Software Engineering Conference Russia
October 2017, St. Petersburg

New approach of network function creation

Ilia Filippov, Intel

Legal Information

- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.
- No computer system can be absolutely secure.
- Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.
- Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/performance.
- Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.
- Cost reduction scenarios described are intended as examples of how a given Intel- based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.
- Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.
- Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.
- Intel, the Intel logo and others are trademarks of Intel Corporation in the U.S. and/or other countries.
- *Other names and brands may be claimed as the property of others.
-

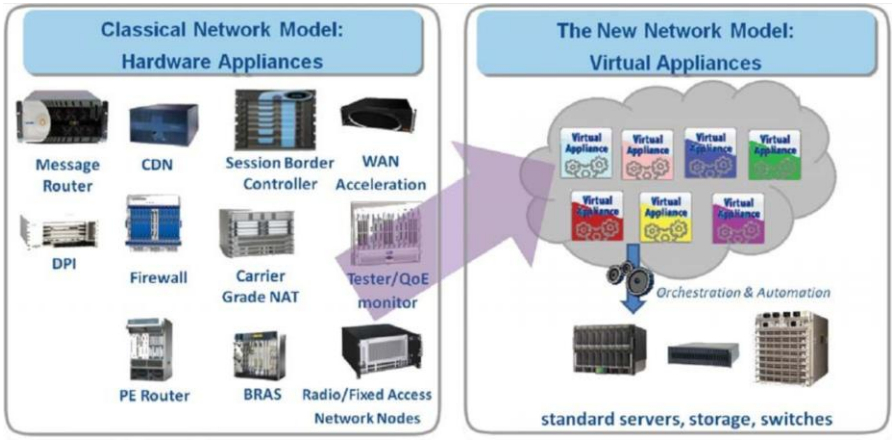
About me

- Ilia Filippov - architect and developer of YANFF
  - Senior Software Engineer at Intel Corporation
  - PhD student of Moscow Institute of Physics and Technology
  - Located in Austin TX, US

NFV – Network Function Virtualization

- Network handles lots of things except packet forwarding!

- Before: Middle-boxes
  - Expensive
  - Inflexible
  - Costly change
  - Non-scalable



- After: SW at commodity HW
  - Cheap
  - Flexible
  - Scalable

- NFV is orthogonal to SDN – Software Defined Network
- NFV deployment is orthogonal to NFV development

NFV development problems

- Changing of whole industry. No hardware devices, only software
  - Who will write functions?
  - How?
  - When?
  - How fast? How often?

- *Can you write a tiny function which will receive->send packets before the end of this presentation?*

YANFF - Yet Another Network Function Framework

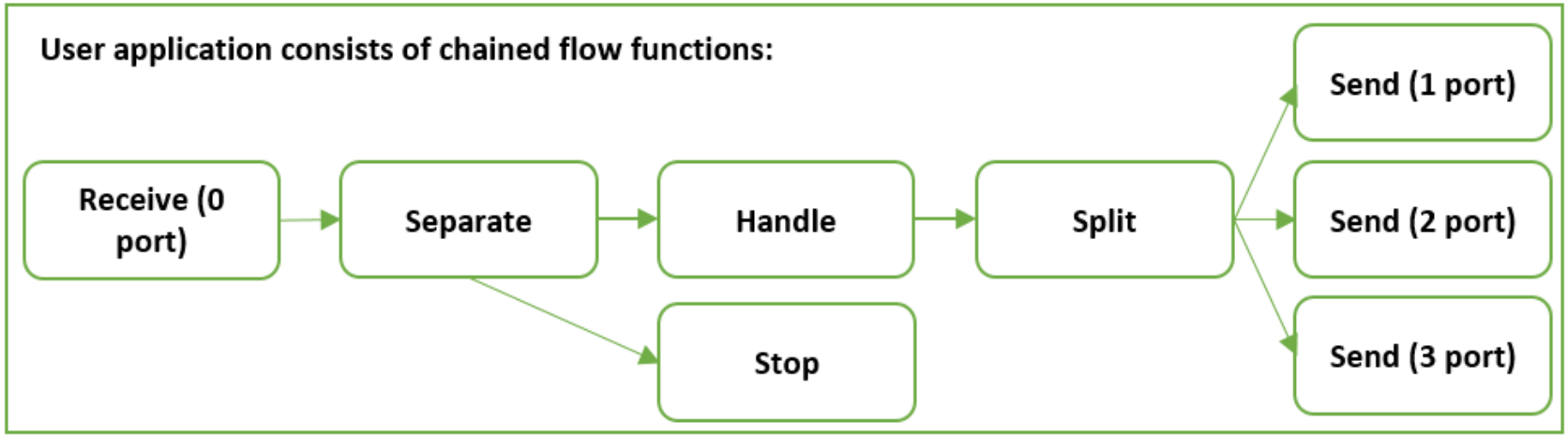
- Framework – high level abstractions
  - Helper:

  set of components for importing while writing network function
  - Execution semantics instead of declaration semantics:

  created network function is a program
- Open Source
  - https://github.com/intel-go/yanff/
- Concurrency, productivity, safety
  - GO language
- Performance
  - DPDK library as network base

Packet processing graph

- Functions which process packets – flow functions
- There is set of 9 flow functions
  - Receive-Send, Generate-Stop
  - Separate/Split/Partition-Merge, Handle
- Processing graph is built statically by chaining functions



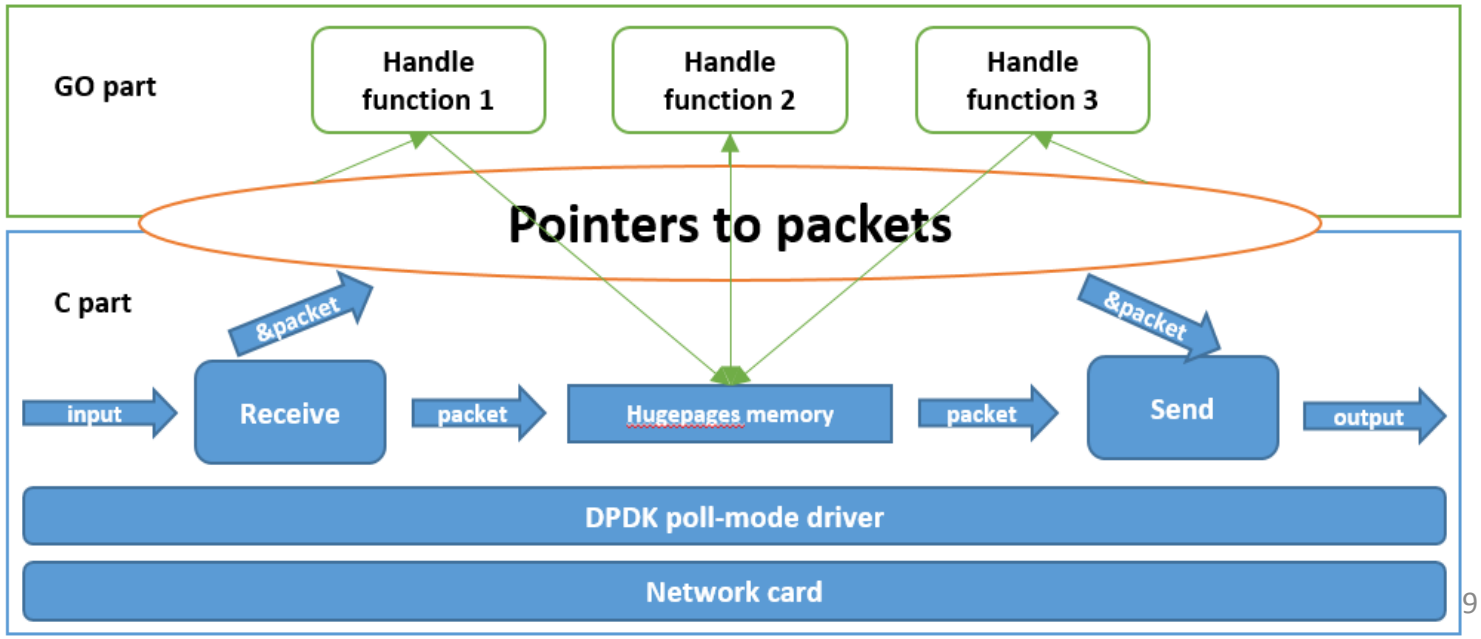
Why DPDK – Data Plane Development Kit

- State of the art in high-perf networking on commodity HW
- http://dpdk.org/

- User-mode drivers
  - No system calls, No context switches
  - No copying between kernel and user
  - More profits from DDIO technology
- Hugepages
  - No page swapping, less TLB misses
- Memory management
  - Preallocated set of constant memory spaces
  - Lockless memory buffers

Using of DPDK

- DPDK: low-level C library (+ user-mode network drivers)
- C calls from GO? CGO!
  - CGO functions calls are expensive
- DPDK functions only *for low level* and *at separate cores*:
  - Receive (allocate), Send/Stop (release)



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 tasks
  - Other tasks are doing well

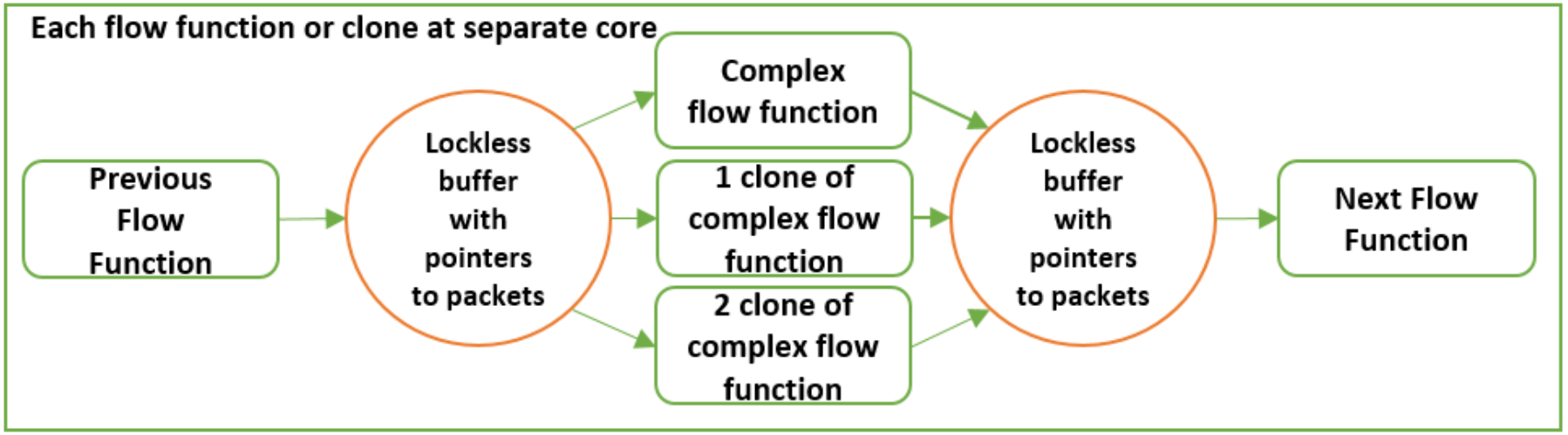
- How
  - GO GC has comparatively small pauses ~1ms
  - Packets are in C (DPDK allocated memory) – no garbage
  - GC can stop everything! Except receives! – They are in C
  - Packet buffers are enough for stop-the-world for 3ms

Customization

- Four flow functions are customizable:
  - Handle, Generate, Separate/Split
  - User function as parameter
- Low-level optimization BUT high-level customization
  - No user involving in prefetches, concurrency, etc.
  - User function gets each packet or vector of packets for SIMD
- Each function can be cloned or stopped
  - According to the strength of packet flow
  - And uses separate core

Deep in details

- Flow functions are chained via lockless ring buffers



- Copy free paradigm
  - Ring buffers transfer only pointers to packets
- Flow function is a goroutine and is bind to exact core

Additional helpers

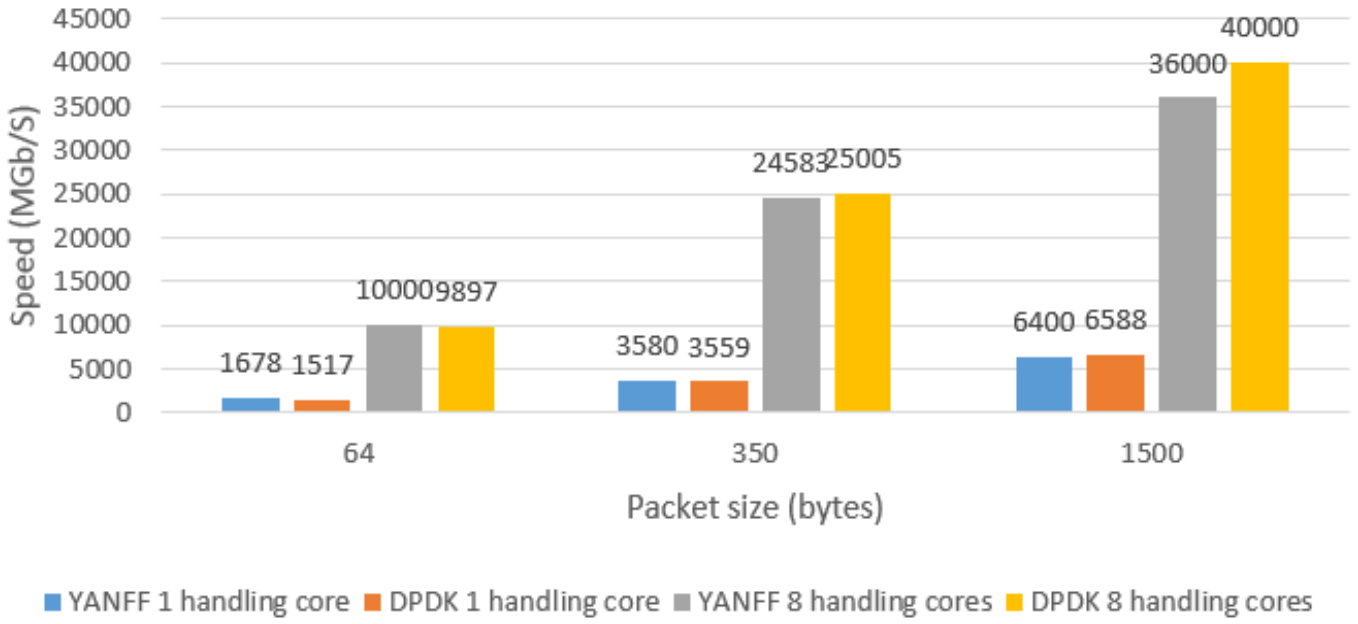
- Packet parsing
  - Not raw bytes, but high-level structure with protocols levels
- Checking rules
  - L2, L3, L4
  - Checking ACL from two file formats
  - Possibility of dynamic ACL changing
- Debugging
  - Writing and reading from PCAP file
  - Real-time packets number statistics

Code example: forwarding/dumping

```
package main
import (
        "fmt"
        "github.com/intel-go/yanff/flow"
        "github.com/intel-go/yanff/packet"
)
func main() {
        flow.SystemInit(&flow.Config{ CPUCoresNumber: 10, })
            inputFlow := flow.SetReceiver(0)
                printFlow := flow.SetPartitioner(inputFlow, 50000000, 1)
                flow.SetHandler(printFlow, hexdumper, nil)
                outputFlow := flow.SetMerger(inputFlow, printFlow)
            flow.SetSender(outputFlow, 1)
        flow.SystemStart()
}
func hexdumper(currentPacket *packet.Packet, context flow.UserContext) {
        fmt.Printf("Raw bytes=%x\n", currentPacket.GetRawPacketBytes())
}
```

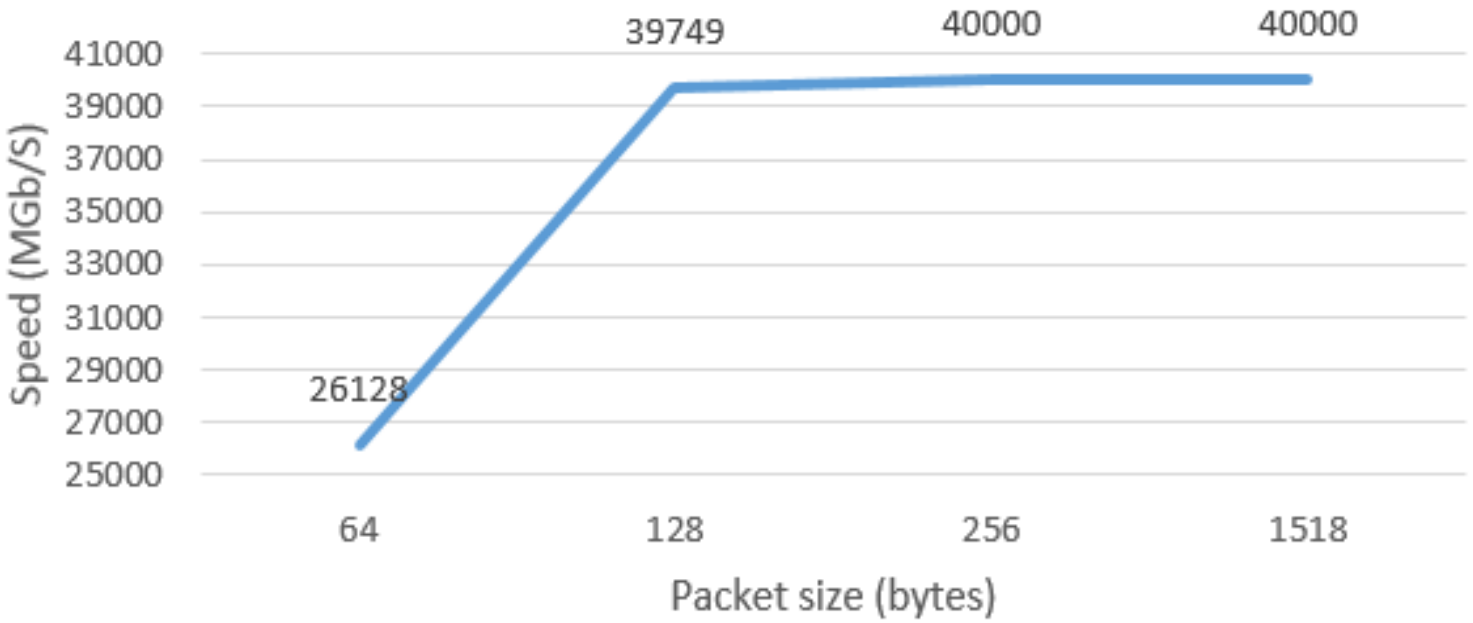
Results (1/2) IPSec

- DPDK performance is state of the art
  - There is no aim to surpass DPDK in performance
  - The aim is to surpass in productivity and scaling without loosing performance
    - ~210 code lines YANFF vs ~1500 code lines DPDK



Results (2/2) Simple forwarding

- Simple Forwarding (9 ACL rules) can handle 100% of NIC speed with packets from 256 bytes
  - *NIC has 28GB/s at 64 byte packets, instead of 40GB/s



Further development: deployment

- We are going to introduce new NFV deployment scheme in the cloud
  - No OpenStack / OpenNFV
  - Per tenant scaling between machines
  - Run to completion model per each tenant on each machine
  - Cloud boundary node as a gate in/out

- This scheme will be suitable for any virtual functions
  - Works natively with YANFF created functions

Conclusion

- New framework for rapid development of network functions
- https://github.com/intel-go/yanff
- Under heavy continuous development
- Questions?

All in all

Flow functions

Receive (Port)
driver (loop) -> Flow

Send (Port)
Flow -> driver (loop)

Stop
Flow -> free

Merge {slow}
Flow ->
Flow -> Flow

Partition (periodicity)
-> Flow
Flow -> calculation -> Flow

Split (SplitFunction) {input closed}
-> Flow
Flow -> SplitFunction -> Flow
-> Flow

Separate(SeparateFunction) {input stay}
-> Flow
Flow -> SeparateFunction -> Flow

Handle (SeparateFunction) {can drop}
-> Stop
Flow -> SeparateFunction -> Flow

Handle (HandleFunction) {can't drop}
Flow -> HandleFunction -> Flow

Generate (GenerateFunction) {can wait}
GenerateFunction -> Flow

Instances (new types)

Flow
Abstraction without public fields, which is used for pointing connections between Flow functions. Opened by Receive / Split / Separate / Counter / Generate. Closed by Send / Merge / Stop.

Packet
High-level representation of network packet. Private field is *mbuf, public fields are mac / ip / data /etc: pointers to mbuf with offsets (zero copy). Is extracted before any User defined function. Can be filled after user request by Packet functions. Can be checked by Rule functions.

Port
Network door, used in Receive, Send.

Rule
Set of checking rules, used in User defined functions.

Packet functions

Parsing packet fields
Parse L2 or/and L3 or/and L4 levels

Initializing packet fields
Initialize L2 or/and L3 or/and L4 levels

Encapsulate / Decapsulate

Rule functions

Create rule
Create checking rule from json / config

Checking packet fields by rule
Check L2 or/and L3 or/and L4 levels

User defined functions

Split Function
-> Packet -> № of Flow -> uint

Separate Function *
-> Packet -> Boolean value -> bool

Handle Function *
-> Packet ->

Generate Function *
Packet ->

All functions take packet and handling context

* Can process vector of packets at one time

All functions at separate cores and can be cloned

Connections

- External (bytes inside network)
- Flow (*mbufs inside rings)
- Packets (as function arguments)

Library External Components

- Flow: type "Flow" Init, Starting, Checking, Flow functions
- Packet: type "Packet", parsing / initializing packet functions
- Rules: type "Rule", parsing rules / checking Packet functions
- User package: user defined functions

Library Internal Components

- Scheduler: Cloning of user defined flow functions
- Asm: assembler functions added to GO
- Common: technical functions shared by other components
- Low: connections with DPDK C implementation

Optimization notice

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2®, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804