
[build](#) [unknown](#) [build](#) [unknown](#) [gitter](#) [developer chat](#) [build](#) [unknown](#) # Network Function Framework for Go (former YANFF)

Wonderful news : we are now supporting AF_XDP and supporting(almost) getting packets directly from Linux. So you do not need to write 3(three) different applications to process packets coming from different type of drivers of PMDs. You just write everything in NFF-Go, and it can dynamically use whatever you would like underneath. Contact us if you need help.

What it is

NFF-Go is a set of libraries for creating and deploying cloud-native Network Functions (NFs). It simplifies the creation of network functions without sacrificing performance. * Higher level abstractions than DPDK. Using DPDK as a fast I/O engine for performance * Go language: safety, productivity, performance, concurrency * Network functions are application programs not virtual machines * Built-in scheduler to auto-scale processing based on input traffic. Both up and down.

Benefits:

- Easily leverage Intel hardware capabilities: multi-cores, AES-NI, CAT, QAT, DPDK
- 10x reduction in lines of code
- No need to be an expert network programmer to develop performant network function
- Similar performance with C/DPDK per box
- No need to worry on elasticity - done automatically
- Take advantage of cloud native deployment: continuous delivery, micro-services, containers

Feel the difference

Simple ACL based firewall

```
1
2 func main() {
3     // Initialize NFF-GO library to use 8 cores max.
4     config := flow.Config{
5         CPUCoresNumber: 8,
6     }
7     flow.CheckFatal(flow.SystemInit(&config))
8
9     // Get filtering rules from access control file.
10    L3Rules, err := packet.GetL3ACLFromTextTable("Firewall.conf")
11    flow.CheckFatal(err)
12}
```

```

13 // Receive packets from zero port. Receive queue will be added
    automatically.
14 inputFlow, err := flow.SetReceiver(uint8(0))
15 flow.CheckFatal(err)
16
17 // Separate packet flow based on ACL.
18 rejectFlow, err := flow.SetSeparator(inputFlow, L3Separator, nil)
19 flow.CheckFatal(err)
20
21 // Drop rejected packets.
22 flow.CheckFatal(flow.SetStopper(rejectFlow))
23
24 // Send accepted packets to first port. Send queue will be added
    automatically.
25 flow.CheckFatal(flow.SetSender(inputFlow, uint8(1)))
26
27 // Begin to process packets.
28 flow.CheckFatal(flow.SystemStart())
29 }
30
31 // User defined function for separating packets
32 func L3Separator(currentPacket *packet.Packet, context flow.UserContext
    ) bool {
33     currentPacket.ParseL4()
34     // Return whether packet is accepted or not. Based on ACL rules.
35     return currentPacket.L3ACLPermit(L3Rules)
36 }

```

NFF-GO is an Open Source BSD licensed project that runs mostly in Linux user land. The most recent patches and enhancements provided by the community are available in the develop branch. master branch provides the latest stable released version under the appropriate tag.

Getting NFF-GO

Starting with release 0.7.0 NFF-Go uses go.mod for getting dependencies, therefore Go version 1.11 or later is required. To checkout NFF-Go sources use the following command

```
1 git clone --recurse-submodules http://github.com/intel-go/nff-go
```

Setting up the build and run environment

DPDK

NFF-GO uses DPDK, so you must setup your system to build and run DPDK. See System Requirements in the DPDK Getting Started Guide for Linux for more information.

By default NFF-Go is build with Mellanox cards support out of the box you need to install additional dependencies required for MLX network drivers. On Ubuntu they are called `libmnl-dev` and `libibverbs-dev`. For more details see MLX drivers respective pages for MLX4 and MLX5. If these dependencies cannot be satisfied, and Mellanox drivers are not needed, you can set variable `NFF_GO_NO_MLX_DRIVERS` to some unempty value to disable MLX drivers compilation.

Additional dependencies are required for pktgen, especially if you are running RedHat or CentOS Linux distributions. See this file for details. LUA section for RedHat and CentOS is in its end.

After building a DPDK driver with the make command, you must register network cards to work with the DPDK driver, load necessary kernel modules, and bind cards to the modules. See Compiling the DPDK Target from Source and How to get best performance with NICs on Intel platforms in the DPDK Getting Started Guide for Linux for more information.

The kernel module, which is required for DPDK user-mode drivers, is built but not installed into kernel directory. You can load it using the full path to the module file: `nff-go/test/dpdk/dpdk/x86_64-native-linuxapp-gcc/kmod/igb_uio.ko`

Go

Use Go version 1.11.4 or higher. To check the version of Go, do:

```
1 go version
```

AF_XDP support

AF_XDP support is enabled by default, and it requires you to install `libbpf` package. At the time of writing Ubuntu doesn't have this library among its packages, so it is necessary to build `libbpf` from sources or disable AF_XDP socket support.

To disable it set variable `NFF_GO_NO_BPF_SUPPORT` to some unempty value. When NFF_GO is built with it, AF_XDP support is disaled and using it results in errors.

If you want to build `libbpf` from sources you can do it in two different ways. * If you are using stock Linux kernel from distribution, download `libbpf` from GitHub, then execute `cd src; make; sudo make install`. Add `/usr/lib64` to your ldconfig path. * If you build Linux kernel from sources, you can build `libbpf` from Linux source tree using commands `cd tools/lib/bpf; make; sudo make install install_headers`. Add `/usr/local/lib64` to your ldconfig path.

Building NFF-GO

When Go compiler runs for the first time it downloads all dependent packages listed in `go.mod` file. This operation cannot be done in parallel because otherwise Go package cache gets corrupted. Because of that it is necessary to run command `go mod download` before first `make` is done. Another option is to use single process `make -j1` when it is run for the first time, but may be quite slow.

```
1    cd nff-go
2    go mod download      # do it once before first build
3    make -j8
```

Building NFF-GO in debug mode

```
1    make debug -j8
```

Running NFF-GO

Documentation

Online API documentation is available on godoc.org site. API usage is explained on our Wiki pages.

Tests

Invoking `make` in the top-level directory builds the testing framework and examples. NFF-GO distributed tests are packaged inside of Docker container images. There are also single node unit tests in some packages that you can run using the command:

```
1    make testing
```

Docker images

To create Docker images on the local default target (either the default UNIX socket in `/var/run/docker.sock` or whatever is defined in the `DOCKER_HOST` variable), use the **make images** command.

To deploy Docker images for use in distributed testing, use the **make deploy** command. This command requires two environment variables:

-
- NFF_GO_HOSTS="hostname1 hostname2 ... hostnameN"* - a list of all hostnames for deployed test Docker images
 - DOCKER_PORT=2375* - the port number to connect to Docker daemons running on hosts in the NFF_GO_HOSTS variable

To delete generated images in the default Docker target, use the **make clean-images** command.

Running tests

After the Docker images are deployed on all test hosts, you can run distributed network tests. The test framework is located in the test/main directory and accepts a JSON file with a test specification. There are predefined configs for performance and stability tests in the same directory. To run these tests, change **hostname1** and **hostname2** to the hosts from the NFF_GO_HOSTS list in these JSON files.

Cleaning-up

To clean all generated binaries, use the **make clean** command. To delete all deployed images listed in NFF_GO_HOSTS, use the **make cleanall** command.

Contributing

If you want to contribute to NFF-Go, check our Contributing guide. We also recommend checking the bugs with 'help-wanted' or 'easyfix' in our list of open issues; these bugs can be solved without an extensive knowledge of NFF-Go. We would love to help you start contributing.

You can reach the NFF-Go development team via our mailing list.