
Netmap: a framework for fast packet I/O

Introduction

Netmap is a framework for very fast packet I/O from userspace. VALE is an equally fast in-kernel L2 software switch using the netmap API. Both are implemented as a single kernel module for FreeBSD and Linux. Netmap/VALE can handle tens of millions of packets per second, matching the speed of 10G and 40G ports even with minimum sized frames.

To learn about netmap, you can use the following resources:

- the man pages (<https://www.freebsd.org/cgi/man.cgi?query=netmap&sektion=4> or [share/man/man4/netmap.4](#) in this repository)
- the papers.
- the tutorials, available at <https://github.com/netmap-unipi/netmap-tutorial>

This repository contains source code (BSD-Copyright) for FreeBSD, Linux and Windows. Netmap, VALE and related applications are already included in FreeBSD since version 10.x. FreeBSD users should use the code included in the FreeBSD src tree rather than the one in this repository, although the two codebases are mostly aligned.

Why should I use netmap?

Netmap is mostly useful for userspace applications that must deal with raw packets: traffic generators, sinks, monitors, loggers, software switches and routers, generic middleboxes, interconnection of virtual machines.

The `apps/` directory includes `pkt-gen.c` (a fast traffic generator/receiver) and `bridge.c`, a simple bidirectional interconnect between two ports. The kernel module itself implements a learning ethernet bridge.

More resources are hosted on other repositories. For example <https://github.com/luigirizzo/netmap-libpcap> contains a netmap-enabled version of libpcap (which is also included in FreeBSD distribution) so you can run any libpcap client on top of netmap at much higher speeds than using bpf. The <https://github.com/luigirizzo/netmap-ipfw> repository contains a userspace version of ipfw and dumynet which can handle several million packets per second in a single thread

QEMU has native netmap support, so it can interconnect VMs at high speed through netmap ports (e.g., using VALE ports or netmap pipes). For maximum performance, it is also possible to pass-through any netmap port into a QEMU VM, as described [here](#). Also the FreeBSD bhyve hypervisor has native support for netmap.

Netmap alone **does not** accelerate your TCP. For that you need to implement your own tcp/ip stack probably using some of the techniques indicated below to reduce the processing costs.

Architecture

netmap uses a number of techniques to establish a fast and efficient path between applications and the network. In order of importance:

- I/O batching
- efficient device drivers
- pre-allocated tx/rx buffers
- memory mapped buffers

Despite the name, memory mapping is NOT the key feature for netmap's speed; systems that do not apply all these techniques do not achieve the same speed *and* efficiency.

Netmap clients use a select()-able file descriptor to synchronize with the network card/software switch, and exchange multiple packets per system call through device-independent memory mapped buffers and descriptors. Device drivers are completely in the kernel, and the system does not rely on IOMMU or other special mechanisms.

Installation instructions

A single kernel module implements the core Netmap functions, including the VALE switch and access to physical NICS using unmodified device drivers (at the price of much lower performance than netmap-aware drivers).

Netmap-aware device drivers are needed to use netmap at high speed on ethernet ports. To date, we have support for Intel ixgbe (10G), ixl (10/40G), e1000/e1000e/igb (1G), Realtek 8169 (1G) and Nvidia (1G). FreeBSD has also native netmap support in the Chelsio 10/40G cards.

FreeBSD

FreeBSD already includes netmap kernel support by default since version 11. If your kernel configuration does not include netmap, you can enable it by adding a `dev netmap` line, and rebuilding the kernel. Alternatively, you can build standalone modules (netmap, ixgbe, em, lem, re, igb, ...).

FreeBSD users will find the netmap example applications in `src/tools/tools/netmap/` within the FreeBSD src tree.

Linux

The `./configure` && `make` build system in the `LINUX/` directory will let you patch device driver sources and build some netmap-enabled device drivers. Please look here for more instructions.

Make sure you have kernel headers matching your installed kernel. The sources for `e1000e`, `igb`, `ixgbe` and `i40e` will be downloaded from the Intel e1000 project on sourceforce. If you need the netmap enabled drivers for `e1000`, `veth`, `forcedeth`, `virtio-net` or `r8169` you will also need the full kernel sources.

Linux users can find the netmap example applications in the `apps/` directory in this repository.

Step 1 Configure netmap. To compile Netmap/VALE and the Intel drivers above:

```
1 ./configure
```

(This will also download the Intel driver sources from sourceforce). To compile only Netmap/VALE (using unmodified drivers):

```
1 ./configure --no-drivers # only netmap, no unmodified drivers
```

If you need the full kernel sources and you have installed them in `/a/b/c/linux-A.B.C/`, then you should do

```
1 ./configure --kernel-dir=/a/b/c/linux-A.B.C/ # netmap+device drivers
```

You can omit `--kernel-dir` if your kernel sources are in a standard place.

If you use distribution packages, full sources and headers may be in different places contain headers (e.g., on debian systems). Use

```
1 ./configure --kernel-sources=/a/b/c/linux-sources-A.B/ --kernel-dir=/a/
  b/c/linux-headers-A.B/
```

Step 2 Build kernel modules and sample applications:

```
1 make
```

Step 3 Install the new modules and the applications:

```
1 sudo make install
```

To have the new netmap-enabled driver modules alongside the original ones, you may want to add `--driver-suffix=-netmap` to the configure command above. The new drivers will then be called `e1000e-netmap`, `ixgbe-netmap`, and so on.

Windows

Netmap has been ported to Windows in summer 2015 by Alessio Faina as part of his Master thesis. You may take a look [here](#) for details, but please be aware that the port has been left behind for years, and is currently unmaintained.

Applications

The directory `apps/` contains some programs that use the netmap API

- `pkt-gen.c` a packet generator/receiver working at line rate at 10Gbit/s
- `vale-ctl.c` utility to configure ports of a VALE switch
- `bridge.c` a utility that bridges two interfaces or one interface with the host stack

For libpcap and other applications look at the `extra/` directory.

Testing

`pkt-gen` is a generic test program which can act as a sender or receiver. It has a large number of options, but the simplest form is:

```
1 pkt-gen -i ix0 -f rx      # receive and print stats
2 pkt-gen -i ix0 -f tx -l 60 # send a stream of 60-byte packets
```

(replace `ix0` with the name of the interface or VALE port). This should be able to work at line rate (up to 14.88 Mpps on 10 Gbit/interfaces, even higher on VALE) but note the following

Operating Speed

Netmap is able to send packets at very high rates, and for simple packet transmission and reception, speed generally not limited by the CPU but by other factors (link speed, bus or NIC hw limitations).

For a physical link, the maximum number of packets per second can be computed with the formula:

```
1 pps = line_rate / (672 + 8 * pkt_size)
```

where “`line_rate`” is the nominal link rate (e.g 10 Gbit/s) and `pkt_size` is the actual packet size including MAC headers and CRC. The following table summarizes some results (in Mpps)

```
1          LINE RATE
2 pkt_size  100M   1G  10G 40G
3
```

4	64	.1488	1.488	14.88	59.52
5	128	.0589	0.589	5.89	23.58
6	256	.0367	0.367	3.67	14.70
7	512	.0209	0.209	2.09	8.38
8	1024	.0113	0.113	1.13	4.51
9	1518	.0078	0.078	0.78	3.12

On VALE ports, there is no physical link and the throughput is limited by CPU or memory depending on the packet size.

Common problems

Before reporting slow send or receive speed on a physical interface, check ALL of the following:

Cannot set the device in netmap mode:

- make sure that the netmap module and drivers are correctly loaded and can allocate all the memory they need (check into `/var/log/messages` or equivalent)
- check permissions on `/dev/netmap`
- make sure the interface is up before invoking `pkt-gen`

Sender does not transmit

- some switches/interfaces take a long time to (re)negotiate the link after starting `pkt-gen`; in case, use the `-w N` option to increase the initial delay to N seconds;

This may cause inability to transmit, or lost packets for the first few seconds of transmission

Receiver does not receive

- make sure traffic uses a broadcast MAC addresses, or the UNICAST address of the receiving interface, or the receiving interface is in promiscuous mode (this must be done with `ifconfig`; `pkt-gen` does not change the operating mode)

Lower speed than line rate

- check that your CPUs are running at the maximum clock rate and are not throttled down by the governor/powerd. On Linux:
`lscpu # shows current cpu speed`
`sudo apt-get install cpufrequtils`

-
- make sure that the sender/receiver interfaces and switch have flow control (FC) disabled (either via `sysctl` or `ethtool`). If FC is enabled and the receiving end is unable to cope with the traffic, the driver will try to slow down transmission, sometimes to very low rates.
 - a lot of hardware is not able to sustain line rate. For instance, `ixgbe` has problems with receiving frames that are not multiple of 64 bytes (with/without CRC depending on the driver); also on transmissions, `ixgbe` tops at about 12.5 Mpps unless the driver prefetches tx descriptors. `igb` does line rate in all configurations. `e1000/e1000e` vary between 1.15 and 1.32 Mpps. `re/r8169` is extremely slow in sending (max 4-500 Kpps)

Host rings do not work

- disable NIC offloads, because `netmap` does not support them and packets exchanged between `netmap` and the kernel stack can be dropped because of invalid checksums. On FreeBSD offloads can be disabled with a command like

```
sudo ifconfig vtnet0 -txcsum -rxcsu6 -tso4 -tso6 -lro -txcsum6 -rxcsu6
```

Check here for the corresponding Linux command.

Credits

`Netmap` and `VALE` are projects of the Università di Pisa, partially supported by various entities including: Intel Research Berkeley, EU FP7 projects `CHANGE` and `OPENLAB`, `Netapp/Silicon Valley Community Foundation`, `ICSI`

Authors: * Luigi Rizzo

Contributors (<https://github.com/netmap-unipi/netmap/graphs/contributors>):

- Giuseppe Lettieri
- Michio Honda
- Marta Carbone
- Gaetano Catalli
- Matteo Landi
- Vincenzo Maffione
- Stefano Garzarella
- Alessio Faina

References

There are a few academic papers describing netmap, VALE and applications. You can find the papers at <http://info.iet.unipi.it/~luigi/research.html>

- Luigi Rizzo, netmap: a novel framework for fast packet I/O, Usenix ATC'12, Boston, June 2012
- Luigi Rizzo, Revisiting network I/O APIs: the netmap framework, Communications of the ACM 55 (3), 45-51, March 2012
- Luigi Rizzo, Marta Carbone, Gaetano Catalli, Transparent acceleration of software packet forwarding using netmap, IEEE Infocom 2012, Orlando, March 2012
- Luigi Rizzo, Giuseppe Lettieri, VALE: a switched ethernet for virtual machines, ACM Conext 2012, Nice, Dec. 2012
- Luigi Rizzo, Giuseppe Lettieri, Vincenzo Maffione, Speeding up packet I/O in virtual machines, IEEE/ACM ANCS 2013, San Jose, Oct. 2013
- Stefano Garzarella, Giuseppe Lettieri, Luigi Rizzo, Virtual device passthrough for high speed VM networking IEEE/ACM ANCS 2015, Oakland, May 2015
- Vincenzo Maffione, Luigi Rizzo, Giuseppe Lettieri, Flexible virtual machine networking using netmap passthrough IEEE Lanman 2016, Rome, June 2016