6WINDGate™
-
Architecture Overview
-
v2.0

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

1.1 PURPOSE OF THE DOCUMENT

This document provides an overview of the 6WINDGate 5 software architecture.

- Section 2 gives a general reminder of network equipment architecture.
- Section 3 provides an overview of the 6WINDGate software architecture.
- Section 4 details the 6WINDGate high performance Data Plane, known as the Fast Path.
- Section 5 explains the synchronization between the Control Plane and the Fast Path.
- Section 6 describes the 6WINDGate Control Plane.
- Section 7 presents the 6WINDGate Management Plane.
- Section 8 introduces the 6WINDGate High Availability modules.
- Section 9 describes how 6WINDGate can be used in different environments.
- Section 10 summarizes the 6WINDGate main components and APIs.

1.2 REFERENCE DOCUMENTS

[1] 6WINDGate Modules – Data Sheets

[2] 6WINDGate supported RFCs

2 GENERAL CONCEPTS OF NETWORK EQUIPMENT ARCHITECTURE

2.1 OVERVIEW

The main concepts of an efficient architecture were defined some years ago, as part of the design of high-speed routers needed to address the explosion of Internet traffic. Now, this architecture has been extended to new services such as Layer 2 protocols, security, mobility, multicast, etc.

IP-based equipment can be partitioned into three basic elements: Data Plane, Control Plane and Management Plane (cf. Figure 1).

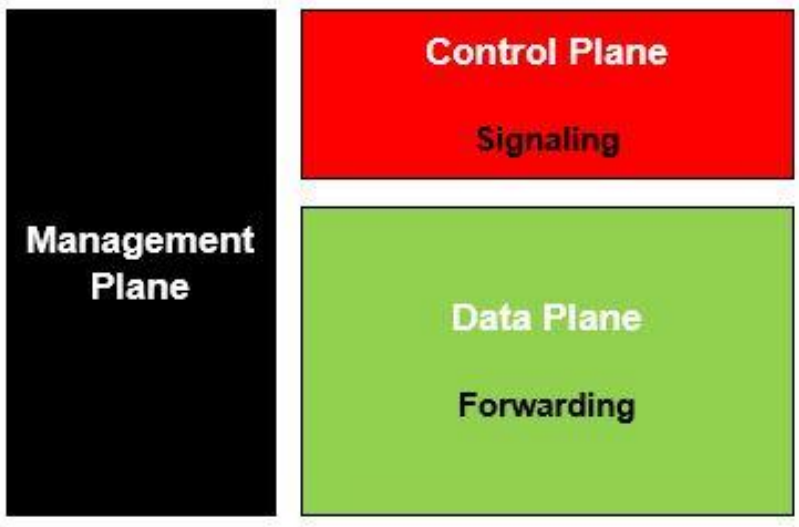

Figure 1: Networking equipment architecture

2.2 DATA PLANE

The Data Plane is a subsystem of a network node that receives and sends packets from an interface, processes them as required by the applicable protocol, and delivers, drops, or forwards them as appropriate. The Data Plane offloads packet forwarding from higher-level processors for most or all of the packets it receives that are not addressed for delivery to the node itself, it performs all the required processing.

For routing functions, it consists of a set of procedures (algorithms) that a router uses to make a forwarding decision on a packet. The algorithms analyze the information from a received packet to find an entry in the forwarding table.

Similarly, for IPsec functions, a security gateway checks if a Security Association is valid for an incoming flow and if so, the Data Plane finds locally the information that is required to process the packet.

2.3 CONTROL PLANE

The Control Plane maintains information that is used to by the Data Plane for packet processing. Maintaining this information requires handling complex signaling protocols. Implementing these protocols in the Data Plane would lead to poor forwarding performance. A common way to manage these protocols is to let the Data Plane detect incoming signaling packets and locally forward them to the Control Plane. The Control Plane signaling protocols can update the Data Plane information and inject outgoing signaling packets into the Data Plane. This architecture works with good performance as signaling traffic is a negligible part of the global traffic.

For routing functions, the Control Plane consists of one or more routing protocols that implement the exchange of routing information between routers, as well as the algorithms that a router uses to convert this information into the forwarding table. RIP, OSPF and BGP are examples of such routing protocols. In the case of virtual routing, multiple instances of forwarding tables are managed at the Data Plane level. As soon as the Data Plane detects a routing packet, it forwards it to the Control Plane so that the routing protocol can compute new routes, add routes or delete routes. Forwarding tables are updated with this new information. When a routing protocol has to send a packet (OSPF Hello packet for instance), it is injected into the Data Plane to be sent in the outgoing flow.

For IPsec security functions, signaling protocols for key exchange management such as IKE or IKEv2 are in the Control Plane. Incoming IKE packets are locally forwarded to the Control Plane. When an IKE negotiation completes, Security Associations and Policies located in the Data Plane are updated by the Control Plane. Outgoing IKE packets are injected into the Data Plane to be sent into the outgoing flow.

2.4 MANAGEMENT PLANE

The Management Plane provides an administrative interface into the overall system. It contains processes that support operational administration, management or configuration/provisioning actions such as:

- Facilities for supporting statistics collection and aggregation,
- Support for the implementation of management protocols,
- Command Line Interface, Graphical User Configuration Interfaces through Web pages or traditional SNMP Management. More sophisticated solutions based on XML can also be included.

2.5 RUNNING ON COMMERCIAL OFF-THE-SHELF HARDWARE AND SOFTWARE

The last concept of modern network equipment is commoditization. The power of the latest standard CPUs from Intel or Arm, the bandwidth of modern PCI NICs (up to 100G) and the versatility and feature-richness of the Linux Networking Stack make it possible to build high performance network equipment from Commercial Off-The-Shelf (COTS) hardware and software, provided that you have the right networking stack to overcome Linux performance limitations. Let's detail this in the next section.

3 6WINDGATE OVERVIEW

3.1 ARCHITECTURE

6WINDGate implements a networking architecture as explained in the previous section. This section gives an overview of the different components of the architecture, which are then described in detail in the following sections.

Even with an efficient implementation of the Linux Networking Stack in SMP mode, a Linux-based solution cannot scale because all the packets are managed by the stack and the Linux architecture is limited by OS latency and lock contention. SMP architectures are not able to fully benefit from the capabilities of many cores.

To go beyond this limitation, the software architecture should implement a Fast Path. CPU cores are dispatched between Linux OS and Control Plane application processing on one side and Fast Path Data Plane processing on the other side.

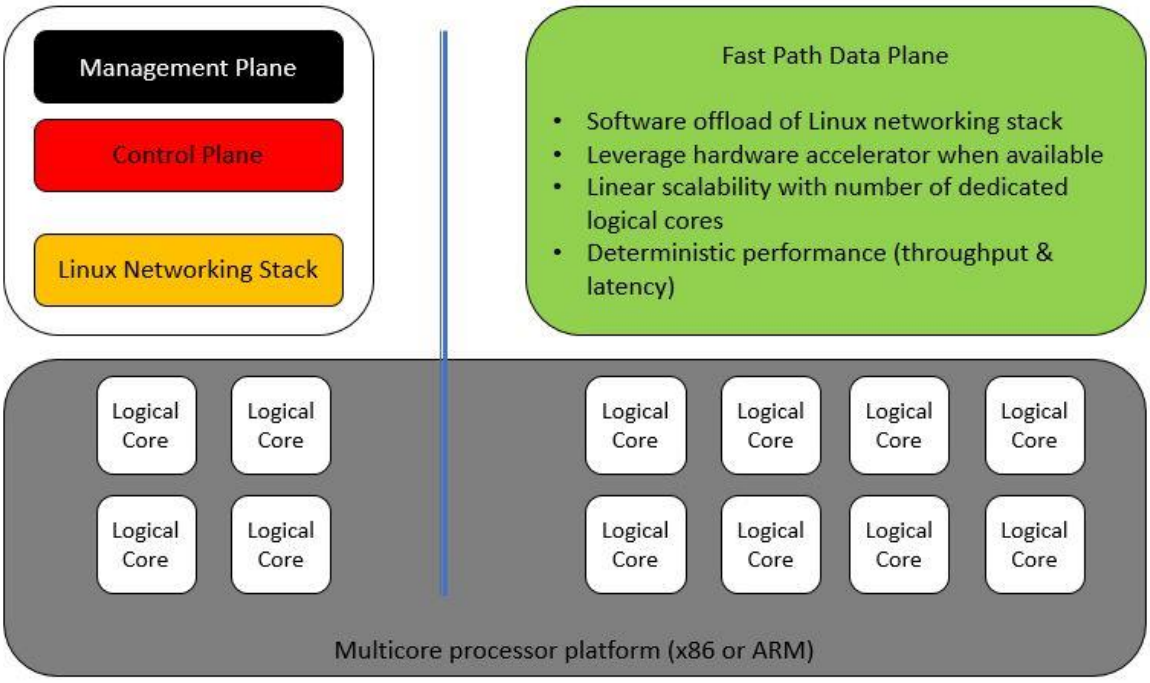

Figure 2: Fast Path Data Plane isolation

Each protocol has to be redesigned to implement time-consuming and recurrent tasks at the Fast Path level while only complex packets are forwarded to the networking stack. The 6WINDGate Data Plane concepts are described in detail in section 4.

Coherency of the system is ensured by the exception strategy and by the continuous synchronization between Linux and the Fast Path, as illustrated in Figure 3.

The goal of this architecture is to provide an offload of the Linux Networking Stack, as transparent as possible to the Control and Management Planes. The Fast Path processes most of the Data Plane packets without involving the underlying Operating System. The Fast Path could be compared to a hardware data plane offload technology, without the limited scalability and programming complexity of an ASIC or FPGA. The Fast Path offers excellent performance and scalability, with all the comfort and flexibility of software.

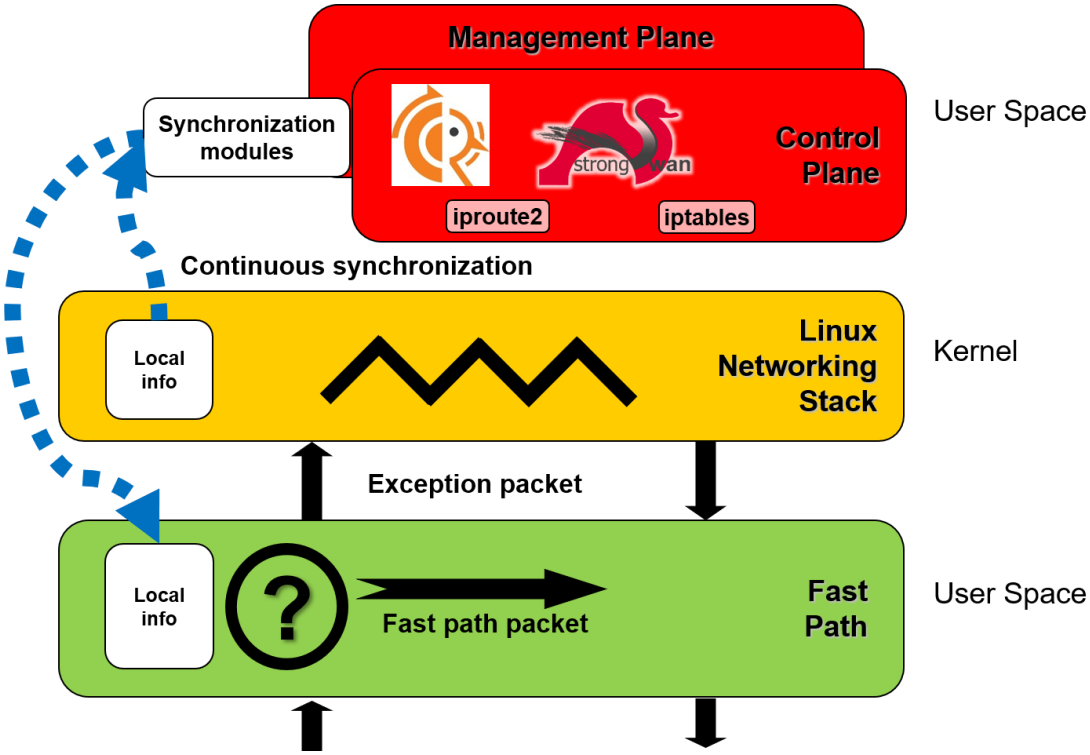

Figure 3: Fast Path-based architecture

The following processing is applied to any incoming packet in the system:

- The packet is received by the Fast Path and processed according to local information if present. The Fast Path implements lock-free packet processing.
- When the Fast Path local information does not allow a received packet to be processed (because it is intended at a Control Plane protocol, or the local information has not been updated yet, or the protocol is not supported in the Fast Path), it is sent as an exception packet to the networking stack, where it is handled using the standard Linux processing. It should be noted that exception packets are only a few percent of the overall traffic, so that there would be no benefit in having a full and complex IP stack at the Fast Path level.
- During standard Linux processing, information learned in the networking stack (ARP entries, L3 routes, IPsec security associations, etc.) is automatically and transparently synchronized to the Fast Path using the synchronization module, so that the next packet from the same flow can be processed by the Fast Path.

The combination of the exception strategy and continuous synchronization mechanism allows the system to process any kind of packet (even those that are not supported or not yet configured in the Fast Path) and transparently update local information in the Fast Path. The result is that almost all packets are finally processed by the Fast Path and only a small minority of packets go to the operating system. The 6WINDGate Linux synchronization concepts are described in section 5.

3.2 MODULES

The 6WINDGate software is modular. You can select the family of modules you need and, within each family, the modules your application require.

Here are the different families of 6WINDGate modules:

- Processor SDK (DPDK)
- FPN-SDK
- Fast Path
- Linux / Fast Path Synchronization
- Control Plane
- Management
- High Availability

The exhaustive list of 6WINDGate modules is represented in Figure 4.

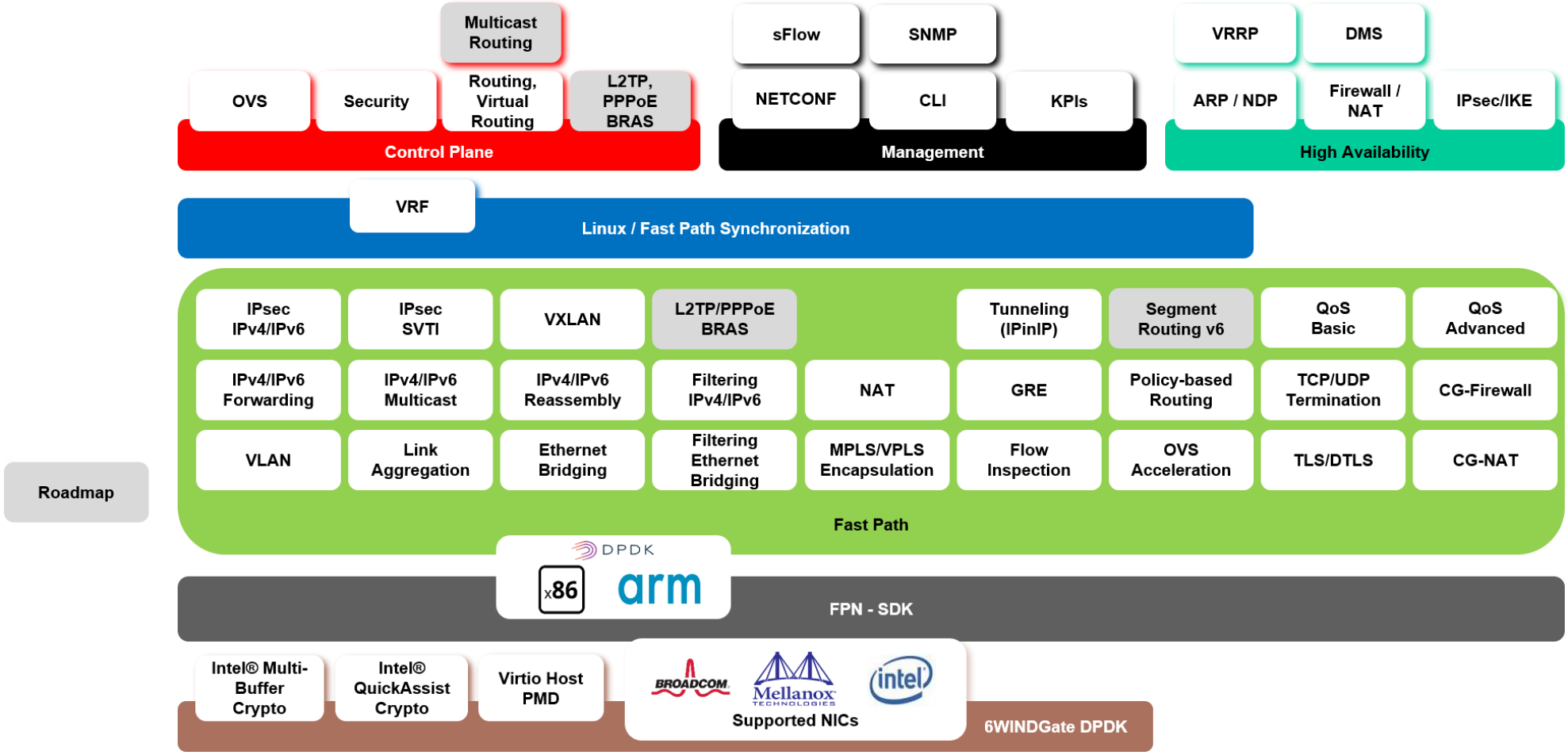

Figure 4: 6WINDGate modules

In the next sections, we introduce the main families and provide details about their architecture.

A detailed description of the different modules is provided in Module Data Sheets that are available from 6WIND on request. Each data sheet includes the following information:

- Module name
- Module features
- Management interfaces
- APIs
- 6WINDGate-related modules
- Implementation details
- Supported RFCs
- Supported hardware platforms

4 6WINDGATE DATA PLANE

The Fast Path is the 6WINDGate's high performance networking stack. It is responsible for processing all packets in the system.

4.1 PROCESSOR SDK (DPDK) AND FPN-SDK

The Fast Path receives and sends packets through the FPN-SDK, the hardware abstraction layer on top of the hardware-dependent 6WINDGate DPDK.

The 6WINDGate DPDK provides drivers and libraries for high performance I/Os on x86 and Arm. It is based on the open source DPDK from dpdk.org, provides virtualized networking and crypto add-ons in addition to the standard features of the open source DPDK, plus commercial support and maintenance.

The FPN-SDK provides zero-overhead APIs that enable Fast Path protocols to receive and send packets on the wire, to receive and send packets to Linux, to manage memory and to interface with multicore hardware such as crypto-engines, hardware queues, etc. These APIs are implemented using the processor SDK:

- Application API
- Packet bulk API
- Checksum Computation API
- Core-set management API
- Control Plane Protection API
- Fpn_flow API
- Garbage collector
- Intercore API
- Mbuf structure API
- Message API
- Hardware offload API
- Tools description, FPVI statistics and Control Plane Protection usage.

Refer to the following Module Data Sheets for details:

- 6WINDGate DPDK,
- 6WINDGate DPDK Intel Multi-Buffer crypto add-on,
- 6WINDGate DPDK Intel QuickAssist crypto add-on,
- 6WINDGate DPDK Virtio Host PMD add-on,
- FPN-SDK Baseline,
- 6WINDGate FPN-SDK DPDK add-on.

4.2 FAST PATH

4.2.1 Packet Processing

The Fast Path implements the "Run to Completion" model: as soon as a packet has been allocated to a core, all the processing on this packet is performed by this specific core.

If the Fast Path can process an incoming packet, it passes it to the relevant module that will perform this processing using information in the Shared Memory.

If the incoming packet cannot be processed at the Fast Path level, the packet is forwarded to the Linux Networking Stack as an exception packet through the FPVI API. The packet is injected at the right location in the Linux Networking Stack to avoid any process duplication.

Figure 5 illustrates the operation of a Fast Path protocol, using IPv4 forwarding packet processing as an example.

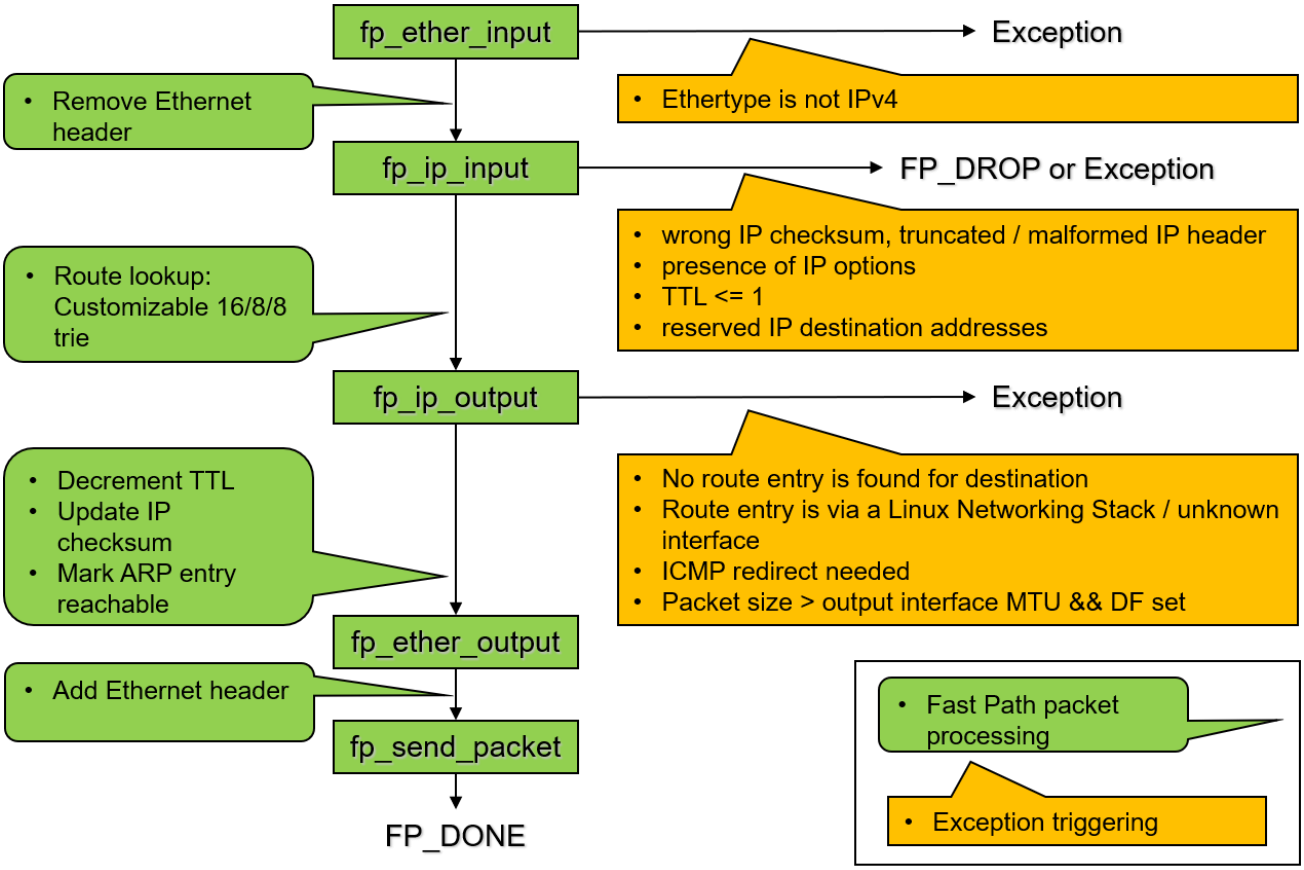

Figure 5: IPv4 forwarding packet processing

The Fast Path packet processing is responsible for parsing the incoming packet and determining whether it can be forwarded with the information available in the local memory.

For that purpose, the following tests have to be performed to achieve the IPv4 forwarding function:

- Firstly, the Fast Path determines whether the incoming packet has an IPv4 ethertype. If the test is negative, the packet is an exception packet and diverted to the 6WINDGate Linux Networking Stack to be handled there.

- Then, the IPv4 packet is examined to detect packets that cannot be managed at the Fast Path level, such as packets with a wrong checksum, incorrect IP options or a TTL at 0. Broadcast/multicast packets as well as packets reserved with IP destination addresses are also detected. If such an event occurs, the packet is an exception and diverted to the 6WINDGate Linux Networking Stack to be processed.
- Once this is completed, if the packet has not been diverted, IP lookup is performed to check if there is valid entry in the routing table. IP lookup in the 6WINDGate IPv4 forwarding Fast Path protocol is implemented as M-trie 16/8/8 so that any flow is looked up within a fixed number of memory accesses, thereby achieving the highest performance. At this stage, a determination is made of:
    - if a route is found for the destination,
    - if the route found in the table is via a Linux Networking Stack interface. For example, it could be for a PCI or WiFi interface managed by the Linux Networking Stack,
    - if the route found in the table is via an unknown interface,

  If such an event occurs, the packet is an exception and diverted to the 6WINDGate Linux Networking Stack to be processed.
- At this stage, the Fast Path has to perform the required processing to mark the ARP entry as reachable, decrement TTL, update the IP packet checksum and add the ethertype before queuing the packet to be forwarded.

This simple example shows how a Fast Path protocol works. Of course, the processing becomes more complex when additional functions are included (e.g. Virtual Routing, Policy-Based Routing, packet bulks, etc.).

Each Fast Path protocol is integrated within the complete 6WINDGate Fast Path architecture to provide a complete solution (refer Figure 6).

This example also illustrates the flexibility of the 6WINDGate architecture and how it can be used to progressively add Fast Path protocols. Protocols that are not supported in the Fast Path can be diverted to the Linux Networking Stack. As soon as the corresponding Fast Path protocol is available, the packet can remain at the Fast Path level for further processing.

4.2.2 Fast Path Virtual Interface

The Fast Path Virtual Interface (FPVI) allows exchanging packets between the Fast Path and the Linux Networking Stack. The FPVI makes Fast Path ports appear as netdevices into the Linux Networking Stack.

The purpose of the FPVI is to:

- Provide a physical NIC representor in Linux for configuration, monitoring and traffic capture.
- Send packets from Linux to the Fast Path (locally generated traffic).
- Exchange exception packets between the Fast Path and Linux.

The FPVI is implemented in Linux using the TUN/TAP driver, and in the Fast Path through the FPN-SDK using the DPDK virtio-user PMD providing a virtual port to each TUN/TAP interface.

Packets to be sent locally by the Linux Networking Stack are directly injected in the outgoing flow to be processed by the Fast Path, using the TUN/TAP Linux driver.

The FPVI implements the exception strategy as follows:

- For Basic Exceptions, the FPVI implements a standard processing through the netif_rx function of the TUN/TAP Linux driver.
- For Special Exceptions, on the ingress path, packets are injected at the right place into the Linux Networking Stack thanks to an eBPF program as explained below. On the egress path, packets are sent directly using the standard sendmsg() API.

Special Exceptions are sent to Linux with a specific trailer called FPTUN, including information about the processing that occurred in the Fast Path. They are sent to an eBPF program called the FPTUN handler, which parses the FPTUN trailer and drives the packets to the right hook inside the Linux Networking Stack for further processing aligned with the work already done by the Fast Path.

4.2.3 Supported Protocols

The list of available Fast Path modules can be found in Figure 4.

For each supported networking feature, a 6WINDGate Fast Path protocol has been designed, according to the following guidelines:

- The Fast Path protocol implementation shall be simple and efficient,
- The Fast Path protocol shall be seamlessly integrated with the Linux Networking Stack and Control Plane,
- The Fast Path protocol shall be portable and optimized for multicore architectures.

These protocols are implemented as high-performance parallel generic software using the FPN-SDK API (refer to previous section).

The following figure represents the high-level Fast Path call flow with the main supported protocols:

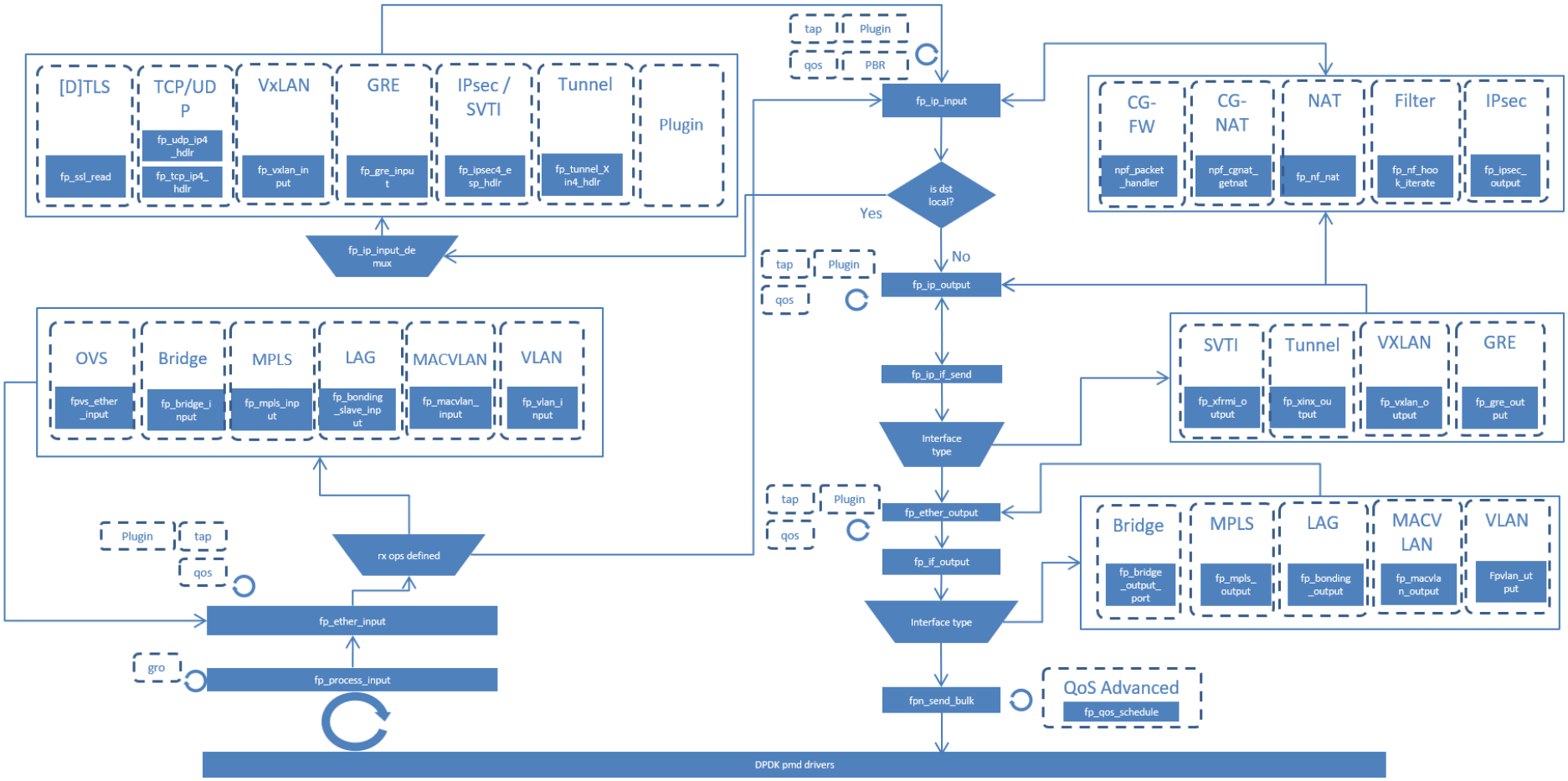

Figure 6: Fast Path call flow

Incoming packets are processed as follows (assuming that all the functions are activated):

- Ingress QoS is applied first.
- Then comes Layer 2 processing.
- The packet is optionally reassembled and filtered (ingress); traffic conditioning (rate limitation) is applied.
- Layer 3 processing (e.g. forwarding, IPsec, NAT) is then applied.
- The packet is filtered and, if necessary, fragmented.
- Layer 2 processing is performed on the packet again.
- Egress QoS is applied.

At each level, an exception will occur if the packet cannot be processed at the Fast Path level.

In cases where Layer 3 encapsulation is applied (IP tunneling, GRE), the packet is considered as coming back on a new interface. If several encapsulations are applied, they are applied recursively.

4.2.4 Scalability According To The Number Of CPU Cores

The packet processing performance scales with the number of cores allocated to the Fast Path by combining:

- RSS (Receive Side Scaling) in NICs to dispatch packets into several reception queues according to the 5-tuple of the incoming packet,
- lockless, run-to-completion design of the Fast Path that makes processing in each core independent of the other.

In some use cases, RSS does not help as the 5-tuple of incoming packets is always the same (for example with IPsec encapsulation). In these cases, processing is load balanced among cores in software (for example IPsec crypto processing is offloaded to idle cores).

Some modules cannot be lockless by design because they need to share resources between cores (for example QoS or TCP use shared software queues). In that case, performance does not scale linearly.

4.2.5 Fast Path Plugins

Fast Path plugins make it possible to customize some part of the Fast Path application without modifying the main Fast Path engine. Plugins override some "hooks" in the Fast Path by using the dynamic linking load mechanism.

Here are some available hooks:

- fp_ether_input
- fp_ether_output
- fp_if_output
- fp_ip_input
- fp_ip_inetif_send
- fp_ip6_input
- fp_ip6_inet6if_send
- fp_process_linux_tx

And some Fast Path plugin examples:

- PPPoE load balancer
- Round-robin load balancer
- TCP client, server and proxy

One of the key advantages of Fast Path plugins is to avoid code rebasing when 6WINDGate is rebased. Also, it is a flexible mechanism to integrate new protocols on top of the 6WIND networking stack.

Refer to the 6WINDGate Fast Path Module Data Sheets for details (one for Baseline, one for plugins and one for each supported protocol).

5 6WINDGATE LINUX SYNCHRONIZATION

This section describes how information between the Control Plane and Data Plane is synchronized in 6WINDGate Fast Path-based implementation.

5.1 EXCEPTION STRATEGY

If the incoming packet cannot be processed at the Fast Path level, the packet is forwarded to the Linux Networking Stack as an exception packet through the FPVI (see section 4.2.2).

In general, packets sent through the FPVI are received in Linux through the TUN/TAP reception driver and processed through the standard Linux Networking Stack. However, in some cases, the original packet cannot be sent to Linux for standard processing. For example, it is the case for IPsec packets that have been deciphered and cannot be processed further (e.g. no route for the clear packet). The clear packets cannot be sent to the standard Linux Networking Stack as they would be dropped because they match a Security Policy. In that case, the packets are sent to Linux through a dedicated eBPF program to inject them at the right location in the Linux networking stack (in our case, after ingress IPsec processing).

As the Fast Path is generally more efficient than Linux, generating many exceptions can overload Linux. To control the exception mechanism, 6WINDGate supports customizing the number of cores responsible for processing exceptions in Linux, plus rate limitation on the exception channel with prioritization of most important Control Plane packets.

5.2 CONTINUOUS SYNCHRONIZATION

In order to avoid any modification to the Linux OS, the Control Plane and the Management Plane when the Fast Path is present, synchronization mechanisms ensure that Linux configuration gets silently synchronized into the Fast Path and, conversely, that Fast Path statistics end up into the right place in the Linux kernel. This way, Linux applications continue to rely on Linux APIs to configure or monitor the system.

5.2.1 Configuration: Cache Manager / Fast Path Manager

The Cache Manager is one of the two software modules that perform synchronization between the Linux Networking Stack and the Fast Path. It monitors the kernel updates performed by the Control Plane protocols (ARP and NDP entries, Layer 3 routing tables, Security Associations etc.) through Netlink messages and synchronizes the Fast Path with this information.

Thanks to the Cache Manager, no change is required to Control Plane protocols when they are integrated with the Fast Path. The Cache Manager is hardware-independent. It provides any required information to be offloaded to the Fast Path toward the FPC API (refer section 9).

The Fast Path Manager (FPM) is the Control Plane software module responsible for Fast Path configuration. The FPM updates Fast Path tables according to messages exchanged with the Cache Manager through the FPC API (refer section 9).

There are two different ways to update the Fast Path: writing into the tables in the shared memory, or sending a direct message to the Fast Path application when an update needs to trigger an event in the Fast Path (for example change MTU).

5.2.2 Statistics And Hitflags

As some packets are processed in the Fast Path and others are processed in the Linux Networking Stack, it is necessary to aggregate Linux and Fast Path statistics. The Fast Path modules update the Shared Memory with statistics. The FPS daemon reads the Shared Memory and updates the Linux kernel statistics using Netlink. Some kernel statistics cannot be updated by Netlink. For these, a pre-loadable library is provided, so that Netlink statistics requests are updated transparently with the Fast Path statistics from the Shared Memory.

This way, Linux applications that read statistics from Linux receive aggregated Linux Networking Stack + Fast Path statistics.

Similarly, for packets processed by the Fast Path, the corresponding Linux object states (ARP entries, conntracks, Linux bridge, etc.) have to be updated to prevent their expiration. The Fast Path modules update the Shared Memory when an entry is hit (hitflag). The Hitflags daemon reads the Shared Memory and updates the corresponding Linux kernel entry using Netlink. Linux applications read object states from Linux as usual.

Refer to the 6WINDGate Linux / Fast Path Synchronization Module Data Sheet for details.

Note that a few modules are not synchronized with Linux:

- TCP, as accelerated TCP applications have to be ported into the Fast Path and are therefore independent from the Linux Networking Stack;
- QoS, as for performance reasons the 6WINDGate architecture is very different from Linux's and synchronization is not possible;
- CG-NAT, as there is no Linux CG-NAT implementation and therefore nothing to synchronize.

5.3 EXAMPLES

5.3.1 IP Forwarding Example

Using IP forwarding and routing as an example, the following sections explain how transparent synchronization between the Fast Path, the Linux Networking Stack and the Control Plane is performed for both static configurations (iproute2, ifconfig, etc.) and dynamic protocols (BGP with FRR, IKE with strongSwan, etc.). It also shows that synchronization does not require any change in the Control Plane protocols, as it relies on standard OS APIs.

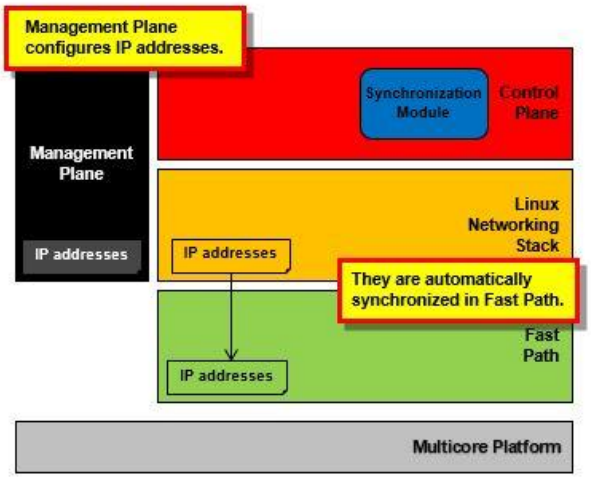

Figure 7: IP forwarding - Step 1

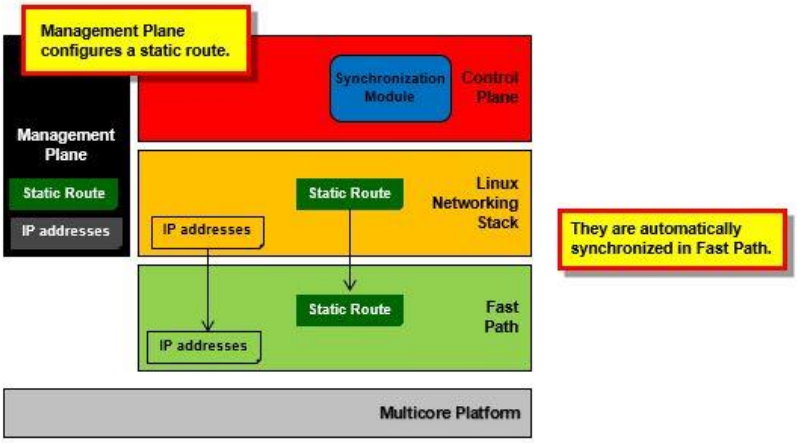

Figure 8: IP forwarding - Step 2

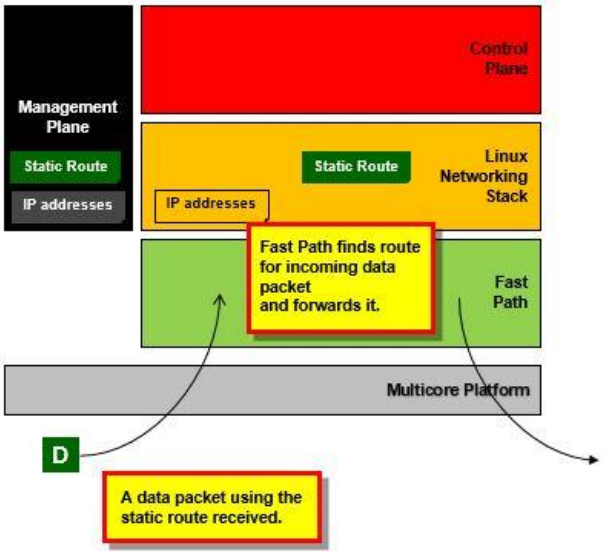

Figure 9: IP forwarding - Step 3

5.3.2 Dynamic Routing Example

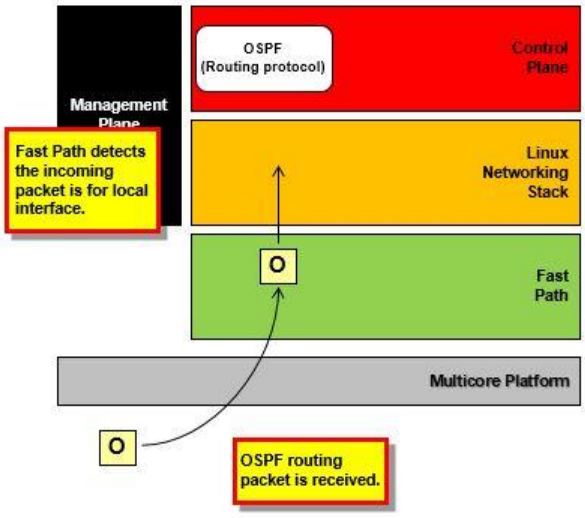

Figure 10: IP routing - Step 1

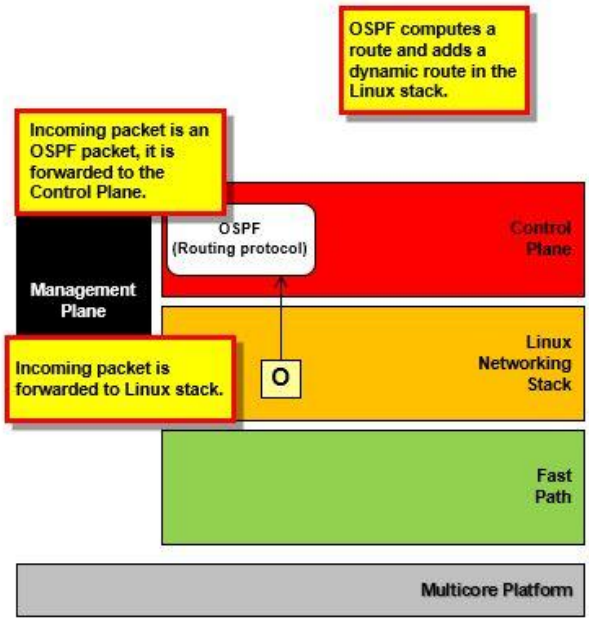

Figure 11: IP routing - Step 2

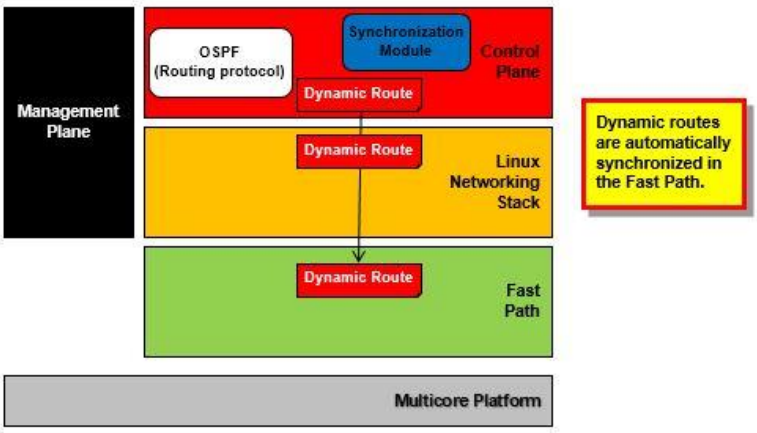

Figure 12: IP routing - Step 3

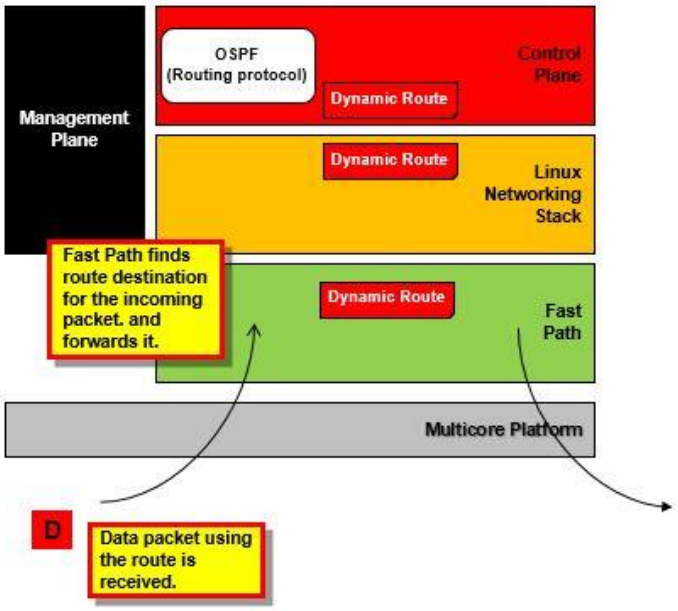

Figure 13: IP routing - Step 4

5.3.3 IPsec Example

The same architecture design is applied to all the protocols supported in 6WIND's product, including IPsec.

6WIND Control Plane synchronization daemons allow replication of Security Associations and Security Policies in the Fast Path Shared Memory. In this way, when a packet matching a security policy is processed by 6WINDGate IPsec, all the necessary information to cipher/decipher it is locally available. The cryptographic operation is then taken care of by 6WIND accelerated Data Plane. When a packet reaches the Fast Path for a SA that is not yet negotiated, this packet is delivered to the Linux kernel stack through the exception mechanism. The kernel can then trigger an acquire message.

Control plane IKE packets are also sent to Control Plane daemon using the same exception mechanism.

On Intel x86 architecture, 6WIND supports DPDK add-ons to benefit from AES-NI Multi-buffer library, or external cryptographic acceleration/offload cards (Intel DH895xCC Coleto Creek, Intel Lewisburg and C3000 integrated QAT module).

6 6WINDGATE CONTROL PLANE

6.1 INTRODUCTION

As explained in detail in section 5, 6WINDGate automatically synchronizes the Linux Networking Stack states with its own local information. This mechanism is based on standard Linux APIs, so that standard Linux Control Plane utilities can be used to manage the system. We can commonly think of iproute2 to configure static IP addresses, routes, neighbors, and create logical interfaces like VLAN, VXLAN, etc. All the common networking utilities are available in the 6WINDGate product.

Additionally, this means that open source Control Plane daemons configuring the Linux network stack can also be used as-is. 6WINDGate leverages this by embedding:

- FRRouting for dynamic routing,
- strongSwan for IKE,
- Open vSwitch for OVS.

6.2 DYNAMIC ROUTING

The 6WINDGate routing Control Plane module enables route management over a wide variety of routing protocols. It is provided by the zebra daemon, from the open source FRR project.

Features:

- RIPv1/RIPv2 for IPv4 and RIPng for IPv6
- OSPFv2 and OSPFv3
- BGP4

Releases of FRR integrated in 6WINDGate follow the upstream versions. 6WIND also contributes to this open source project by submitting bug fixes back to the community.

Refer to the 6WINDGate Control Plane Routing Module Data Sheet for details.

6.3 IKE

The 6WINDGate security Control Plane module implements the IKEv1 and IKEv2 protocols. It allows negotiating keying material (IPsec SAs) for the use of IPsec VPNs. It is based on the latest version of the open source strongSwan IPsec-based VPN solution.

All strongSwan features are supported by 6WIND.

Refer to the 6WINDGate Control Plane Security IKE add-on Module Data Sheet for details.

6.4 OVS

The 6WINDGate OVS Control Plane module provides virtual switching, flow matching, and packet manipulation, configured via an OpenFlow controller or the command line. It is based on http://www.openvswitch.org/. This module follows the open source releases.

Features:

- The Control Plane OVS module provides the following capabilities:
    - Manage, control and debug the OVS datapath
    - Support for statistics synchronization with the Fast Path
    - Support for hitflags (a flow not hit by the kernel datapath, but hit by the Fast Path datapath, stays alive)
    - Support for direct addition/deletion of flows in the Fast Path Shared Memory

Refer to the 6WINDGate Control Plane OVS Module Data Sheet for details.

7 6WINDGATE MANAGEMENT PLANE

7.1 OVERVIEW

The 6WINDGate Management Plane (cf. Figure 14) comprises three main building blocks:

- Engine and Data Store: YAMS, a Python-based engine configuring and monitoring all network components. The YAMS engine uses a YANG model data store.
- Configuration: A NETCONF server providing a standard API to interface with NETCONF-based configuration tools. 6WIND has developed its own CLI tool as a NETCONF client to configure 6WINDGate features.
- Monitoring / Analytics: Traditional SNMP and sFlow monitoring services as well as advanced services through a KPI (Key Performance Indicator) agent collecting and streaming statistics.

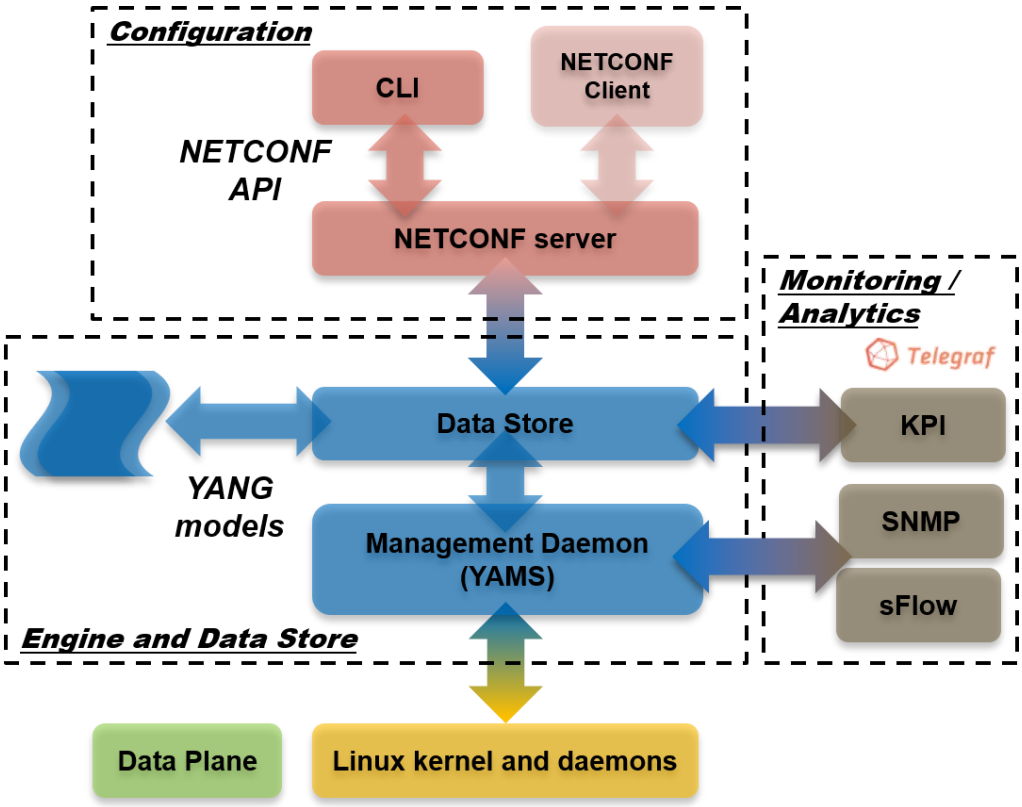

Figure 14: 6WINDGate management architecture

The 6WINDGate Management Plane consists of the following modules:

- Management Baseline provides:
  - NETCONF server
  - YANG-based datastore
  - YAMS

  - SNMP
  - Support of Fast Path features (Control Plane features supported through add-ons)
- Management Routing add-on provides management of the Control Plane Routing module,
- Management Security add-on provides management of the Control Plane Security module,
- Management VRRP add-on provides management of the High Availability VRRP module,
- Management HA add-on provides management of other High Availability modules,
- Management CLI provides the Command-Line Interface (NETCONF client),
- Management sFlow provides the sFlow protocol and its management,
- Management KPIs provides the Key Performance Indicators YANG-models, engine and agent to stream the data to an external Time-Series Database.

7.2 YAMS: NETCONF/YANG-BASED MANAGEMENT ENGINE

NETCONF is a network management protocol standardized by the IETF. It defines mechanisms to install, manipulate and delete the configuration of network devices. It uses Extensible Markup Language (XML)-based data encoding for the configuration data as well as the protocol messages. More information is available in RFC 6241.

YANG is a language used to model data for the NETCONF protocol. A YANG module defines a hierarchy of data that can be used for NETCONF-based operations, including configuration, state data, Remote Procedure Calls (RPCs), and notifications for network management protocols. More information is available in RFC 7950.

The NETCONF API can be used from any NETCONF client to configure and monitor the router remotely, therefore enabling automation and orchestration.

NETCONF/YANG-based engine

The management engine comprises a YANG-based datastore and a NETCONF server. It supports all the required protocol operations to read and write the configuration: <get>, <get-config>, <edit-config>, <copy-config> and so on.

Clear separation between configuration and state data

The management engine stores separate configuration and state data for each feature. The state part includes additional runtime information compared to the configuration part; typically, statistics. It is possible to display the state data from anywhere in the CLI using the get command, so that the user can review data from the current state while building his configuration.

VRFs

The networking configuration is natively split into VRFs in order to provide a high level of isolation between Management Plane and networking plane. Each VRF has its own interfaces, IP addresses, routing table, firewall, etc.

This approach ensures a good isolation of services and will allow in the future to define limits in term of CPU resource or memory for a given VRF. It relies on Linux network namespaces (netns).

Compatibility with existing Linux Day-1 configuration

Cloud-init can be embedded in 6WINDGate for Day-1 configuration, that is, the initial configuration of 6WINDGate to enable basic console access. It can be used to configure the management interface, basic networking services such as DHCP and SSH, provision the SSH keys, etc.

The management engine is compatible with such cloud-init configuration, as it does not touch the configuration of network services (SSH, DNS, DHCP, etc.) as long as there is no configuration statement for them. When a configuration statement is present, it takes precedence over any existing, external configuration. Finally, a known service like SSH will be recognized and will not be restarted if it is not necessary.

Refer to the 6WINDGate Management Baseline Module Data Sheet for details.

7.3 CLI

The CLI is the common user interface to interact with 6WINDGate. It can be used to configure, monitor and troubleshoot. The CLI provides help and completion, as well as the management of configuration files to save and restore a complete and consistent configuration in one command.

The CLI is actually a NETCONF client that communicates with 6WINDGate's YANG-based configuration engine. Its command names and statements follow the syntax and the hierarchical organization of the 6WINDGate YANG models. Data consistency is checked against the YANG model, so that syntax errors are detected early. The configuration engine supports transactions and rollback on error.

The CLI comes with traditional features, such as completion, history and contextual help. Users can walk the configuration tree as they would browse a file system, for example, "/" jumps to the root of the configuration, ".." moves one level up. Relative and absolute paths can be used to refer to configuration data, making browsing very efficient.

Refer to the 6WINDGate Management CLI Module Data Sheet for details.

7.4 MONITORING / ANALYTICS

6WINDGate provides a complete portfolio of services for network monitoring and analytics (refer Figure 14).

7.4.1 Traditional Monitoring: SNMP

SNMP (Simple Network Management Protocol) is an Internet-standard protocol for collecting and organizing information about managed devices on IP networks.

It exposes management data in the form of variables on the managed systems organized in a MIB (Management Information Base) that describes the system status. These variables can then be remotely queried by management applications.

The 6WINDGate SNMP management module provides the support of SNMP monitoring, based on the net-snmp (http://www.net-snmp.org) open source project.

6WINDGate supports SNMPv1, SNMPv2c (basic authentication with community strings) and SNMPv3 (authentication with SNMP users).

Supported MIBS include standard system and networking MIBs (interface, IP, IPv6, IP forward, etc.), routing MIBs (BGP, OSPF, RIP), VRRP MIBs and 6WIND-developed IPsec MIB.

Refer to the 6WINDGate Management Baseline SNMP Module Data Sheet for details.

7.4.2 Data Plane Analytics: Sflow

sFlow is a technology for monitoring traffic in data networks containing switches and routers. In particular, it defines the traffic sampling mechanisms implemented in sFlow Agents and the format of the sFlow Datagram that carries traffic measurement data from sFlow Agents to an sFlow Collector.

This module is based on host-sflow, which is an open source implementation of the sFlow standard. It is patched by 6WIND for Fast Path offload.

Refer to the 6WINDGate Management sFlow Module Data Sheet for details.

7.4.3 Next-Gen Monitoring: KPIs

In addition to traditional monitoring, 6WINDGate provides an advanced monitoring solution based on time series collection and visualization of Key Performance Indicators (KPIs). With such a solution, it is much easier to understand problems that happened in the past and to correlate them to past events. And it may even be used to predict the future as the user directly visualizes the dynamics of the system.

6WINDGate KPIs are pre-integrated with the InfluxDB time-series database and the Grafana analytics frontend. An example of InfluxDB/Grafana setup is described on 6WIND's github. Integration with other TSDB or analytics frontends is possible. Telegraf is used to collect KPIs and export them to InfluxDB.

KPIs are modelized using YANG and exposed using NETCONF or a local API. The KPIs module consists in:

- The KPIs daemon (kpid), for both Linux system and Fast Path modules,
- The monitoring engine (YAMS), exposing the KPIs with local API and NETCONF/YANG API in a consistent way,
- The tool (kpi-tool) to query the API for direct use by common existing tools (JSON, Telegraf/InfluxDB).

Refer to the 6WINDGate Management KPIs Module Data Sheet for details.

7.5 MANAGEMENT EXTENSIBILITY

The 6WINDGate management architecture can be customized to extend the 6WINDGate management services. These extensions are summarized in Figure 15.

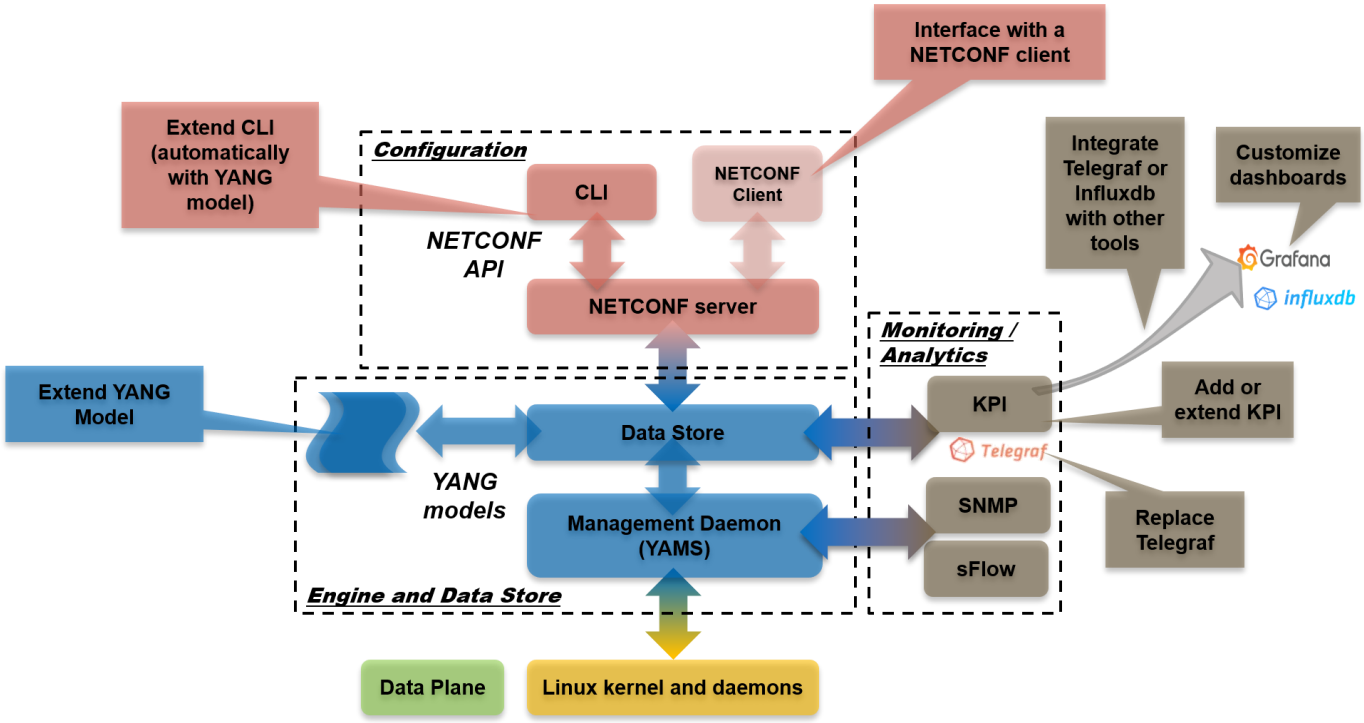

Figure 15: 6WINDGate management extensibility

7.5.1 YANG Model And Configuration

As 6WINDGate configuration services rely on a NETCONF API, all tools including automation and orchestration platforms based on a NETCONF client can be easily integrated with 6WIND.

The 6WINDGate CLI can also be extended to develop new commands for an existing 6WINDGate feature. For instance adding a new CLI command for routing configuration or routing information display.

Users can also use the 6WINDGate configuration framework to integrate the configuration of a user feature in a consistent manner with all existing services. For instance, a user can develop a 6WINDGate management extension for a user GTP module integrated with 6WINDGate.

Extensions of the 6WINDGate CLI are automatically available after extending the corresponding YANG models. The CLI documentation (command reference) is also automatically generated.

Examples of extensions of YANG models and CLI are provided in 6WINDGate management documentation.

7.5.2 Monitoring / Analytics

The 6WINDGate open architecture can be used in different ways to extend monitoring and analytics services.

First of all, users can extend the services and data available in 6WINDGate to:

- Add or extend the 6WINDGate available KPIs,
- Create new KPIs for user features integrated with 6WINDGate,
- Customize the Grafana dashboards available under 6WIND github for displaying 6WINDGate or user features.

It is also possible to use different tools:

- The Telegraf agent can be integrated with another Time Series Data Base tool like ElasticSearch instead of Influxdb,
- Influxdb can be integrated with another graphical front-end like Kibana instead of Grafana,
- The 6WINDGate Telegraf agent can be replaced by another KPI agent such as collectd.

8 6WINDGATE HIGH AVAILABILITY

The list of 6WINDGate High Availability modules is the following one:

- HA Baseline
- HA ARP/NDP Synchronization
- HA Firewall / NAT Synchronization
- HA IPsec/IKE Synchronization
- Daemon Monitoring System
- VRRP

Modules data sheets are available upon request for more information.

8.1 HA BASELINE

The HA Baseline module provides libraries and scripts common to other High Availability synchronization modules.

Refer to the 6WINDGate High Availability Baseline Module Data Sheet for details.

8.2 HA ARP/NDP SYNCHRONIZATION

The HA ARP/NDP Synchronization module allows synchronizing ARP (Address Resolution Protocol)/NDP (Neighbor Discovery Protocol) tables between two different machines.

Refer to the 6WINDGate High Availability ARP/NDP synchronization Module Data Sheet for details.

8.3 HA FIREWALL / NAT SYNCHRONIZATION

6WIND HA Firewall / NAT Synchronization module allows synchronizing conntracks between two different machines.

It is based on the conntrackd open source daemon.

This module synchronizes the stateful session table between the active firewall and the backup firewall. After a failover, the backup firewall session table is already up-to-date and the traffic continues to be forwarded without session interruption.

Refer to the 6WINDGate High Availability Firewall / NAT Synchronization Module Data Sheet for details.

8.4 HA IPSEC/IKE SYNCHRONIZATION

The HA IPsec/IKE Synchronization module enables to synchronize IKE (Internet Key Exchange) and IPSEC (Internet Protocol Security) between two different machines.

Refer to the 6WINDGate High Availability IPsec/IKE Synchronization Module Data Sheet for details.

8.5 DAEMON MONITORING SYSTEM

The HA Daemon Monitoring System provides crash recovery and proactive health check of daemons.

Refer to the 6WINDGate High Availability Daemon Monitoring System Module Data Sheet for details.

8.6 VRRP

The 6WINDGate VRRP Control Plane module provides a way, for a set of routers, to control a virtual IPv4 and MAC addresses, including automatic failover mechanism. Such an address may be used by hosts for some service access, e.g. as static default gateway. The gain from using VRRP is a higher availability of the service without requiring automatic reconfiguration of end hosts.

6WINDGate integrates the open source keepalived daemon. 6WIND's Fast Path supports the synchronization of macvtap interfaces created by the keepalived daemon.

Features:

- VRRPv2 RFC 3768 (https://tools.ietf.org/html/rfc3768.html) (IPv4 only)
- Automatic election of master router
- Preemption when a backup server with higher priority than master is present
- Advertisement interval (to indicate that master is still in service) from 1 to 255 seconds
- Ability to create several VRRP groups and to synchronize them: a router, belonging to different groups, has the same state (master or backup) in any group
- Authentication (Simple Text Password or IP Authentication Header) as defined in VRRPv1 RFC 2338 (https://tools.ietf.org/html/rfc2338.html)
- Remote management with SNMP

Refer to the 6WINDGate High Availability VRRP Module Data Sheet for details.

9 USING 6WINDGATE IN DIFFERENT ENVIRONMENTS

To address a wide variety of applications, 6WINDGate can be used in different environments including:

- Bare metal
- Virtual machines
- Containers

9.1 BARE METAL

6WINDGate can be used in a bare metal environment. All the Linux resources are directly accessible and used by 6WNDGate. For example, 6WINDGate DPDK directly interfaces with physical drivers of the NICs.

9.2 VIRTUAL MACHINES

Figure 16 shows how 6WINDGate can be used in a virtual machine. Each virtual machine has its own and dedicated Linux kernel and embeds a complete instance of 6WINDGate.

It is recommended to use SR-IOV to bypass and remove performance bottlenecks in the hypervisor (KVM in this example). The actual physical NIC is split into several Virtual Functions (Eth-VF) that are attached and dedicated to the Virtual Machine. The Fast Path interacts directly with the Eth-VFs through the 6WINDGate DPDK.

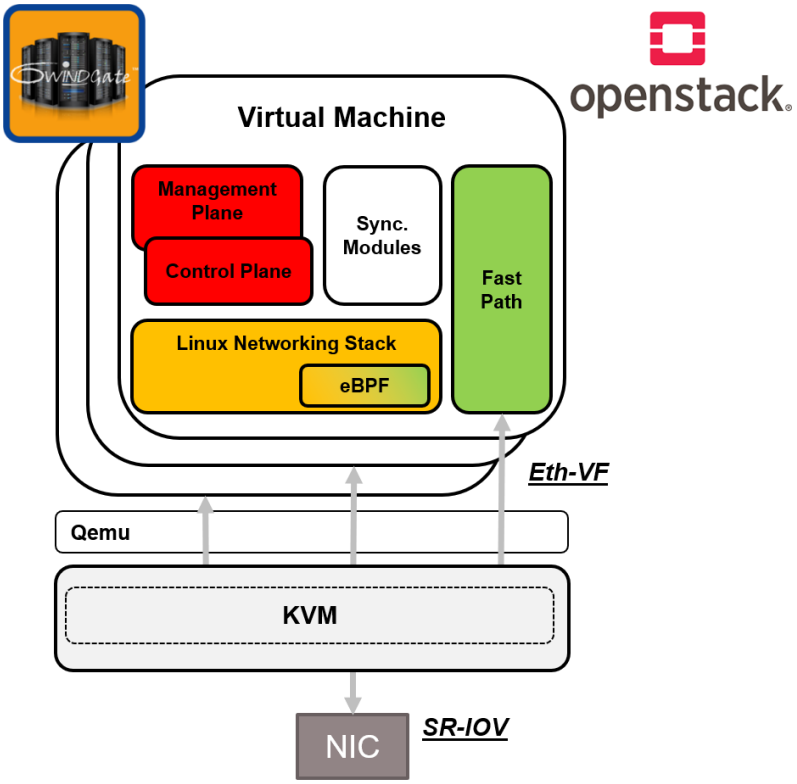

Figure 16: Using 6WINDGate in virtual machines

9.3 CONTAINERS

Figure 17 shows how 6WINDGate can be used in containers. Containers are userland application isolated in a Linux network namespace (netns), sharing a single Linux kernel.

Similarly to VM deployments, it is recommended to use SR-IOV to dedicate VFs to container instances in order to get the best performance.

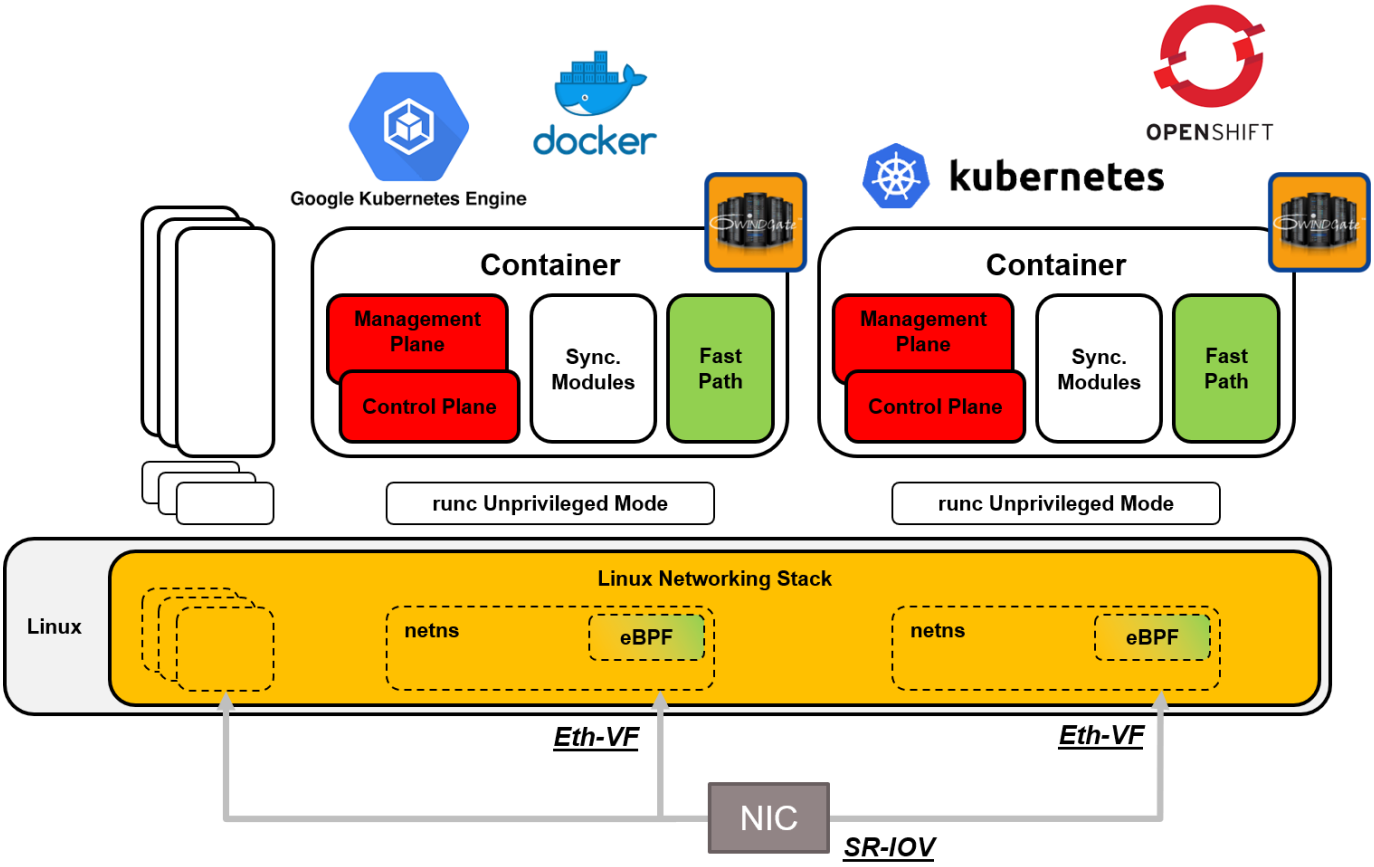

Figure 17: Using 6WINDGate in containers

10 6WINDGATE MAIN COMPONENTS AND APIs

10.1 OVERVIEW

6WINDGate is designed to be extensible at all levels. Figure 18 details the main components and APIs of the 6WINDGate packet processing software.

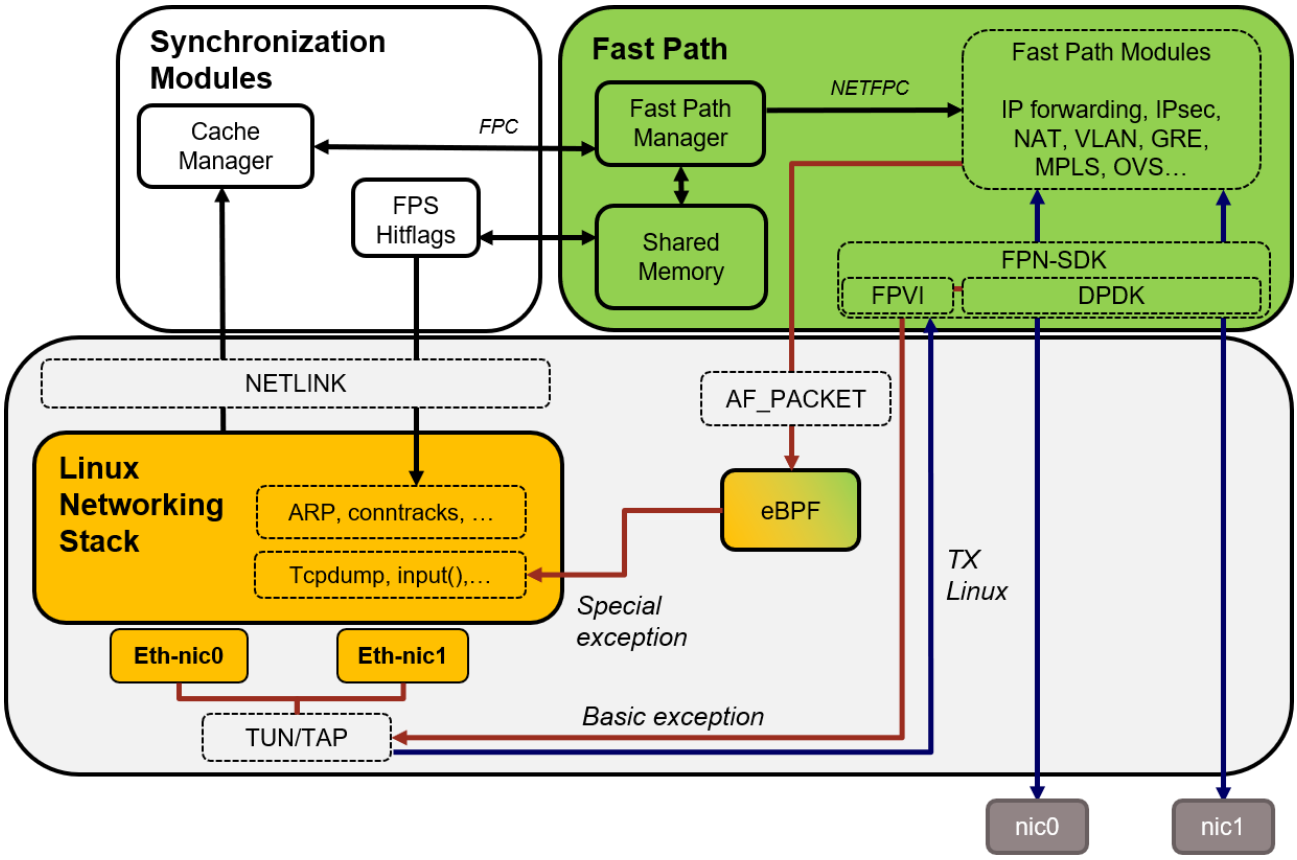

Figure 18: 6WINDGate detailed architecture

A 6WINDGate user can customize the source code for miscellaneous needs.

10.2 MAIN COMPONENTS AND APIs REFERENCE

The following table summarizes the main components and APIs available in 6WINDGate for quick reference.

Name	Purpose
FPN-SDK (Fast Path Networking SDK)	The FPN-SDK is the abstraction layer on top of the underlying hardware architecture. It provides the FPN API to the Fast Path modules and implements it using the HW-specific SDK (today DPDK for Intel and Arm architectures). See section 4.1 for more information.
FPN API (Fast Path Networking API)	The FPN API is the generic API between the Fast Path Modules and the CPU, NICs, Shared Memory and Linux. It provides: 1. Packet buffer (mbuf) 2. Access to Shared Memory (userland / kernel / Fast Path) 3. FPVI (see below) 4. Crypto with HW support and SW fallback 5. Offloads: checksum, TCP (LRO, TSO) 6. And more: Control Plane protection, fast and scalable timers, memory pool and ring, lock and synchronization, atomic operations, CPU usage monitoring, function calls tracking for debugging, inter-core packet distribution… It is provided by the FPN-SDK module.
Fast Path Modules	Actual implementation of the Fast Path networking stack, taking care of high-performance packet processing. See section 4.2 for more information.
Fast Path Plugin API	Fast Path plugins make it possible to customize some part of the Fast Path application without modifying the main Fast Path engine. 6WINDGate developers can implement their own plugins using the plugin API to hook into the packet processing and using the FPN API to actually process packets.
FPVI	The FPVI provides a NIC representor (netdevice) in Linux for interaction with the Fast Path Modules. It is implemented by each Fast Path Module on the Fast Path side and as a TUN/TAP device and an eBPF program in Linux. The FPVI implements the exception strategy for all exceptions as described in section 5.1. Extended exceptions imply the use of the FPTUN proprietary protocol and a FPTUN handler (eBPF program) to inject packets into the Linux Networking Stack. It is also used by Linux to send data packets through the Fast Path. 6WINDGate developers extend the FPVI and the FPTUN handler in case they need to add new exception types for their own Fast Path Modules.

Netlink API	The Netlink API is used by the Cache Manager to monitor kernel events and state changes for interfaces, Layer 2 / Layer 3 tables, IPsec, etc. It is a standard Linux notification mechanism. Refer to section 5.2 for detailed information about the Linux / Fast Path Continuous Synchronization mechanism.
FPC (Fast Path Control) API	The FPC API is the communication API between the Cache Manager and the Fast Path Manager. The FPC synchronizes the states of the Linux Networking Stack with the Fast Path without any modification of the existing services that distribute the information. Therefore, there is no need to modify any existing daemons or kernel modules to interface to the Fast Path when you use the FPC and the Cache Manager. Refer to section 5.2 for detailed information about the Linux / Fast Path Continuous Synchronization mechanism.
Shared Memory	The Shared Memory contains structures for: • Physical ports • Forwarding table • Statistics • IPsec processing • etc. It is used by: • the Fast Path Manager to write local information received from Cache Manager through FPC messages • the Fast Path to read local information used for packet processing (L2/L3 entries, IPsec SAs, etc.) and write statistics • the FPS to read statistics
NETFPC	The NETFPC API triggers events in the Fast Path from the Fast Path Manager as part of the Linux / Fast Path Continuous Synchronization mechanism.
FPS (Fast Path Statistics) and Hitflags	The Fast Path Statistics (FPS) and Hitflags daemon read statistics and protocol state information from the Shared Memory and update the corresponding stats and state information in the Linux kernel using Netlink. For statistics that cannot be updated by Netlink, a pre-loadable library is provided, so that Netlink statistics requests are updated transparently with the Fast Path statistics.
NETCONF/YANG API	The NETCONF API is used by the CLI to configure and monitor 6WINDGate. YANG models are fully documented and the NETCONF API can indeed be used from any NETCONF client (local or remote). The CLI is automatically generated from the YANG data model, so that extending the model automatically results into new CLI commands.

YAMS (YANG-based Management System)	YAMS is the interface between the YANG-based management engine and the Linux system. When a new configuration is received in the datastore, YAMS configures it in Linux using standard Linux APIs. It also retrieves monitoring information from Linux and stores it into the datastore according to the YANG data model, so that it can be queried from a NETCONF client.