# VNB

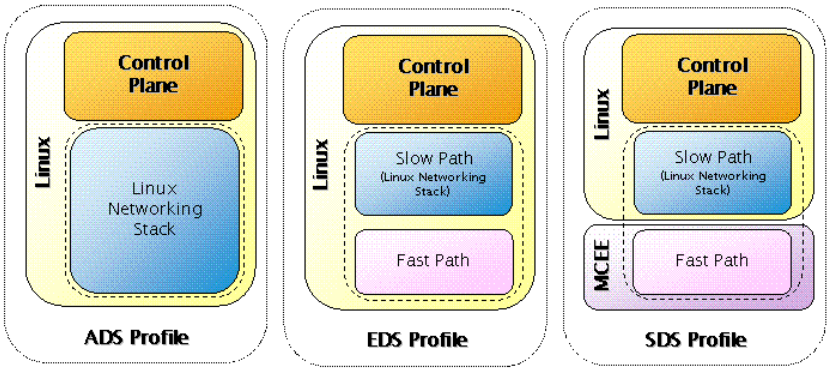
Kevin Gao

October 15, 2013

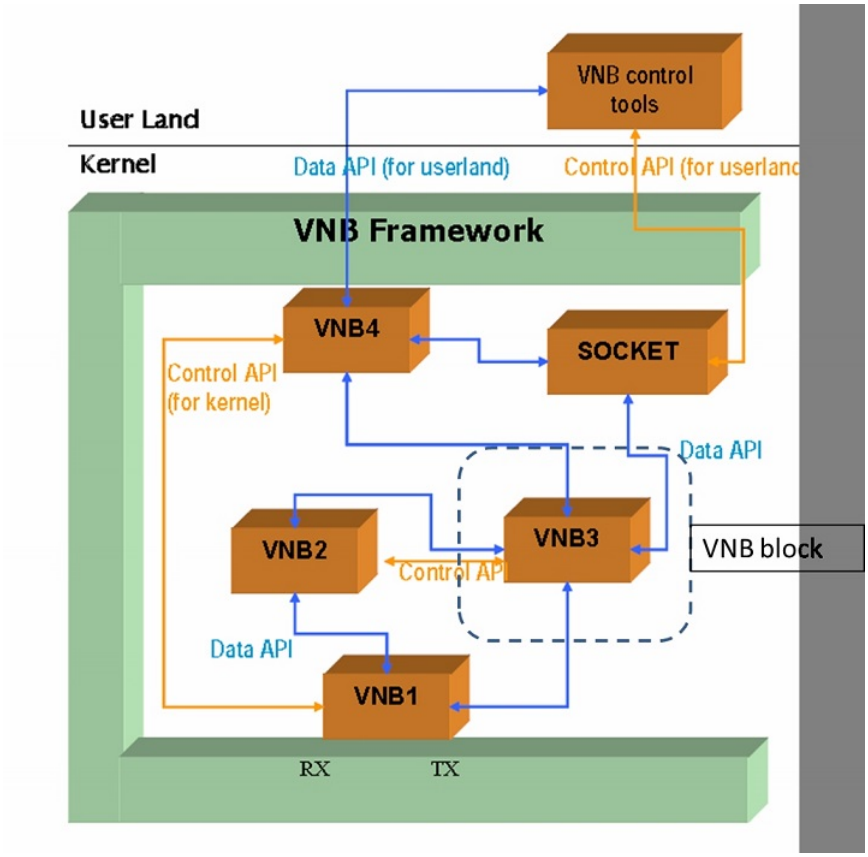
## Contents

# 1 Overview

**Virtual Networking Blocks** technology (VNB) is an enhancement of Linux Networking stack to make kernel software development modular and extensible. VNB concept has also been extended by 6WIND to Fast Path to bring similar benefits and architecture for Fast Path-less 6WIND solutions (6WINDGate ADS) and Fast Path-based solutions (6WINDGate EDS and 6WINDGate SDS);



## 1.1 Content




A VNB Block is made by a node, which is a processing unit and one or more hooks, which are interconnection units. Data packets flow bi-directionally along hooks from node to node. When a node receives a data packet, it performs some processing on it, and then (usually) forwards it to another node. The processing may be something as simple as adding/removing headers, or it may be more complicated or involve other parts of the system such as userland control modules.

## 1.2 Entity in our equipment

### 1.2.1 Slow path

- Two kernel modules

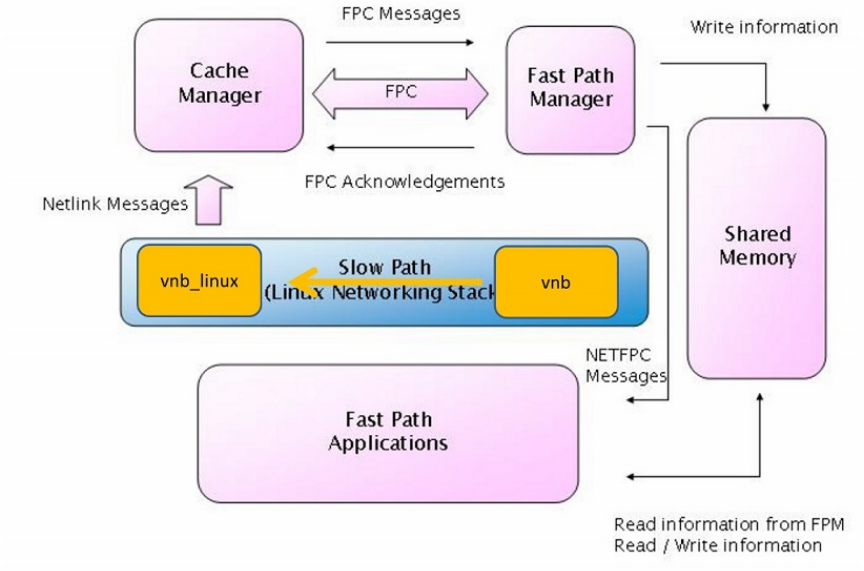
```
[root@EIPU-0(BCN_151) /root]
# lsmod |grep vnb
vnb_linux              3475  1 vnb
vnb                  466358  4
[root@EIPU-0(BCN_151) /root]
# modinfo vnb
filename:       /lib/modules/2.6.34.9-WR4.3.fp_octwnd_6880_standard-202183-g3165a46/drivers/net/vnb.ko
license:        6WIND
description:    VNB for Linux
author:         JMG
depends:        vnb-linux
vermagic:       2.6.34.9-WR4.3.fp_octwnd_6880_standard-202183-g3165a46 SMP preempt mod_unload OCTEON 64BIT
parm:           ng_vlan_afterconnect_enable:Enable vlan afterconnect method (bool)
[root@EIPU-0(BCN_151) /root]
# modinfo vnb_linux
filename:       /lib/modules/2.6.34.9-WR4.3.fp_octwnd_6880_standard-202183-g3165a46/drivers/net/vnb-linux.ko
license:        GPL
depends:
vermagic:       2.6.34.9-WR4.3.fp_octwnd_6880_standard-202183-g3165a46 SMP preempt mod_unload OCTEON 64BIT
```

  - the location of vnb_linux /SS_6Wind/src/6WINDGate/COMPONENTS/np/vnb/sys/netgr
  - the location of vnb

/SS_6Wind/src/6WINDGate/COMPONENTS/np/vnb/sys/netgraph

### 1.2.2 Fast path

There are only flow handling mechanism in fast path, all the Nodes and hooks are created and synchronized from slow path, sync path is



# 2 Node and hook

Node creation flow: take ng_ether as example: ng_ether_attach–>ng_make_node_common –>malloc priviate date –>registe node private date the node name and interface name is the same

## 2.1 Node

```
/*
 * Structure of a node
 */
struct ng_node {
    char   *name;       /* optional globally unique name */
    struct  ng_type *type;  /* the installed 'type' */
    int flags;      /* see below for bit definitions */
    vnb_atomic_t    refs;   /* number of references to this node */
    u_int   numhooks;   /* number of hooks */
    int colour;     /* for graph colouring algorithms */
    void   *private;    /* node type specified information */
    ng_ID_t     ID; /* Unique per node */
    LIST_HEAD(hooks, ng_hook) hooks;    /* linked list of node hooks */
    LIST_ENTRY(ng_node)   nodes;    /* linked list of all nodes for linear search */
    LIST_ENTRY(ng_node)   namenodes;    /* linked list of all nodes for name hash */
    LIST_ENTRY(ng_node)   idnodes;  /* ID hash collision list */
};
typedef struct ng_node *node_p;
```

### 2.1.1 ng_type

The structure that contail type specificed informations include the name,eg: ether, and all method pointers

```
/*
 * Structure of a node type
 */
struct ng_type {

    u_int32_t   version;    /* must equal NG_VERSION */
    const char  *name;      /* Unique type name */
    void *  mod_event;  /* not used */
    ng_constructor_t *constructor;  /* Node constructor */
```

```
    ng_rcvmsg_t *rcvmsg;    /* control messages come here */
    ng_shutdown_t   *shutdown;  /* reset, and free resources */
    ng_newhook_t    *newhook;   /* first notification of new hook */
    ng_findhook_t   *findhook;  /* only if you have lots of hooks */
    ng_connect_t    *connect;   /* final notification of new hook */
    ng_connect_t    *afterconnect;  /* final notification of new hook */
    ng_rcvdata_t    *rcvdata;   /* date comes here */
    ng_rcvdata_t    *rcvdataq;  /* or here if being queued */
    ng_disconnect_t *disconnect;    /* notify on disconnect */
    ng_rcvexception_t *rcvexception; /* exceptions come here */

    const struct    ng_cmdlist *cmdlist;    /* commands we can convert */

    /* R/W data private to the base netgraph code DON'T TOUCH! */
    LIST_ENTRY(ng_type) types;      /* linked list of all types */
    int         refs;       /* number of instances */
};
```

### 2.1.2 refs

An atomic operated number, that is used to idendify, whether some node are used by private date

### 2.1.3 private data

The node type specified information, all supported hook pointer is stored here.

```
/* Per-node private data */
struct private {
    struct ifnet    *ifp;       /* associated interface */
    hook_p      upper;      /* upper hook connection */
    hook_p      lower;      /* lower OR orphan hook connection */
    hook_p      lower_in[NG_ETHER_MAX_LOWER_IN];    /* lower input hooks */
    u_char      lowerOrphan;    /* whether lower is lower or orphan */
    u_char      autoSrcAddr;    /* always overwrite source address */
    uint8_t     promisc;    /* promiscuity counter */
#ifdef __FreeBSD__
    u_long      hwassist;   /* hardware checksum capabilities */
#endif
};
```

```
typedef struct private *priv_p;
```

## 2.2 Hooks

The process of create a hook for some node take ng_vlan as an example ng_vlan_newhook–>Get the private of the node –>Check different type of hook

```
/* Hook names */
#define NG_VLAN_HOOK_LOWER        "lower"    /* the lower hook */
#define NG_VLAN_HOOK_LINK_PREFIX  "link_"    /* append decimal integer */
#define NG_VLAN_HOOK_LINK_FMT     "link_%d"  /* for use with printf(3),
                                                %d is the tag           */
#define NG_VLAN_HOOK_NOMATCH      "nomatch"  /* the unknown packets */
#define NG_VLAN_HOOK_ORPHANS      "orphans"  /* the unknown tags */
```

–>allocate the hook name, eg:link_807 –>registe the per link private data

```
/* Store tag value in private zone of each hook */
struct ng_vlan_hook_private {
    uint16_t tag;
    uint8_t dscp_enable;
    uint8_t nfmark_enable;
    uint8_t dscp_to_priority[DSCP_MAX_SIZE];
    uint8_t nfmark_to_priority[NFMARK_MAX_SIZE];
    uint8_t pko_internal_port;
#define DEFAULT_VLAN_PKO_PORT   0xff
};
```

–>store the hook pointer into the node's private –>registe the receive data hook

The process of connect hook ng_connect

```
if (hook2->hook_rcvdata)
    hook1->peer_rcvdata = hook2->hook_rcvdata;
else
    hook1->peer_rcvdata = hook2->node->type->rcvdata;
```

### 2.2.1 Data structure

```
/*
```

```
 * Structure of a hook
 */
struct ng_meta;

struct ng_hook {
    struct  ng_hook *peer;  /* the other end of this link */
    int (*peer_rcvdata)(struct ng_hook *, struct mbuf *, struct ng_meta *);
    int (*hook_rcvdata)(struct ng_hook *, struct mbuf *, struct ng_meta *);
    void   *private;    /* node dependant ID for this hook */
#ifdef NG_NODE_CACHE
    void   *node_cache;     /* node dependant cache info for this hook */
#endif
    void   *node_private;   /* pointer to node private structure */
    int flags;      /* info about this hook/link */
    int refs;       /* dont actually free this till 0 */
    char   *name;       /* what this node knows this link as */
    struct  ng_node *node;  /* The node this hook is attached to */
    LIST_ENTRY(ng_hook) hooks;  /* linked list of all hooks on node */
    LIST_ENTRY(ng_hook) namehooks; /* linked list of hooks for name hash */
};
typedef struct ng_hook *hook_p;
```

# 3 Data flow

## 3.1 Data message

Data Messages are passed in mbuf chains along the edges in the VNB Graph, one edge at a time. Each VNB Node decides how to handle data coming in on its hooks. In fact, mostly, the hook decide how to handle the date, but exactly speaking, the hook is also part of the node.

```
hook->hook_rcvdata = ng_vlan_rcv_tag;
```

## 3.2 Control message

```
.rcvmsg =     ng_vlan_rcvmsg,           /* control messages come here */
```

## 3.3 Flow in graph

the message(both data and control) is deleverred by function call, but not queue or mailbox. eg: A wishes to send a data mbuf to neighboring VNB

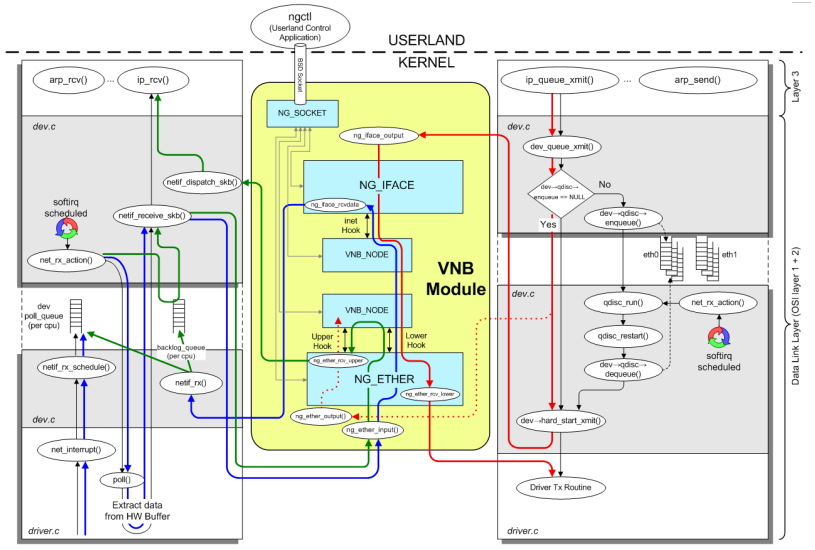
Node B, it calls the generic VNB data delivery function. This function in turn locates VNB Node B and calls B's receive data method Communication between VNB Nodes is based on the "send and forget" principle. Any VNB Node sending information on a hook from VNB Node A to VNB Node B is managed by node B only. This means:

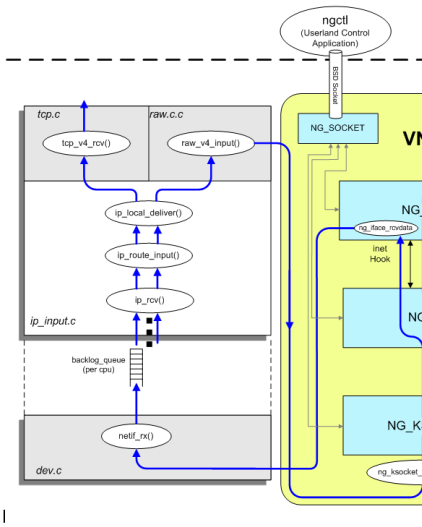
- In case of error, node B will free the mbuf and return a code error.
- In nominal case, node B will return 0.
- In both cases, node A should not use m reference anymore (granted by the use of

NG_SEND_DATA).

### 3.4 In our equipment

### 3.5 Slow path




### 3.6 Fastpath

Call trace

```
fpn_main_loop()-->fpn_process_input()-->fp_process_input()-->fp_ether_input()-->ng_ethe
```

```
eitp_udp_output()-->eitp_fp_ip_output()--> fp_ip_send_fragment() -->fp_ip_if_send()-->f
--> ng_ether_rcv_lower() ->fp_if_output()
```

### 3.7 Flags in ip link

#### 3.7.1 fpout

- whether goto fastpath with fptun

#### 3.7.2 infra

- Whether handled by nfhook.

## 4 Net graph configuration

### 4.1 CM is responsible for sync the configuration from slow-path to fastpath

#### 4.1.1 The configuation from kernel

The following devices are from kernel

```
1: lo: <LOOPBACK,UP,RUNNING,LOWER_UP> <INFRA> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    IPv4 forwarding: on IPv6 forwarding: off
    IPv4 force reassembly: no IPv6 force reassembly: no
2: mgmt0: <BROADCAST,MULTICAST,UP,RUNNING,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:d0:c9:c5:f5:06 brd ff:ff:ff:ff:ff:ff
    IPv4 forwarding: on IPv6 forwarding: off
    IPv4 force reassembly: no IPv6 force reassembly: no
3: tunl0: <NOARP> mtu 1480 ttl 0 tos 0x0 qdisc noop
    link/ipip 0.0.0.0 brd 0.0.0.0
    IPv4 forwarding: on IPv6 forwarding: on
    IPv4 force reassembly: no IPv6 force reassembly: no
4: svti_cfg@NONE: <POINTOPOINT,MULTICAST,NOARP> mtu 16416 ttl 0 tos 0x0 qdisc noop
    link/svti 0.0.0.0 peer 0.0.0.0
    IPv4 forwarding: on IPv6 forwarding: on
    IPv4 force reassembly: no IPv6 force reassembly: no
5: ip6tnl0: <NOARP> mtu 1460 ttl 0 tos 0x0 qdisc noop
    link/tunnel6 :: brd ::
```

```
    IPv4 forwarding: on IPv6 forwarding: on
    IPv4 force reassembly: no IPv6 force reassembly: no
6: xaui0: <BROADCAST,MULTICAST,PROMISC,UP,RUNNING,LOWER_UP> <INFRA,FP_OUTPUT> mtu 9000
    link/ether 00:d0:c9:c5:f5:07 brd ff:ff:ff:ff:ff:ff
    IPv4 forwarding: on IPv6 forwarding: off
    IPv4 force reassembly: no IPv6 force reassembly: no
7: xaui1: <BROADCAST,MULTICAST,PROMISC,UP,RUNNING,LOWER_UP> mtu 9000 qdisc noqueue
    link/ether 00:d0:c9:c5:f5:08 brd ff:ff:ff:ff:ff:ff
    IPv4 forwarding: on IPv6 forwarding: off
    IPv4 force reassembly: no IPv6 force reassembly: no
8: oct0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop
    link/ether 00:00:00:00:01:01 brd ff:ff:ff:ff:ff:ff
    IPv4 forwarding: on IPv6 forwarding: on
    IPv4 force reassembly: no IPv6 force reassembly: no
9: fpn0: <BROADCAST,MULTICAST,NOARP,UP,RUNNING,LOWER_UP> <INFRA> mtu 1500 qdisc noqueue
    link/ether 00:00:00:00:03:0c brd ff:ff:ff:ff:ff:ff
    IPv4 forwarding: on IPv6 forwarding: on
    IPv4 force reassembly: no IPv6 force reassembly: no
10: internal: <BROADCAST,MULTICAST,UP,RUNNING,LOWER_UP> <INFRA> mtu 9000 qdisc noqueue
    link/ether mode virtual 00:d0:c9:c5:f5:07 brd ff:ff:ff:ff:ff:ff
    IPv4 forwarding: off IPv6 forwarding: off
    IPv4 force reassembly: no IPv6 force reassembly: no
```

#### 4.1.2 Call tracing

```
-> register_netdev()
  -> register_netdevice()
     -> call_netdevice_notifiers() : to protocols
     -> rtmsg_ifinfo()  : to netlink
        -> rtnl_notify() : RTNLGRP_LINK
           -> nlmsg_notify()
              -> nlmsg_multicast()
                 -> netlink_broadcast()
                    -> do_one_broadcast()
                       -> netlink_broadcast_deliver()
                          -> __netlink_sendskb()
```

#### 4.1.3 The configuration from ngctl

The devices

```
10: internal: <BROADCAST,MULTICAST,UP,RUNNING,LOWER_UP> <INFRA> mtu 9000 qdisc noqueue
    link/ether mode virtual 00:d0:c9:c5:f5:07 brd ff:ff:ff:ff:ff:ff
    IPv4 forwarding: off IPv6 forwarding: off
    IPv4 force reassembly: no IPv6 force reassembly: no
11: ethtest20: <BROADCAST,MULTICAST,UP,RUNNING,LOWER_UP> <FP_OUTPUT> mtu 1500 qdisc noo
    link/ether mode virtual 00:d0:c9:c5:ac:ea brd ff:ff:ff:ff:ff:ff
    IPv4 forwarding: on IPv6 forwarding: off
    IPv4 force reassembly: no IPv6 force reassembly: no
12: rnchanet: <BROADCAST,MULTICAST,UP,RUNNING,LOWER_UP> <FP_OUTPUT> mtu 9000 qdisc noqu
    link/ether mode virtual 00:d0:c9:c5:f5:07 brd ff:ff:ff:ff:ff:ff
    IPv4 forwarding: off IPv6 forwarding: off
    IPv4 force reassembly: no IPv6 force reassembly: no
13: cfeigw: <BROADCAST,MULTICAST,UP,RUNNING,LOWER_UP> <FP_OUTPUT> mtu 9000 qdisc noqueu
    link/ether mode virtual 00:d0:c9:c5:f5:07 brd ff:ff:ff:ff:ff:ff
    IPv4 forwarding: on IPv6 forwarding: off
    IPv4 force reassembly: no IPv6 force reassembly: no
```

#### 4.1.4 call tracing

ngctlkernel ng_eiface_rcvmsg() ngcltmsgdeviceregister_netdevice() ngctl-register_netdevice()CM

```
main()
-> fpm_init()
  -> fpm_connect()
     -> cm_init()
        -> cm_netlink_cmd_init() : ngctl : NETLINK_ROUTE (Routing/device hook)
           -> cm_netcmd: cm_nl_recv()
               -> cm_nl_dispacth()
                  -> cm_nl_link()
                     -> cm_eth_create() # cm_iface_handler_lookup()
                        -> cm2cp_iface_create()
                           -> post_msg() : fpm_enqueue() : UNIX socket ---> FPM
       -> cm_netlink_route_init()  ==> kernel : NETLINK_ROUTE (Routing/device hook)
          -> cm_netlink: cm_nl_recv()
```

```
 1:  void cm_netlink_route_init(void)
 2:  {
 3:      /*
 4:       * We want to receive
 5:       *    iface info
```

```
 6:       *    addresses
 7:       *    routes
 8:       * This needs to be updated with additional
 9:       * netlink resources
10:       */
11:      cm_netlink_sock (NETLINK_ROUTE,
12:              &cm_netlink,
13:              (RTMGRP_LINK | RTMGRP_NEIGH | RTMGRP_NOTIFY |
14:               RTMGRP_IPV4_ROUTE | RTMGRP_IPV6_ROUTE |
15:               RTMGRP_IPV4_IFADDR | RTMGRP_IPV6_IFADDR
16:  #ifdef RTNLGRP_VNB
17:               | RTMGRP_VNB
18:  #endif
19:               ), 1, CM_BULK_READ);
20:  }
21:
22:       _PF(CMD_RESET)
23:         _PF(CMD_FLUSH)
24:         _PF(CMD_IF_CREATE)
25:         _PF(CMD_IF_DELETE)
26:         _PF(CMD_IF_MTU)
27:         _PF(CMD_IF_STATE_UPDATE)
28:         _PF(CMD_IF_MAC)
29:         _PF(CMD_IF_BLADEINFO)
30:         _PF(CMD_IF_ADDR)
31:         _PF(CMD_IF_VRF_UPDATE)
32:         _PF(CMD_INTERFACE_IPV4_ADDR_ADD)
33:         _PF(CMD_INTERFACE_IPV4_ADDR_DEL)
34:         _PF(CMD_INTERFACE_IPV6_ADDR_ADD)
35:         _PF(CMD_INTERFACE_IPV6_ADDR_DEL)
36:         _PF(CMD_TUN_TTL)
37:         _PF(CMD_TUN_TOS)
38:         _PF(CMD_ROUTE4_ADD)
39:         _PF(CMD_ROUTE4_DEL)
40:         _PF(CMD_ROUTE6_ADD)
41:         _PF(CMD_ROUTE6_DEL)
42:         _PF(CMD_ARP_UPDATE)
43:         _PF(CMD_NDP_UPDATE)
44:         _PF(CMD_XIN4_CREATE)
45:         _PF(CMD_XIN4_DELETE)
```

```
46:         _PF(CMD_XIN4_UPDATE)
47:         _PF(CMD_XIN6_CREATE)
48:         _PF(CMD_XIN6_DELETE)
49:         _PF(CMD_XIN6_UPDATE)
50:         _PF(CMD_ISATAP_CREATE)
51:         _PF(CMD_ISATAP_DELETE)
52:         _PF(CMD_6TO4_START)
53:         _PF(CMD_6TO4_STOP)
54:         _PF(CMD_NATPT_ADD_PFX)
55:         _PF(CMD_NATPT_DEL_PFX)
56:         _PF(CMD_NATPT_ADD_ADDR)
57:         _PF(CMD_NATPT_DEL_ADDR)
58:         _PF(CMD_NATPT_ADD_SESSION)
59:         _PF(CMD_NATPT_DEL_SESSION)
60:         _PF(CMD_MCAST_START)
61:         _PF(CMD_MCAST_STOP)
62:         _PF(CMD_MCAST_ADD_VIF)
63:         _PF(CMD_MCAST_DEL_VIF)
64:         _PF(CMD_MCAST_ADD_MFC)
65:         _PF(CMD_MCAST_DEL_MFC)
66:         _PF(CMD_MCAST_JOIN_GROUP)
67:         _PF(CMD_MCAST_LEAVE_GROUP)
68:         _PF(CMD_NF_DISABLE)
69:         _PF(CMD_NF_UPDATE)
70:         _PF(CMD_NF_CTADD)
71:         _PF(CMD_NF_CTDELETE)
72:         _PF(CMD_NF_CTFLUSH)
73:         _PF(CMD_NF6_UPDATE)
74:         _PF(CMD_NF6_CTADD)
75:         _PF(CMD_NF6_CTDELETE)
76:         _PF(CMD_NF_CTDISABLE)
77:         _PF(CMD_IPSEC_SA_CREATE)
78:         _PF(CMD_IPSEC_SA_DELETE)
79:         _PF(CMD_IPSEC_SA_FLUSH)
80:         _PF(CMD_IPSEC_SA_REPLAYWIN)
81:         _PF(CMD_IPSEC_SP_CREATE)
82:         _PF(CMD_IPSEC_SP_DELETE)
83:         _PF(CMD_IPSEC_SP_FLUSH)
84:         _PF(CMD_PORT_CREATE)
85:         _PF(CMD_PORT_DELETE)
```

```
86:         _PF(CMD_PORT_STATE)
87:         _PF(CMD_SWITCH_STATE)
88:         _PF(CMD_SWITCH_BIND)
89:         _PF(CMD_SWITCH_UNBIND)
90:         _PF(CMD_SWITCHPORT_STATE)
91:         _PF(CMD_ADD_SNOOPING_ENTRY)
92:         _PF(CMD_DEL_SNOOPING_ENTRY)
93:         _PF(CMD_VNB_MSGHDR)
94:         _PF(CMD_BLADE_CREATE)
95:         _PF(CMD_BLADE_DELETE)
96:         _PF(CMD_BLADE_FPIB_IF_SET)
```

#### 4.1.5 About the vlan handling

fptunvlanslowpath fptunfp slowpathVNB

# 5 One testing result

## 5.1 TCP packet testing result

In two different box BCN513 _FPT_1.40.1.13

| CSPU | sample 1 | sample 2 | sample 3 | sample 4 | average |
|---|---|---|---|---|---|
| rnchanet | 1.52 | 1.55 | 1.52 | 1.53 | 1.53 |
| xaui0 | 2.55 | 2.84 | 2.48 | 2.78 | 2.6625 |
| EIPU | | | | | |
| rnchanet+fpout | 0.762 | 0.784 | 0.756 | 0.825 | 0.78175 |
| rnchanet+nofpout | 0.735 | 0.683 | 0.679 | 0.76 | 0.71425 |
| rnchanet+xaui0+nofpout | 0.732 | 0.787 | 0.766 | 0.755 | 0.76 |
| xaui0+fpout | 0.784 | 0.786 | 0.816 | 0.788 | 0.7935 |
| xaui0+nofpout | 1.36 | 1.37 | 1.35 | 1.37 | 1.3625 |

In two box BCN82 FPT_2.3.1.0

| EIPU | sample 1 | sample 2 | sample 3 | sample 4 | average |
|---|---|---|---|---|---|
| rnchanet + fpout | 1.07 | 1.06 | 1.04 | 1.07 | 1.06 |
| rnchanet + nofpout | 1.36 | 1.37 | 1.35 | 1.36 | 1.36 |
| rnchanet + xaui0 +nofpout | 1.18 | 1.19 | 1.18 | 1.18 | 1.1825 |
| xaui0 + fpout | 0.82 | 0.818 | 0.821 | 0.817 | 0.819 |
| xaui0 + nofpout | 1.44 | 1.43 | 1.43 | 1.43 | 1.4325 |

in one box BCN75 FPT_2.3.1.0

| EIPU | sample 1 | sample 2 | sample 3 | sample 4 | average |
|---|---|---|---|---|---|
| rnchanet + fpout | 1.06 | 1.06 | 1.06 | 1.07 | 1.0625 |
| rnchanet + nofpout | 1.35 | 1.35 | 1.36 | 1.35 | 1.3525 |
| rnchanet + xaui0 + nofpout | 1.18 | 1.18 | 1.18 | 1.17 | 1.1775 |
| xaui0 + fpout | 0.818 | 0.820 | 0.819 | 0.818 | 0.81875 |
| xaui0 + nofpout | 1.45 | 1.44 | 1.44 | 1.44 | 1.4425 |

### 5.2 bottle neck

| EIPU | fpout | nofpout |
|---|---|---|
| rnchanet | tx | tx |
| xaui0 | tx | rx |
| CSPU | | |
| rnchanet | rx | |
| xaui0 | rx | |

1. VNB rx bottleneck is 1.3G
2. VNB tx bottleneck is 0.78M

## 6 Code tracing

### 6.1 TX direction

#### 6.1.1 VLAN interface with fpout flag

- Code tracing

```
dev_queue_xmit(dev=vlan interface, rnchanet with fpout flag)
-->dev_fp_output=rfpvi_xmit(buid fptun packet)
-->dev_queue_xmit(dev=fpn0, without fpout flag)
-->dev_hard_start_xmit
-->cvmx_oct_xaui_pow_netdev_ops(always_use_pow == 1)
-->cvmx_pow_xmit-->cvmx_pow_work_submit
```

### 6.1.2 VLAN interface without fpout flag

```
dev_queue_xmit(dev=vlan interface, rnchanet without fpout flag)
-->dev_hard_start_xmit
-->ndo_start_xmit(for vlan interface, ndo_start_xmit=ng_eiface_start_xmit)
-->ng_eiface_start_xmit
-->
```

## 6.2 RX direction