

# 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](http://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.

# NFF-Go (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/nff-go>

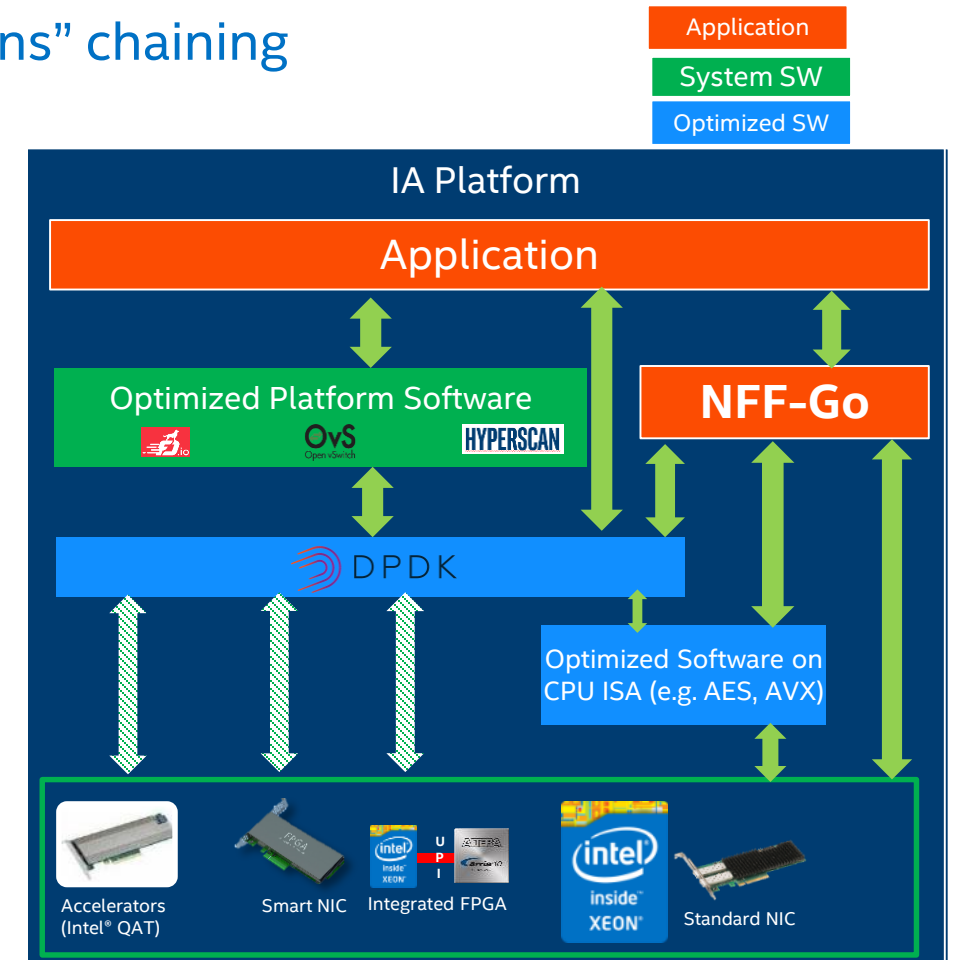
# Technical Motivation

- 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***

# NFF-Go: 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 <stdio.h>	162	{0, 0, 2},	270	/* - PORTS (LVL, ADDR, etc) */	655	static inline void	1905
2	#include <stdlib.h>	163	{0, 1, 2},	271	/*	656	prepare_ones_packet(struct rte_mbuf *pktb_in, struct acl_search_t *acl,	1906
3	#include <stdint.h>	164	{0, 2, 2},	272	*/	657	int index)	1907
4	#include <ctype.h>	165	{1, 0, 2},	273	enum {	658		1908
5	#include <sys/types.h>	166	{1, 1, 2},	274	RTE_ACL_IPV4VLAN_VLAN,	659	struct ipv4_hdr *ipv4_hdr;	1909
6	#include <string.h>	167	{1, 2, 2},	275	RTE_ACL_IPV4VLAN_VLAN_SCTP,	660	struct rte_mbuf *pktb_in[index];	1910
7	#include <sys/socket.h>	168	{2, 0, 2},	276	RTE_ACL_IPV4VLAN_SCTP,	661		1911
8	#include <unistd.h>	169	{3, 0, 3},	277	RTE_ACL_IPV4VLAN_PROTO,	662	if (RTE_ETH_IS_IPV4_HDR(pktb_in->packet_type)) {	1912
9	#include <errno.h>	170	{3, 1, 3},	278	RTE_ACL_IPV4VLAN_NUM	663	ipv4_hdr = rte_pktmbuf_mtdata_offset(pktb_in, struct ipv4_hdr *);	1913
10	#include <getopt.h>	171	};	279	};	664	sizeof(struct ether_hdr, hdr));	1914
11		172		280		665		1915
12	#include <rtm/common.h>	173	static struct acl_params {	281	struct rte_acl_field_def ipv4_data[NUM_FIELDS_IPV4] = {	666		1916
13	#include <rtm/hypervisor.h>	174	static struct acl_params {	282	static struct acl_params {	667	/* Check to make sure the packet is valid (RFC8121) */	1917
14	#include <rtm/log.h>	175	static struct acl_params {	283	static struct acl_params {	668	if (is_valid_ipv4_pkt(ipv4_hdr, pktb_in->len) == 0) {	1918
15	#include <rtm/memory.h>	176	static struct acl_params {	284	static struct acl_params {	669	/* Update time to live and header checksum */	1919
16	#include <rtm/memmap.h>	177	static struct acl_params {	285	static struct acl_params {	670	--input index = RTE_ACL_IPV4VLAN_PROTO,	1920
17	#include <rtm/memmap.h>	178	static struct acl_params {	286	static struct acl_params {	671	--input index = RTE_ACL_IPV4VLAN_PROTO,	1921
18	#include <rtm/memmap.h>	179	static struct acl_params {	287	static struct acl_params {	672	--offset = 0,	1922
19	#include <rtm/memmap.h>	180	static struct acl_params {	288	static struct acl_params {	673	/* Fill acl structure */	1923
20	#include <rtm/memmap.h>	181	static struct acl_params {	289	static struct acl_params {	674	acl->data_ipv4[acl->num_ipv4] = NULL;	1924
21	#include <rtm/memmap.h>	182	static struct acl_params {	290	static struct acl_params {	675	acl->data_ipv4[acl->num_ipv4] = NULL;	1925
22	#include <rtm/memmap.h>	183	static struct acl_params {	291	static struct acl_params {	676	acl->data_ipv4[acl->num_ipv4] = NULL;	1926
23	#include <rtm/memmap.h>	184	static struct acl_params {	292	static struct acl_params {	677	/* Not a valid IPv4 packet */	1927
24	#include <rtm/memmap.h>	185	static struct acl_params {	293	static struct acl_params {	678	rte_pktmbuf_free(pktb_in);	1928
25	#include <rtm/memmap.h>	186	static struct acl_params {	294	static struct acl_params {	679		1929
26	#include <rtm/memmap.h>	187	static struct acl_params {	295	static struct acl_params {	680		1930
27	#include <rtm/memmap.h>	188	static struct acl_params {	296	static struct acl_params {	681	/* Fill acl structure */	1931
28	#include <rtm/memmap.h>	189	static struct acl_params {	297	static struct acl_params {	682	/* Fill acl structure */	1932
29	#include <rtm/memmap.h>	190	static struct acl_params {	298	static struct acl_params {	683	/* Fill acl structure */	1933
30	#include <rtm/memmap.h>	191	static struct acl_params {	299	static struct acl_params {	684	/* Fill acl structure */	1934
31	#include <rtm/memmap.h>	192	static struct acl_params {	300	static struct acl_params {	685	/* Fill acl structure */	1935
32	#include <rtm/memmap.h>	193	static struct acl_params {	301	static struct acl_params {	686	/* Fill acl structure */	1936
33	#include <rtm/memmap.h>	194	static struct acl_params {	302	static struct acl_params {	687	/* Fill acl structure */	1937
34	#include <rtm/memmap.h>	195	static struct acl_params {	303	static struct acl_params {	688	/* Fill acl structure */	1938
35	#include <rtm/memmap.h>	196	static struct acl_params {	304	static struct acl_params {	689	/* Fill acl structure */	1939
36	#include <rtm/memmap.h>	197	static struct acl_params {	305	static struct acl_params {	690	/* Fill acl structure */	1940
37	#include <rtm/memmap.h>	198	static struct acl_params {	306	static struct acl_params {	691	/* Fill acl structure */	1941
38	#include <rtm/memmap.h>	199	static struct acl_params {	307	static struct acl_params {	692	/* Fill acl structure */	1942
39	#include <rtm/memmap.h>	200	static struct acl_params {	308	static struct acl_params {	693	/* Fill acl structure */	1943
40	#include <rtm/memmap.h>	201	static struct acl_params {	309	static struct acl_params {	694	/* Fill acl structure */	1944
41	#include <rtm/memmap.h>	202	static struct acl_params {	310	static struct acl_params {	695	/* Fill acl structure */	1945
42	#include <rtm/memmap.h>	203	static struct acl_params {	311	static struct acl_params {	696	/* Fill acl structure */	1946
43	#include <rtm/memmap.h>	204	static struct acl_params {	312	static struct acl_params {	697	/* Fill acl structure */	1947
44	#include <rtm/memmap.h>	205	static struct acl_params {	313	static struct acl_params {	698		

```

/* Define MAX_PACKET_LEN used
#define MAX_PACKET_LEN 1500
*/
#define MAXPOOL_CACHE_SIZE 126
/*
 * This expression is used to calculate the number of mbufs needed
 * depending on user input, taking into account memory for rx and tx hardware
 * rings, cache per queue and stable net per core.
 */
/* RTE_MBUF is used to ensure that NO_MBUF never goes below a
 * minimum value of 1020
 */
/*
 * Define MB_MBUF RTE_MAX()
 */
/*
 * nb ports + nb rx queues + RTE_TX_DESC_DEFAULT +
 * nb ports + nb cores + MAX_PKT_BURST +
 * nb ports + n tx queues + RTE_TX_DESC_DEFAULT +
 * nb cores + MAXPOOL_CACHE_SIZE),
 * (unsigned)1020);
 */
/*
 * Define MAX_PACKET_BURST 32
 */
/*
 * Define BURST_TX_DRAIN_US 100 /* TX drain every -100us */
 */
/*
 * Define MB_SOCKETS 8
 */
/* Configure how many packets ahead to prefetch, when reading packets */
#define PREFETCH_OFFSET 3
/*
 * Configurable number of RX/TX ring descriptors
 */
/*
 * Define RTE_TEST_RX_DESC_DEFAULT 128
 */
/*
 * Define RTE_TEST_TX_DESC_DEFAULT 512
 */
static uint16_t nb_rxd = RTE_TEST_RX_DESC_DEFAULT;
static uint16_t nb_tx_d = RTE_TEST_TX_DESC_DEFAULT;
/* ethernet addresses of ports */
static struct ether_addr ports_eth_addr[RTE_MAX_ETHPORTS];
/* mask of enabled ports */
static uint32_t enabled_port_mask;
static int promiscuous_mode /*== Ports set in promiscuous mode off by default.
static int num_rx = 3; /*== NUMA is disabled by default. */
*/
/*
 * ACU rules should have higher priorities than route ones to ensure ACU
 * always be found when input packs hit the rulebase.
 * A exception case is performance measure, which can define route rules w
 * Higher priority and route rules will always be returned in each lookup.
 * Reserve range from ACU RULE_PRIORITY_MAX + 1 to
 * up to new maximum size value relative to performance measure
 */

```



# NFF-Go – 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**.

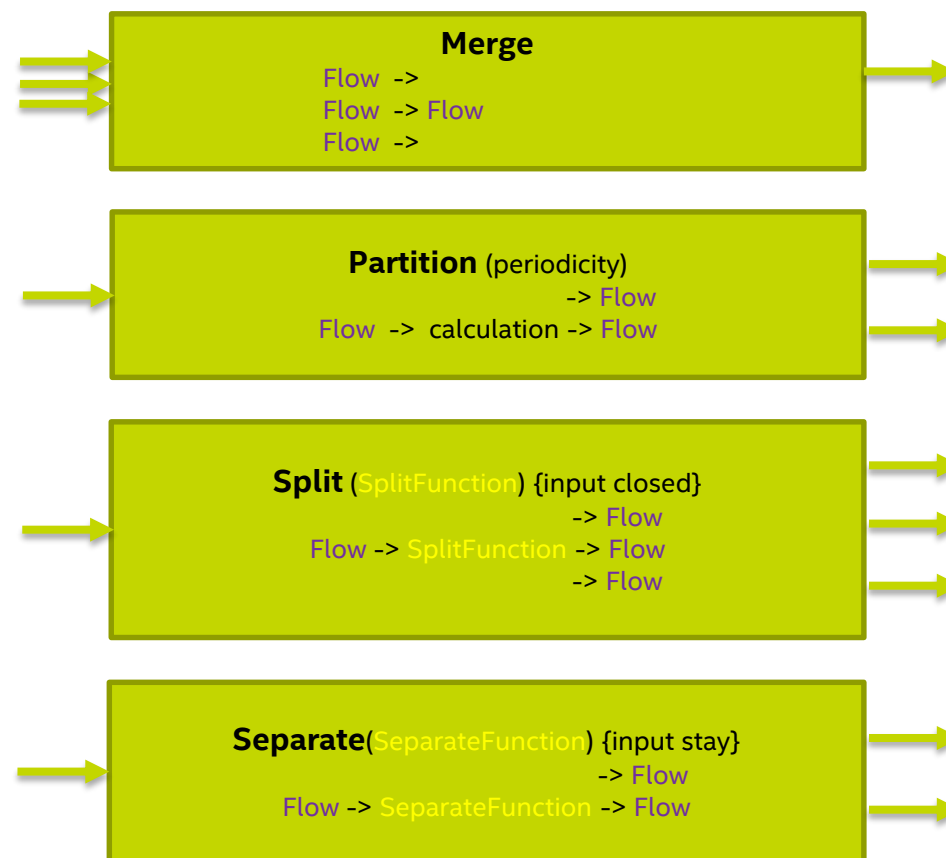
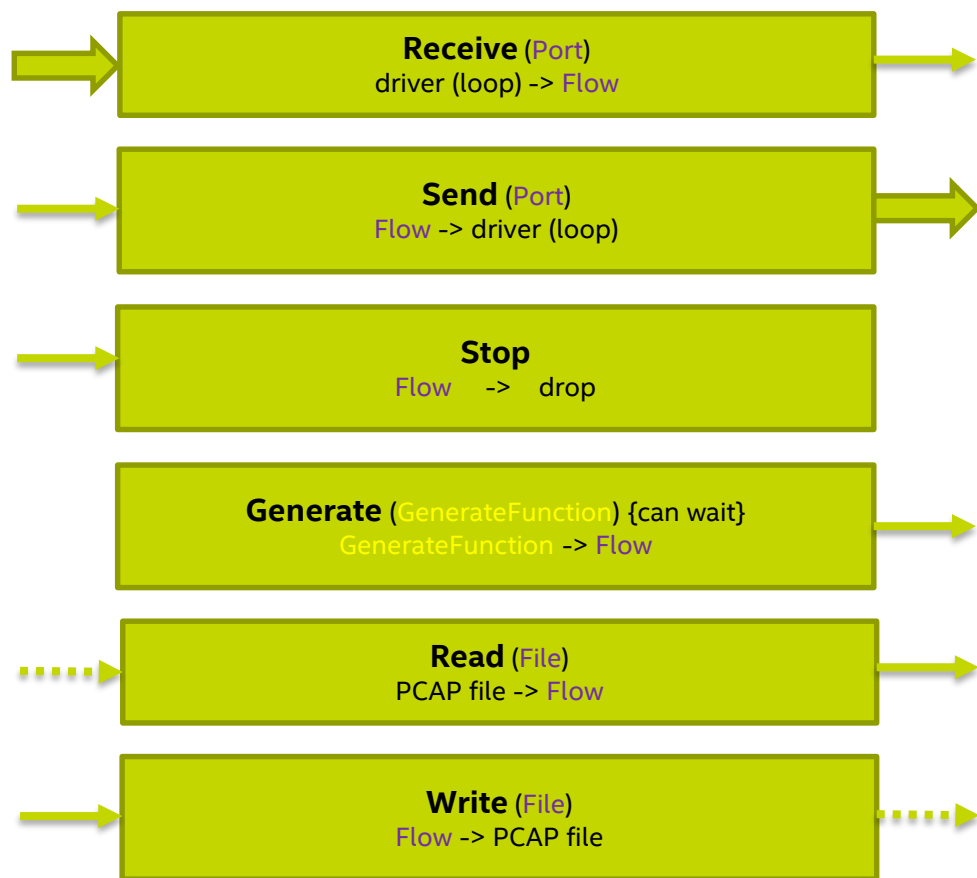
## Port

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

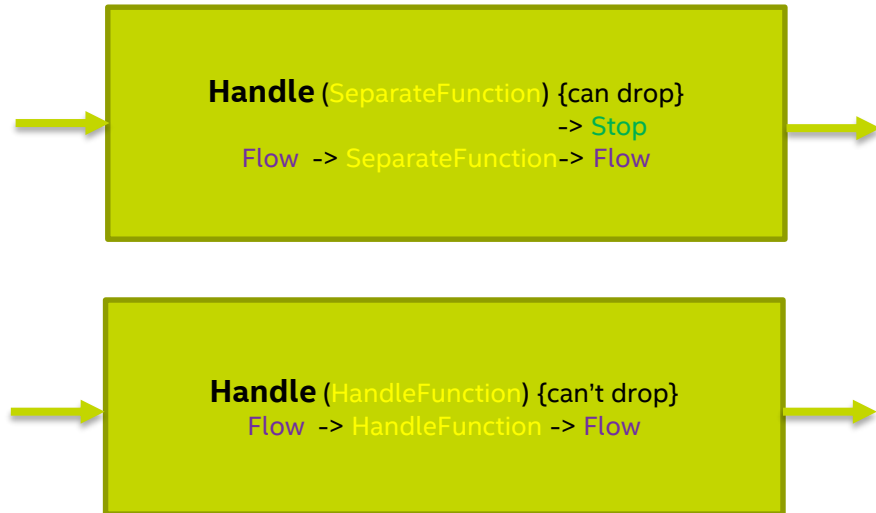
## Rule

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

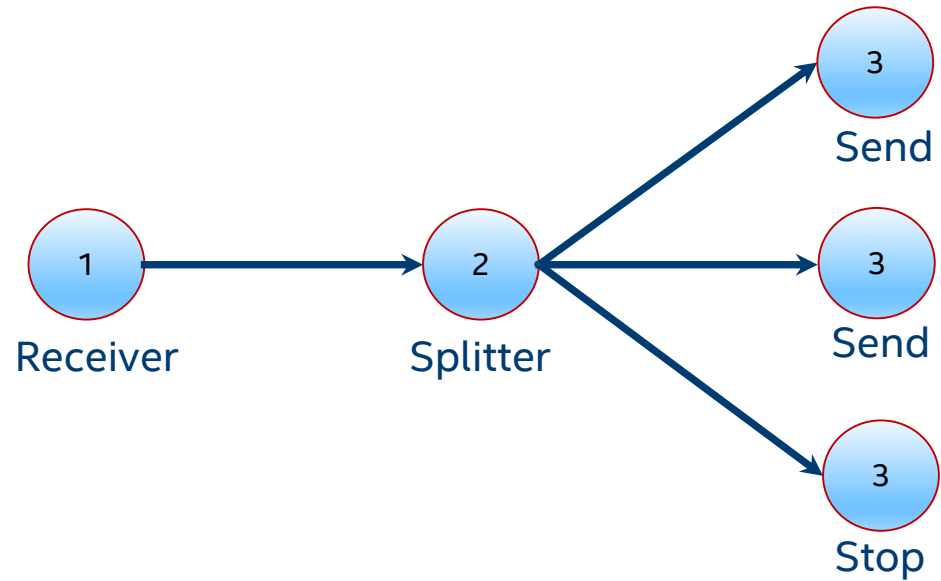
**Create rule**  
Create checking rule from json / config

**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  
nff-go-1$ bindports
```

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

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

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

# Let's try (01 of 11)

Flow graph:

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

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

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

.....

```
Pktgen:/> quit
```

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

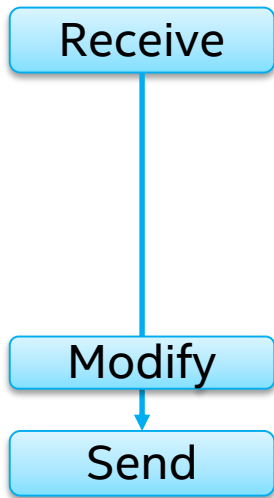
func main() {
    // Init NFF-Go system
    config := flow.Config{}
    checkFatal(flow.SystemInit(&config))

    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/nff-go/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))
```

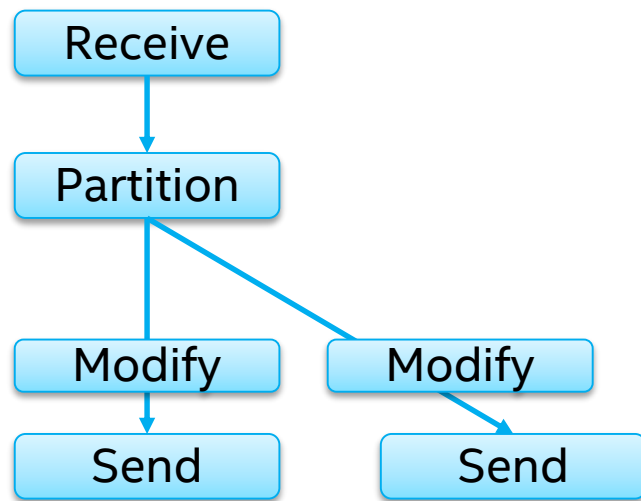
```
    checkFatal(flow.SystemStart())
```

```
}
```



# 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
```

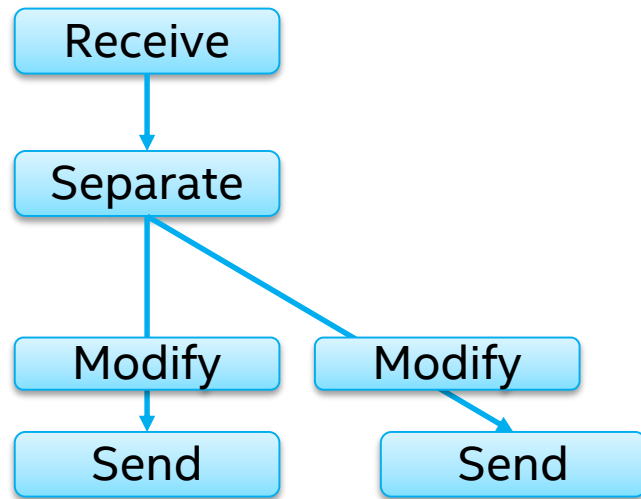
...

```
Pktgen: /> quit
```

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

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

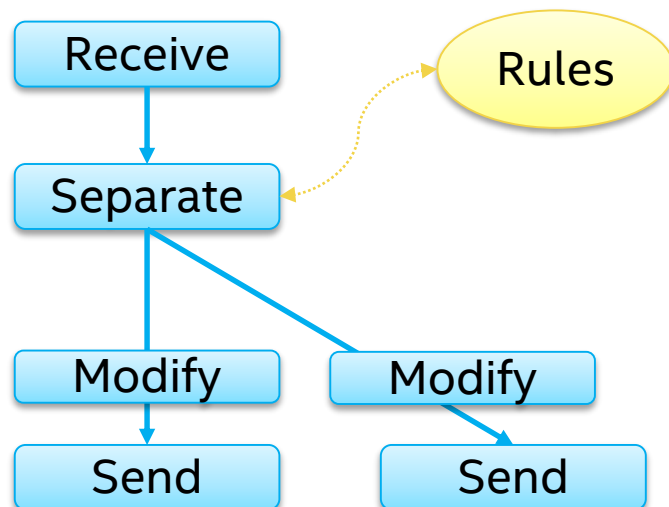
    firstFlow, err := flow.SetReceiver(0)
    checkFatal(err)
    secondFlow, err := flow.SetSeparator(firstFlow, mySeparator, nil)
    checkFatal(err)
    checkFatal(flow.SetHandler(firstFlow, modifyPacket[0], nil))
    checkFatal(flow.SetHandler(secondFlow, modifyPacket[1], nil))
    checkFatal(flow.SetSender(firstFlow,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

```
...
```

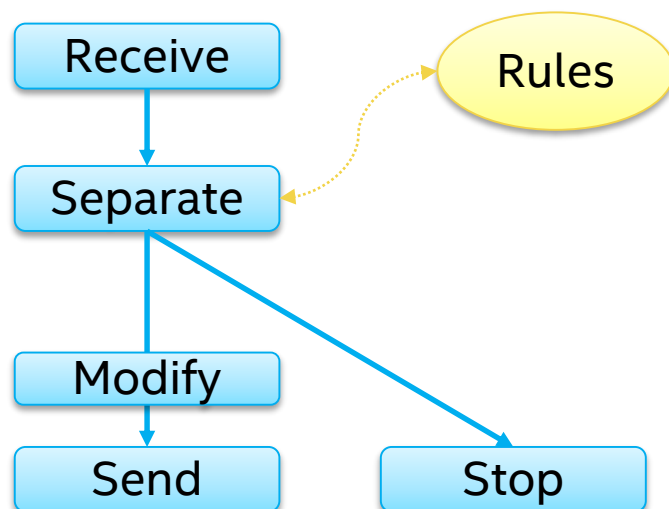
```
Pktgen:/> quit
```

```
... ..  
import "github.com/intel-go/nff-go/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
```

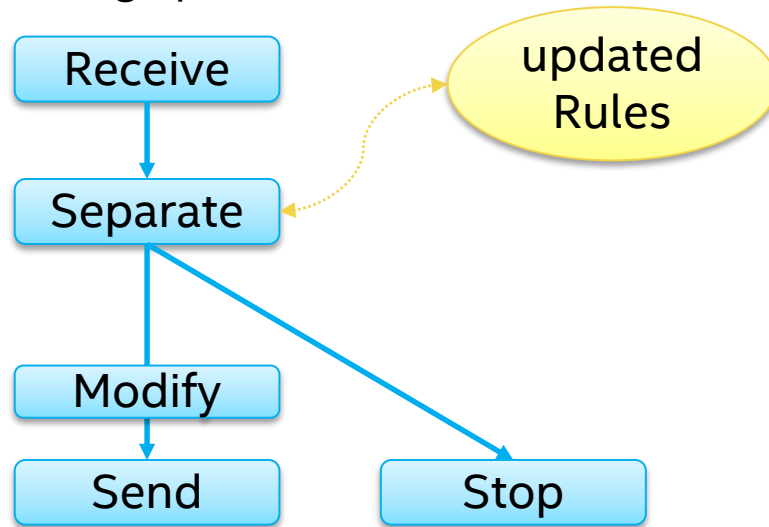
...

```
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()
... ..

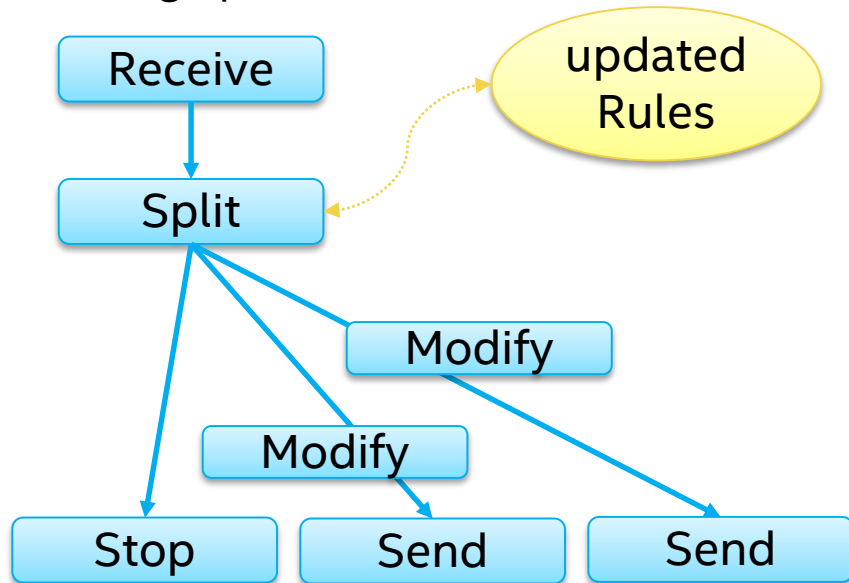
func MySeparator(cur *packet.Packet, ctx flow.UserContext) bool {
    localL3Rules := (*packet.L3Rules)(atomic.LoadPointer(&rulesp))
    return cur.L3ACLPermit(localL3Rules)
}

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 NFF-Go 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
```

```
...
```

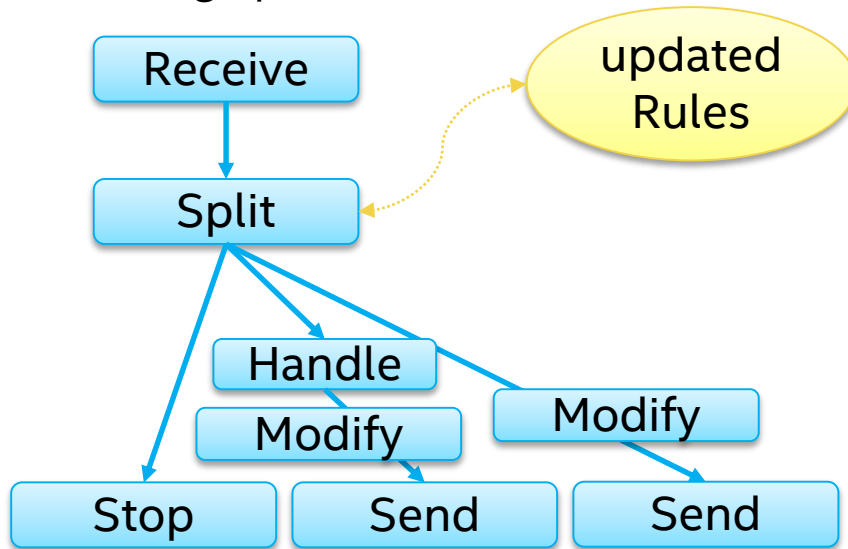
```
Pktgen:/> quit
```

```
... ..  
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 NFF-Go 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
```

```
...
```

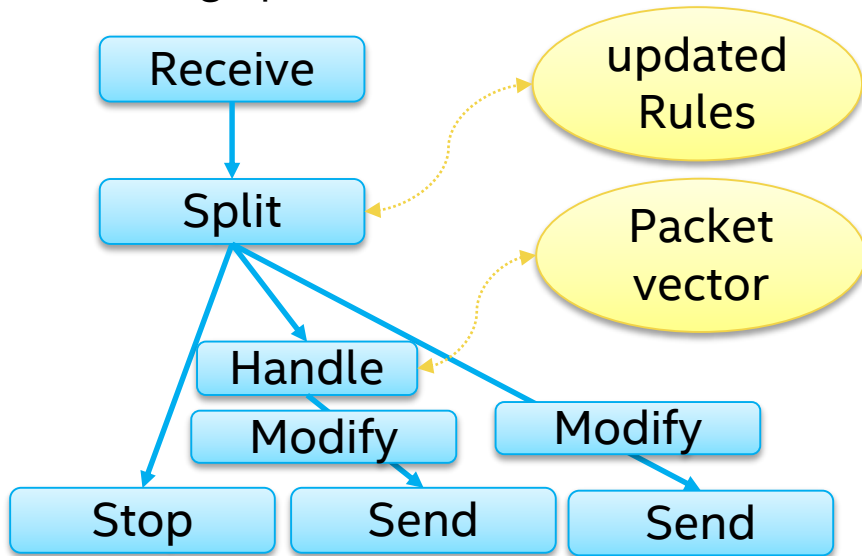
```
Pktgen:/> quit
```

```
... ..
import "github.com/intel-go/nff-go/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 NFF-Go 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
```

...

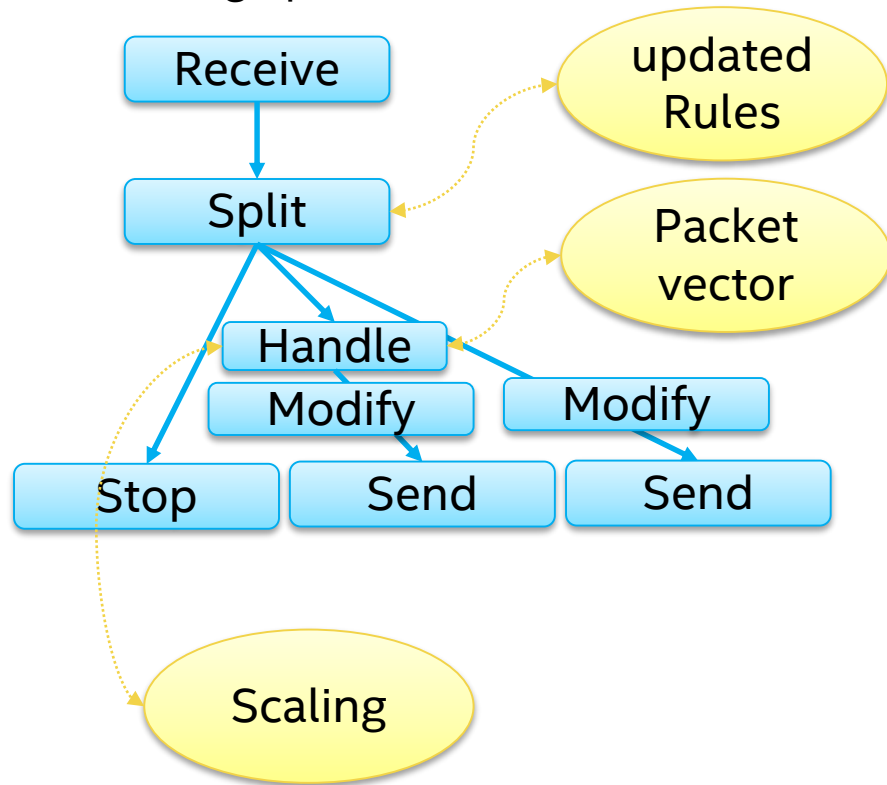
```
Pktgen:/> quit
```

*To make changes in rules2.conf file it is necessary to connect to target VM in another window or run NFF-Go 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 NFF-Go 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
```

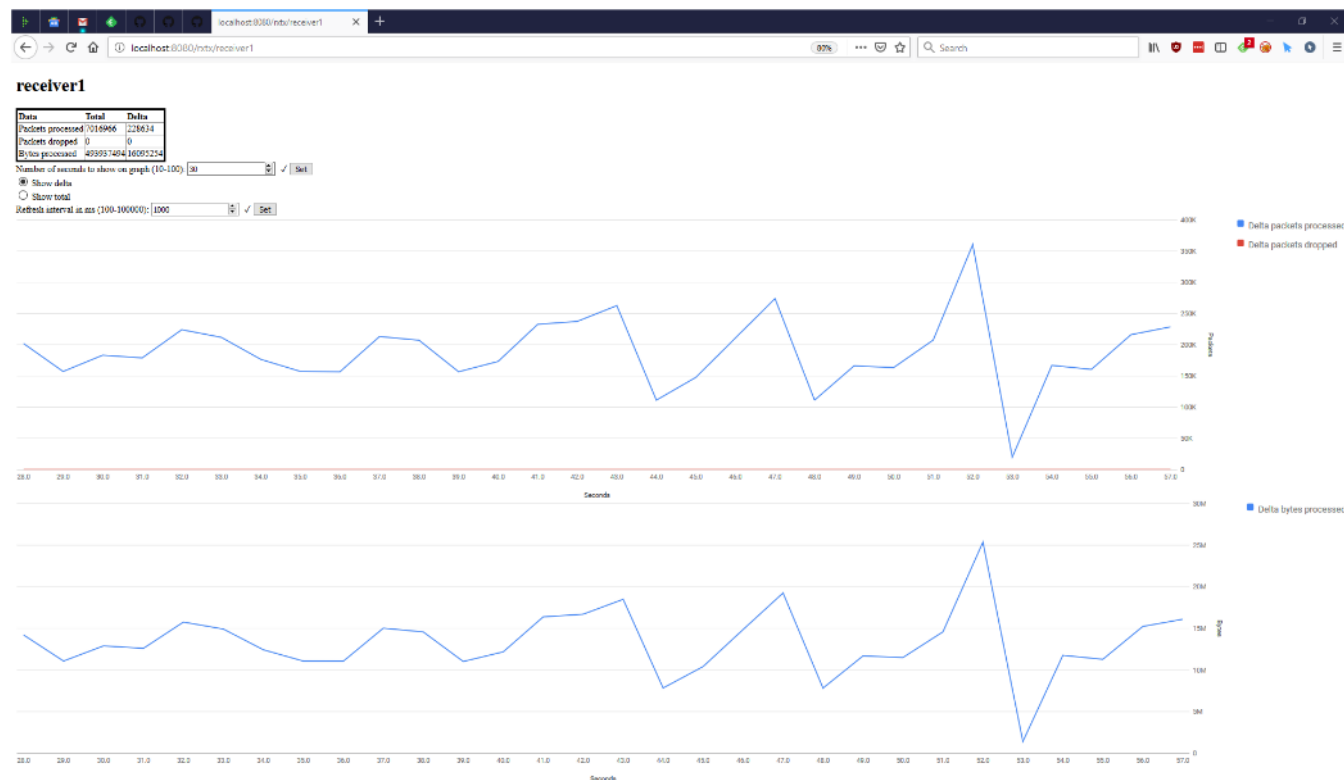
...

```
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 ?

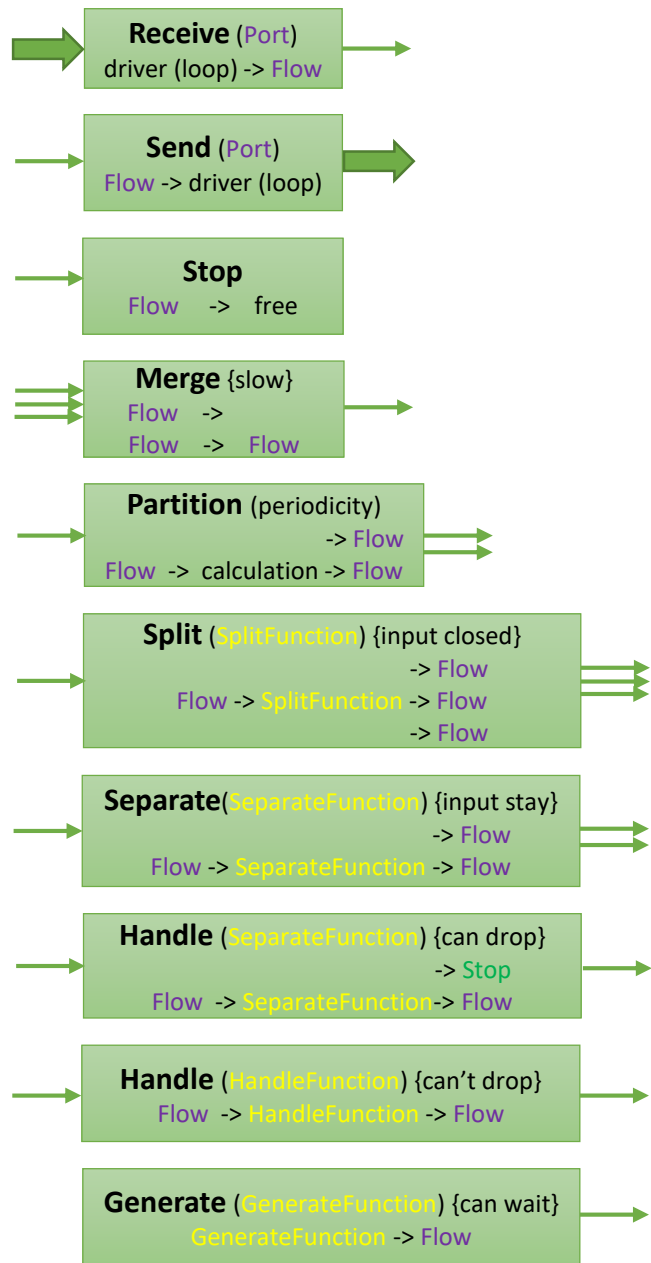
# Optimization Notice

## 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.

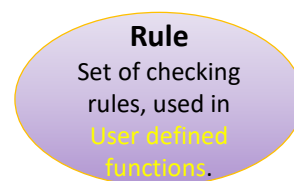
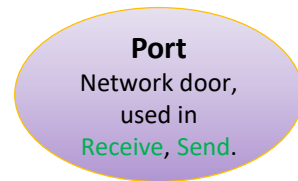
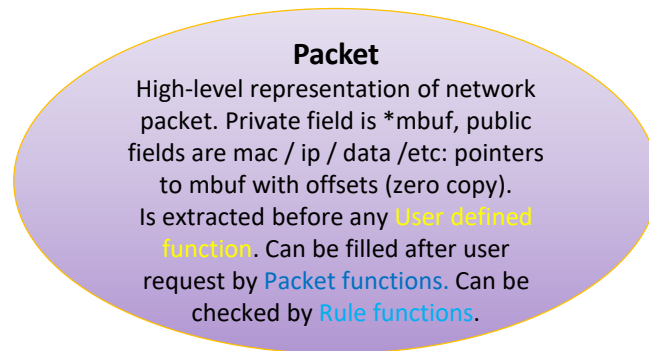
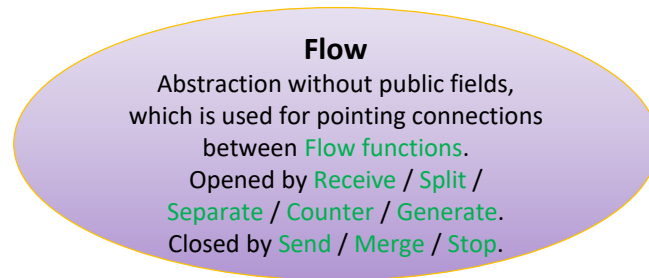
Notice revision #20110804

## Flow functions

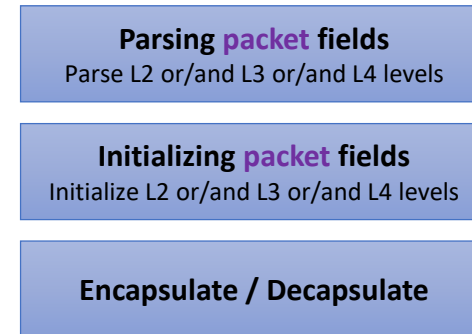


# Basic components

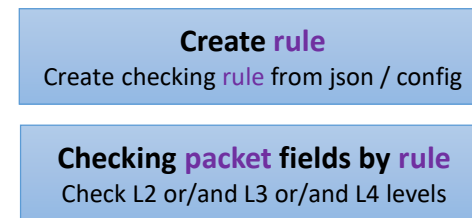
## Instances (new types)



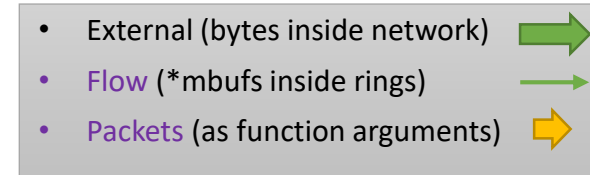
## Packet functions



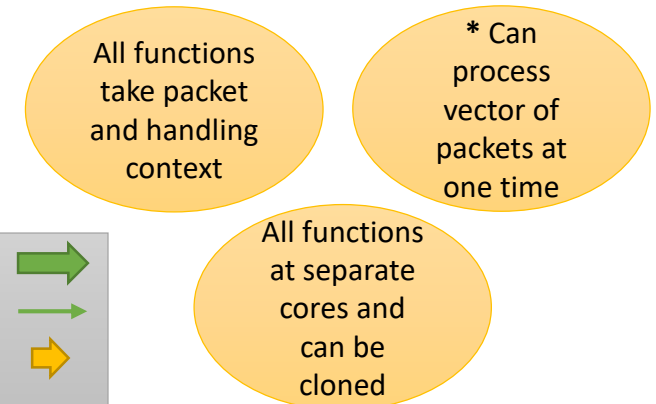
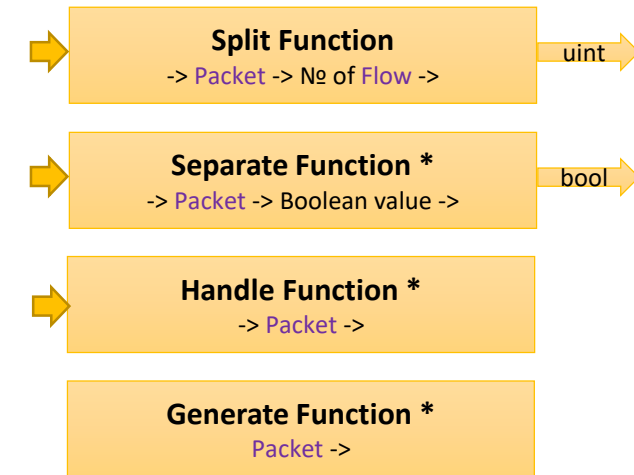
## Rule functions



## Connections



## User defined functions



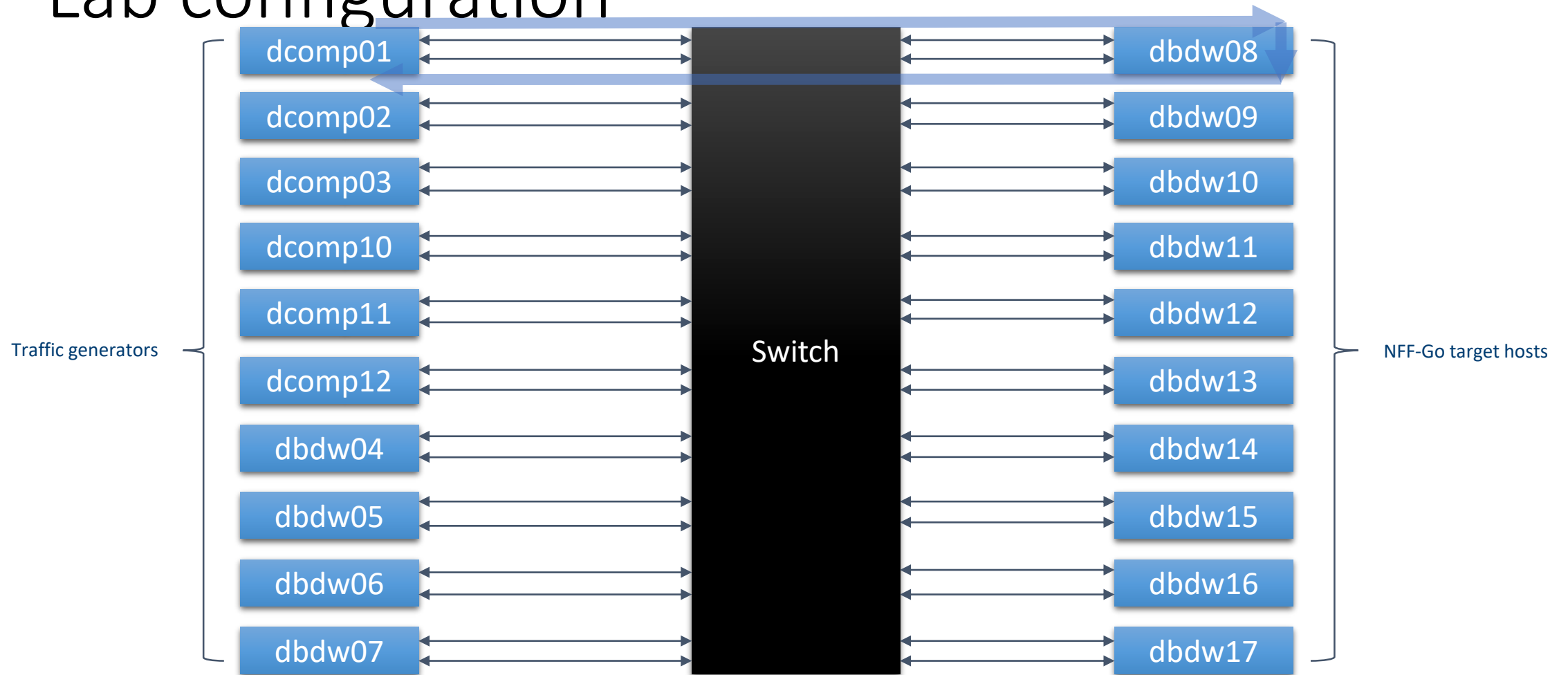
## Library External Components

- 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