

NFF-GO (YANFF) — YET ANOTHER NETWORK FUNCTION FRAMEWORK LABS

Intel Golang Team

Gregory Shimansky

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.

© 2017 Intel Corporation.

YANFF - Yet Another Network Function Framework

Framework for building performant native network functions

- Open-source project
- Higher level abstractions than DPDK
- Go language: productivity, performance, concurrency, safety
- Network functions are application programs and not virtual machines

Benefits:

- Easily leverage IA HW capabilities: multi-cores, AES-NI, CAT, QAT, DPDK
- 10x reduction lines of code
- No need to be expert network system programmer
- Similar performance with C
- Take advantage of cloud native deployment: continuous delivery, micro-services, containers

<https://github.com/intel-go/yanff>

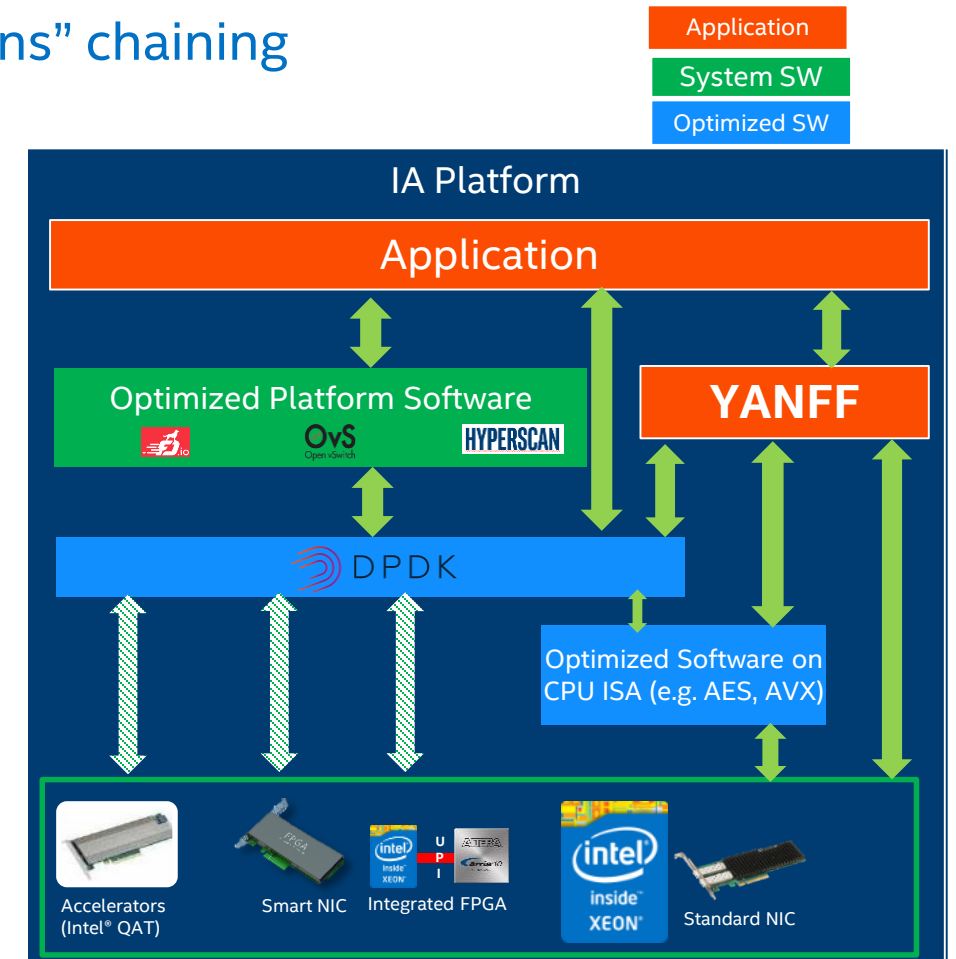
Technical Motivation

- Developers need framework to shorten development cycle of VNFs
 - Currently VNFs are monolithic - “virtual appliances” instead of network functions
 - Significant part of VNF is about plumbing. Plumbing VNFs to CommSPs network is an art. Should be abstracted from VNFs
- Lack of stable and unified APIs for VNF control and data plane
- Challenges with access to HW Accelerators in cloud environment.
- Cloud-friendly APIs and designs needed.

**Accelerating transition to from rule-based networking to
*imperative networking***

YANFF: Yet Another Network Function Framework

- Simple but powerful abstractions:
 - Flow, Packet
- User builds packet processing graph using “flow functions” chaining
 - SetReceiver -> SetHandler -> SetSender
 - Several predefined possibilities of adding user processing inside packet processing graph
 - Split, Separate, Generate, Handle
- Can leverage predefined functions which parse packets, check ACL rules, etc.
- Run to completion – NFs can be expressed in the flow functions and natural chaining
- Auto-scaling, ease of development
- Zero-copy between NFs
- Flexible incoming flow handling – sources can be anything: network port, memory buffer, remote procedure call, etc.



L3 Simple Forwarding Example

```
var L3Rules *rules.L3Rules

func main() {
    flow.SystemInit(16)
    L3Rules = rules.GetL3RulesFromORIG("Forwarding.conf")
    inputFlow := flow.SetReceiver(0)
    outputFlows := flow.SetSplitter(inputFlow, L3Splitter, uint(3))
    flow.SetStopper(outputFlows[0])
    for i := 1; i < 3; i++ {
        flow.SetSender(outputFlows[i], uint8(i-1))
    }
    flow.SystemStart()
}

// User defined function for splitting packets
func L3Splitter(currentPacket *packet.Packet) uint {
    currentPacket.ParseL4()
    return rules.L3_ACL_port(currentPacket, L3Rules)
}
```

Configuration file for Forwarding

```
# Source address, Destination address, L4 protocol ID, Source port, Destination port, Output port
111.2.0.0/31      ANY      tcp      ANY      ANY      1
111.2.0.2/32     ANY      tcp      ANY      ANY      Reject
ANY              ANY      udp      3078:3964 56:61020 2
```

1	#include <code>ethtool.h</code>	142	{0, 0, 2},	267	/* -> ETH_HLEN (incl. eth_hdr) */	655	static inline void	1005
2	#include <code>ethtool.h</code>	143	{0, 1, 2},	268	/* */	656	prepare_one_packet(struct rtw_mbuf *pkt, int, struct acl_search_t *acl,	1006
3	#include <code>ethtool.h</code>	144	{0, 2, 2},	269	enum {	657	int index)	1007
4	#include <code>ethtool.h</code>	145	{1, 0, 2},	270	RTW_ACL_IPV4VLAN_PROTO,	658	struct ipv4_hdr *ipv4_hdr;	1008
5	#include <code>ethtool.h</code>	146	{1, 1, 2},	271	RTW_ACL_IPV4VLAN_VLAN,	659	struct rtw_mbuf *pkt = pkt; int index;	1009
6	#include <code>ethtool.h</code>	147	{1, 2, 2},	272	RTW_ACL_IPV4VLAN_SMC,	660	struct ipv4_hdr *ipv4_hdr;	1010
7	#include <code>ethtool.h</code>	148	{2, 0, 2},	273	RTW_ACL_IPV4VLAN_SGT,	661	struct rtw_mbuf *pkt = pkt; int index;	1011
8	#include <code>ethtool.h</code>	149	{3, 0, 2},	274	RTW_ACL_IPV4VLAN_PROTO,	662	if (RTW_ETH_IS_IPV4_HDR(pkt->packet_type)) {	1012
9	#include <code>ethtool.h</code>	150	{3, 1, 2},	275	RTW_ACL_IPV4VLAN_NUM	663	ipv4_hdr = rtw_get_mbuf_offset(pkt, struct ipv4_hdr *,	1013
10	#include <code>ethtool.h</code>	151	};	276	};	664	sizeof(struct ether_hdr));	1014
11	#include <code>ethtool.h</code>	152		277		665		1015
12	#include <code>ethtool.h</code>	153	static struct icore_params *icore_params = icore_params; icore_params; default;	278	struct rtw_acl_field_def ipv4_def(MAP_FIELDS_IPV4) = {	666		1016
13	#include <code>ethtool.h</code>	154	static struct icore_params *icore_params = icore_params; icore_params; default;	279		667		1017
14	#include <code>ethtool.h</code>	155	static struct icore_params *icore_params = icore_params; icore_params; default;	280	.type = RTW_ACL_FIELD_TYPE_BITMAP,	668		1018
15	#include <code>ethtool.h</code>	156	static struct rtw_acl_def port_def = {	281	.size = sizeof(int),	669		1019
16	#include <code>ethtool.h</code>	157	.name = "port",	282	.field_index = PHOTO_FIELD_IPV4,	670		1020
17	#include <code>ethtool.h</code>	158	.name = "port",	283	.input_index = RTW_ACL_IPV4VLAN_PROTO,	671		1021
18	#include <code>ethtool.h</code>	159	.req_node = ETH_HQ_RX_RX,	284	.offset = 0,	672		1022
19	#include <code>ethtool.h</code>	160	.max_rx_pkt_len = ETH_HQ_RX_RX,	285	},	673		1023
20	#include <code>ethtool.h</code>	161	.split_hdr_size = 0,	286		674		1024
21	#include <code>ethtool.h</code>	162	.header_split = 0, /* Header Split disabled */	287	.type = RTW_ACL_FIELD_TYPE_MASK,	675		1025
22	#include <code>ethtool.h</code>	163	.hw_is_checksum = 1, /* IP checksum offload enabled */	288	.size = sizeof(int),	676		1026
23	#include <code>ethtool.h</code>	164	.hw_vlan_filter = 0, /* VLAN filtering disabled */	289	.field_index = SMC_FIELD_IPV4,	677		1027
24	#include <code>ethtool.h</code>	165	.jumbo_frame = 0, /* Jumbo Frame Support disabled */	290	.input_index = RTW_ACL_IPV4VLAN_SGT,	678		1028
25	#include <code>ethtool.h</code>	166	.hw_strip_crc = 1, /* CRC stripped by hardware */	291	.offset = offsetof(struct ipv4_hdr, src_addr) -	679		1029
26	#include <code>ethtool.h</code>	167		292	offsetof(struct ipv4_hdr, next_proto_id),	680		1030
27	#include <code>ethtool.h</code>	168	.rx_adv_conf = {	293	},	681		1031
28	#include <code>ethtool.h</code>	169	.rx_adv_conf = {	294		682		1032
29	#include <code>ethtool.h</code>	170	.rx_adv_conf = {	295	.type = RTW_ACL_FIELD_TYPE_MASK,	683		1033
30	#include <code>ethtool.h</code>	171	.rx_adv_conf = {	296	.size = sizeof(int),	684		1034
31	#include <code>ethtool.h</code>	172	.rx_adv_conf = {	297	.field_index = SGT_FIELD_IPV4,	685		1035
32	#include <code>ethtool.h</code>	173	.rx_adv_conf = {	298	.input_index = RTW_ACL_IPV4VLAN_SGT,	686		1036
33	#include <code>ethtool.h</code>	174	.rx_adv_conf = {	299	.offset = offsetof(struct ipv4_hdr, dst_addr) -	687		1037
34	#include <code>ethtool.h</code>	175	.rx_adv_conf = {	300	offsetof(struct ipv4_hdr, next_proto_id),	688		1038
35	#include <code>ethtool.h</code>	176	.rx_adv_conf = {	301	},	689		1039
36	#include <code>ethtool.h</code>	177	.rx_adv_conf = {	302		690		1040
37	#include <code>ethtool.h</code>	178	.rx_adv_conf = {	303	.type = RTW_ACL_FIELD_TYPE_RANGE,	691		1041

[illegible]

YANFF – Main Architectural Concepts

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**.

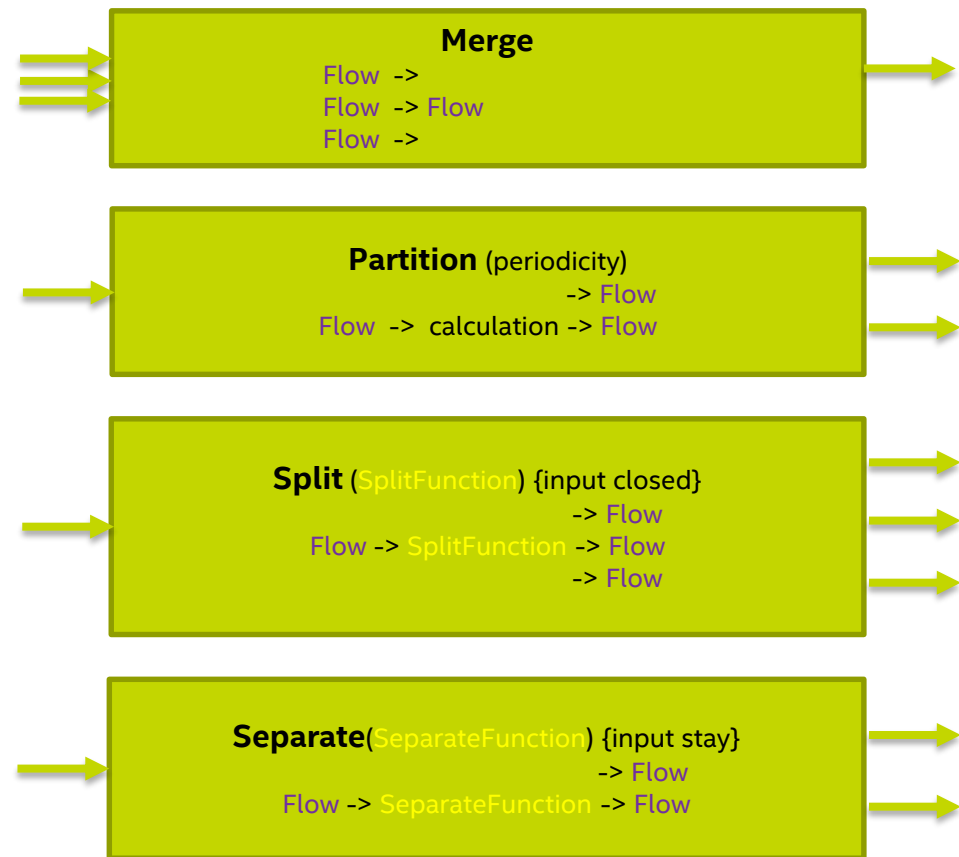
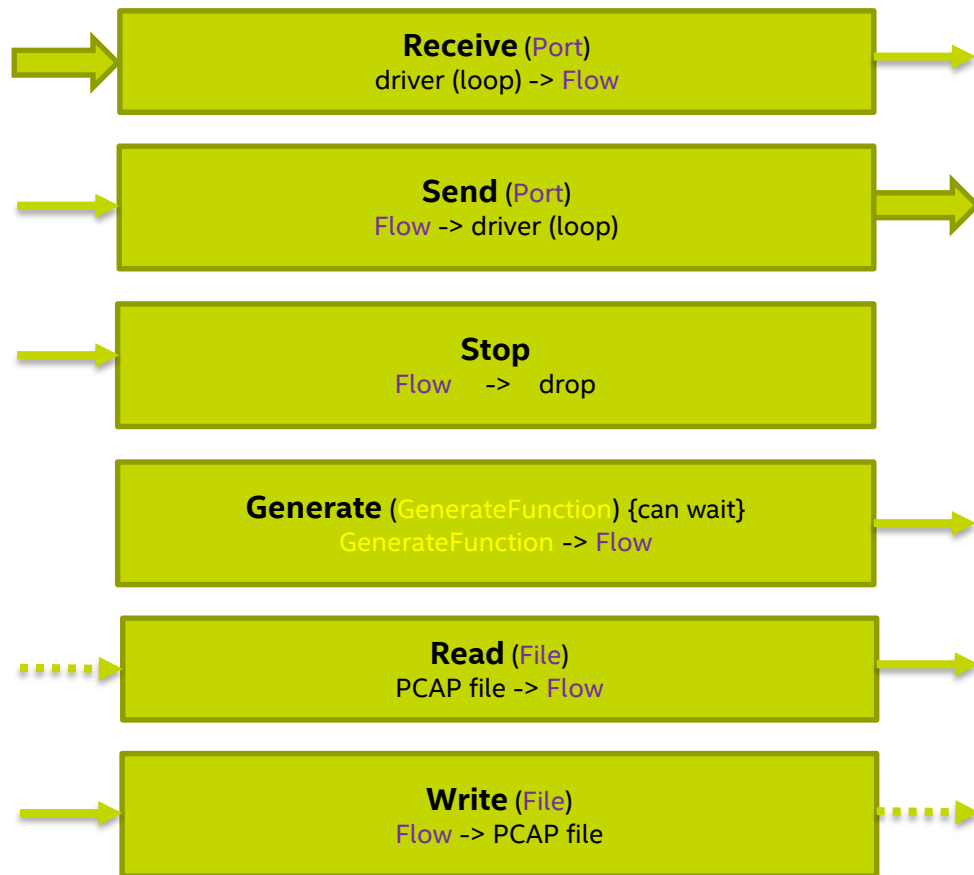
Port

Network door, used in **Receive, Send**.

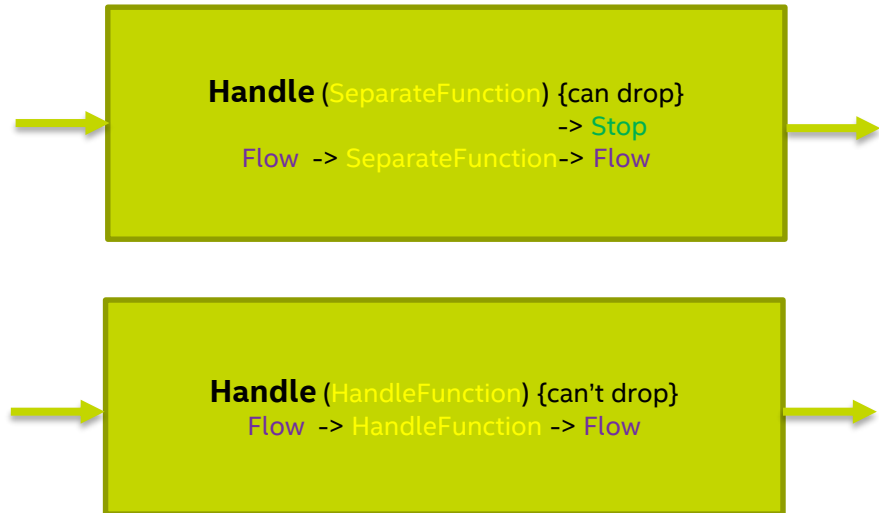
Rule

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

Building Processing Graph



Packet modification 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

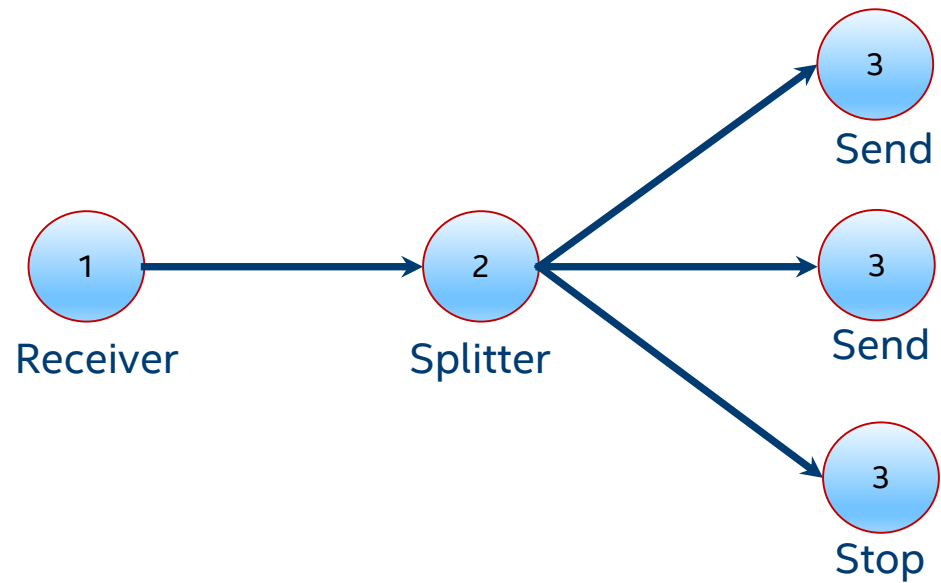
Create rule
Create checking rule from json / config

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

Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Flow Graph Example - Forwarding



Let's build some functions!

Create test VMs

1. Create and provision two test VMs:

```
$ cd nff-go/vagrant
```

```
$ vagrant up
```

2. Open two terminal windows

3. cd to vagrant directory below

4. run “vagrant ssh nff-go-”**VM_number**” to connect to pktgen VM and target VM, e.g.

```
$ vagrant ssh nff-go-1  
yanff-1$ bindports
```

```
# NFF-Go test program host  
# if ports not bound yet
```

```
$ vagrant ssh nff-go-0  
yanff-0$ bindports
```

```
# pktgen host  
# if ports not bound yet
```

Let's try (01 of 11)

Flow graph:

```
nff-go-1$ cd $YANFF/examples/tutorial }  
nff-go-1$ sudo ./step01
```

```
nff-go-0$ cd $YANFF/examples/tutorial  
nff-go-0$ ./genscripts  
nff-go-0$ ./runpktgen.sh
```

```
Pktgen:/> start 0
```

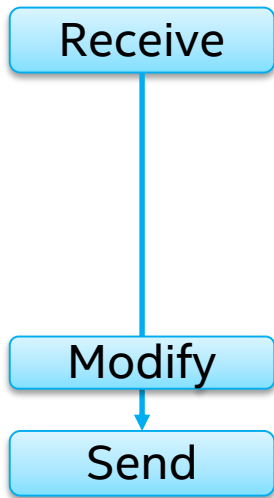
.....

```
Pktgen:/> quit
```

```
package main  
import "github.com/intel-go/yanff/flow"  
  
func main() {  
    // Init YANFF system  
    config := flow.Config{}  
    checkFatal(flow.SystemInit(&config))  
  
    initCommonState()  
  
    checkFatal(flow.SystemStart())  
}
```

Let's try (02 of 11)

Flow graph:



```
nff-go-1$ sudo ./step02
```

```
nff-go-0$ ./runpktgen.sh
```

```
Pktgen:/> load step02.pg
```

```
Pktgen:/> start 0
```

```
...
```

```
Pktgen:/> quit
```

```
package main
```

```
import "github.com/intel-go/yanff/flow"
```

```
func main() {
```

```
    config := flow.Config{}
```

```
    checkFatal(flow.SystemInit(&config))
```

```
    initCommonState()
```

```
    firstFlow, err := flow.SetReceiver(0)
```

```
    checkFatal(err)
```

```
    checkFatal(flow.SetHandler(firstFlow, modifyPacket[0], nil))
```

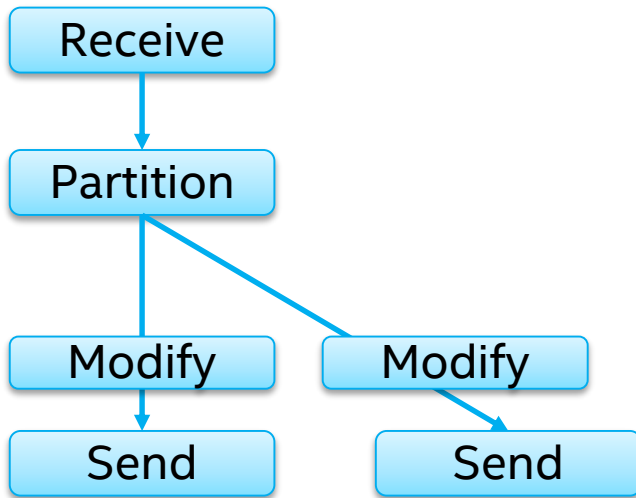
```
    checkFatal(flow.SetSender(firstFlow, 0))
```

```
    checkFatal(flow.SystemStart())
```

```
}
```


Let's try (03 of 11)

Flow graph:



```
nff-go-1$ sudo ./step03
```

```
nff-go-0$ ./runpktgen.sh  
Pktgen: /> load step03.pg  
Pktgen: /> start 0
```

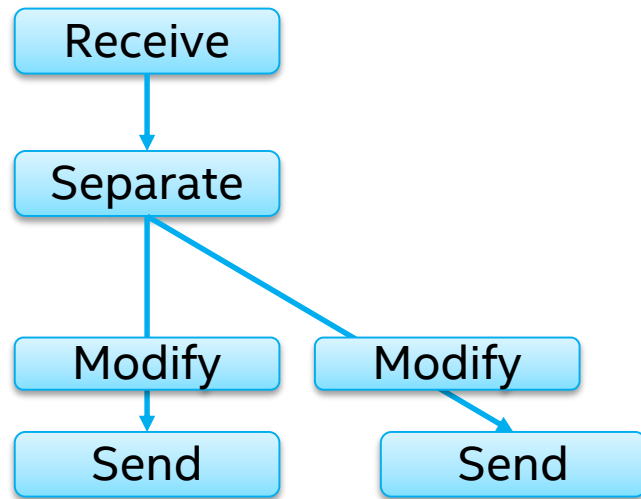
...

```
Pktgen: /> quit
```

```
package main  
import "github.com/intel-go/yanff/flow"  
  
func main() {  
    config := flow.Config{}  
    checkFatal(flow.SystemInit(&config))  
  
    initCommonState()  
  
    firstFlow, err := flow.SetReceiver(0)  
    checkFatal(err)  
    secondFlow, err := flow.SetPartitioner(firstFlow, 300, 300)  
    checkFatal(err)  
    checkFatal(flow.SetHandler(firstFlow, modifyPacket[0], nil))  
    checkFatal(flow.SetHandler(secondFlow, modifyPacket[1], nil))  
    checkFatal(flow.SetSender(firstFlow, 0))  
    checkFatal(flow.SetSender(secondFlow, 1))  
  
    checkFatal(flow.SystemStart())  
}
```

Let's try (04 of 11)

Flow graph:



```
nff-go-1$ sudo ./step04
```

```
nff-go-0$ ./runpktgen.sh
```

```
Pktgen:/> load step04.pg
```

```
Pktgen:/> start 0
```

...

```
Pktgen:/> quit
```

```
package main
import "github.com/intel-go/yanff/flow"
import "github.com/intel-go/yanff/packet"
```

```
func main() {
    config := flow.Config{}
    checkFatal(flow.SystemInit(&config))
    initCommonState()

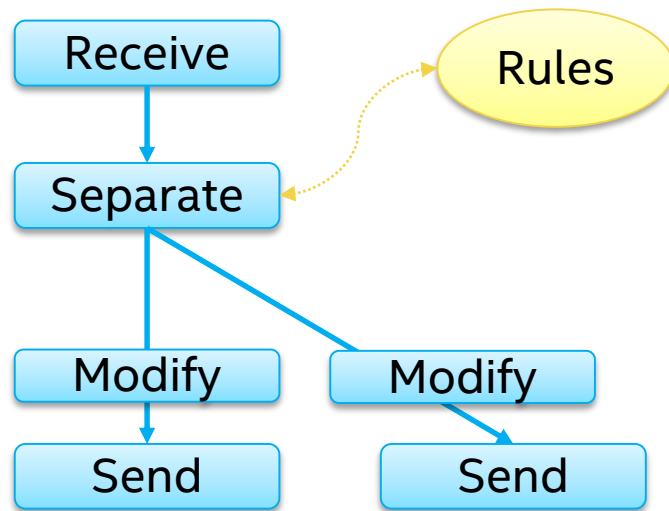
    firstFlow, err := flow.SetReceiver(0)
    checkFatal(err)
    secondFlow, err := flow.SetSeparator(firstFlow, mySeparator, nil)
    checkFatal(err)
    checkFatal(flow.SetHandler(firstFlow, modifyPacket[0], nil))
    checkFatal(flow.SetHandler(secondFlow, modifyPacket[1], nil))
    checkFatal(flow.SetSender(firstFlow, 0))
    checkFatal(flow.SetSender(secondFlow, 1))

    checkFatal(flow.SystemStart())
}

func mySeparator(cur *packet.Packet, ctx flow.UserContext) bool {
    cur.ParseL3()
    if cur.GetIPv4() != nil {
        cur.ParseL4ForIPv4()
        if cur.GetTCPForIPv4() != nil &&
            packet.SwapBytesUint16(cur.GetTCPForIPv4().DstPort) == 53 {
            return false
        }
    }
    return true
}
```

Let's try (05 of 11)

Flow graph:



```
nff-go-1$ sudo ./step05
```

```
nff-go-0$ ./runpktgen.sh
Pktgen:/> load step05.pg
Pktgen:/> start 0
...
Pktgen:/> quit
```

```
... ..
import "github.com/intel-go/yanff/rules"
var L3Rules *rules.L3Rules
```

```
func main() {
    var err error
    config := flow.Config{}
    checkFatal(flow.SystemInit(&config))
    initCommonState()

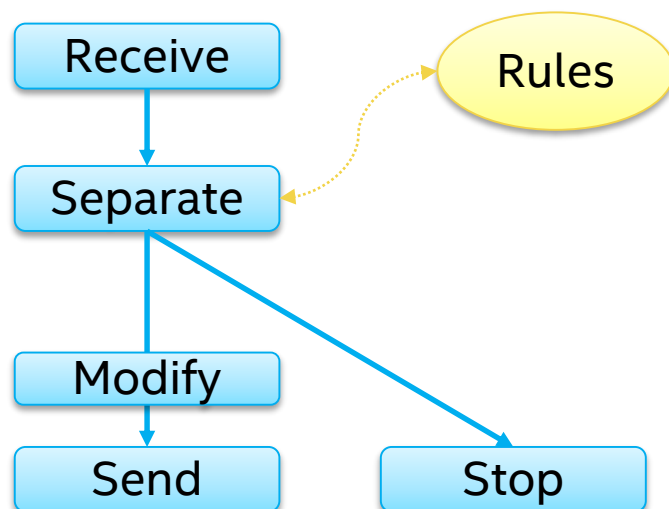
    l3Rules, err = packet.GetL3ACLFromORIG("rules1.conf")
    checkFatal(err)

    firstFlow, err := flow.SetReceiver(0)
    checkFatal(err)
    secondFlow, err := flow.SetSeparator(firstFlow, mySeparator, nil)
    checkFatal(err)
    checkFatal(flow.SetHandler(firstFlow, modifyPacket[0], nil))
    checkFatal(flow.SetHandler(secondFlow, modifyPacket[1], nil))
    checkFatal(flow.SetSender(firstFlow, 0))
    checkFatal(flow.SetSender(secondFlow, 1))
    checkFatal(flow.SystemStart())
}

func MySeparator(cur *packet.Packet, ctx flow.UserContext) bool {
    return cur.L3ACLPermit(l3Rules)
}
```

Let's try (06 of 11)

Flow graph:



```
nff-go-1$ sudo ./step06
```

```
nff-go-0$ ./runpktgen.sh
```

```
Pktgen:/> load step06.pg
```

```
Pktgen:/> start 0
```

```
...
```

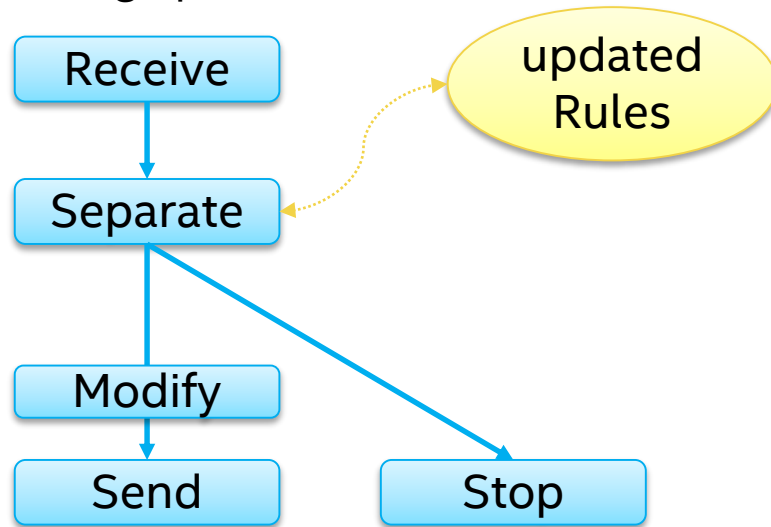
```
Pktgen:/> quit
```

```
func main() {
    var err error
    config := flow.Config{}
    checkFatal(flow.SystemInit(&config))
    L3Rules = rules.GetL3RulesFromORIG("rules1.conf")
    checkFatal(err)
    firstFlow, err := flow.SetReceiver(0)
    checkFatal(err)
    secondFlow, err := flow.SetSeparator(firstFlow, mySeparator, nil)
    checkFatal(err)
    checkFatal(flow.SetHandler(firstFlow, modifyPacket[0], nil))
    checkFatal(flow.SetSender(firstFlow, 0))
    checkFatal(flow.SetStopper(secondFlow))
    checkFatal(flow.SystemStart())
}

func MySeparator(cur *packet.Packet, ctx flow.UserContext) bool {
    return cur.L3ACLPermit(l3Rules)
}
```

Let's try (07 of 11)

Flow graph:



```
nff-go-1$ sudo ./step07
```

```
nff-go-0$ ./runpktgen.sh
```

```
Pktgen:/> load step07.pg
```

```
Pktgen:/> start 0
```

```
...
```

```
Pktgen:/> quit
```

```
... ..
import "time"
var rulesp unsafe.Pointer

... ..
    l3Rules, err := packet.GetL3ACLFromORIG("rules1.conf")
    checkFatal(err)
    rulesp = unsafe.Pointer(&l3Rules)
    go updateSeparateRules()
... ..

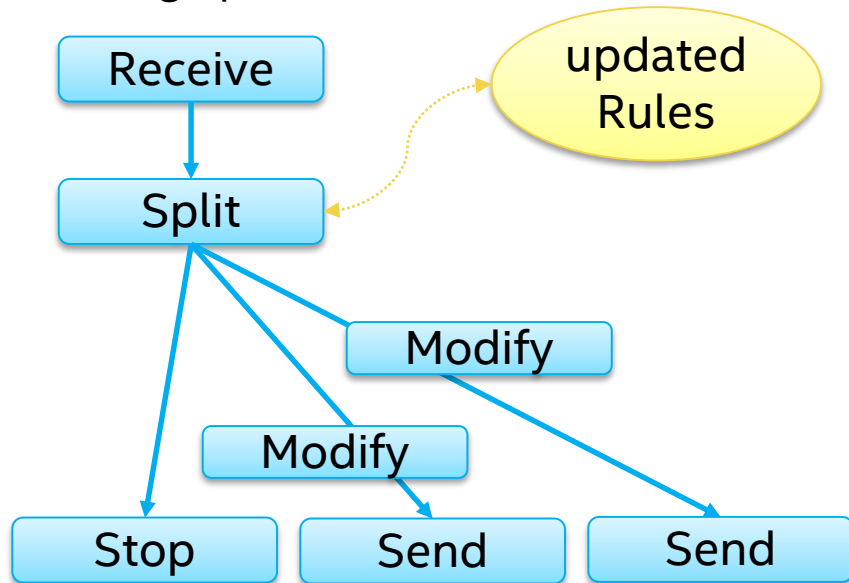
func MySeparator(cur *packet.Packet, ctx flow.UserContext) bool {
    localL3Rules := (*packet.L3Rules)(atomic.LoadPointer(&rulesp))
    return cur.L3ACLPermit(localL3Rules)
}

func updateSeparateRules() {
    for {
        time.Sleep(time.Second * 5)
        localL3Rules, err := packet.GetL3ACLFromORIG("rules1.conf")
        checkFatal(err)
        atomic.StorePointer(&rulesp, unsafe.Pointer(localL3Rules))
    }
}
```

To make changes in rules1.conf file it is necessary to connect to target VM in another window or run YANFF executable in screen terminal multiplexer.

Let's try (08 of 11)

Flow graph:



```
nff-go-1$ sudo ./step08
```

```
nff-go-0$ ./runpktgen.sh
```

```
Pktgen:/> load step08.pg
```

```
Pktgen:/> start 0
```

```
...
```

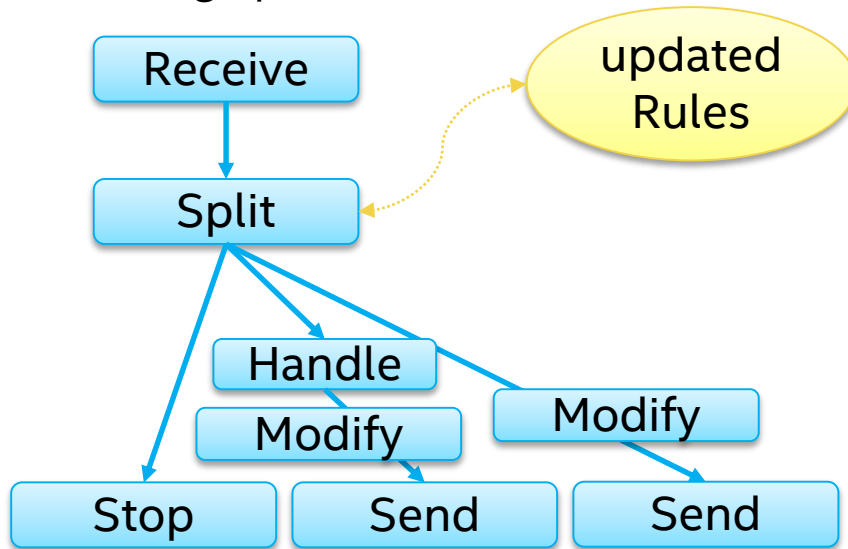
```
Pktgen:/> quit
```

```
... ..  
const flowN = 3  
... ..  
firstFlow, err := flow.SetReceiver(0)  
checkFatal(err)  
outputFlows, err := flow.SetSplitter(firstFlow, mySplitter, flowN, nil)  
checkFatal(err)  
checkFatal(flow.SetStopper(outputFlows[0]))  
for i := uint8(1); i < flowN; i++ {  
    checkFatal(flow.SetHandler(outputFlows[i], modifyPacket[i-1], nil))  
    checkFatal(flow.SetSender(outputFlows[i], i-1))  
}  
... ..  
func mySplitter(cur *packet.Packet, ctx flow.UserContext) uint {  
    localL3Rules := L3Rules  
    return cur.L3ACLPort(localL3Rules)  
}  
... ..
```

To make changes in rules2.conf file it is necessary to connect to target VM in another window or run YANFF executable in screen terminal multiplexer.

Let's try (09 of 11)

Flow graph:



```
nff-go-1$ sudo ./step09
```

```
nff-go-0$ ./runpktgen.sh
```

```
Pktgen:/> load step09.pg
```

```
Pktgen:/> start 0
```

```
...
```

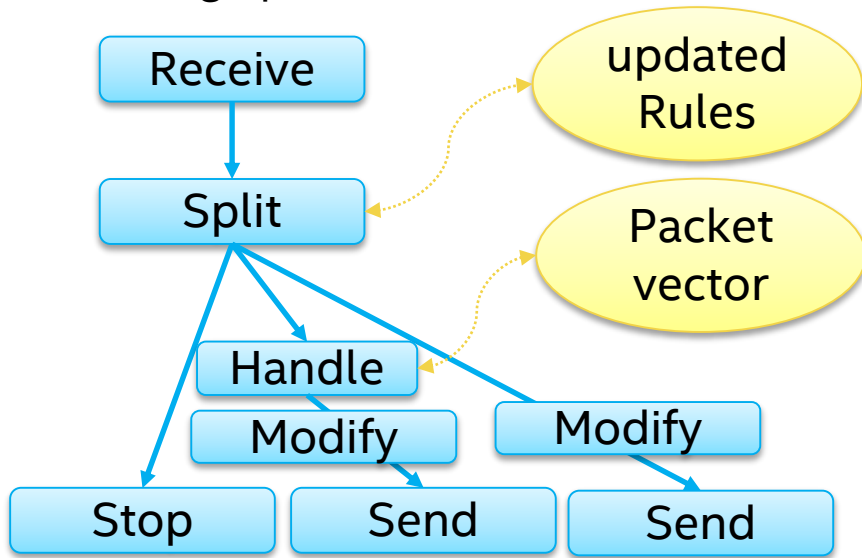
```
Pktgen:/> quit
```

```
... ..  
import "github.com/intel-go/yanff/common"  
... ..  
firstFlow, err := flow.SetReceiver(0)  
checkFatal(err)  
outputFlows, err := flow.SetSplitter(firstFlow, mySplitter, flowN, nil)  
checkFatal(err)  
checkFatal(flow.SetStopper(outputFlows[0]))  
checkFatal(flow.SetHandler(outputFlows[1], myHandler, nil))  
for i := uint8(1); i < flowN; i++ {  
    checkFatal(flow.SetHandler(outputFlows[i], modifyPacket[i-1], nil))  
    checkFatal(flow.SetSender(outputFlows[i], i-1))  
}  
... ..  
func myHandler(cur *packet.Packet, ctx flow.UserContext) {  
    cur.EncapsulateHead(common.EtherLen, common.IPv4MinLen)  
    cur.ParseL3()  
    cur.GetIPv4NoCheck().SrcAddr = packet.BytesToIPv4(111, 22, 3, 0)  
    cur.GetIPv4NoCheck().DstAddr = packet.BytesToIPv4(3, 22, 111, 0)  
    cur.GetIPv4NoCheck().VersionIhl = 0x45  
    cur.GetIPv4NoCheck().NextProtoID = 0x04  
}
```

To make changes in rules2.conf file it is necessary to connect to target VM in another window or run YANFF executable in screen terminal multiplexer.

Let's try (10 of 11)

Flow graph:



... ..

```
func myHandler(curV []*packet.Packet, num uint, ctx flow.UserContext) {
    for i := uint(0); i < num; i++ {
        cur := curV[i]
        cur.EncapsulateHead(common.EtherLen, common.IPv4MinLen)
        cur.ParseL3()
        cur.GetIPv4NoCheck().SrcAddr = packet.BytesToIPv4(111, 22, 3, 0)
        cur.GetIPv4NoCheck().DstAddr = packet.BytesToIPv4(3, 22, 111, 0)
        cur.GetIPv4NoCheck().VersionIhl = 0x45
        cur.GetIPv4NoCheck().NextProtoID = 0x04
    }
}
```

```
nff-go-1$ sudo ./step10
```

```
nff-go-0$ ./runpktgen.sh
```

```
Pktgen:/> load step10.pg
```

```
Pktgen:/> start 0
```

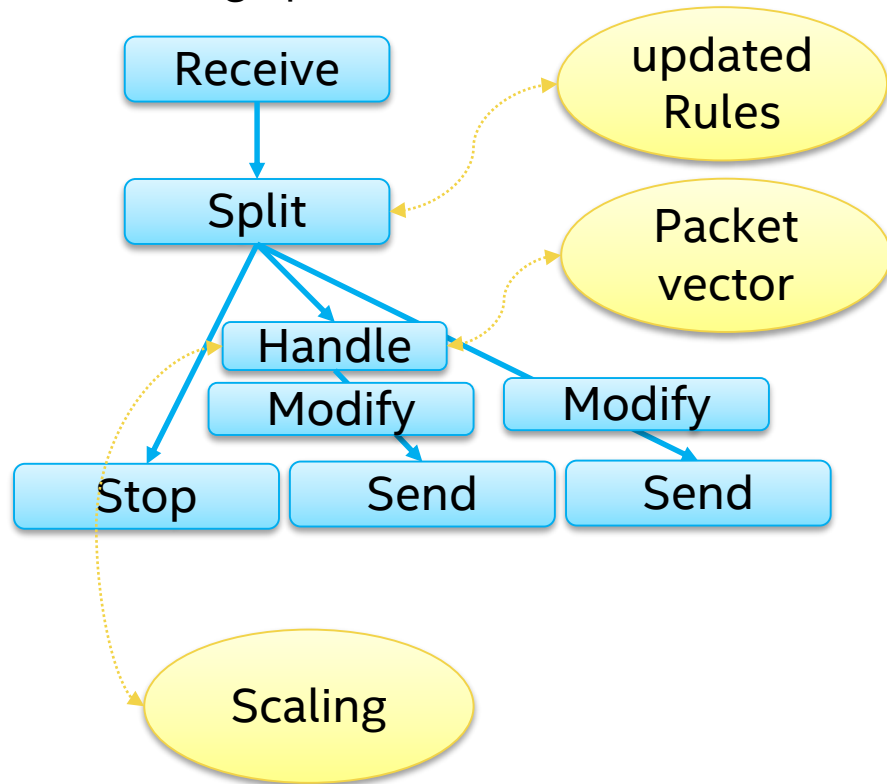
...

```
Pktgen:/> quit
```

To make changes in rules2.conf file it is necessary to connect to target VM in another window or run YANFF executable in screen terminal multiplexer.

Let's try (11 of 11)

Flow graph:



To make changes in rules2.conf file it is necessary to connect to target VM in another window or run YANFF executable in screen terminal multiplexer.

... ..

```
func myHandler(curV []*packet.Packet, num uint, ctx flow.UserContext) {
    for i := uint(0); i < num; i++ {
        cur := curV[i]
        cur.EncapsulateHead(common.EtherLen, common.Ipv4MinLen)
        cur.ParseL3()
        cur.GetIPv4NoCheck().SrcAddr = packet.BytesToIPv4(111, 22, 3, 0)
        cur.GetIPv4NoCheck().DstAddr = packet.BytesToIPv4(3, 22, 111, 0)
        cur.GetIPv4NoCheck().VersionIhl = 0x45
        cur.GetIPv4NoCheck().NextProtoID = 0x04
    }
    // Some heavy computational code
    heavyCode()
}
```

```
nff-go-1$ sudo ./step11
```

```
nff-go-0$ ./runpktgen.sh
```

```
Pktgen:/> load step11.pg
```

```
Pktgen:/> start 0
```

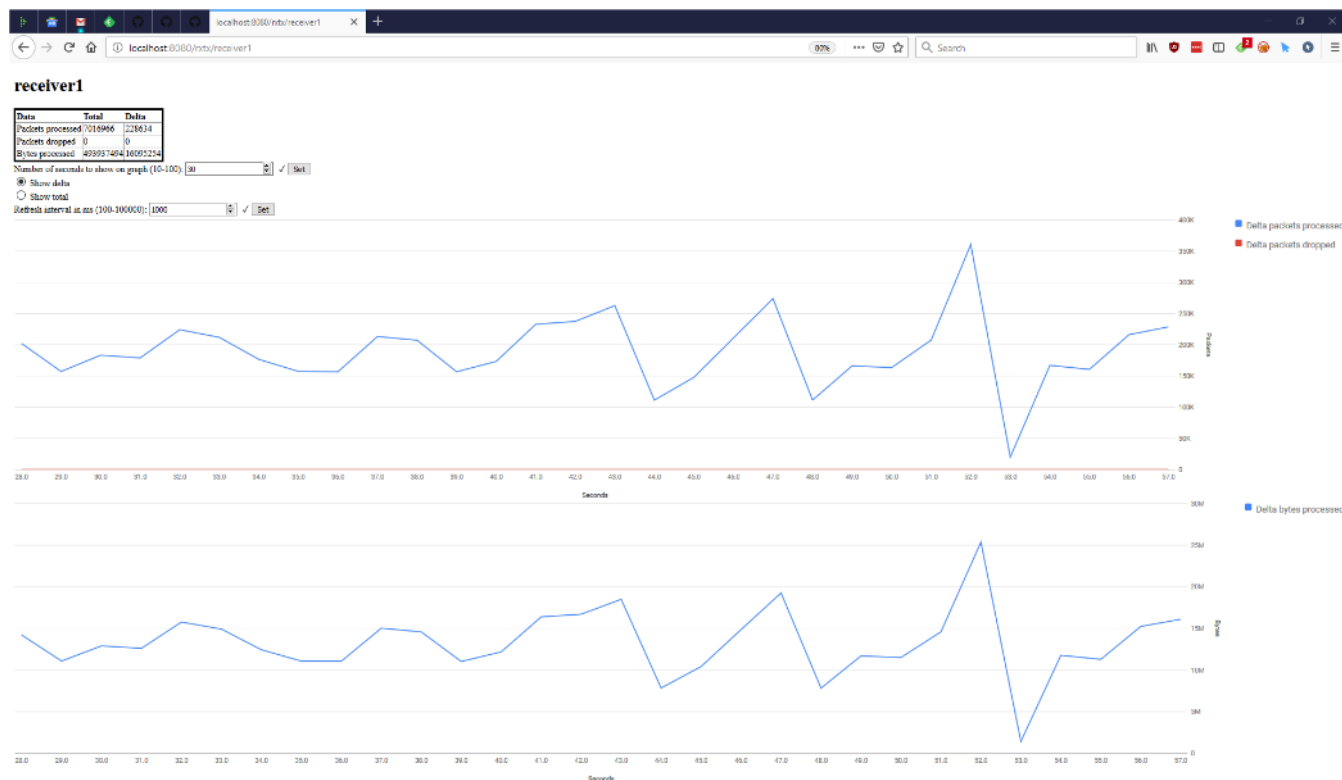
...

```
Pktgen:/> quit
```

Alternative network packet IO

- [KNI interfaces \(examples/kni.go\)](#)
- [Linux raw sockets \(examples/OSforwarding.go\)](#)
- [PCAP files \(examples/clonablePcapDumper.go\)](#)
- Linux XDP (coming soon)

Statistic counters



... ..

```
// Set up address for stats web server
statsServerAddress = &net.TCPAddr{
    Port: 8080,
}
```

```
config := flow.Config{
    StatsHTTPAddress: statsServerAddress,
}
```

... ..

Finally: NAT

```
nff-go-1$ ./genscripts -pktgen direct  
nff-go-1$ sudo ../nat/main/nat -config nat.json
```

```
nff-go-0$ ./runpktgen.sh  
Pktgen:/> load nat.pg  
Pktgen:/> start 0  
Pktgen:/> start 1  
...  
Pktgen:/> quit
```

Q & A ?

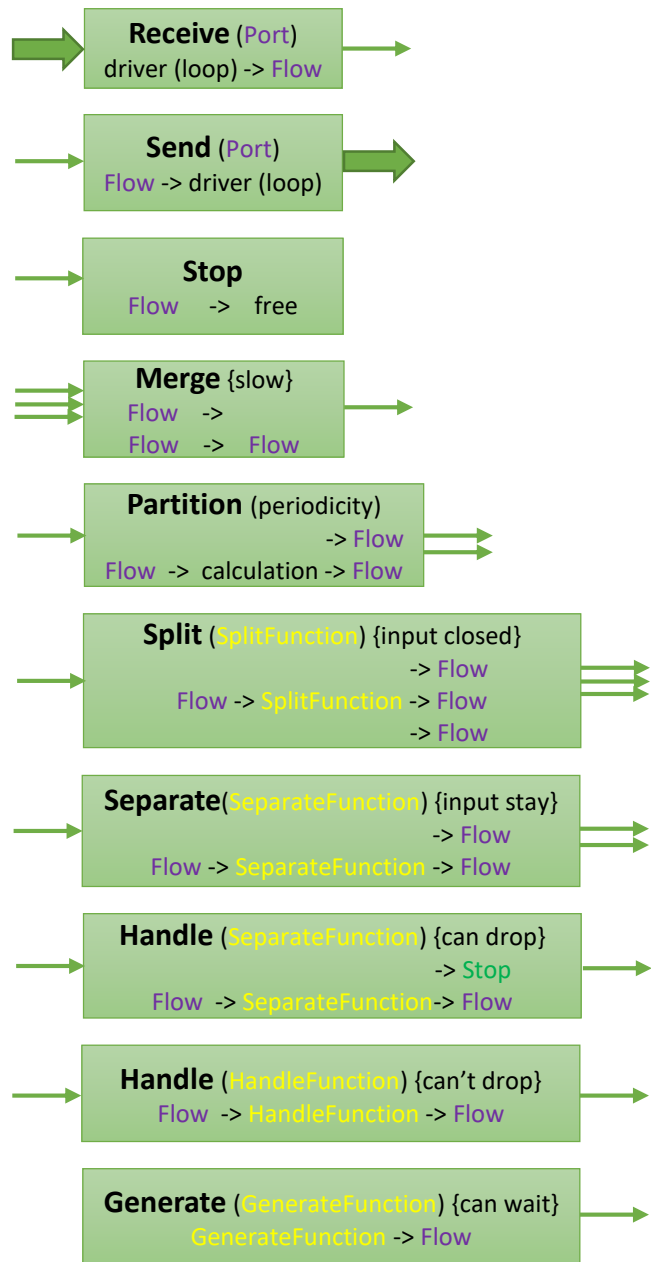
Optimization Notice

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.

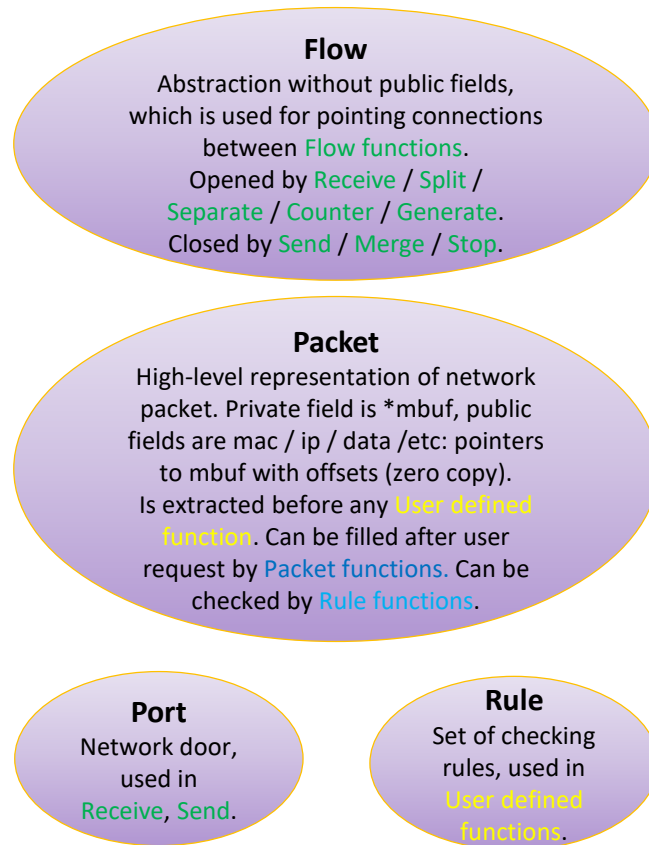
Notice revision #20110804

Flow functions

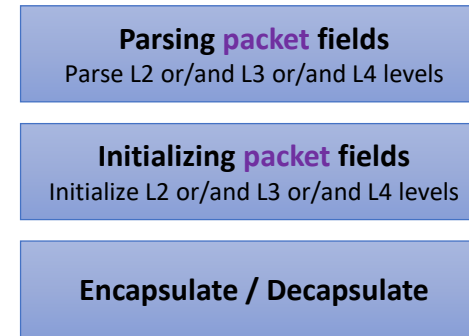


Basic components

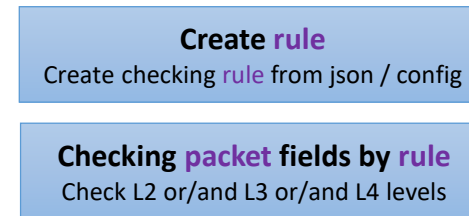
Instances (new types)



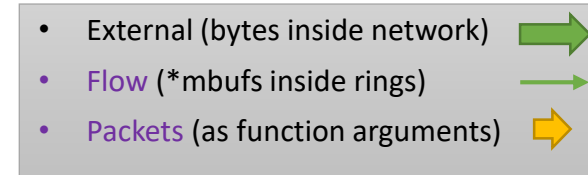
Packet functions



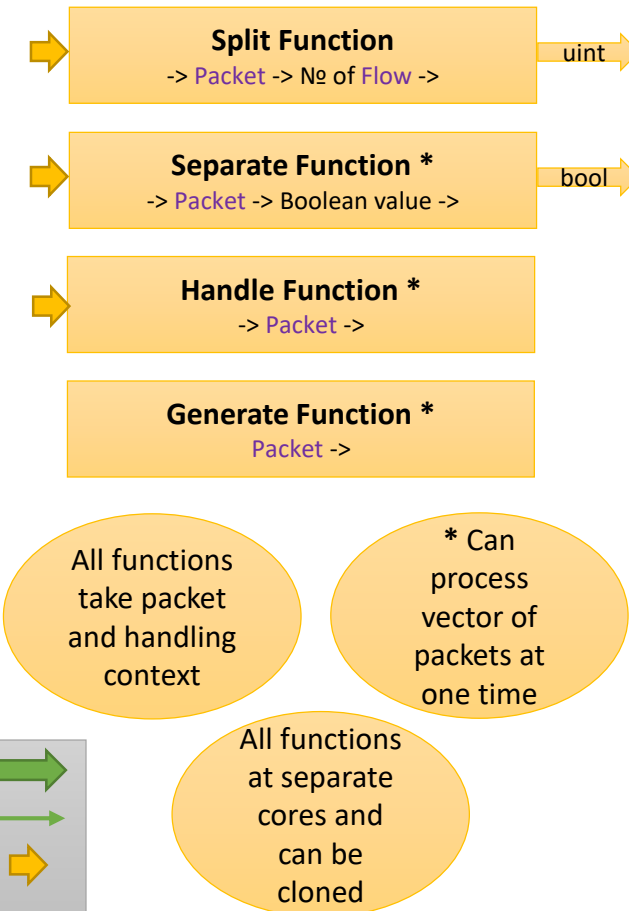
Rule functions



Connections



User defined functions



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

Lab configuration



Jump host: , Login: gashiman, Password:

Finally (2 of 2): ipsec

- Showing ipsec example