PF_RING Tutorial

Alfredo Cardigliano <cardigliano@ntop.org>
Overview

• Introduction

• Installation

• Configuration

• Tuning

• Use cases
• Open source packet processing framework for Linux.

• Originally (2003) designed to accelerate packet capture on commodity hardware, using patched drivers and in-kernel filtering.

• Today it supports almost all Intel adapters with kernel-bypass zero-copy drivers and almost all FPGAs capture adapters.
PF_RING’s Main Features

• PF_RING consists of:

  • Kernel module (pf_ring.ko)
  • Userspace library (libpfring)
  • Userspace modules implementing multi-vendor support
  • Patched libpcap for legacy applications
Standard Drivers

- Standard kernel drivers, NAPI polling.
- 1-copy by the NIC into kernel buffers (DMA).
- 1-copy by the PF_RING kernel module into memory-map’ed memory.
PF_RING ZC Drivers

- Userspace drivers for Intel cards, kernel is bypassed.
- 1-copy by the NIC into userspace memory (DMA).
- Packets are read directly by the application in zero-copy.
PF_RING ZC API

- PF_RING ZC is not just a zero-copy driver, it provides a flexible API for creating full zero-copy processing patterns using 3 simple building blocks:
  - Queue
    - Hw Device Queue
    - Sw SPSC Queue
  - Pool: DMA buffers resource.
  - Worker: execution unit able to aggregate traffic from M ingress queues and distribute it to N generic egress queues using custom functions.
PF_RING ZC API - zbalance Example
• Code for aggregation and load-balancing using ZC:

```c
zc = pfring_zc_create_cluster(ID, MTU, MAX_BUFFERS, NULL);
for (i = 0; i < num_devices; i++)
   inzq[i] = pfring_zc_open_device(zc, devices[i], rx_only);
for (i = 0; i < num_slaves; i++)
   outzq[i] = pfring_zc_create_queue(zc, QUEUE_LEN);
zw = pfring_zc_run_balancer(inzq, outzq, num_devices,
                            num_slaves, NULL, NULL, !wait_for_packet, core_id);
```
Currently PF_RING natively supports the following vendors (1/10/40/100 Gbit)

Example:

- pfcount -i eth1  [Vanilla Linux adapter]
- pfcount -i zc:eth1 [Intel ZC drivers]
- pfcount -i nt:1  [Napatech]
- pfcount -i myri:1 [Myricom]
- pfcount -i exanic:0 [Exablaze]
Many modules, single API.
Overview

• Introduction

• Installation

• Configuration

• Tuning

• Use cases
• Two options for installing PF_RING:
  • Source Code (GitHub)
  • Packages
    ‣ Stable
    ‣ Dev (aka “nightly builds”)
• Download
  
  # git clone https://github.com/ntop/PF_RING.git

• Installation:
  
  # cd PF_RING/kernel
  
  # make && make install
  
  # cd ../userland
  
  # make && make install

• ZC drivers installation (optional):
  
  # cd PF_RING/drivers/intel/<model>/<model>-<version>-zc/src
  
  # make && make install

• Support for FPGAs (Napatech, Myricom, etc) is automatically enabled if drivers are installed.
Installation - Packages

- CentOS/Debian/Ubuntu stable/devel repositories at http://packages.ntop.org

- Installation:
  
  # wget http://apt.ntop.org/16.04/all/apt-ntop.deb
  
  # dpkg -i apt-ntop.deb
  
  # apt-get clean all
  
  # apt-get update
  
  # apt-get install pfring

- ZC drivers installation (optional):
  
  # apt-get install pfring-drivers-zc-dkms

- Support for FPGAs (Napatech, Myricom, etc) is already there.
Overview

• Introduction

• Installation

• Configuration

• Optimisation

• Use cases
Loading PF_RING

• If you compiled from source code:
  
  # cd PF_RING/kernel
  
  # insmod ./pf_ring.ko

• If you are using packages:
  
  # tree /etc/pf_ring/
  
  |-- pf_ring.conf
  
  `-- pf_ring.start
  
  # /etc/init.d/pf_ring start
Loading ZC Drivers

- ZC drivers are available for almost all Intel cards based on e1000e, igb, ixgbe, i40e, fm10k

- ZC needs hugepages for memory allocation, the pf_ring init script takes care of reserving them.

- A ZC interface acts as a standard interface (e.g. you can set an IP on ethX) until you open it using the “zc:” prefix (e.g. zc:ethX).
• If you compiled from source code:

# cd PF_RING/drivers/intel/<model>/<model>-<version>-zc/src
# ./load_driver.sh

• In essence the script loads hugepages and dependencies and load the module with:

# insmod <model>.ko RSS=1,1 [other options]

• You can check that the ZC driver is actually running with:

# cat /proc/net/pf_ring/dev/eth1/info | grep ZC

Polling Mode: ZC/NAPI
• If you are using packages (ixgbe driver in this example):

```bash
# tree /etc/pf_ring/
|-- hugepages.conf
|-- pf_ring.conf
|-- pf_ring.start
  `-- zc
    `-- ixgbe
      |-- ixgbe.conf
      `-- ixgbe.start
```

• Where:

```bash
# cat /etc/pf_ring/hugepages.conf
node=0 hugepagenumber=1024

# cat /etc/pf_ring/zc/ixgbe/ixgbe.conf
RSS=1,1
```
• RSS distributes the load across the specified number of RX queues based on an hash function which is IP-based (or IP/Port-based in case of TCP)
• Set the number of RSS queues using the insmod option or ethtool:

```
# ethtool --set-channels eth1 combined 4
# cat /proc/net/pf_ring/dev/eth1/info | grep Queues
TX Queues:  4
RX Queues:  4
```

• In order to open a specific interface queue, you have to specify the queue ID using the "@<ID>" suffix.

```
# tcpdump -i zc:eth1@0
```

Note: when using ZC, “zc:eth1” is the same as “zc:eth1@0”! This happens because ZC is a kernel-bypass technology, there is no abstraction (queues aggregation) provided by the kernel.
Indirection Table

• Destination queue is selected in combination with an indirection table:

\[
\text{queue} = \text{indirection\_table}[\text{rss\_hash}(\text{packet})]
\]

• It is possible to configure the indirection table using `ethtool` by simply applying weights to each RX queue.
# ethtool --set-channels eth1 combined 4
# ethtool -x eth1

RX flow hash indirection table for eth1 with 4 RX ring(s):

<table>
<thead>
<tr>
<th>Destination Queue ID</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 1 2 3 0 1 2 3</td>
</tr>
<tr>
<td>8</td>
<td>0 1 2 3 0 1 2 3</td>
</tr>
<tr>
<td>16</td>
<td>0 1 2 3 0 1 2 3</td>
</tr>
<tr>
<td>24</td>
<td>0 1 2 3 0 1 2 3</td>
</tr>
<tr>
<td>32</td>
<td>0 1 2 3 0 1 2 3</td>
</tr>
<tr>
<td>40</td>
<td>0 1 2 3 0 1 2 3</td>
</tr>
<tr>
<td>48</td>
<td>0 1 2 3 0 1 2 3</td>
</tr>
<tr>
<td>56</td>
<td>0 1 2 3 0 1 2 3</td>
</tr>
<tr>
<td>64</td>
<td>0 1 2 3 0 1 2 3</td>
</tr>
<tr>
<td>72</td>
<td>0 1 2 3 0 1 2 3</td>
</tr>
<tr>
<td>80</td>
<td>0 1 2 3 0 1 2 3</td>
</tr>
<tr>
<td>88</td>
<td>0 1 2 3 0 1 2 3</td>
</tr>
<tr>
<td>96</td>
<td>0 1 2 3 0 1 2 3</td>
</tr>
<tr>
<td>104</td>
<td>0 1 2 3 0 1 2 3</td>
</tr>
<tr>
<td>112</td>
<td>0 1 2 3 0 1 2 3</td>
</tr>
<tr>
<td>120</td>
<td>0 1 2 3 0 1 2 3</td>
</tr>
</tbody>
</table>
# ethtool -X eth1 weight 1 0 0 0
# ethtool -x eth1

<table>
<thead>
<tr>
<th>hash</th>
<th>destination</th>
<th>queue ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>48</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>56</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>64</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>72</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>88</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>96</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>104</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>112</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>120</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Overview

• Introduction
• Installation
• Configuration
• Tuning
• Use cases
Xeon Architecture
QPI

• QPI (Quick Path Interconnect) is the bus that interconnects the nodes of a NUMA system.

• QPI is used for moving data between nodes when accessing remote memory or PCIe devices. It also carries cache coherency traffic.
Memory

- Each CPU has its local memory directly attached.
- Accessing remote memory is slow as data flows through the QPI, which has lower bandwidth and adds latency.

E5-2687Wv4
9.6 GT/s QPI
76.8 GB/s RAM DDR4 2400
QPI latency: hundreds of nanosec

Example:
8.0 GT/s QPI - bandwidth 32 GB/s ~32 GB/s
PCIe

- Each node has its dedicated PCIe lanes.
- Plug the Network Card (and the RAID Controller) to the right slot reading the motherboard manual.
Memory Channels

- Multi-channel memory increases data transfer rate between memory and memory controller. You can use n2membenchmark as benchmark tool.

- Check how many channels your CPU supports and use at least as many memory modules as the number of channels (check dmidecode).
CPU Cores

• CPU pinning of a process/thread to a core is important to isolate processing and improve performance.

• In most cases dedicating a physical core (pay attention to hyper-threading) to each thread is the best choice for optimal performance.
Core Affinity

• All our applications natively support CPU pinning, e.g.:

  # nprobe -h | grep affinity
  [--cpu-affinity|-4] <CPU/Core Id> | Binds this process to the specified CPU/Core

• When not supported, you can use external tools:

  # taskset -c 1 tcpdump -i eth1
NUMA Affinity

- You can check your NUMA-related hw configuration with:
  - `lstopo`
  - `numactl --hardware`

- Configuring CPU pinning, usually the application allocates memory to the correct NUMA node, if this is not the case you can use external tools:

  ```
  # numactl --membind=0 --cpunodebind=0 tcpdump -i zc:eth1
  ```

- You can check your QPI bandwidth with:

  ```
  # numactl --membind=0 --cpunodebind=1 n2membenchmark
  ```
PF_RING ZC Driver NUMA Affinity

- PF_RING ZC drivers allocate data structures (RX/TX ring) in memory, setting NUMA affinity is important. You can do that at insmod:

  # insmod <model>.ko RSS=1,1,1,1 numa_cpu_affinity=0,0,8,8

- Or if you are using packages:

  # cat /etc/pf_ring/zc/ixgbe/ixgbe.conf
  RSS=1,1,1,1 numa_cpu_affinity=0,0,8,8
Traffic Recording - Wrong Configuration

1. NIC to memory
2. Processing in memory
3. Memory to Storage
Traffic Recording - Correct Configuration
Overview

- Introduction
- Installation
- Configuration
- Tuning
- Use cases
RSS Load Balancing

Network Card

Linux

Consumer Thread 0

PF_RING ZC

Core 0

Consumer Thread 1

PF_RING ZC

Core 1

Consumer Thread 2

PF_RING ZC

Core 2

Consumer Thread 3

PF_RING ZC

Core 3
RSS: When it can be used

• Flow-based traffic analysis (multi-threaded or multi-process) and all the applications where Divide and Conquer strategy is applicable.

• Examples:
  • nProbe (Netflow probe)
  • nProbe Cento
  • Suricata
  • Bro
• nProbe instances example with 4 RSS queues:

```bash
# nprobe -i zc:eth1@0 --cpu-affinity 0 [other options]
# nprobe -i zc:eth1@1 --cpu-affinity 1 [other options]
# nprobe -i zc:eth1@2 --cpu-affinity 2 [other options]
# nprobe -i zc:eth1@3 --cpu-affinity 3 [other options]
```
• Bro node.cfg example with 8 RSS queues:

# [worker-1]
type=worker
host=10.0.0.1
interface=ztc:eth1
lb_method=pf_ring
lb_procs=8
pin_cpus=0,1,2,3,4,5,6,7

This is expanded into ztc:eth1@0 .. ztc:eth1@7
RSS: When it can NOT be used

• Applications where packets order has to be preserved (also across flows), especially if there is no hw timestamping.

• For example in n2disk (traffic recording) we have to keep the original order for packets dumped on disk.
ZC Load Balancing (zbalance_ipc)
ZC Load Balancing: When it is useful

- When RSS is not available or not flexible enough (with ZC you can build your distribution function/hash)
- When you need to send the same traffic to multiple applications (fan-out) while using zero-copy
- When you need to aggregate from multiple ports and then distribute
• zbalance_ipc is an example of multi-process load balancing application:

```
# zbalance_ipc -i zc:eth1,zc:eth2 -c 99 -n 1,2 -m 1 -g 0
```

• Consumer applications example:

```
# taskset -c 1 tcpdump -i zc:99@0

# nprobe -i zc:99@1 --cpu-affinity 2 [other options]
# nprobe -i zc:99@2 --cpu-affinity 3 [other options]
```
Bro node.cfg example with 8 ZC queues:

```
# [worker-1]
type=worker
host=10.0.0.1
interface=zc:99
lb_method=pf_ring
lb_procs=8
pin_cpus=0,1,2,3,4,5,6,7
```

This is expanded into zc:99@0 .. zc:99@7
Other processing patterns

• Using the ZC API you can create any multithreaded or multi-process processing pattern. Pipeline example:

![Diagram showing ZC API usage with components like Packet Dispatcher, App A, App B, Packet Forwarder, PF_RING, and Core 0-3 with 1/10G connections.](image-url)
• Any hypervisor is supported: KVM, VMWare (Direct I/O), Xen, etc.
(Host) $ zpipeline_ipc -i zc:eth2,0 -o zc:eth3,1 -n 2 -c 99 -r 0 -t 2 -Q /tmp/qmp0

(VM) $ zbounce_ipc -c 99 -i 0 -o 1 -u
• ZC is a Kernel-Bypass technology: what if we want to forward some traffic to the Linux Stack?
Thank you!