Suricata Extreme Performance 调优

（带着难以置信的勇气……

第一章

介绍

本指南的工作是由 SuriCon 2016 启动的。

本指南代表并包含我们在与 Suricata 一起在商品/COTs硬件上进行 20Gbps IDSing 的高性能协调工作中的发现。

多年来，关于如何从 Suricata 获得最佳性能以及在什么条件下可能以及使用什么硬件，已经存在了很多问题/猜测和想法。

本指南尝试介绍一种从头开始的方法，并强调和描述 Suricata IDS 高性能调优的首要必要步骤。

Suricata 的性能有 4 个主要变量：

- Suricata 版本
- 流量类型
- 使用的规则
- 硬件功能

不过，当我们想将性能提升到一个新的水平时，不要低估作系统类型/内核版本/NUMA-NIC 位置/CPU L3 缓存。这些将在本文后面更详细地描述。理解和解决的最重要部分是，数据包在到达 Suricata 进行进一步处理之前需要跳转的跃点数量。

我们的设置

- 在长期持续的峰值中高达 20Gbit/s 的流量。大约 2 000 000 pkt/sec。
- Suricata 3.2dev（实际上是 3.2）

- 惠普 ProLiant DL360 Gen9
- 2 个双端口 X710 和 X520 - 使用的每张卡上各有一个端口。卡安装到单独的 NUMA 节点中。
- 来自 Intel 的 Ixgbe-4.4.6，来自 Intel 的 e40i 驱动程序，而不是来自内核的驱动程序。
- 禁用 RSS，每张卡一个队列，以避免数据包重新排序
- 4.4.0-38-generic #57~14.04.1-Ubuntu 及更高版本。较低版本有一个AF_Packetbug - 我们将在本文后面更详细地介绍它。
- 从 ET Pro 启用 21332 规则，10490 检查南北向，10842 检查内部流量（以及外部流量）。
- 128GB RAM，8 个 DIMM，每个插槽 4 个。
- 2 个 Intel（R） Xeon（R） CPU E5-2697 v3 @ 2.60GHz - 总共 28 个内核，启用 HT 并用于 56 个硬件线程。建议使用 Haswell，最少使用 Sandy Bridge。

我们最终不需要如此大的系统的所有资源来进行 20Gbps 检测。

第 2 章

宏伟计划

我们的宏伟计划可以总结为这样的 - 使用 CPU L3 缓存作为数据总线。

因此，重现本文结果的一个主要要求是具有 L3 缓存的 CPU。（例如 E5）

我们的意思是，让你的卡直接将数据包推送到每个套接字上的 L3 中，当内核和 Suricata 稍后想要处理这个数据包时，它已经在 L3 中，新鲜、温暖并准备好使用。随着我们在本文中的进展，您将看到我们的大部分工作都花在了确保上述内容是正确的，并且数据包在使用/处理之前不会从缓存中驱逐。

为什么我们如此关注 CPU L3 缓存?

当你的 CPU 想要处理数据时，它需要先读取数据。它可以从多个地方读取数据 - 各种级别的缓存、内存缓冲区，然后是 RAM 本身。每次它试图获取该数据但无法快速找到它时，都会发生 CPU 停顿（执行等待）。这意味着浪费了周期，你的 CPU 什么都不做，等待从另一个地方（例如 RAM）获取所需的数据。它引入了匿名高效/简化的 CPU 使用率。

在这项研究中，我们使用英特尔性能计数器监控工具测量了 IPS - 每秒指令数。在现代 CPU 上，您希望这个数字尽可能接近 4。我们从 0.7 IPC（当 CPU 大量等待数据时）变为

2.7 在我们高度优化的环境中。访问延迟示例取决于数据的位置 - 数字是近似的，Home Snoop 模式，您可以使用 EarlySnooping 在这里和那里节省几纳秒：

- 本地 L3 CPU 缓存 - 20ns

数据位于与处理它的 CPU 位于同一 NUMA 节点上的 L3 缓存上。

- 远程 L3 - 80ns

位于远程 NUMA 节点上的 L3 缓存上的数据，与 CPU 处理它相关。

- 本地 RAM - 96ns

位于 RAM 中的数据与处理它的 CPU 位于同一 NUMA 节点上。

- 远程 RAM - 140ns

位于远程 NUMA 节点上的 RAM 中的数据与处理它的 CPU 相关。

有趣的观察

从 RAM 获取数据所需的时间几乎是从本地 L3 缓存中获取数据的时间的 5 倍。不丢弃缓存至关重要。从远程 L3 缓存中获取数据所需的时间几乎与从本地 RAM 获取数据所需的时间一样长。NUMA 位置很重要。

在处理每秒数十 Gb 的网络时，您有 65 到几百纳秒的时间来处理单个数据包。

通过尽可能小的缓存未命中，我们可以实现端到端的平滑数据包处理，而不会出现可能导致数据包丢失的读取/获取延迟，从而影响 Suricata IDS/IPS 检查。

我们是如何做到的

- 双 CPU 服务器和每个插槽相同大小的所有 DIMM。不用说，如果每个插槽有四个内存通道，则必须使用 4 个 DIMM。

- 两个网络接口，位于单独的卡上，每个 NUMA 节点一个卡。NIC RSS、MQ、FDIR、ATR 保持禁用状态，以帮助避免数据包被 FDIR（流导向器）重新排序。

- 将所有内核与任何任务隔离开来，并使用内核 0 线程 0 作为内务处理内核。是的，正确 - 隔离内核，以便除了 Suricata 完成的工作外，没有任务在它们上运行。

- 隔离硬件 IRQ 处理和软件 IRQ 处理，并将其全部AF_Packetand固定在专用内核上 - 每个卡/NUMA 节点一个。上述所有任务通常都可以由单个内核处理。没有其他任何东西在这个内核上运行。

- 最后，我们运行固定到“剩余”隔离内核的 Sulicata。卡 0 将数据包发送到节点 0，从头到尾处理数据包，卡 1 将数据包发送到节点 1。我们在工作模式下使用 Suricata，从头到尾在同一个内核和线程上处理一个流。

• 每个缓冲区、环、描述符结构都尽可能小，以避免缓冲区膨胀并确保它保持在 CPU 缓存中（我们的生命线）。

数据包的生命周期

一个理想的世界。数据包的生命周期：

1. 数据包在网络上。卡将其提取到 FIFO 队列中。

2. 卡将数据包描述符和数据包本身发送到卡所连接的 CPU 的 L3 缓存。

3. 在某个时候，该数据会回写到同一 NUMA 节点上的主机内存。此时，数据包位于一个空闲缓冲区中。这就是人们所说的“硬件缓冲区”。除了 FIFOqueue 之外，卡上没有硬件缓冲区。我们还将卡将数据包写入“DMAarea”的内存区域称为“DMAarea”。

4. 卡发送或不发送中断，让 Linux 知道它应该开始使用 NAPI 池化数据包。Linux 确认该中断（如果已引发）并返回到 NAPI 轮询。一切都发生在专用于中断的核心上。

5. NAPI 从 DMA 区域（环形缓冲区，参见上面的 2 和 3）获取数据包，通过对底层 memorypages 执行魔术，导致 L3 命中并进入 SKB。驱动程序重用同一页面的一半，或分配一个新页面来替换已使用的页面。NAPI 将数据包交付给订阅者，如 AF_Packet。我们仍然处于“中断核心”上。

6. AF_Packet计算哈希值，并将数据包放入同一 CPU 上不同内核上相应工作线程的缓冲区中，从而导致 L3 命中。TPacket V3（AF_Packet Suricata 中的 V3）努力避免数据复制。

7. Suricata worker 时不时地切换到处理一个环中充满数据包的下一个块，使用简单的指针数学，而无需发出单个系统调用或进行复制。该机制旨在分摊每个数据包要完成的工作量 - 这就是 Suricata 批量访问它们的原因。在 Suricata 完成数据包后，它会对其进行标记AF_Packet以便可以重用底层页面。它无需在用户空间和内核空间之间迁移即可做到这一点。这对性能非常重要。
Suricata 没有自己的数据包缓冲区 - 内核缓冲区与用户空间共享。

这可以总结如下：

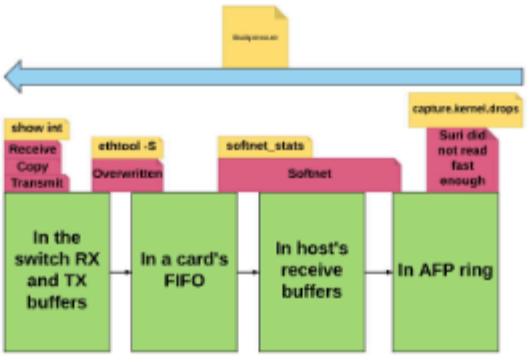
关于 AF_Packet 的常见误解

- AF_Packet很慢。使用 TPacket V2，尤其是 TPacket V3，可以在单个物理服务器上超过 20Gbit/sec。我们相信 40Gbit/sec 是可能的，我们只是没有足够的流量！！请给我们提供流量进行测试:)
如果您不相信。

- AF_Packet 表示每个数据包有一个中断，10Gbit/s 相当于 14Mpps。

首先，您的流量是混合的，平均帧大小约为 900-1000 字节，因此您不需要为 14Mpps 提供服务。其次，我们观察到 NIC 大部分时间都在禁用中断的情况下工作，我们在下面进行了解释。

- Linux 内核会复制数据包，每个数据包复制几次。


show int
Receive
Copy
Transmit
ethtool -S
Overwritten
softnet_stats
Softnet
capture.kernel.drops
Suri did not read fast enough
In the switch RX and TX buffers
In a card's FIFO
In host's receive buffers
In AFP ring

在这项研究中，我们花费了无数小时来了解从线路到 Suricata 的数据包路径。我们可以说 Linux 网络堆栈令人印象深刻（至少）。它避免了复制数据包，除非执行 acopy 比移动指针更便宜，例如，当数据包小于 SKB（套接字缓冲区）标头时。

- 专用卡是必要的，因为它们避免了 Linux 对每个数据包所做的复制数据包、内部、系统调用、用户和内核空间之间的转换。

中断不再是问题，正确配置的 Linux 系统可以很好地避免该列表的大部分内容。它尽可能接近将 DMA 数据包直接卡入 CPU 上的 L3，在那里数据包将被消耗。实际上，其中一些专用卡在主机 CPU 上作弊并做了大量工作，这项工作隐藏在二进制内核模块中。例如，Myricom 计算您所有内核的 RSS。一些卡确实可以在硬件中完成大部分工作（然后是一些），但您正在寻找 10 - 25 000 美元。相反，购买更多的服务器。这些硬件功能有很多不明显的限制，其中大多数都无法通过软件升级来帮助。您可以随时更改您的 Linuxkernel :)

第 3 章

优化步骤

在我们的设置中，我们决定将内核硬编码为某些职责，以避免缓存抖动，并确保我们的假设在大多数情况下是正确的。以下是它的工作原理：

- 内核 0 和 14 - 内务管理内核。每个 NUMA 节点一个。它们没什么特别的，只是第一个 CPU 的第一个内核的第一个线程和第二个 CPU 的第一个内核的第一个线程。

它只做通用的作系统职责，与我们的工作负载没有直接关系。ssh 或 audit 或各种内核线程都在这里运行。计时也是在 thiscore 上完成的。Suricata 流管理器线程也在这里运行。

- 核心 1 和 15 - IRQ 处理的硬件和软件部分以及 AFP 的大部分都在这里运行。

这个内核将在用户空间和内核空间之间频繁切换。由于它是负载最多的一对内核，我们建议不要在 HT 对等体上运行任何东西。

- 剩余颜色。

他们运行 Suricata 工人。如下图所示：

硬件

让我们从基础开始。每个 NUMA 节点使用一个网卡。单 CPU 是可以的，双 CPU 也可以。不要超过两个 CPU - 这些平台的扩展性很差。购买 2 或 4 台服务器，而不是一台具有 4 个 CPU 的服务器。它也更便宜。

使用 Intel E5 处理器，即使对于单个 CPU 配置也是如此。理想情况下 Haswellor 以后。他们有 E3 没有的 L3 缓存，而且缓存是性能的关键部分。不要买 E7，没有理由。虽然他们有更多的内核，但由于环形架构，它的扩展性不是那么好。此外，它们很昂贵（接受它们的服务器也是如此）- 购买两台 E5 服务器。

使用您能买到的最快的内存。每个通道有一个 DIMM，避免 2 个或更多 DPC（每个通道的 Dimm），并确保使用所有内存通道。是的，购买 8 个 16GB 比购买 16 个 8GB 更贵，但购买后者后，内存访问延迟会增加，频率和吞吐量会下降。

使用 Intel X710（推荐）或 Intel X520 卡。Mellanox 卡看起来也很有趣，我们还没有测试过它们。

确保每个卡都进入一个单独的 NUMA 节点，并连接到至少具有 x8 Gen 3.0 到 root 的 PCIeslot。避免在任何额外插槽中安装任何其他东西。

我的 NUMA 在哪里

安装 hwloc 软件包并创建一些漂亮的图表。确保在 BIOS/EFI 中启用了 acpi slit（见下文）。

在类似 Debian/Ubuntu 的系统上：

```
apt-get install hwloc
```

试一试：

```
lstopo --logical --输出格式 txt
```

对于上面的 ascii 艺术使用，如果你使用 libcairo 支持构建它，你可以这样做：

```
lstopo --logical --output-format png > 'hostname'.png
```

要查看您的卡连接到哪个 NUMA 节点：

```
cat /sys/class/net/<INTERFACE>/device/numa_node
```

要查看核心列表及其所属位置：

```
cat /sys/devices/system/node/node[X]/cpulist
```

要查看每个接口或 PCI 功能：

```
cat /sys/class/net/[接口]/device/numa_nodecat /sys/devices/[PCI 根]/[PCIe 功能]/numa_node
```

固件。 EFI 的。BIOS 的。

- 禁用 IOMMU（输入输出内存管理单元）

它提供的安全性很低，一些功能在我们的例子中没有任何用处，并且会给所有内存作带来巨大的延迟问题。

- 禁用 ASPM （Active State Power Managemen） 和 QPI （QuickPath Intercon-nect） 电源管理以及类似的 PCIe 节能状态。

我们在这里谈论的是每秒数十吉比特，数百万个数据包。你的卡会很忙。ASPM（与 C 状态相反）名声不佳，喜欢在最糟糕的时刻让 PCIe 总线“消失”。您的 CAR 想从 FIFO 向环形缓冲区发送数据包，因为 FIFO 已满？Ups，暂时没有 PCIe。数据包被覆盖。QPI 省电也是如此 - 好处不大，我们需要 QPI 始终处于活动状态。

- 禁用 VT-x 和 VT-d 以及 SR-IOV。

您不会使用它们，有时很难理解它们在静默中启用了什么，以及内核在看到它们时将启用什么样的解决方法 - 并且它们增加了攻击面。

- 禁用节点交错。

保持通道交错启用状态。您希望本地分配的内存保持本地，而不是同时在两个 NUMA 节点上分配。

- 启用 IOAT - Intel I/O 加速技术（DDIO - 数据直接 I/O 技术，DCA - 直接高速缓存访问）。

从技术上讲，DDIO 是更大的 IOAT“功能包”的一部分。它由两部分组成：

- 由专用 DMA 引擎（想想硬件 acceleratedmemcpy（） ）创建的加速副本 - 它不被网络堆栈使用，因为它具有巨大的延迟
- DDIO - 让您的卡将数据包直接发送到 L3 缓存。这就是我们想要的。

- 禁用某些类型的预取。

大多数预取机制都以类似 shotgun 的模式工作，预取视线中的所有内容，刷新缓存，并错误地显示它认为下一步需要的内容。DDIO 将执行此作。

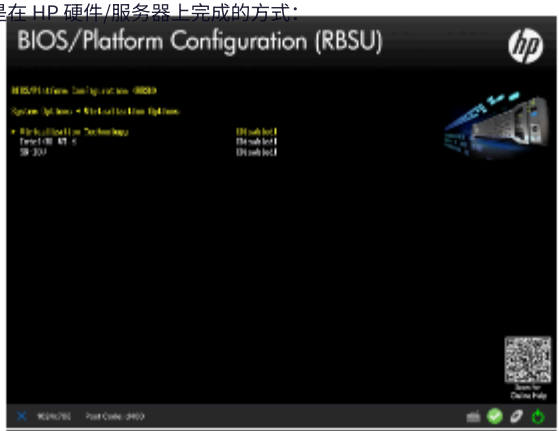
以下是我们禁用的预取程序类型：

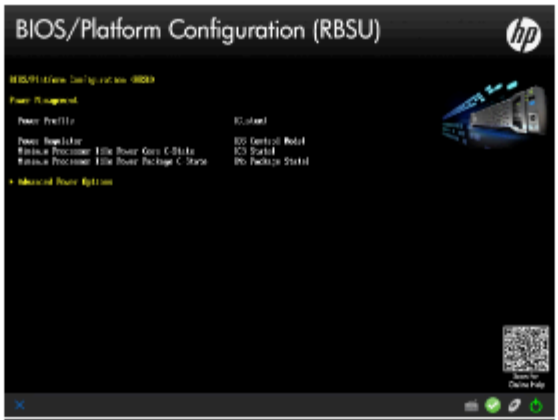
- HW 预取器
- 相邻扇区预取

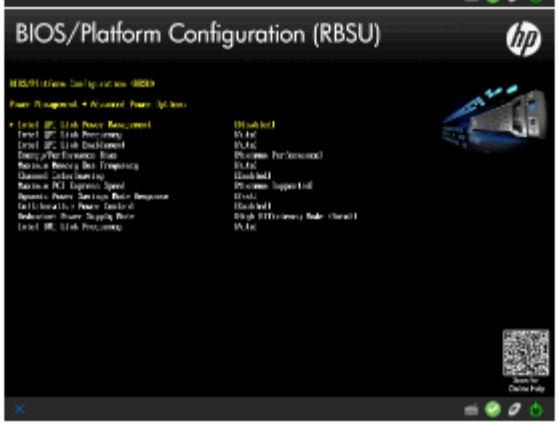
- DCU 流预取器
- 保持 C 状态和 P 状态处于启用状态。

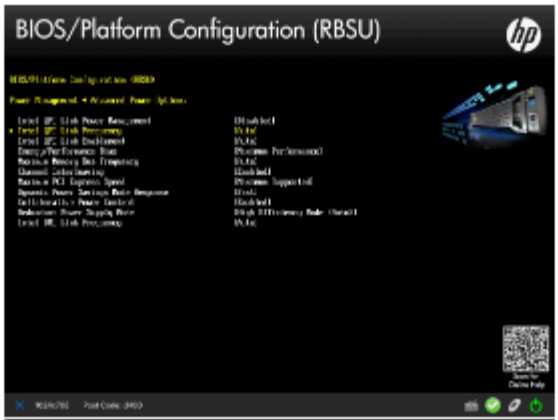
无论如何，Kernel 都可能会覆盖你的决定。你稍后会看到为什么我们需要它们。

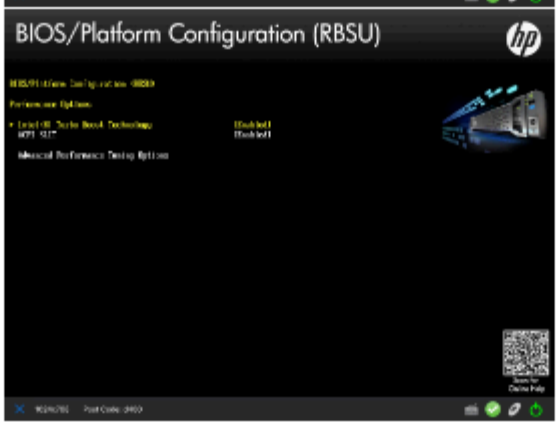
这是在 HP 硬件/服务器上完成的方式：




BIOS/Platform Configuration (RBSU)


BIOS/Platform Configuration (RBSU)


BIOS/Platform Configuration (RBSU)

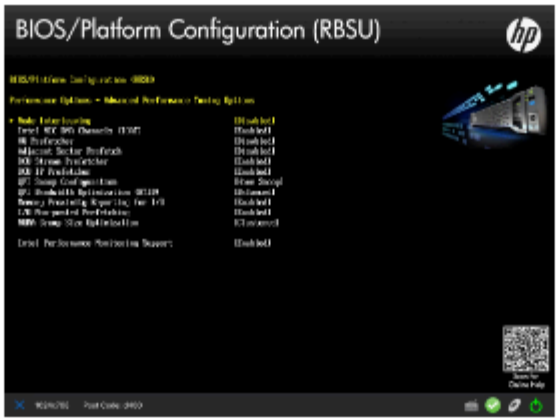

BIOS/Platform Configuration (RBSU)

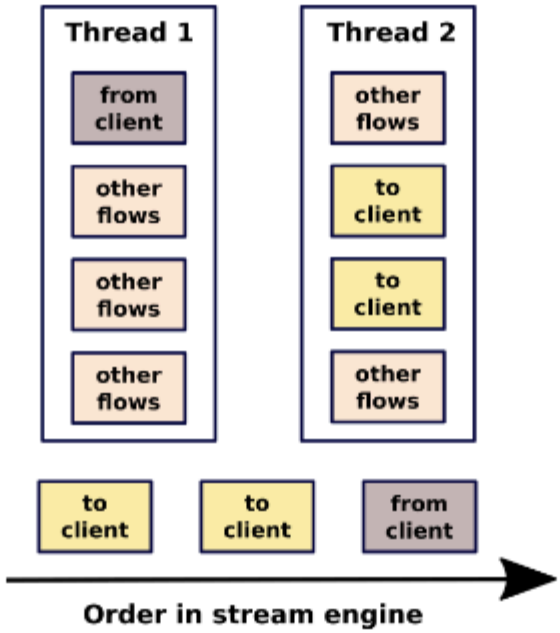
AF-Packet 对称哈希

对于某些内核版本，存在与对称哈希相关的错误，如修复它的提交中所述。

我们强烈建议您首先使用 Justin Azoff（Bro IDS 项目）的 excel-lent 验证工具验证您的内核没有错误。

此问题最好在此处进行解释，如下所示：


BIOS/Platform Configuration (RBSU)



内核 <=4.2 或 4.4.16+、4.6.5+ 或 4.7+ 是安全的。为了确保您的内核版本不存在问题，您可以使用上面的验证工具。

删除可能产生意外效果的应用程序

- IRQ余额

它试图平衡内核之间的中断，一直在改变亲和力，反过来破坏我们精心的设置。只需删除它，不要中继将任何内容列入白名单。

• Dhcpd、dhclient、dhcprelay、lldpd、iperf 和类似软件其中一些通过 libpcap 打开 AFP 套接字，其模式会使整个 AFP 处理路径发生偏移，因此效率要低得多。执行 ps -ef 并查看您可以删除什么

- 不要运行任何非绝对必要的应用程序。

使用正确的参数进行正确的 NIC 驱动程序

正如我们之前所说，多队列 NIC 功能会导致数据包重新排序。例如，您可以使用 ethtool -l eth0 进行检查。多队列、RSS、Flow Director 和

应用程序定向接收从来都不是为 IDS/IPS 设计的，而是用于扩展 Web 或文件服务器等工作负载。在这些情况下，它们表现出色。对于 IDS/IPS，它们必须保持禁用状态。

禁用所有可以禁用的内容。下面的 ITR 编号对我们来说效果很好，可能需要您进行调整。每个参数都有 4 个值，因为系统上有 4 个端口。

```
Modprobe IXGBE Mac=0,0,0,0 RSS=1,1,1,1,1 VMDK=0,0,0,0
\interrupta=12500,12500,12500 \FCoE=0,0,0,0,0 Lauro=0,0,0,0,0
VXLN_Rx=0,0,0,0：0ModProbe I40A
```

在 i40e 卡上，这些设置中的大多数不再使用 driver's moduleparameters 进行管理，而是使用 ethtool 进行管理。

要在 X520 和 X710 上强制执行单个 RX 队列：

```
ethtool -L <interface> 组合 1
```

使用以下方式进行验证：

```
ethtool -l <interface>
```

和

```
cat /proc/interrupts （猫 /proc/interrupts） |egrep <interface>（英语）
```

并查看是否有一行用于组合的 RxTx 中断，类似于以下内容：

```
p2p1-TxRx-0i40e-p3p1-TxRx-0
```

并且没有 -1、-2、-3 等。使用 ixgbe 查看内核日志：

```
dmesg |egrep ixgbe
```

查找如下所示的行：

```
[        9.200226] ixgbe 0000：04：00.0 eth0：  \
启用的功能：  RxQ：  1 TxQ：  1 DCA

[        9.220935] ixgbe：禁用多队列支持

[        9.220940] ixgbe：接收端缩放 （RSS） 设置为 1
```

对于 i40e：

```
[493063.749599] i40e 0000：82：00.1：RSS 计数/硬件最大 RSS 计数：              1/64
```

可能还有其他类似的中断，对于我们不使用的各种卡的功能，比如 management 或 virtualization，这些都可以安全地忽略。

本节的便捷词典：

- MQ - 多队列。我们只使用单个队列，因此数据包将由单个内核处理

- DDIO - 卡将使用数据包描述符和 packetdata 预热最低级缓存。这意味着，当内核想要获取数据包时，它已经在 L3 中，以后不必复制。

- VMDQ - 虚拟机设备队列。云不会与高性能环境混合使用。

- InterruptThrottleRate - 控制硬件中断。我们稍后从作系统对其进行了调整，因此这主要是为了展示多种方法中的一种。

- FCoE - 以太网光纤通道。我们不在乎，所以我们禁用了它。

- LRO - 必须与其他所有卸载功能一起禁用，以便 IDS 可以看到数据包的原样。如果启用，卡会将数据包合并为一个巨大的数据包，并将其发送到环，以避免复制开销。不幸的是，这会阻止 IDS 正确地重组流 - 有人说“LRO merges everythingin sight”

- Vxvlan_rx - 我们不使用，因此我们禁用了它。

请注意，没有 DCA，因为这是一种较旧的技术，现在被 DDIO 取代。使用 DCA，驱动程序必须管理将数据包预取到 L3 缓存中，而 DDIO 会自动执行。DCA 的唯一优势是它可以将数据包发送到远程 NUMA 节点的 L3 缓存，而 DDIO 始终发送到本地节点。如果这是一个问题，只需更正您的 CPU 固定或移动卡。

如果您的硬件支持 DDIO，则默认启用。Sandy Bridge 及更高版本使用 Intel 卡自动执行此作。对于其他供应商 - 请咨询供应商。

管理中断

我们不希望卡向我们发送太多的硬件中断。请记住，他们唯一做的就是启动 NAPI 处理循环，以便它开始处理数据包。由于我们的目标是高速 - NAPI 将在大部分时间运行，无论如何都服从内部机制以避免匮乏，当我们将处理隔离到一组内核时，这并不那么重要。

```
ethtool -C p1p1 adaptive-rx 在 rx-usecs 100ethtool -C p3p1 adaptive-rx 在 rx-usecs 100 上
```

虽然 ethtool 中有大量的中断管理开关，但现代网络卡几乎会静默地忽略所有这些开关。像这样的设置将在两个连续中断之间强制间隔。换句话说，它们将限制每秒的最大中断数。

减小 NIC 环描述符大小

```
ethtool -G p3p1 rx 512ethtool -G p1p1 rx 512
```

为什么？ 将其从 4096 降低使 DDIO 正常工作 - 在核心处理 IRQ（软 + 硬）上，L3 缓存未命中率从 16% 下降到 0.2%，对于 Suricataworker 线程核心下降到 0.46%。这意味着数据包由 L3 缓存传输。对 L1 和 L2 缓存没有影响。

较小的缓冲区意味着中断应该更频繁地运行，因此我们将卡无法发出中断的阈值时间从 80 次降低到 20 次。这并不会导致 CPU 使用率增加，因为大部分时间都花在禁用 hardwareinterrupts 的软中断中。

缓存未命中

环描述符大小对 DDIO 和 L3 命中的影响。在内核处理软件和硬件中断上测量。环描述符大小 4096 个缓冲区。每个缓冲区的大小为 2048 字节：

```
性能统计 \
-e LLC-加载，LLC-加载未命中，LLC-商店，LLC-预取 -C 1

“CPU（s） 0”的性能计数器统计信息：

1939117135      LLC 负载                                                (66.67%)
289305806       LLC 未命中#             占所有 LL 缓存命中率的 14.92% （66.67%）
356341541       有限责任公司商店                                        (66.67%)
<不支持>LLC 预取

       114.342786481 秒已用
```

环描述符大小 512 个缓冲区。在处理软件和硬件中断的核心上测量。每个缓冲区的大小为 2048 字节：

```
性能统计 \
-e LLC-loads，LLC-load-misses，LLC-stores，LLC-prefetches -C 1 休眠 60

“CPU（s） 1”的性能计数器统计信息：

659135009LLC 负载                                                       (66.67%)
1372230         LLC-load-misses # 所有 LL 缓存命中的0.21%               (66.67%)
124004599       有限责任公司商店                                        (66.67%)
<不支持>LLC 预取

       60.001419120 秒已用
```

在处理 Suricata 工作线程的核心上测量：

```
性能统计 \
-e LLC-loads，LLC-load-misses，LLC-stores，LLC-prefetches -C 10 休眠 60

“CPU（s） 10”的性能计数器统计信息：

1981142209        LLC 负载                                                (66.66%)
9128245LLC-load-misses # 占所有 LL 缓存命中的0.46%（66.67%）62828203LLC-stores
（66.66%）<不支持>LLC-prefetches


          60.006355454 秒
```

L3 缓存成为卡和 Suricata 之间的传输总线。请注意数据包处理的核心处理内核端的 LLC-load-misses 如何从 14.92% 下降到 0.21%。

禁用暂停帧

暂停帧将掩盖某些计数器（它们永远不会上升）并导致不准确的掉落统计数据

```
ethtool -A p1p1 rx off tx offethtool -A p3p1 rx off tx off
```

禁用卸载功能

```
IP 链路集 p1p1 promisc on arp off upip 链路集 p3p1 promisc
on arp off upecho 1 >
/proc/sys/net/ipv6/conf/p1p1/disable_ipv6echo 1 >
/proc/sys/net/ipv6/conf/p3p1/disable_ipv6ip 链路集 dev p1p1
mtu 9000ip 链路集 dev p3p1 mtu 9000


for i in rx tx autoneg tso ufo gso gro lro \txnocachecopy rxhash
ntuple sg txvlan rxvlan;\do ethtool -K <INTERFACE> $i 关闭 2>&1 >
/dev/null;完成;
```

确保它们被禁用：

```
ethtool -k $IFACE
```

并且：

```
ethtool -S p1p1 |egrep 'fdir_miss'
```

在我们的系统上为 0

```
NSM16 系列 ~/i40e-1.5.23/scripts       ethtool -k p3p1
p3p1 的功能：rx-
checksuming：  offtx-
checksuming：  off
        tx-checksum-ipv4：  offtx-checksum-ip-
        generic：  关闭 [修复]tx-checksum-ipv6：
        offtx-checksum-fcoe-crc：  关闭 [修复] tx-
        checksum-sctp：  关闭


scatter-gather：关闭
        tx-scatter-gather：  关闭tx-scatter-gather-fraglist：  关闭 [已修复]


tcp-segmentation-offload：关闭
        tx-tcp 分段：  offtx-tcp-ecn-
        segmentation：  offtx-tcp6-
        segmentation：  关闭
udp-fragmentation-offload：  关闭 [固
定]generic-segmentation-offload：  offgeneric-
receive-offload：  offlarge-receive-offload：
关闭 [固定]rx-vlan-offload：  offtx-vlan-
offload：  offntuple-filters：  offreceive-
hashing：  offhighdma：  onrx-vlan-filter：
onvlan-challenged：  关闭 [固定]tx-lockless：
关闭 [固定]netns-local：  关闭 [固定]tx-gso-
robust：  关闭 [固定]tx-fcoe-segmentation：  关
闭 [固定]tx-gre-segmentation：  ontx-ipip-
segmentation：  关闭 [固定]tx-sit-
segmentation：  关闭 [固定]tx-udp_tnl-
segmentation：  onfcoe-mtu：  关闭 [固定]tx-
nocache-copy：  offloopback：  关闭 [固定]rx-
fcs：  关闭 [固定]rx-all：  关闭 [固定]tx-vlan-
stag-hw-insert：  关闭 [固定]rx-vlan-stag-hw-
parse：  关闭 [固定]rx-vlan-stag-filter：  关闭
[修复]
```

```
l2-fwd-offload: 关闭 [修
复]busy-poll: 关闭 [修复]hw-tc-
offload: 关闭 [修复]
```

为什么？ 所有卸载函数都会以某种方式破坏数据包，通常会合并数据包并丢失标头信息。然后，IDS 无法正确地重组流，甚至根本无法重组流。

Pin 中断

他们需要在哪里：
p1p3 -> NUMA 1，CPU 1，核心数 1，因此 ID 为 141

p3p3 -> NUMA 1，CPU 1，核心数 15，因此 ID 14

方法如下：

```
set_irq_affinity 1 p1p1set_irq_affinity 14 p3p1
```

（set_irq_affinity 脚本位于 NIC 源驱动程序文件夹中）使用以下方式进行验证：

```
cat /proc/interrupts （猫 /proc/interrupts） |egrep <interface>（英语）
```

并观察中断在单个列中增长，对应于您将它们固定到的核心。在我们的系统上：

```
~/i40e-1.5.23/scripts       ./set_irq_affinity 15 页3p1
IFACE CORE MASK -> FILE======================p3p1
15 8000 -> /proc/irq/284/smp_affinity
```

```
~/i40e-1.5.23/scripts       ./set_irq_affinity 1 p1p1
IFACE CORE MASK ->
FILE======================p1p1 1 2 ->
/proc/irq/132/smp_affinity
```

常见误解 - 第 1 部分

- RSS 很好 - 使用它。

不，不要使用多个队列 - X520 哈希默认是不对称的（尽管它可以对称），FlowDirector 对数据包进行重新排序。这是由软件 IRQ 处理和 Linux 调度器之间的交互引起的。未来的工作可能会展示在更改哈希值后如何仍然使用 RSS 和多队列，并仔细地固定进程。我们相信并使用 Bro IDS 对其进行了简短的测试——对称哈希（可以使用 ethtool 设置），固定的进程没有给我们带来数据包重新排序的明显迹象。不过，这种设置的性能值得怀疑 - 每个工作内核都会在用户空间和内核空间之间进行多次转换。如果有的话，应该使用多个“中断处理内核”，与任何其他处理隔离，而此设置的其余部分保持原样。这可能是一个有趣的实验！另请参阅下一节。

- 固定所有内容并启用多队列是否有助于数据包重新排序？

我们不知道，可重复的测试是必要的。理论上它应该有效。在实践中，它的性能可能会更差，唯一有意义的配置是使用少量队列，每个队列都有一个专用核心，而不是通常将每个核心的时间划分为 Suricata 和 softirq 处理。

- 没有 RSS 的 X520 可以处理与使用 RSS 一样多的处理。

它不能。单个队列可以处理高达 2Mpps 的吞吐量，并且只能在非常快的 CPU（每个内核 3.4-3.6Ghz）上处理

- 现代 Linux 服务器使用 IO/AT 来加速数据复制。

他们可以，但如果有的话，只有大副本（如 256KB）才有优势。Ixgbe不使用它。

- IO/AT 自动透明地用于所有数据副本。

事实并非如此。驱动程序必须显式使用它，并且有 API 来实现这一点。“嘿 IOAT将这么多数据数据从这里复制到那里，好吗”——IOAT 确实这样做了。Linux 还做了其他事情。IOAT 返回并表示复制完成（或未完成）。

- 卡上有硬件队列。多队列卡具有多个缓冲区。

接收和发送队列存在于您的主机内存中，而不是卡上。它们只是环形缓冲区（附加了缓冲区的描述符，而这些缓冲区又附加了页面）。您的硬件队列实际上是指向多个缓冲区的描述符。每个队列有一个描述符。卡唯一的“缓冲区”是 FIFO，它只是为了购买足够的时间将传入的帧转换为 PCIe 数据包。如果禁用 FlowDiris，则 X520 具有 512KB FIFO。

如果我的 CPU 速度很慢怎么办？

在不同的内核上使用 RPS（Receive Packet Steering） 和 ACK 硬件中断。这样，软中断的一些处理可以移动到不同的内核，例如：

- 将帧从 DMA 区域发送到 SKB
- 将框架和 SKB 连接到AF_packet

如何正确地做到这一点？使用相同的 NUMA 节点，以便数据包位于 L3 中。这创建了一个非常有效的通信通道。在 2.0GHz CPU 和 netsniff-ng 上测试。

- 不要在多个 CPU/内核之间拆分硬 + 软 irq 或 RPS 处理，否则会发生数据包重新排序。

您需要回显位掩码，表示将处理软中断的核心，例如，处理来自核心 2 上的 p1p1 和核心 16 上的卡 p3p1 的数据：

```
cd /sys/class/netecho 4 > ./p1p1/queues/rx-
0/rps_cpusecho 10000 > ./p3p1/queues/rx-
0/rps_cpus
```

注意：使用 RSS 可以配置对称哈希，使用 RPS 无法控制哈希，并且可能会发生数据包重新排序。始终对 RPS 使用单个核心（线程）perNIC 卡。

内核隔离

添加到 grub（类似 Debian/Ubuntu 的系统）

```
audit=1 processor.max_cstate=3 intel_idle.max_cstate=3 selinux=0 \apparmor=0
nohz_full=1-13,15-55 isolcpus=1-13,15-55 \rcu_nocbs=1-13,15-55 mce=ignore_ce
```

我们还在此处添加了一些额外的参数，以便进行更通用的调整。

- mce=ignore_ce - 阻止 Linux 每五分钟启动一次“Machine Check Banks”轮询可纠正的错误，这可能会导致延迟峰值。ECC 和其他严重错误仍将被报告和处理 - 您将不会有抢先通知。
- audit=1 - 我们在此处审核每次执行，禁用会为每个系统调用提供一些性能提升。 轮到你了。
- Selinux=0， apparmor=0 - 增加每个系统调用两次的时间，以获得一点安全优势。
- C 状态将进一步描述。
- isolcpus - 隔离 CPU 1-13,15-55.从内核调度算法中删除这些 CPU。将进程移动到或移出“隔离”CPU 的唯一方法是通过 CPU 关联系统调用（这是 Suricata CPU 关联性发挥作用的地方）。

• nohz_full、rcu_nocbs - 省略仅运行 onetask 的 CPU 的计划时钟周期。取决于您的内核版本，以及是否在内核构建期间启用了它，则在 Ubuntu 上不受支持。

使用以下方式进行验证：

```
猫 /proc/cmdlinedmesg | 头
```

我们系统的示例：

```
BOOT_IMAGE=/vmlinuz-4.4.0-45-generic.efi.signed \root=UUID=dedcba7d-1909-4797-bd57-
663a423a6a2f \ro processor.max_cstate=3 intel_idle.max_cstate=3 selinux=0 apparmor=0
\mce=ignore_ce nohz_full=1-13,15-55 \isolcpus=1-13,15-55 rcu_nocbs=1-13,15-55
```

```
[        0.000000] 命令行：\
BOOT_IMAGE=/vmlinuz-4.4.0-45-generic.efi.signed \root=UUID=dedcba7d-1909-
4797-bd57-663a423a6a2f \ro processor.max_cstate=3
\intel_idle.max_cstate=3 selinux=0 apparmor=0 mce=ignore_ce \nohz_full=1-
13,15-55 isolcpus=1-13,15-55 rcu_nocbs=1-13,15-55
```

注意：带有单个进程的隔离内核不需要精确的时间保持，因此 Linux 不会在该内核上滴答作响 - 可能需要定期进行一些内务处理。调度器没有任务可以切换，为什么呢？每个滴答通常是用户和内核空间之间的过渡，部分缓存刷新与部分 TLB（翻译后备缓冲区）刷新（它们被标记，因此损害被摊销），然后返回用户空间。现在想象一下你的内核每秒这样做 1000 次，你明白了。最好经常打扰每个 NUMAnode 的单个内核，然后打扰所有内核。在我们的系统上，TLB 命中率为 100%。

拥有一个专用核心，用于硬 + 软 IRQ 加上系统任务以及任何日志记录并在剩余部分上运行工作人员。比尝试在硬 IRQ + 软 IRQ + Suricata + 系统剩余部分之间共享核心效果更好。经过测试和验证 - 从您的系统中窃取核心。

内核之间的不断迁移会导致缓存未命中 - 数据包位于 core5 的 L1 中，但我们刚刚迁移到内核 8。上下文切换部分刷新缓存和 TLB，因此我们不仅需要将 Suricata worker 固定到内核，还需要防止其他任何东西在 worker 专用内核上运行。

引脚 IRQ

这就是所谓的 - 内务处理。将所有 IRQ 固定到核心 0，线程 0：

```
for D in $（ls /proc/irq | egrep -v '<接口 1>|<接口 2>'）
```

```
的
        如果 [[ -x “/proc/irq/$D” && $D ! = “0”
        ]]thenecho $Decho 1 >
        /proc/irq/$D/smp_affinityfi


    做
```

将 rcu 任务（如果有）固定到核心 0、线程 0、NUMA 0：

```
对于 i 在 ’pgrep rcu’ ;do taskset -pc 0 $i ; 做
```

更多内核线程固定到核心 0、线程 0：

```
echo 1 > /sys/bus/workqueue/devices/writeback/cpumaskecho 0 >
/sys/bus/workqueue/devices/writeback/numa
```

注意，由于不明显的原因，“isolcpus” 参数与使用 cgroups 或 taskset 手动移动进程不同。做后者实际上会损害性能。一些内核线程需要用 taskset 移动，比如 rcu，这很好。

确保隔离配置正确：

```
ps -eHO psr --sort=psr
```

注意第二列 - 它应该只列出所有用户空间进程和大多数内核线程的 'housekeeping' 核心。

绿色时 Turbo 模式

这怎么可能呢？您的 CPU 具有多种性能增强技术 - 自我超频和省电 - 它们都可以协同工作。

- P 状态 - 在某些限制内更改频率 - 对于 2.6GHz（标称）CPU，在 1.6GHz 和 3.6GHz 之间。
- C 状态 - 禁用 CPU 的某些部分。

你问我为什么在乎？为了省电。省电使 maximumTurboBoost 频率能够处理流量高峰。这是它的工作原理。某些内核速度减慢的 CPU 或内核部分关闭的 CPU 可以节省电量。这意味着电压较低是可以的，它会变得更凉爽。当它较低时，一些内核可以超频一段时间，然后再次回到较低的速度以冷却，然后在必要时超频，依此类推。

为什么？ 这与 IDS 的通常流量模式完美配合 - 您不太可能在 100% 的时间内拥有最大流量（如果您这样做，这些设置将不起作用，因此启用它们是安全的）。流量来来去去，突发发生，如果单个内核的流量饱和而其他内核处于空闲状态，我们可以允许最繁忙的内核超频以处理该流量。

缺点是 - CPU 需要时间才能进入任何 C 状态，然后再返回。这就是为什么我们发现（通过反复试验）您希望 CPU 达到 C3 状态（而不是 C7 状态）。将 CPU 限制为 C1 不允许它在 TurboBoost 中达到最大速度。需要注意的是，CPU 只有在空闲足够长的时间时才会进入 C 状态。繁忙的核心将保持在 C0 状态。

```
对于 i in {0..55};do taskset -c $i cpupower -c $i frequency-set -g performan
```

使用以下方式进行验证：

```
对于 i 在 {0..55};do cpupower -c $i frequency-info; 做;
```

还要确保使用 Intel pstate 而不是 acpi pstate - acpi 将一次为所有内核设置频率。

```
dmesg |egrep intelcpupower 频率信息 |egrep 驱动程序

  司机：intel_pstate
```

在我们的系统上：

```
分析 CPU 0：
 驱动程序： intel_pstateCPUs 运行相同的硬件 频率： 0需要通过软件协调其频率的
 CPU： 0最大转换延迟： 0.97 毫秒.硬件限制： 1.20 GHz - 3.60 GHz可用 CPU频率调节
 器： 性能， 省电当前策略： 频率应在 1.20 GHz 和 3.60 GHz 之间。
                    调控器 “performance” 可以决定在此范围内使用哪个速度。
 当前 CPU 频率为 3.10 GHz（通过调用硬件断言）。boost 状态支持：
       支持： 是有效： 是
```

对于 Turbo 模式，您必须注意有多少个内核可以以什么样的频率运行。对于我们的 CPU：

```
1-2 核 - 3.6GHz3 核 -
3.4GHz4 核 - 3.4GHz5 核
- 3.2GHz所有核 3.1Ghz.最
大观察到 3.3Ghz。
```

如果执行 AVX（密集型矢量任务），则为 3.3/3.1/3/2.9。在这样的系统上，AVX 将被 hyperscan 使用，因此 Suricata 内核的频率可能低于执行硬件和软件 IRQ 的内核。

常见误解 - 第 2 部分

- 您应该禁用所有 C 状态，否则您的 CPU 将无法全功率运行。

不，C 状态是空闲状态 - 当内核无事可做时激活。CPU 不会强迫自己进入 C 状态（前提是没有温度问题）。在正常作期间唯一改变的状态是 P 状态 - 它会改变 CPU 运行的频率。

- 限制 C 状态（或禁用它们），因为在它们之间切换会花费大量时间。

在状态之间切换会产生一些成本，但它们允许内核冷却并收集热余量，并在必要时切换到 Turbo 模式。可以将 C 状态限制为 C3，从而允许完整的 Turbo 模式 - 所有内核为 3.0GHz 或

3.3 在少数情况下。使用允许达到完全 Turbo Boost 的最小数量的 C 状态 - 例如，放弃 C6。

- Ark.intel.com 会告诉您有关 CPU Turbo 的所有信息。

不，它不会 - 找到英特尔至强®处理器 E5 v3 产品系列处理器规格更新，有一个表格“Turbo bins”，它会告诉你 CPU 可以超频其内核多高，与其中有多少个内核处于活动状态。

Suricata 相关配置

由于我们将 Suricata 与 AF_Packet 一起使用，因此关键部分是：

- 将 Suricata 工作线程固定到隔离的 CPU （请参阅上面的核心隔离）-CPU 关联性

- 启用新的（在3.2dev中）本地旁路功能 — 如果相应的流是本地旁路的，则它只需跳过所有流、检测和输出，数据包在IDS模式下直接传出，并在IPS模式下判定。

本地旁路

什么是本地旁路
“本地旁路的概念很简单：Suricata 读取数据包，对其进行解码，在流表中进行检查。如果相应的流是本地旁路的，那么它只需跳过所有流、检测和输出，数据包在 IDS 模式下直接传出，在 IPS 模式下 toverdict。

一旦流被本地绕过，它就会应用特定的超时策略。Idea是我们无法干净地处理流的末尾，因为我们不再执行 streamingreassembly。因此，Suricata 可以在没有看到数据包时超时流。由于流应该真正处于活动状态，因此我们可以设置一个超时，该超时时间短于

established 超时。这就是为什么默认值等于紧急 establishedtimeout 值的原因。

本地旁路 conf：

```
流：
 memcap：12gbprealloc-
 sessions：200000checksum-
 validation：否                          # 拒绝错误的 csum
 内联：否                                # 无内联模式
 bypass： yes重新组装：

    内存容量：24GB深度：1MB
```

AF 数据包

AF 数据包：

```
# Linux 高速采集 supportAF-packet：

- interface： plpl# 接收线程数。“auto” 使用核心线程数： 11# 默认 clusterid。AF_PACKET将根据 flow.cluster-id： 99# 默认AF_PACKET集群类型对数据包进行负载均衡。AF_PACKET可以按流进行负载均衡或#这仅支持 Linux 内核 > 3.1# 可能的值为：#* cluster_round_robin：循环负载均衡#* cluster_flow：给定流的所有数据包都发送到同一个 socke#* cluster_cpu：所有在内核中被 CPU 处理的数据包都发送到 #* cluster_qm：所有通过网卡链接到 RSS 队列的数据包都是 se#socket。至少需要 Linux 3.14.#* cluster_random：数据包被随机发送到套接字，但带有 equ#至少需要 Linux 3.14.#* cluster_rollover：内核在套接字之间轮换，将每个套接字填充到下一个套接字#。至少需要 Linux 3.10.# 在大多数机器上cluster_flow推荐模式，cluster_cpu 或 clu# 使用 RSS 的采集卡（需要 cpu 亲和性调整和系统 irqcluster-type： cluster_flow# 在某些碎片情况下，无法计算哈希值。如果 “defrag” i# 为 yes，内核将在发送 thdefrag 之前进行所需的碎片整理： yes# 在 Linux 内核 3.10 之后，可以激活 rollover 选项：# full 然后内核将在下一个套接字上发送数据包，房间为 avai# 可以最小化数据包丢弃并增加单个 i#rollover：yes
```

要使用 AF_PACKET 的环功能，请将 'use-mmap' 设置为 yesuse-mmap： yes# 锁定内存映射以避免它被交换。小心，过度订阅# 你的系统#mmap-locked，yes# 使用 TPacket_v3，捕获模式，仅当 user-mmap 为 truetpacket-v3 时才激活tpacket-v3： yes# 将根据线程的 max_pending_packets 和 numb# 计算环大小。你可以手动设置环的大小（以数据包的数量为单位）#以下值。如果你正在使用 flow cluster-type，并且有非常#密集的单流，你可能想独立设置环的大小# of threads-ring-size： 400000# 块大小仅供tpacket_v3使用。它应该设置为一个高值 en# 相当数量的包。大小以字节为单位，因此请考虑您的 MT# 是 2 的幂，并且它必须是页面大小的倍数（4096）.block-size： 393216# tpacket_v3 block timeout： 如果打开的块在 block-timeout 毫秒后被 # 填充，则会传递给用户空间。#block-timeout： 10# 在繁忙的系统上，这有助于将其设置为 yes 以从 pack# 阶段恢复。这将导致一些数据包（最多一次环刷新）是非#use-emergency-flush： yes# recv 缓冲区大小，增加值可以提高性能#buffer-size： 1048576##buffer-size： 262144# 设置为 yes 以禁用混杂模式# disable-promisc： no# 为接口选择校验和验证模式。在捕获的那一刻#，由于#卸载到校验和计算的网卡，一些数据包可能具有无效的校验和。# 可能的值是：#- kernel，使用内核为每个数据包发送的指示（默认）#- yes，强制校验和验证#- no：校验和验证被禁用#- auto，Suricata 使用统计方法来检测何时使用#checksum off-loading。# 警告：'checksum-validation'必须设置为 yes 才能将任何 validatio#checksum-checks： kernel# BPF 过滤器应用于此接口。pcap 过滤器语法应用 her#bpf-filter： port 80 或 udp# 您可以使用以下变量来激活 AF_PACKET tap 或 IPS mo# 如果 copy-mode 设置为 IPS 或 tap，则进入当前#接口的流量将被复制到 copy-iface 接口。如果设置了 'tap'，# 复制完成。如果设置了 'IPS'，则不会复制与 'drop'作 # 匹配的数据包。

```
  #copy 模式：ips#copy-iface：eth1

# 将默认值放在这里。这些将用于上面列表中不是 # 的接口。

- interface： p3p1threads：
11cluster-id： 98use-mmap：
yestpacket-v3： yesring-size：
400000block-size： 393216#buffer-
size： 1048576##buffer-size：
262144# 128KB beforecluster-
type： cluster_flow



- 接口： p1p1线程： 11集群 ID：
99use-mmap： yestpacket-v3：
yesring-size： 400000block-size：
393216#buffer-size：
1048576##buffer-size： 262144集群
类型： cluster_flow



- 接口： p3p1线程：
11cluster-id： 98use-
mmap： yestpacket-v3：
yesring-size：
400000block-size：
393216
```

线程：

```
线程：
  set-cpu-affinity： 是 cpu-affinity：

        - management-cpu-set：cpu：
        [ 0,28,14,42 ]mode： “平
        衡”prio：default： “低”


        - worker-cpu-set：cpu： [“2-13”，“16-27”，
        “30-41”，“44-55”]mode： “独占”
```

普里奥：
默认值： “high”

注意：NIC、CPU 和 Suricata 工作线程必须驻留在同一个 NUMAnode 上。在上面的例子中，我们有 p1p1 和 p3p1 在不同的 NUMA 节点上，因此使用 CPU 亲和性相应地传播工作线程。

注意：在上面的配置中，cluster-id 也会针对接口 99/98/99/98 进行轮换

AF 数据包内存消耗计算

详细描述和细分可以在 这篇文章 中找到
内存方程式（以下所有数字均以字节为单位）：

```
<number_of_total_detection_threads> *<
 （（780）+（default_packet_size））> *<max-
pending-packets>

+

<defrag.memcap >

+

<host.memcap 中>

+

<ippair.memcap >

+

<flow.memcap >
```

```
+

<number_of_threads> *<216（TcpSession 结构为 192 字节，PoolBucket 为 24 字节）>
*<prealloc-sessions>


+

[每个 AF 数据包接口已启用]<af-
packet_number_of_threads>*<ringsize>*<
（（780）+（default_packet_size））>


+

<stream.memcap 中>

+

<stream.reassembly.memcap>

+

<app-layer.protocols.dns.global-memcap>

+

<app-layer.protocols.http.memcap >

=Suricata 估计的总内存消耗量 （+/- 1-2%）
```

常见误解 - 第 3 部分

- 必须在 BIOS 中禁用 Linux 上的超线程，因为它们会影响性能。

超线程实际上有助于提高性能，最大限度地减少整个内核空闲等待数据包的时间。谁知道呢，也许可以在 core 等待数据包 143 时处理 nr 205 数据包？做你的测试，我们不需要用 HT 来达到 20Gbit/sec，也许它对你来说真的很好。值得一试。

- 在 Linux 上，启用 HT 时，每个其他“内核”都是超线程。在内核 2.6 上也是如此。从 3.x 和 4.x 开始，Linux 首先列出第一个 CPU 的第一个 N 线程，然后列出第二个 CPU 的第一个 N线程，然后返回到第一个 CPU（如果机器有两个 CPU）。

- 给定一对 0-28（同一内核上的两个线程），0 是真正的内核，28 是超线程，因此您应该避免在 28 上调度任何内容，因为它只会在 0 空闲时运行。

不，0 和 28 从一开始就是超线程，哪个线程可以访问哪些资源是内部处理器状态的函数 - 没有一个超线程是受欢迎的。这背后的通用想法是 - “如果超线程 0 不使用 executionunit A，那么超线程 28 也许可以？

- 超线程将核心资源一分为二，导致性能降低。

实际上，如果您不使用超线程，每个内核上都有一些未使用的重复资源。

- NUMA 串扰是错误的，因为 CPU 之间的带宽有限。

QPI （QuickPath Interconnect） 上有足够的带宽。问题出在其他地方 - 在频繁的缓存未命中中 - NIC 卡已经用数据预热了 CPU 0 上的 L3，但 CPU 0 上的 L3 需要通过 QPI 获取相同的数据包，从而导致停顿。

数据包丢弃

数据包可以在多个位置丢弃。监控所有这些数据包非常重要，否则您将无法准确描述数据包丢失，从而导致错过事件。

- 如果使用 span 端口，则可能会发生大量数据包丢失，尤其是在 Juniper 和较旧的 Cisco 上，通常为 10-20%。新的 Cisco 交换机将数据包丢失率保持在低个位数，甚至低于 1%。

- 如果使用数据包代理，例如 Arista、Netoptics / Ixia xStream / xDirector、Gigamon 等，则在输入缓冲区或输出缓冲区

- 如果数据包格式错误 （ethtool -S），则在卡的 MAC 层

- 在卡的 FIFO 缓冲区（rx_missed且无缓冲区计数）处，如果 RX 缓冲区中没有位置，则会导致 FIFO 内容被覆盖

- 在软中断层 - 内核无法足够快地从 RX 缓冲区提取数据包并进入 SKB 和 AF_Packet - /proc/net/softnet_stat

- 在AF_Packet，如果数据包无法足够快地移动到 mmaped 环
  - getsockopt（ptv->套接字、SOL_PACKET、PACKET_STATISTICS、&kstats、&len）

- 最后，如果数据包的 CRC 无效，Suricata 可以丢弃一个经过所有折磨的数据包到达这里

Suricata 仅显示最后部分，在大多数情况下，那里的任何下降都很可能是其他地方发生的下降的间接影响。

您可以（并且应该）在多个位置监控这些下降。

卡的 FIFO 缓冲器 drops/losss：

```
ethtool -S p1p1 |egrep 'rx_dropped|rx_missed|rx_packets|错误'
```

并观察 rx_dropped 或 rx_missed 计数器的增长与否（取决于卡片）。

SoftIRQ 的：

```
cat /proc/net/softnet_stat
```

遗憾的是，值是十六进制的，并且仅记录在内核源代码中。读取 kernel/net/core/net-procfs.c for columns 描述中的内核函数 kernel softnet_seq_show（）。在我们写这篇论文的那一刻，列是这样的：

1. 处理的帧总数
2. 丢帧数
3. softirq 有更多帧要处理但用完了预配置时间或帧数太大的次数。如果这种情况持续增长，那么增加 netdev_budget 可能会有所帮助（如下所示）。

4. 零
5. 零
6. 零
7. 零
8. 零（认真）
9. 当传输路径尝试获取设备锁定时，发生多次冲突。这里不重要。

10. CPU 间中断数，用于在远程 CPU 上启动积压队列的处理，仅由 RPS 和 RFS 使用，默认情况下不启用

11. 仅供 RFS 使用，不在此处使用

还有一个脚本可以实时显示和翻译统计数据，我们推荐该脚本 - 称为 softnet_stats.pl。

要增加预算（如有必要），请执行以下作：

```
sysctl -w net.core.netdev_budget=3000
```

（默认为 300）如果不查看驱动程序代码，就无法判断这意味着什么，但默认情况下，softirq 循环在处理 ixgbe_poll（） 或类似函数的 300 / 64 ~ 5 次运行后退出，默认情况下，它尝试一次将 64 个数据包出列，因此大约有 320 个数据包。还有另一个限制 - 2 个 jiffies，因此在 1000Hz 系统上为 2ms（这是默认值）。

但是 ifconfig 丢弃计数器正在增长：忽略 ifconfig - 它会读取难以解释的计数器，并在最令人惊讶的地方增加。以下是最常见的原因（我们相信所有原因，但列出一个完整的列表意味着读取整个网络堆栈）：

- 软网积压已满（但我们已经描述了详细的故障排除）

- 错误的 VLAN 标签（您确定您在 promisc 开启的情况下运行吗?

- 以未知或未注册协议接收的数据包（再次 promisc 或网络上的 strangeframes ，请咨询您的网络管理员）

- 当服务器仅配置为 IPv4 时，IPv6 数据包 - 不应该是这种情况在这里，我们运行 promisc 并只是嗅探所有内容

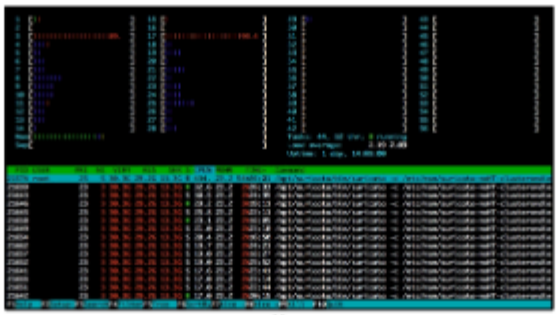
第 4 章

结论要点

- 在轮到 Suricata 处理之前，缓冲区中的数据包丢弃/丢失或覆盖可能发生很多地方。
- 为实现高性能而进行的 Suricata 调整是一个过程，而不是从某个地方复制/粘贴的配置。
- 调优本身是针对整个系统完成的 - 而不仅仅是 Suricata。
- 在订购该硬件之前进行测试和分析。

在 20 Gbps 持续峰值以下的调谐系统上的性能：

在 20 Gbps 持续峰值以下的调谐系统上的丢包数：



SURICON


0.00137%
capture.kernel_packets | Total | 18887981902
capture.kernel_drops | Total | 260649
decoder.pkts | Total | 18880400881
On 20Gbps

在暴民中，我们信任。

第 5 章

作者

关于作者：

米哈尔・普尔津斯基 （@MichalPurzynski）

- 威胁管理，Mozilla

彼得・马涅夫 （Peter Manev） （@pevma）

- Suricata 核心团队
- 首席 QA 和培训讲师
- Stamus 网络
- Mobster 布道者

第 6 章

谢谢

没有他们，本指南就不可能完成：

- Mozilla（时间、交通、硬件）
- 英特尔 - 亚历山大・杜伊克
- Eric Leblond（@Regit – Suricata cpu 关联性/本地旁路）
- 丹尼尔・博克曼 （netsniff-ng， AFPacket）
- AFPacket 的 Dave Miller： -）
- SuriCon 2016！！