NFF-GO: NOVEL APPROACH TO NETWORK FUNCTION DEVELOPMENT

Maria Bulatova, Daria Kolistratova

Background

Network Function (NF) – a component of a network infrastructure with well defined interfaces and behavior (routing, network address translation (NAT), firewall, etc.).

Traditional NFs:

- expensive
- not flexible
- not scalable.

Background

Solution: Network Function Virtualization (NFV) technology.
NFV involves implementing network functions in software that can run on industry standard server hardware.

VNFs:

- cheap
- can be moved to various locations in the network
- behavior can be changed easily
- can run in parallel.

Problem

There are few instruments for NFV development, but no one provides at once:

- rapid and simple development
- easy learning
- fast prototyping
- not sufficient overhead
- scalability

Required easy-to-learn performant **framework** for NFV development

Solution: NFF-Go!

NFF-GO

DPDK based

- DPDK stands for Data Plane Development Kit
- DPDK is set of highly optimized libraries and drivers to accelerate packet processing
- DPDK uses kernel bypass

- GO language based
- Open Source
- Framework
- For Network Function Development
- By smart chaining of customized, highly optimized, predefined blocks

Current status

- 6 releases
- 488 stars at GitHub
- “Pathfinding project with product quality”
- Has LPM, NAT, IPSec, anti DDoS, L3 reassemble, KNI support, protocols: ARP, VLAN, ICMP, UDP, etc.

DPDK usage

To achieve high-performance we are using DPDK.

- DPDK is a C library
- CGO calls are expensive
  - DPDK functions only for low level Receive, Send
- store packets in C memory
- use pointers to packets without direct calls to C from GO.

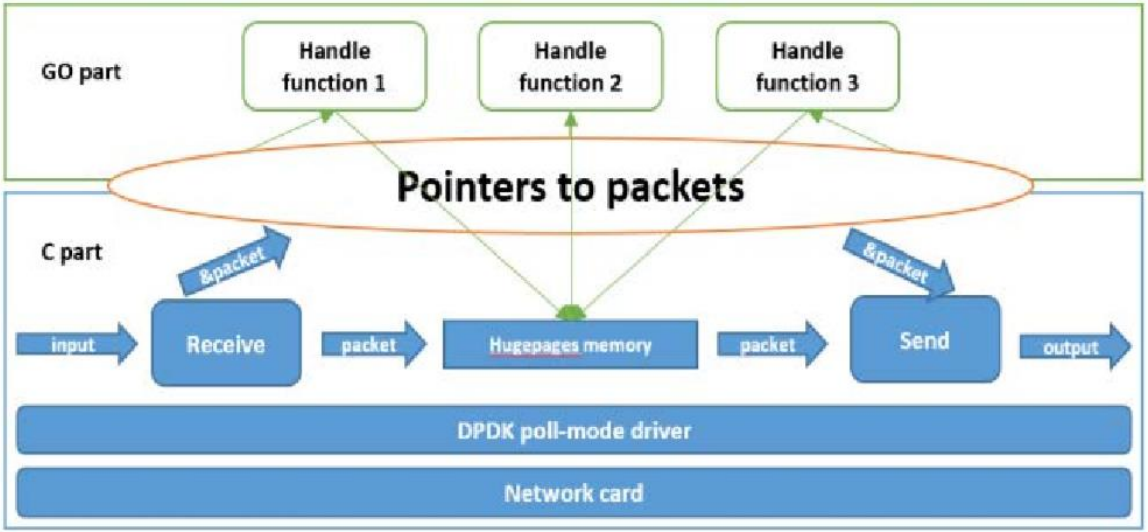


Figure 3. C and GO interaction for packet handling.

Flow Functions

There is an abstraction – flow function (FF).

- Each FF is a goroutine pinned to thread by go runtime
- FFs are cloned to idle cores to achieve given speed
- FFs are chained through lockless rings

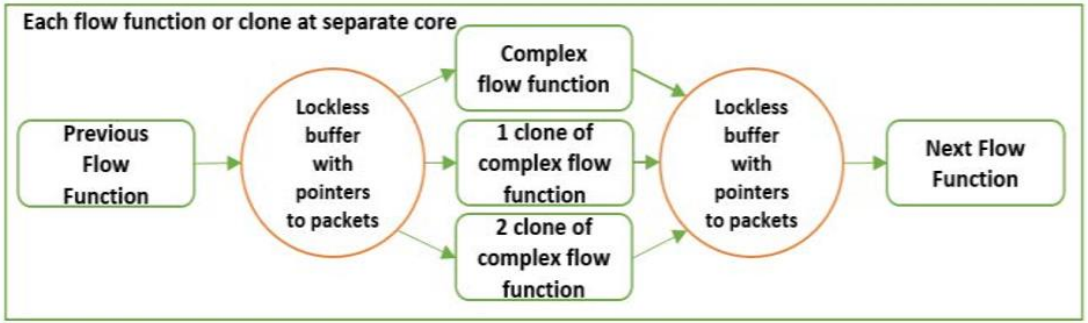


Figure 1. Cloning of Flow Functions

Packet Processing graph

Is built from FFs.
Five predefined FFs:

- receive
- send
- stop
- merge
- partition
- copy

The developer can configure their parameters but can't change the functionality.

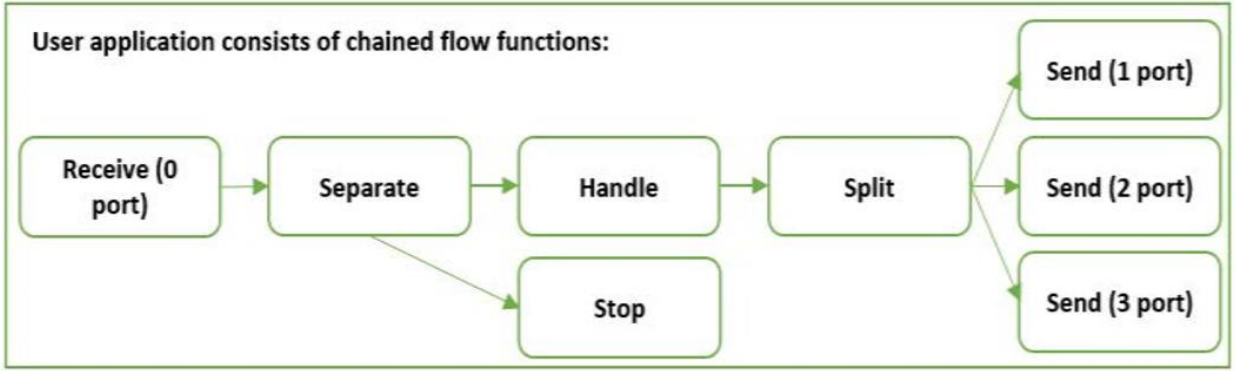


Figure 2. Blocks in a packet processing graph of user application.

Packet Processing graph

Four user-defined flow functions (and their vector versions):

- handle
- handleDrop
- separate
- split
- generate

They get user-defined function as a parameter, acting as a flow function.

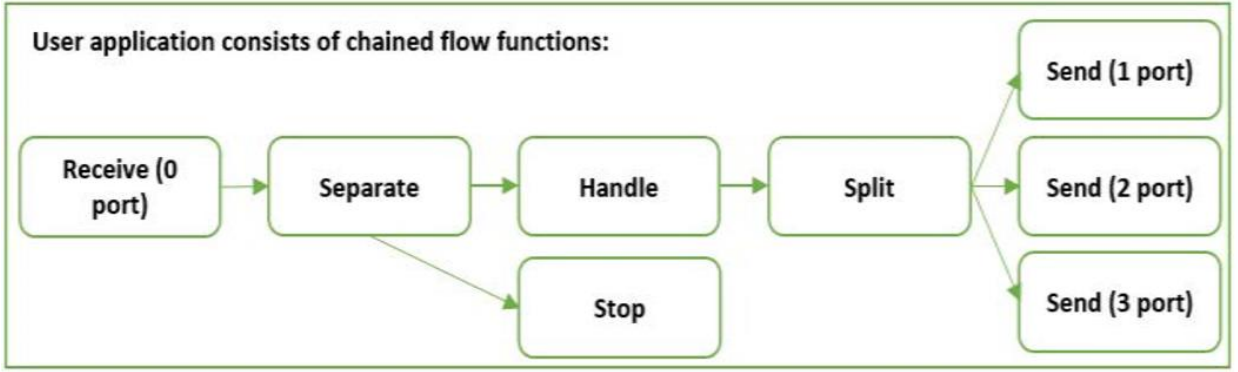
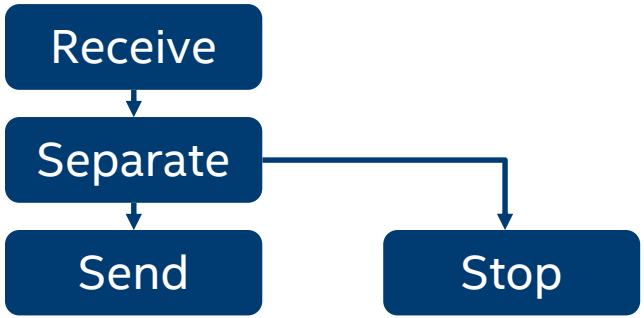


Figure 2. Blocks in a packet processing graph of user application.

L3 simple firewall example

Config file example:

#	Source addr,	Destination addr,	L4 protocol ID,	Src port,	Dsr port,	Decision
	10.10.0.5/24	ANY	TCP	46	ANY	Accept
	111.2.0.4/32	ANY	TCP	49:122	ANY	Accept
	ANY	21.23.45.10/32	UDP	ANY	ANY	Accept
	ANY	ANY	UDP	ANY	4080	Accept



The same app on DPDK is ~ 1500 lines!

```go
package main

import (
    "github.com/intel-go/nff-go/flow"
    "github.com/intel-go/nff-go/packet"
)

var l3Rules *packet.L3Rules

func main() {
    config := flow.Config{}
    flow.SystemInit(&config)
    l3Rules, _ = packet.GetL3ACLFromORIG("firewall.conf")
    inputFlow, _ := flow.SetReceiver(uint16(0))
    rejectFlow, _ := flow.SetSeparator(inputFlow,
        l3Separator, nil)
    flow.SetStopper(rejectFlow)
    flow.SetSender(inputFlow, uint16(1))
    flow.SystemStart()
}

func l3Separator(currentPacket *packet.Packet,
    context flow.UserContext) bool {
    return currentPacket.L3ACLPermit(l3Rules)
}
```

How to start

Join and star us on GitHub https://github.com/intel-go/nff-go

Read a developers guide https://github.com/intel-go/nff-go/wiki/Developers-Guide

View a tutorial https://github.com/intel-go/nff-go/blob/master/examples/tutorial/YANFF%20tutorial.pdf

And start coding!

If you have any question, feel free to open issues on GitHub.

References

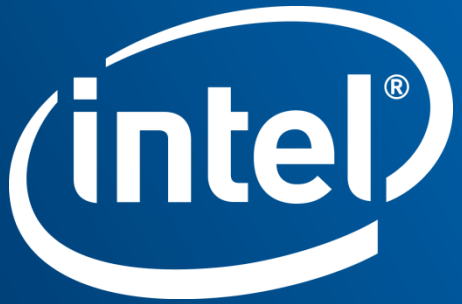
NFF-GO on GitHub: https://github.com/intel-go/nff-go

DPDK: https://www.dpdk.org/

An article about NFF-GO: https://doi.org/10.1145/3166094.3166111

Ilya Philippov and Areg Melik-Adamyan. 2017. Novel approach to network function development. In Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR '17). ACM, New York, NY, USA, Article 17, 6 pages.

About NFV: https://www.etsi.org


intel®

BACKUP SLIDES

What is NFF-Go

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.

NFF-Go 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

Implementation details

- FFs are chained via lock-free ring buffers.
- Clone when buffer is full.
- Copy free – buffers transfer only pointers.
- FF is a separate goroutine and is bind to exact core.

Go Garbage collector

- GO language has safe memory release by GC
- Real time library based on language with GC? Really?
  - Yes, it is not a framework for mission critical latency-sensitive tasks
  - Ok for other tasks
- How?
  - GO GC has comparatively small pauses ~1ms
  - Packets are in C (DPDK allocated memory) – no garbage
  - GC can stop everything! Except receives! – They are in C
  - Packet buffers are enough for stop-the-world for 3ms