Functional Network Programming

Nicola Bonelli

NetGroup@Unipi

nicola@pfq.io

Agenda

- Network programming
- A functional approach
- Enif-lang
- Enif-lang overview
- Enif-lang examples

Beba

BEhavioural BAsed forwarding

Network Programming Context

- Commodity hardware running open-source software
  - Linux OS
- 10/40/100 G NICs with multiple queues
- Modern multi-core architectures
  - Multi-core CPU, NUMA and multiple packages, cluster of servers
- Why multi-core is so important?
  - Non-trivial applications can spend lots of per-packet cpu-cycles
    - Most of clocks are spent in memory I/O
  - Stateful processing causes continuous cache invalidation
- Is legacy software able to exploit multi-core architectures, fast links and all?

The recipe for high-performance network applications

- Fast packet I/O? (DPDK, PF_RING, Netmap, PFQ…)
- Concurrency? Most applications are still a single process
- Lock-less programming
  - lock-free/wait-free algorithm (no mutexes or spinlocks)
  - Amortized atomic operations (CAS etc.) -> copies are better than sharing
- Memory
  - Zero dynamic memory allocations (0-malloc)
  - Minimize true sharing, avoid false-sharing of cache-line
  - NUMA and memory-aware applications?
  - Memory models (C11, C++11-14) ? STM?
  - HugePages for TLB?
- Affinity
  - Interrupt affinity of multi-queues NICs and process/thread pinning...

High-Performance Network programming

- It’s challenging
  - Scale linearly with the number of cores/threads
  - Too much tricks and techniques
  - Programming at kernel level is hard to debug
  - Deep knowledge about the architectures
  - Hard to profile

- No automatisms are possible (at the present moment)
  - Compilers are not aware about parallel architectures
  - Memory models are too complex

We are far from making things easy for the network developer...

Functional Network Programming

- High level domain-specific language
  - Designed for building programmable middleware
    - Switches, network filters, traffic shapers, load balancers
    - Orchestrate NIDS, DPI analyzers etc.
  - Compact, expressive and easy to use (non-error prone)

- Functional programming principles
  - Strongly, statically typed (no undefined-behavior)
  - Functions and packets are first-class citizens
  - Pure (no mutation to packet is possible)
    - Copy-on-write
  - Concurrency
    - allow multiple applications to works on the same data source

Enif-lang (aka pfq-lang)

Enif-lang

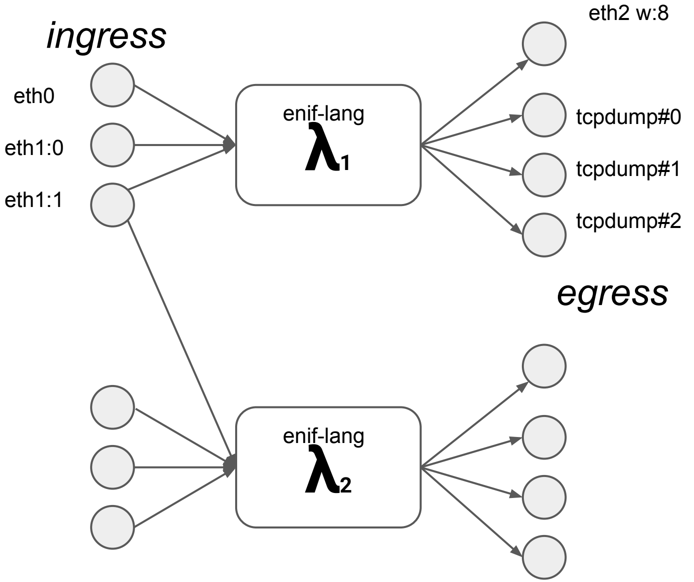
- Declarative language as eDSL (AST) (syntax borrowed from Haskell)
- Compositions of effectful functions (monads)

```
function :: Arg1 -> Arg2 -> … -> QBuff -> Action QBuff
```

- The Action context is responsible to:
  - Forwarding, filtering, steering, logging, sharing state across functions
  - Meters and counters
- Experimental compiler
  - Grammar parser
  - IR as JSON/binary
  - Backend to C/DPDK, or native P4 lang (future work)
- Runtime functional engine
  - Implementation exists within PFQ kernel module (BPF on steroid)

Enif-lang in a nutshell...

- Instantiate an engine
  - A master thread/process with PFQ API
  - Declared in a pcap config. file
  - By means of run-enif application
- Define the endpoints
  - Threads, processes or NICs…
    - Ingress-points bind to a group
    - Egress-points join a group to receive traffic
  - Applications via PFQ/pcap sockets
    - No modifications to applications are required
- Load an enif-lang program...
  - Dynamic update (hot-swappable)



Enif-lang:
functions overview

Enif-lang functions overview: 1/4

- Predicates :: Arg1 -> Arg2 -> ... -> Qbuff -> Bool (used in conditional expressions):
  - is_ip, is_udp, is_tcp, is_icmp, has_addr CIDR, is_rtp, is_rtcp, is_sip, is_gtp...

```
if (not is_tcp)                 when (has_addr “192.168.0.0/24”) do
     then pass                       forward “eth1”
     else drop
```

- Combinators :: (Qbuff -> Bool) -> (Qbuff -> Bool) -> (Qbuff Bool)
  - .||., .&&., .^^., not...

```
                when (is_tcp .||. is_udp)...
```

- Combinators (action):
  - `inv :: (Qbuff -> Action Qbuff) -> (Qbuff -> Action Qbuff)`   `e.g.  inv ip >-> kernel`
  - `par :: (Qbuff -> Action Qbuff) -> (Qbuff -> Action Qbuff)`

```
        -> (Qbuff -> Action Qbuff)                     e.g.  par tcp udp >-> kernel
```

Enif-lang functions overview: 2/4

- Properties :: Arg1 -> Arg2 -> ... -> Qbuff -> Word:
  - ip_tos, ip_tot_len, ip_id, ip_frag, ip_ttl, get_mark, get_state, tcp_source, tcp_dest...

```
filter (ip_ttl .< 5)              when (get_state .== 10) kernel
```

- Comparators:
  - `.<, .<=, .==, ./=, .>, .>=, <., <=., ==., /=., >., >=.`
  - `any_bit :: (Qbuff -> Word) -> Word -> Bool`
  - `all_bit :: (Qbuff -> Word) -> Word -> Bool`

```
if (any_bit tcp_flags (SYN.|.ACK))...
```

Enif-lang functions overview: 3/4

- Filters (effectful functions): {pass, drop}
  - ip, udp, tcp, icmp, rtp, rtcp, sip, voip, gtpv1, gtpv2, no_frag... :: (Arg..-> Qbuff -> Action Qbuff)
  - l3_proto, l4_proto :: Int16 -> Qbuff -> Action Qbuff

```
l3_proto 0x842 >-> log_msg "Wake-on-LAN!"
```

  - vlan_id_filter :: [Int] -> Qbuff -> Action Qbuff

```
flan_id_filt [1,2,3] >-> kernel
```

  - port, src_port, dst_port :: Int16 -> Qbuff -> Action Qbuff

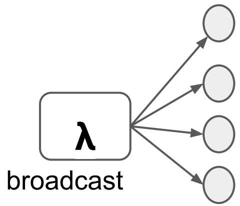
```
port 80 >-> log_msg "http packet!”
```

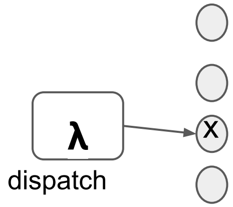
  - addr, src_addr, dst_addr :: CIDR -> Qbuff -> Action Qbuff

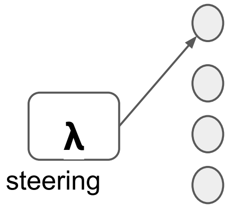
```
src_addr "192.168.0.0/24" >-> log_packet
```

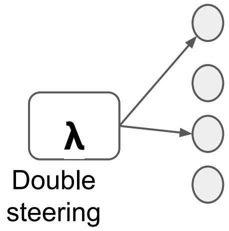
Enif-lang steering

- Provides a bunch effectful functions for steering
  - Fine-grained control of packet dispatching across end-points
- Steering the traffic
  - To fully exploit the hardware of multi-core architectures
  - To let core work independently from each other
    - Reduce or avoid the inter-core communications
    - Find a trade-off between correctness and performance
- Different kind of (weighed) steering
  - Pass (unit), Drop (filtering)
  - Broadcast (deterministic)
  - Dispatch CLASS (deterministic)
  - Steer HASH (randomized)
  - DoubleSteer Hash1 Hash2 (randomized)









Enif-lang functions overview: 4/4

- Steering :: Arg1 -> Arg2 … -> Qbuff -> Action Qbuff
  - broadcast, drop, kernel, forward (e.g. forward “eth1”)
  - steer_rrobin (useful only for stateless operations)
  - steer_link, steer_local_link (GW)
    - `steer_local_link "4c:60:de:86:55:46"`
  - double_steer_mac
  - steer_vlan
  - steer_p2p, double_steer_ip, steer_local_ip :: CIDR -> Qbuff -> Action Qbuff
    - `steer_local_ip "192.168.1.0/24"`
  - steer_flow, steer_field, steer_field_symm, double_steer_filed
  - etc..

Beba
BEhavioural BAsed forwarding

Enif-lang syntax

- do notation desugaring:

```
dns = port 53
main = do  dns                               main = dns >-> log_packet >-> forward “eth1”
           log_packet
           forward “eth1”
```

- shifting computations (tunnels)
  - Support for IPIP, IPv6IP, (IPGRE, MPLS? future work)

```
main = steer_flow -- doesn’t work            main = shift $ steer_flow

main = shift $ shift $ do
           addr “192.168.0.0/24”
           kernel
```

Enif-lang syntax

- High-order functions
  - Functions that can takes functions or effectful functions as argument
  - `conditional, when, unless, inv, par, filter` etc.
  - `if-then-else` is a built-in support borrowed from Haskell

- Context (high-order functions)
  - All functions that operates on src or dst fields can be restricted
  - `src, dst, local, remote` create restricted contexts:

```
src $ do
      addr “192.168.0.0/24”
      Kernel
```

```
local “192.168.0.1/24” $ do
      if (has_port 80)
            then steer_p2p
            else drop
```

Enif-lang examples

Stateless firewall

```
import Network.PFQ.Lang.Default

1ocal_networks   = [ “192.168.0.0”, “192.168.1.0” , “192.168.42.0” ]
unipi            = [ “113.114.52.0” ]
home             = [ “216.58.198.4” ]

http = port 80

engine = Engine { gid = 1, core = 0, policy = Restricted, input  = [ dev "eth0"], output = []}

main = do
     par4 has_port http
          bloom_filter 1024 local_networks 24
          bloom_filter 1024 unipi 22
          bloom_filter 128  home 32
     kernel
```

BEhavioural BAsed forwarding

Snort load balancer

```
import Network.PFQ.Lang.Default

is_dns = has_port 53
my_network = “192.168.0.0/24”

engine = Engine
  { gid    = 2
  , core   = 1
  , policy = Shared
  , input  = [ dev "eth0", dev "eth1.1" ]
  , output = [ dev "eth2" .^ 4]
  }

main = if is_dns
           then broadcast
           else steer_local_ip my_network
```

Questions?

www.pfq.io

- Nicola Bonelli, Andrea Di Pietro, Stefano Giordano, Gregorio Procissi **"Packet Capturing on Multicore Architectures"** [IEEE M&N]
- Nicola Bonelli, Andrea Di Pietro, Stefano Giordano, Gregorio Procissi **"On Multi-Gigabit Packet Capturing With Multi-Core Commodity Hardware"** [PAM]
- N. Bonelli, S. Giordano, G. Procissi and L. Abeni:**"A Purely Functional Approach to Packet Processing"** [ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)]
- N. Bonelli, S. Giordano, G. Procissi:**"Network Traffic Processing with PFQ"** [JSAC-SI-MT/IEEE]