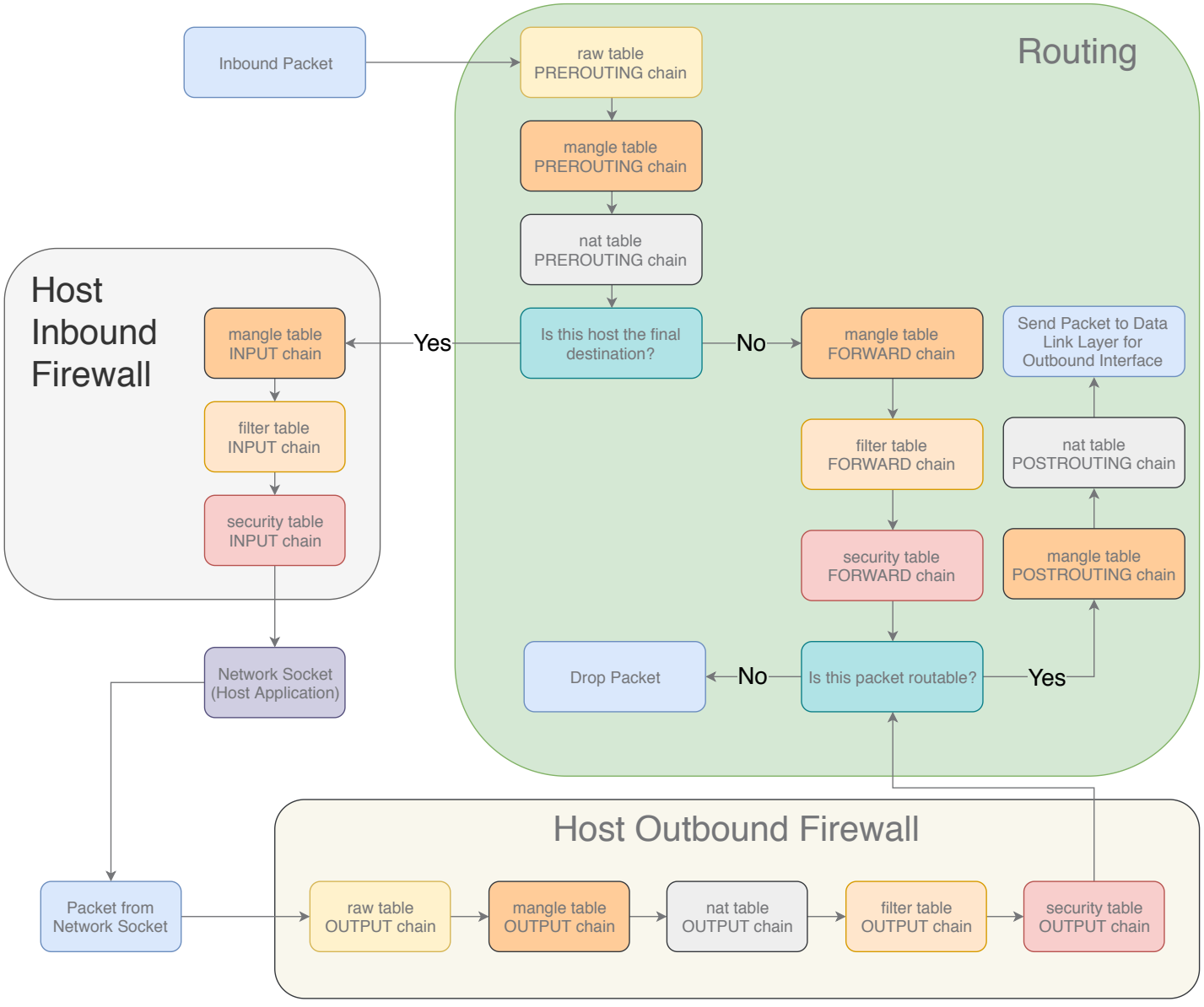


FIGURE 1 How iptables work

- The idea is to forward packets from LAN to WAN and vice-versa
- First, allow the kernel to forward packets

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

- A basic setup requires 3 rules
  - 1 in `nat` table, 2 in `filter` table
- These rules allow packets from internal network to be forwarded to external (LAN → WAN)

```
iptables -t nat -A POSTROUTING -o eth-wan -j MASQUERADE

iptables -A FORWARD -i eth-wan o eth-lan -m state --state RELATED
    ,ESTABLISHED -j ACCEPT

iptables -A FORWARD -i eth-lan -o eth-wan -j ACCEPT

echo 1 > /proc/sys/net/ipv4/ip_forward
```

- Check that all is working by pinging an external IP from PC

Summary

- Run Gentoo on ARM board
- Create and run container
- Configure container as router
- Access external network through container from a machine in local network
- Application of containers?

Gentoo Linux on ARM platforms

davor.popovic@sartura.hr · **mak.krnic@sartura.hr**

info@sartura.hr · www.sartura.hr