Interface Packet Processing Service (IPPS)

|  |  |
| --- | --- |
| Version | 0.1 |
| Author | Tom Sumardi |
| TODO: | References,REST API, Communication handshakes, split this doc to ipps and ppp |

Table of Contents

Introduction 1

Abbreviations 1

Conventions/Requirements 1

Summary 2

Components 2

Architecture 3

Component descriptions: 3

Communication handshakes: 4

# Introduction

This documentation intended to serve the purpose of giving detail level architecture description.

# Abbreviations

TBD

# Conventions/Requirements

* Performance bound
* IPPS operation will involve partial kernel and user space
* Bidirectional communication interface from MS to IPPS will be through AMQP
* Unidirectional communication interface from IPPS to PPP will be through ZeroMQ
* PPP is a service that plugs into IPPS component
* Packet outgoing interface will go through different interface than packet incoming interface

# Summary

Pcap

Filter

SKB

Eth0

Eth0

Load

Balancer

(MAC/IPaddr)

L2 Filter

Rulesets

(MAC Addr)

L3 Filter

Rulesets

(IP Addr)

Packet Payload

Processing (PPP)

Figure 1 (packet flow)

IPPS will be launched as daemon that pulls packets from SKB using pfring based on interface name within kernel space. First filtering being done on the packet layer by specifying libpcap filter syntax at the kernel level. After the packets are pulled, the packets go through series of filters on the user space. A load balancer is also added to load balance the packets per flow based on 4 tuples (src/dst MAC address, src/dst IP address). For example, any interesting packets will be passed from one pipeline to the next from IPPS to PPP. Any uninteresting packets are ignored. Future work might allow user level to tell kernel to ignore the flow.

# Components

Config/Register

Pfring

AMQP

Netw Intf

Load Balancer

(Thread Mgmt)

L2/L3 Filters

Logging

PPP

Figure 2 (component stack)

* Background configuration/registration thread
* Pfring configuration
* Thread management and load balancing
* Packet L2-L3 processing and filtering
* Logging

# Architecture

## Component descriptions:

1. Background configuration/registration thread

Background thread that is responsible for configuring and registering IPPS daemon, which uses AMQP as communication channel to the MS.

Responsibilities:

* The only thread that is up and running when the daemon first started
* Pfring configuration
* Spawning multiple worker threads based on configuration given
* Configures L2/L3 Filters
* Idle once configuration and registration have been successfully applied

1. Pfring configuration

Performs kernel level configurations such mmap, interface, pcap filtering, ring buffers , stream flows, etc.

1. Thread management and load balancing

The threads responsible for pulling packets from all the interfaces as fast as it can from the ring buffers. It utilizes pfring library in pulling the packets from kernel SKB and perform any necessary memory mapping the resource into the user space. Pfring library also responsible for segregating network interface, ring buffer creation, kernel-user space memory mapping, etc. The ring buffers are mapped to specific interface in one-to-many relationships with each ring buffer consuming packets per flow in 4 tuple relationships (src/dst IP address and src/dst MAC address).

Figure 1 example, n-thread instances are pulling packet from interface name “eth0”, which may have n-threads processing the packets with each thread assign its own stream flow. In case of network bonding, the interface will be broken down into each own separate interfaces.

1. Packet L2-L3 processing and filtering

Packet L2-L3 processing and filtering functionality (context) will exist within each load balancer (worker) thread. The idea is to pipeline incoming packets into a series of packet processing component and filters. As illustrated on figure 1, the packets are dissected and inspected from layer 2 up to layer 3.

The filters will decide whether packets will be:

* + dropped,
  + moved to Packet Payload Processing (PPP) component

Figure 3 (state machine)

above explanation can be summarized with figure 3 diagram of state machine.

1. Logging

TBD

## Communication handshakes:

Ladder Diagram: TODO

- Daemon to Management Server (MS)

-> POST http://localhost:9494/api/v1/1/endpoint/register

'{"arguments":

{

"component": "ipps", "guid":"392846198346912"

}

}'

<-- OK {"sts":"success", "message":"success"}, {"sts": "error", "message":"resource unavailable"}

-> POST http://localhost:9494/api/v1/1/endpoint/configure

'{"arguments":

{

"component": "ipps", "guid":"392846198346912", "config":"{ }"

},

"config":"{ }"

}'

<-- OK {"sts":"success", "message":"success"}, {"sts": "error", "message":"resource unavailable"}

- Management Server (MS) to Daemon

-> POST http://localhost:9494/api/v1/1/endpoint/alive

'{"arguments":

{

"component": "ms", "guid":"56e56546646454"

}

}'

<-- OK {"sts":"success"}, {"sts": "error", "message":"resource unaivaiable"}