Sugon

中科曙光 NetFirm 智能加速卡

用户手册

声 明

本手册的用途在于帮助您正确地使用曙光服务器产品(以下称“本产品”)，在安装和第一次使用本产品前，请您务必先仔细阅读随机配送的所有资料，特别是本手册中所提及的注意事项。这会有助于您更好和安全地使用本产品。请妥善保管本手册，以便日后参阅。

本手册的描述并不代表对本产品规格和软、硬件配置的任何说明。有关本产品的实际规格和配置，请查阅相关协议、装箱单、产品规格配置描述文件，或向产品的销售商咨询。

如您不正确地或未按本手册的指示和要求安装、使用或保管本产品，或让非曙光授权的技术人员修理、变更本产品，曙光将不对由此导致的损害承担任何责任。

本手册中所提供照片、图形、图表和插图，仅用于解释和说明目的，可能与实际产品有些差别，另外，产品实际规格和配置可能会根据需要不时变更，因此与本手册内容有所不同。请以实际产品为准。

本手册中所提及的非曙光网站信息，是为了方便起见而提供，此类网站中的信息不是曙光产品资料的一部分，也不是曙光服务的一部分，曙光对这些网站及信息的准确性和可用性不做任何保证。使用此类网站带来的风险将由您自行承担。

本手册不用于表明曙光对其产品和服务做了任何保证，无论是明示的还是默示的，包括(但不限于)本手册中推荐使用产品的适用性、安全性、适销性和适合某特定用途的保证。对本产品及相关服务的保证和保修承诺，应按可适用的协议或产品标准保修服务条款和条件执行。在法律法规的最大允许范围内，曙光对于您的使用或不能使用本产品而发生的任何损害(包括，但不限于直接或间接的个人损害、商业利润的损失、业务中断、商业信息的遗失或任何其他损失)，不负任何赔偿责任。

对于您在本产品之外使用本产品随机提供的软件，或在本产品上使用非随机软件或经曙光认证推荐使用的专用软件之外的其他软件，曙光对其可靠性不做任何保证。

曙光已经对本手册进行了仔细的校勘和核对，但不能保证本手册完全没有任何错误和疏漏。为更好地提供服务，曙光可能会对本手册中描述的产品之软件和硬件及本手册的内容随时进行改进和/或修改，恕不另行通知。如果您在使用过程中发现本产品的实际情况与本手册有不一致之处，或您想得到最新的信息或有任何问题和想法，欢迎致电我们或登陆曙光服务网站垂询。

商标和版权

目录

第一章 加速卡简介

中科曙光 NetFirm 系列智能加速卡是曙光针对高速网络数据处理类应用，专门定制研发的加速卡，属于专用设备，本章将简单介绍中科曙光 NetFirm 系列智能加速卡的产品功能、技术指标和系统要求。

1.1 产品功能

中科曙光 NetFirm 系列智能加速卡具有以下功能：

- 支持丰富的网络接口，包括 POS(2.5G、10G)、ETHERNET；支持千兆和万兆
- 支持硬件的多网口链路层数据聚合，多网口收包时可保证报文在线路上的顺序
- 支持硬件的多核服务器平台负载均衡，可根据用户配置的分流方法把报文传输到每个 CPU 核心的缓冲区内
- 支持多应用业务处理，可实现多个应用对同一个报文缓冲区处理，以及可实现多个应用对不同报文缓冲区处理
- 支持硬件直接转发和多应用多线程高速转发报文，可实现实时的敏感数据处理
- 支持在线升级加速卡硬件芯片，可方便实现加速卡硬件功能的升级和扩展
- 支持硬件基于以太协议、传输层端口的报文采样和基于五元组的报文分类
- 支持硬件发送连接阻断报文和日志报文。
- 支持硬件实现的 tcp 会话状态管理功能。
- 支持协议栈功能，可实现将管理业务和数据业务分开处理。
- 支持 IPV4 和 IPV6 双栈报文处理。
- 支持链路保护，在设备异常时支持数据透传转发

1.2 技术规格

中科曙光 NetFirm 系列智能加速卡的技术规格如表 1-1 所示：

表 1-1 加速卡技术规格表

	NetFirm-C304	NetFirm-C313	NetFirm-C320
网口接入	4 个千兆 Ethernet	3 个千兆 Ethernet	2 个万兆 POS /2 个 2.5GPOS
	4 个 2.5GPOS	1 个万兆 Ethernet 1 个万兆 POS	2 个万兆 POS
最大网络带宽	10Gbps	10Gbps	20Gbps
支持最大队列	64	64	64
尺寸(长*宽)	190mm * 112mm		
主机接口	8x PCI-E 2.0 规范		
最大功率	10W	12W	15W
输入电压	12V±10%		
工作温度	-5℃~55℃		
相对湿度	5%~90% RH		

1.3 系统要求

服务器配置要求：

- CPU 主频 1.8GHz 以上；
- 内存容量 4 GB 以上；
- 支持全高半长扩展卡的 8x 的 PCI-E 槽位。

操作系统兼容要求：Linux 系统，内核版本 2.6 以上(含 2.6)

第二章 加速卡安装

中科曙光 NetFirm 系列智能加速卡在设计和制造过程中遵循了严格的标准，以保证加速卡拥有卓越的品质。但是 NetFirm 系列智能加速卡属于精密电子设备，在使用过程中仍然可能因为各种原因而导致异常，所以请务必遵守下列注意事项。

- 只有经过培训的维护技术人员才能安装、卸载或升级加速卡。
- 安装加速卡前，必须先关闭服务器电源。
- 安装过程中请正确佩戴防静电手套，防止静电对加速卡造成损坏，请勿触摸焊接点、引脚或裸露的电路。
- 从防静电包装袋中取出加速卡后，应立即进行安装操作，将加速

2.1 拆封检查

请参照表 2-1 列出的物品清单检查配件是否齐全，是否无氧化、无化学腐蚀、无元器件脱落、无运输损坏、无外观破损等情况。

表 2-1 中科曙光 NetFirm 系列智能加速卡配件清单

品名	数量
加速卡	1
光模块	根据用户配置确定
光盘（驱动程序、fpga 逻辑和文档）	1
用户手册	1

【注】实际包装配件以具体的合同配置为准。

2.2 安装加速卡

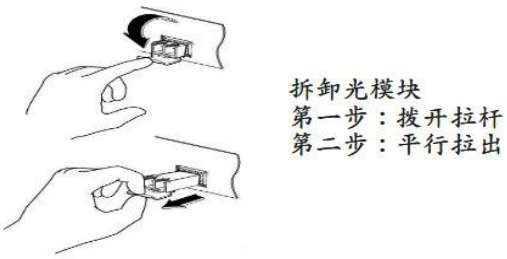
请参照以下步骤安装加速卡：

步骤 1 准备好螺丝刀等工具，佩戴好防静电手套或手镯。

步骤 2 断开机箱电源，打开机箱。

【注】请参照服务器或者 PC 机的操作说明进行操作。

步骤 3 从防静电包装袋中取出加速卡，摘下光纤接口模块。



步骤 4 把加速卡安装在主板或者转接卡上。

直接安装方式：对于机架式服务器高度在 3U 以上、塔式服务器和 PC 机，只要内部空间足够，加速卡可直接安装在主板上。

转接卡方式：对于机架式服务器为 1U 或 2U 的情况下，加速卡需要通过转接卡安装到主板的 PCI-E 槽位上。

【注】不同型号的服务器其转接卡外形会有所不同，其安装方式也可能会有所不同，请参照服务器提供的转接卡安装步骤。

步骤 5 固定加速卡。

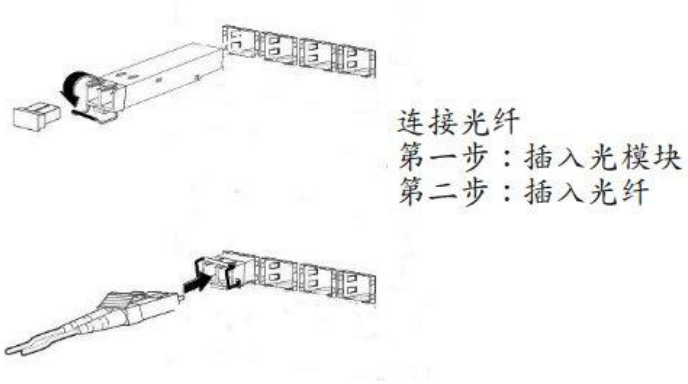
通过加速卡档片把加速卡固定在主板插槽上。

【注】请参照服务器或者 PC 机的操作说明进行操作，通常是用螺丝或卡扣固定加速卡。

步骤 6 盖好机箱。

【注】请参照服务器或者 PC 机的操作说明进行操作。

步骤 7 插入光纤接口模块，插上光纤线。



【注】加速卡上每个网口有固定的编号，编号为 0 的网口是加速卡最上端的（离 pci-e 插槽最远的），向下依次编号，如下图所示，以 NetFirm-C304 加速卡为例：

插光纤时要注意方向，光模块上一般会标识光纤信号的接收和发送端。

2.3 安装加速卡驱动

安装完加速卡之后，请参照以下步骤安装加速卡驱动。

步骤 1 打开服务器，检查硬件架构、操作系统及内核

以管理员账号登陆操作系统，检查服务器是否是 X86-64 架构，检查操作系统的类型及内核版本号是否是 NetFirm 系列智能加速卡所支持的操作系统。用 uname –a 和 lsb_release –a 命令即可，如下所示：

```
[root@localhost ~]# uname -a

Linux localhost.localdomain 2.6.18-8.el5 #1 SMP Fri Jan 26 14:15:14 EST 2007
x86_64 x86_64 x86_64 GNU/Linux

[root@localhost ~]# lsb_release -a
LSB Version:core-3.1-amd64: core-3.1-ia32: core-3.1-noarch:
graphics-3.1-amd64:graphics-3.1-ia32:graphics-3.1-noarch
Distributor ID: RedHatEnterpriseServer
Description:    Red Hat Enterprise Linux Server release 5 (Tikanga)
Release:    5
Codename:   Tikanga
```

步骤 2 检验加速卡硬件是否被操作系统识别

1. 运行 lspci 命令，若看到 Network controller: Growth Networks Unknown device 4e43 表示设备已经被识别，否则表示加速卡硬件未被识别。
2. 可能导致系统不能识别加速卡的原因和解决方法如下：

可能原因	解决方法
加速卡和主板接触不良	断电后重新插拔加速卡和转接卡，并有效固定加速卡。
加速卡和主板不兼容	使用曙光服务器，可以保证主板和加速卡的兼容性。
加速卡运输或安装过程中损坏	更换加速卡。

步骤 3 查找相应的安装包。

在光盘中的/driver 目录下找到相应操作系统的安装包。如操作系统是 Red Hat Enterprise Linux AS5，则找到相应的软件包 netfirm_bin-AS5.0-2.6.18-8.el5-x86_64.tgz。AS5.0 代表该 OS 是 Red Hat Enterprise Linux 5.0，2.6.18-8.el5 代表其内核版本，x86_64 代表系统是 64 位系统。其他系统类似。

步骤 4 解压缩安装文件

```
#cd $work_dir
#tar zxvf $media_path/driver/netfirm_bin-AS5-2.6.18-8.el5.tgz
```

注：$work_dir 是安装包解压后的工作位置，会在工作目录生成一个 netfirm_bin-AS5.0-2.6.18-8.el5-x86_64 的目录，$media_path 是光盘的路径。

步骤 5 进入安装目录，运行安装程序 。

```
#cd netfirm_bin-AS5.0-2.6.18-8.el5-x86_64
#./install.sh
```

经过步骤 5，会在根目录下生成一个安装文件目录/pag，并将编程需要的头文件和库文件拷贝到系

统目录下：头文件拷贝到/usr/include 目录下，库文件拷贝到/usr/lib64 目录下，并将驱动模块加载到内核。

上述步骤是以默认参数把驱动模块加载到内核的，一般用户采取此种操作即可。当然，用户也可以根据不同的需求，修改驱动模块的参数，可配置的参数可通过 netfirm_bin-AS5.0-2.6.18-8.el5-x86_64/driver/drv.conf 来配置，配置完成运行该目录下的 updrv.sh 使配置生效。

下面是 drv.conf 的配置项列表：

```
max_recvbuf_num = 4
max_recvbuf_size = 256
max_sendbuf_num = 5
按屏幕显示内容
```

在 drv.conf 中，max_recvbuf_num 和 max_sendbuf_num 代表收包缓冲区数目和发包缓冲区数目，max_recvbuf_size 和 max_sendbuf_size 分别代表缓冲区的大小。

当然，还可以通过 insmod 命令在加载模块时制定配置参数。加速卡驱动提供的配置参数可通过 modinfo 命令查看：

```
#cd /netfirm/driver
#modinfo netfirm.ko
filename:       netfirm.ko
license:        GPL
description:    Sugon Netfirm NIC driver
author:         SUGON
srcversion:     34FC5498B43474339205664
alias:          pci:v00004943d00004E43sv*sd*bc*sc*i*
depends:
vermagic:       2.6.18-8.el5 SMP mod_unload gcc-4.1
parm:      debug:Debug message level, default=0, no message (int)
parm:      use_numa:use numa mem allocation, default=1, use (int)
parm:      use_rule:use tuple5 rule filter, default=1, use (int)
parm:      use_port:use port bitmap filter, default=1, use (int)
parm:      use_tcp:use tcp stream, default=1, use (int)
parm:      use_regex:use regex, default=1, use (int)
parm:      rx_stream_buf_size:buffer size in MB for each rx stream (int)
parm:      tx_stream_buf_size:buffer size in MB for each tx stream (int)
parm:       rx_stream_num:rx stream number in driver, default = cpu
number (int)
parm:        tx_stream_num:tx stream number in driver, default =
rx_stream_num + 1 (int)
parm:       tcp_stream_num:tcp stream number in driver, default =
rx_stream_num (int)
```

在驱动中可修改的参数有收发包缓冲区的大小和数目。具体操作如下：

```
#rmmod netfirm
#insmod netfirm.ko [rx_stream_buf_size=n1] [rx_stream_num =n2]
                   [tx_stream_buf_size=n3] [tx_stream_num =n4]
```

说明：

[]中为可选项.

rx_stream_buf_size 为每个收包线程的缓冲区大小，以 M 字节为单位的，n1 取值范围[1，1024]，默认值为 256M

rx_stream_num 为收包缓冲区的最大个数，n2 取值范围为[1，64]，默认值为当前服务器 CPU 核心的个数。

tx_stream_buf_size 为每个发包缓冲区的大小，以 M 字节为单位的，n3 取值范围[1，1024]，默认值为 64M。

tx_stream_num 为收包缓冲区的最大个数，n4 取值范围为[0，32]，默认值为：rx_stream_num 加 1。

步骤 6 确认驱动安装查看加速卡初始状态。

首先，检查驱动安装是否安装成功，若运行命令 lsmod | grep netfirm 之后，显示含 netfirm 的一行，说明驱动加载成功，否则，驱动加载失败，请运行 dmesg 查看失败的原因，若原因为分配内存失败，请重启服务器后再重新加载驱动。

```
# lsmod | grep netfirm

netfirm                68516  0
```

其次，查看加速卡初始状态：曙光加速卡命名为 pag?，例如 pag0。经过以上步骤，可以通过 ifconfig 命令来查看加速卡状态。

```
#ifconfig pag0
pag0     Link encap:Ethernet   HWaddr 6E:66:FF:FF:FF:7F
          inet6 addr: fe80::200:ff:fe00:0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)
          Memory:fd000000-fe000000
```

2.4 确认安装成功

在/pag/pag_samples 目录下，有多个示例程序，可以编译并运行这些例程，以确认加速卡工作正常，例如:可以运行收包示例程序./rx_sample，查看当前到达加速卡的流量。若程序可以正常运行，且输出结果正常，则表示智能加速卡已经正确安装，可以正常工作了。

若示例程序显示的流量与实际流量不符，则首先判断网络连接是否可靠，光信号的强度是否满足加速卡要求。

```
#cd /netfirm/pag_samples
#make
#./rx_sample
Current speed-----
     Thread[0]:         0 pps,         0 M bps
     Thread[1]:         0 pps,         0 M bps
     Thread[2]:         0 pps,         0 M bps
     Thread[3]:         0 pps,         0 M bps
```

```
Pkts recieved in threads:
     Thread[0]:  <       0> pkts,  <       0 M> bytes
     Thread[1]:  <       0> pkts,  <       0 M> bytes
     Thread[2]:  <       0> pkts,  <       0 M> bytes
     Thread[3]:  <       0> pkts,  <       0 M> bytes
Total : 0 pkts,   0 bytes
Speed : < 0 > pps,  < 0 M > bps
```

第三章 加速卡卸载

3.1 卸载驱动软件

请参照以下步骤卸载加速卡驱动软件。

步骤 1 以管理员账号登陆操作系统；

步骤 2 确保所有基于加速卡软件的应用业务均被关闭；

步骤 3 进入安装目录，运行卸载程序；

```
#cd /root/netfirm_bin-AS5.0-2.6.18-8.el5-x86_64
#./uninstall.sh
```

3.2 拆卸加速卡

请按照以下步骤拆卸加速卡。

步骤 1 佩戴好防静电手套。

步骤 2 关闭机箱电源，打开机箱。

【注】请参照服务器或者 PC 机的操作说明进行操作。

步骤 3 拆开加速卡的固定装置。

【注】请参照服务器或者 PC 机的操作说明进行操作。通常是拧开加速卡的固定螺丝或者松开卡扣。

步骤 4 沿着平行于槽位的方向，将加速卡从主板或者转接卡上拔出。

步骤 5 将取出的加速卡放回防静电包装袋。

步骤 6 盖好机箱。

【注】请参照服务器或者 PC 机的操作说明进行操作。

第四章 驱动软件

4.1 驱动软件目录结构

加速卡驱动安装好以后，在目录 /pag/下可以看到所有文件，目录结构如下:

doc :存放加速卡相关文档

driver: 存放加速卡驱动

include :存放 API 头文件

lib:存放 libpag.so 库、libpcap.so 库以及 libpcap.a 库

tools:存放加速卡各种功能的标准的管理工具

sbin: 存放加速卡各种功能的客户定制的管理工具

sample :存放使用 libpag 库的示例代码以及使用 libpcap 库的示例代码

4.2 配置文件

4.2.1 收包配置文件

中科曙光 NetFirm 系列智能加速卡支持 libpcap 接口库和 libpag 接口库，每个接口库有自己的配置文件，应用软件调用接口库打开设备时，接口库使用对应配置文件中的参数设置加速卡属性。

4.2.1.1 调用 libpag 库收包

配置文件 pag.conf 存放在应用当前目录下，文件内容示例如下(可根据实际情况修改)，pag.conf 配置项分为基本的基本配置项、高级配置项和可选项配置。在基本配置项中定义了设备名（dev_name）、应用优先级（priv）、应用 id 以及应用名字等。高级选项配置包括（1）收包配置，（2）发包配置，（3）五元组规则过滤配置，（4）tcp 流还原配置，（5）端口过滤配置，（6）正则表达式配置。

```
##############   libpag application config file  ################

######### basic rx config ##############
##device name, libpag can only support one device now
dev_name = pag0
##stream number
stream_num = 4
##application's priority, 0=master 1=slave, default=0, unused from nf1.1
#priv = 1
##application's id, application DFB use it
app_id = 2
##applicaion's name, application DFB use it
app_name=start
################ advance funcitons ######
###### rx_configure  ########
##upload pkt ctrl, 32bites.
#bit0:  ipv4 upload,         bit1:  ipv6 upload,         bit2:  vlan ipv4
upload     bit3:  vlan ipv6 upload,
#bit4:  dvlan ipv4 upload,  bit5:  dvlan ipv6 upload,  bit6:  ipv4 in ipv6
```

```
upload, bit7:  ipv6 in ipv4 upload,
#bit8:  pppoe ipv4 upload,  bit9:  pppoe ipv6 upload,  bit10: mpls1 ipv4
upload,   bit11: mpls1 ipv6 upload,
#bit12: mpls2 ipv4 upload,  bit13: mpls2 ipv6 upload,  bit14: mpls3 ipv4
upload,   bit15: mpls3 ipv6 upload,
#bit63: other unknown pkts upload, 0x00000000: all pkt upload,
#default value = 0x01, only rx ipv4 pkts.
#pkt_upload_enable = 0x0
##whether getting inner ip header or outer ip header from a tunneling
protocols, such as ip in ip.
##0 means outer ip header, 1 means inner ip header; default value = 0, outer
ip header.
#iplayer_config = 1
##type of hash function for stream divided
##0 means getting hash value by nic computing, 8 means getting hash value
from source MAC.
##default value = 0x0, getting hash value by nic computing.
#hash_mode = 8
##tuple4 bitmap for hash algorithm, one bit for a byte in tuple4,
##36 bytes dport-sport-dip-sip in network order, 1 = used byte, 0 = unused
byte,
##default = 0xffffffff, use tuple2.
#hash_t4_bitmap = 0xffffffffff
##use ip defragment, default = 0
#defrag_enable = 1
##upload frag pkt, when defrag_enable = 0,this configure is valid
## 1 = upload, 0 = discard ; default = 1
#ipfrag_upload_enable = 0
##keep pkt order for rx pkts
## 1 = keep order, 0 not keep order; default = 1
#keep_pkt_order = 0
##threshold of cpu percentage in slave application, default = 100, not drop
any pkts
#cpu_dropthreshold = 80
###### tx_configure  ########
##send pkt idle cycle gap, used to test cases where nic send pkts to slow
host,
##more gap, more slow send speed
#send_gap = 1000
###### rule_configure ########
##use filter rule, 1=use 0=not use, default = 0
#usefilterrule = 1
##max rule num, default = 250000
#maxrulenum = 200000
###### port_configure ########
##use port filter, 1 = use 0 = not use, default = 0
#useportfilter = 1
##tcp dst port filter config file path, default not use port filter
#tcp_dport_config_file = tcp_dports.conf
##udp dst port filter config file path, default not use port filter
#udp_dport_config_file = udp_dports.conf
##tcp src port filter config file path, default not use port filter
#tcp_sport_config_file = tcp_sports.conf
```

```
##udp src port filter config file path, default not use port filter
#udp_sport_config_file = udp_sports.conf
###### tcp_rebuild_configure ########
##use tcp rebuild function, default = 0
#tcp_rebuild_enable = 1
##use udp rebuild function, default = 0
#udp_rebuild_enable = 1
##max tcp stream num for each tcp thread, default = 100000
#max_tcp = 100000
##max udp stream num, default = 0
#max_udp = 100000
###### regex_configure ########
##use regex, default = 0
#regex_enable = 1
############ optional ###############
##thread and cpu affinity, default=1
#cpu_affinity = 1
##dma valid buffer, default = 40960 Byte
#valid_buf_size = 40960;
##control rx pointer,
##1 = check slave read_ptr, 0 = ignore slave read_ptr, default = 1
#rx_ctrl = 0
##default filt action for packets that can not hit tuple-5 rules
##0 = upload, 1 = drop, default = 0
#filt_default_action = 1
#scrambling for POS, 0 means enable scramble, 1 means disable scramble
#default value 0, enable scramble
#add_scramble = 1
#length of FCS for PPP. 0 means 4byte, 1 means 2byte.
#default value 0, 4byte
#ppp_fcs_len = 1
```

参数说明：

- 以#开头的行是注释信息行。
- dev_name: 设备名，必须指定。
- stream_num: 收包数据流数目，不能超过驱动分配的最大收包缓冲区队列的个数。
- usefilterrule: 是否启用规则过滤功能，1 表示启用，0 表示禁用。
- max_rulenum: 最大规则数；
- tcp_rebuild_enable: TCP 流还原使能是否打开，0 为关闭，1 为打开；
- hash_t4_bitmap：hash 分流算法使用的 4 元组位图，36bits，每一位表示 4 元组的一个字节，1 表示使用，0 表示不用，36 个字节的顺序为目端口源端口目 ip 源 ip，比如，要使用源 ip 分流，则为 0x00ffff。
- pkt_upload_enable: 制定用户感兴趣的报文类型。默认值为只接受 ipv4 报文。
- useportfilter：指明是否启用端口过滤功能。
- add_scramble 和 ppp_fcs_len 是针对 POS 报文的。
- 根据用户定制的功能，配置文件中还可能包含其它参数。

4.2.1.2 调用 libpcap 库收包

当应用软件仅仅使用智能加速卡的端口采样和高速收包功能时，可以调用标准的 libpcap 接口，libpcap

接口打开设备时也可以使用配置文件，配置文件存放在/etc/netfirm/下，文件名为“设备名.conf”，比如对设备加速卡 pag0，配置文件为 pag0.conf，如果没有该文件，将按默认值处理。

文件内容示例如下：

```
#### config file for libpcap #######
##tcp dst port filter config file path, default not use port filter
#tcp_dport_config_file = tcp_dports.conf
##udp dst port filter config file path, default not use port filter
#udp_dport_config_file = udp_dports.conf
##tcp src port filter config file path, default not use port filter
#tcp_sport_config_file = tcp_sports.conf
##udp src port filter config file path, default not use port filter
#udp_sport_config_file = udp_sports.conf
##tuple4 bitmap for hash algorithm, one bit for a byte in tuple4,
##36 bytes dport-sport-dip-sip in network order, 0 = unused byte, 1 = used
byte,
##e.g. 0x00f = only use sip network addr a.a.a.x, use whole tuple4
#hash_t4_bitmap = 0xfffffffff
##upload pkt ctrl, 32bites.
#bit0:  ipv4 upload,         bit1:  ipv6 upload,         bit2:  vlan ipv4
upload     bit3:  vlan ipv6 upload,
#bit4:  dvlan ipv4 upload,  bit5:  dvlan ipv6 upload,  bit6:  ipv4 in ipv6
upload, bit7:  ipv6 in ipv4 upload,
#bit8:  pppoe ipv4 upload,  bit9:  pppoe ipv6 upload,  bit10: mpls1 ipv4
upload,   bit11: mpls1 ipv6 upload,
#bit12: mpls2 ipv4 upload,  bit13: mpls2 ipv6 upload,  bit14: mpls3 ipv4
upload,   bit15: mpls3 ipv6 upload,
#bit63: other unknown pkts upload, 0x00000000: all pkt upload,
#default value = 0x01, only rx ipv4 pkts.
#pkt_upload_enable = 0x0
```

参数说明：

以#开头的行是注释信息行

1. tcp_port_config_file: tcp 端口采样的配置文件，默认不使用 tcp 端口采样功能，所有报文上传。
2. udp_port_config_file: udp 端口采样的配置文件，默认不使用 udp 端口采样功能，所有报文上传。
3. hash_t4_bitmap：hash 分流算法使用的 4 元组位图，36 位，每一位表示 4 元组的一个字节，1 表示使用，0 表示不用，36 个字节的顺序为目端口源端口目 ip 源 ip，比如，要使用源 ip 分流，则为 0x00ffff。

4.2.2 发包和日志配置文件

使用 libpag 接口库的发包或五元组分类功能时，需要配置在应用软件当前目录下的 sendlog.conf 文件，其中的参数如下:

```
#************************************************************
#local netcard name
LocalHostIPDevice = eth0
#send pkt port
LogSendDevice=pag0:0
RstSendDevice=pag0:0
```

```
MacSendDevice=pag0:0
#LogDestMacAddr=00:04:22:5a:e4:7f
LogDestMacAddr=00:04:22:5a:e4:1f
RstDestMacAddr=00:04:22:5a:e4:9f
Device0SrcMacAddr=00:A0:34:01:EF:00
Device1SrcMacAddr=00:A0:34:01:EF:02
Device2SrcMacAddr=00:A0:34:01:EF:03
Device3SrcMacAddr=00:A0:34:01:EF:04
#Src Port
Srcport = 3456
#the host's IP
#ServerIP = 192.168.1.222
#ServerIP = 10.50.134.122
#ServerIP = 10.50.134.123
#ServerIP = 10.50.134.124
#ServerIP = 10.50.134.125

ServerIP = 192.168.1.1
#Server's Port
ServerPort = 6009

#MTU for tx,  default=1500
#SendMTU = 2000

#hw compute tcp checksum enable, 1 check, 0 no check,  default=1
#Tcpchecksum = 0
```

参数说明:

以#开头的行是注释信息行。

1. LocalHostIPDevice：为本地通用加速卡名，该加速卡的 ip 地址将作为转发日志报文的源 ip 地址。
2. LogSendDevice：发送日志包的网口，pag0:0 表示从设备 pag0 的 0 口发送日志包。
3. RstSendDevice：发送封堵包的网口，pag0:0 表示从设备 pag0 的 0 口发送封堵包。
4. LogDestMacAddr：发送日志包时目标设备（对端接收设备）的 MAC 地址。
5. RstDestMacAddr：发送封堵包是目标设备（对端接收设备）的 MAC 地址。
6. Device0SrcMacAddr ~ Device3SrcMacAddr：0 ~ 3 网口发包时的源 MAC 地址。
7. ServerIP：日志服务器的 IP 地址， 日志接收机最大个数为 64 个，智能加速卡可保证不同日志机之间的负载均衡。
8. ServerPort: 日志服务器的 UDP 端口。
9. SendMTU: 加速卡发包的最大报文长度。
10. Tcpchecksum：硬件对发送报文进行 Tcp 校验的使能

4.2.3 五元组规则配置文件

使用 libpag 接口库的五元组分类功能时，可能需要从五元组规则配置文件中向加速卡导入分类规则，规则文件中的规则一般是不会动态变化的，对于每类规则使用单独的一个配置文件，规则配置文件的格式如下:

```
/*****************************************************************/
```

#ruletype;ruleid;filtaction;sendaction;logaction;prototype;srcip;dstip;srcport;dstport;

1;5;0;1;0;6;10.0.1.23;123.0.3.5;80;1230;

1;5;0;2;2;17;10.0.1.23;0;0;0;

/***/

参数说明:

#号开始的行是注释信息行。

ruletype: 智能加速卡定义的 16 类规则，取值范围为 0 ~ 15;

ruleid: 规则编号，在同一类规则中是唯一的;

filtaction：过滤动作，指定加速卡对报文上传还是丢弃，0 表示上传，1 表示丢弃;

sendaction：发包动作，指定加速卡发送阻断连接的报文的类型，0 表示不阻断，1 表示发送 rst 阻断;

logaction:日志动作，指定发送不同类型的日志包，0 表示不发日志包，1 表示发送只有连接信息的日志包，2 表示发送带原始报文的日志包;

prototype:协议类型;0 代表任意，17 代表 UDP，6 代表 TCP;

srcip:源 IP;0 代表任意;

dstip:目的 IP;0 代表任意;

srcport:源端口;0 代表任意;

dstport:目的端口;0 代表任意;

4.2.4 规则类型配置文件

当使用 libpag 接口的五元组分类功能时，需要使用应用软件当前目录下的配置文件 ruletype.conf 。智能加速卡支持 16 类五元组规则，对各种不同类型的规则，可以在配置文件中指定它们的优先级（用于规则替换）和生存期（规则加入后的有效时间，超时后规则自动删除），格式如下:

```
#ruletype;priority; livetime;
1;5;0;
#2;4;0;
3;2;0;
4;3;0;
5;1;0;
```

参数说明:

#号开始的是注释信息行。

每行用分号隔开 3 列数字，分别表示规则类型、优先级、和生存期。

ruletype: libpag 接口定义的规则类型，从 1 ~ 16;

priorit: 规则的优先级，从 1~16 数字越大，优先级越高; 配置规则时，如果两条规则发生冲突，高优先级的规则会覆盖低优先级的规则；如果是同优先级的规则，后写入的规则覆盖原来的规则;

livetime: 规则的生存期，0 代表该类规则是静态规则，永久有效；其它整数代表生存的秒数，从配置下发开始生效，到达时间后规则自动删除;

4.2.5 端口采样配置文件

调用 libpcap 接口库时，可以通过配置文件，对特定 TCP 和 UDP 端口的数据进行采样，加速卡可以通过配置文件向加速卡中配置需要采样的端口，只有指定端口的报文才上传给主机，该配置文件中保存需

要接收的报文的端口列表。比如要接收 tcp 端口为 80，8080，25，110 的报文，则要在应用配置文件中指定端口采样配置文件，tcp_port_config_file = tcp_ports.conf，在端口采样配置文件 tcp_ports.conf 中列出上述端口：

```
80
8080
25
110
```

4.2.6 加速卡升级配置文件

智能加速卡的硬件芯片是可重构的，可以通过升级管理工具升级硬件逻辑，升级管理工具会读取用户指定的升级配置文件中的信息，文件内容如下：

```
##device identification, 32bits in hex, maybe the serial number
##this string will appear in mac addr, .e.g. 6e:66:12:34:ab:cd
dev_id = 1234abcd

##hardware version, default = 0
#hw_ver = 0

##logic file *.hex used to update firmware, do not update if no file
logic_file = /root/TestStable.hex

##string burn into firmware, maybe device status or others,
##warning: no more than 100 charactors, and no space or # is allowed!
remark_str = user_remark_string_for_identification_this_card
```

参数说明:

#号开始的是注释信息行。

dev_id: 标识设备的 8 个十六进制数，配置后会显示到加速卡的 mac 地址中;

hw_ver: 硬件板卡的生产版本，可用于标识板卡生产的批次等;

logic_file: 升级芯片所用的逻辑文件的路径;

remark_str：额外的标识字符串，需要在 100 个字符内，且不允许有空格。

【注】升级硬件是专业性很强的工作，升级之前需要将正在运行的应用程序停止，升级过程中不允许其他系统操作，升级后需要重启，升级过程大约需要 3-5 分钟，升级过程一旦出错会带来不可恢复的后果，因此，强烈建议只有经过专门培训的技术支持人员，才能进行加速卡升级操作。

4.3 管理工具

4.3.1 加速卡端口类型选择工具 pagconfig

主要功能：选择加速卡端口类型，在 sbin 目录下。。

命令格式：pagconfig <dev_name>[:port] <up|down>

dev_name：加速卡设备名，名称为 pag?。

port:可以直接连接在设备名后面，比如 pag0:1，代表 pag0 设备的 1 号端口。

up or down: 制定某个口 up 或者 down。如果不指定 port 值就默认使能所有端口。

4.3.2 设备管理工具 nf_dev

主要功能：进行加速卡配置、监控运行状态、查看加速卡信息等。

命令格式：nf_dev　cmd

其中 cmd 以下格式之一：

<list>

<stat> <dev_name> [interval]

<info> <dev_name>

<reset> <dev_name> [flag]

<mac_ports> <dev_name> <mac_ports_bitmap>

参数说明：

list， stat， info， reset， mac_ports 为子命令的关键字。

dev_name：加速卡设备名，名称为 pag?。

interval：显示信息的刷新时间，单位为秒。

flag：　　复位标识。

1：复位芯片逻辑；

2：复位统计寄存器；

3：两者都复位（默认）。

mac_ports_bitmap ：网口使能位图，4 位。1：开启；0：关闭。

【注】加速卡上每个网口有固定的编号，编号为 0 的网口是加速卡最上端的（离 pci-e 插槽最远的），向下依次编号。

功能描述：

nf_dev <list>：列出主机中加速卡信息。

nf_dev <stat> <dev_name> [interval]：显示加速卡的网口和各功能模块状态。

nf_dev <info> <dev_name>：显示加速卡软件硬件信息。

nf_dev <reset> <dev_name> [flag]：复位加速卡。

nf_dev <mac_ports> <dev_name> <mac_ports_bitmap>：加速卡网口的使能配置位图。

例如：1101，表示关闭 2 号网口，开启其它三个网口。

4.3.3 发包管理工具 nf_tx

主要功能：查看发包状态

命令格式：nf_tx　cmd

其中 cmd 以下格式：

<stat> <dev_name> [interval]

参数说明：

stat 为子命令的关键字 。

dev_name ：加速卡设备名，名称为 pag?。

interval ：显示信息的刷新时间，单位为秒。

功能描述：

显示发包功能的相关信息，如：发包的报文数和字节数，发包缓冲区读写指针。

4.3.4 收包管理工具 nf_rx

主要功能：收包的负载均衡分流配置、状态监控。

命令格式：nf_rx　　cmd

其中 cmd 以下格式之一：

　　<stat> <device> [interval]

参数说明：

stat，ratio：为子命令的关键字。

device: 加速卡设备名，名称为 pag?。

interval ：显示信息的刷新时间，单位为秒。

功能描述：

<stat> <device> [interval]显示收包流的状态，如收包数，收包字节数，读写指针等，interval 为多少秒钟刷新一次显示信息。

4.3.5 五元组规则管理工具 nf_rule

主要功能：五元组规则加载、导出、使能、状态监控等。

命令格式： nf_rule cmd

其中 cmd 以下格式之一：

<load> <device> <hash_calcnt> <rulemask_file> <rule_file>

<dump> <device> <rule_type>

<clear> <device> <rule_type>

<stat> <device> [interval]

<enable> <device>

<disable> <device>

参数说明：

load，dump，clear，stat，enable，disable 为子命令的关键字。

device：加速卡设备名，名称为 pag?。

rule_file：规则文件名。

rule_type：规则类型，取值范围 0 ~ 16。

hash_calcnt: hash 计算次数，默认值为 12。

rulemask_file:规则掩码文件。

inveral：显示信息的刷新时间，单位为秒。

功能描述：

<load> <device> <hash_calcnt> <rulemask_file> <rule_file> 从规则配置文件 rule_file 中加载五元组规则。

<dump> <device> <rule_type> 打印已经加载的规则，规则类型由 rule_type 指定，0 表示打印所有的规则。

<clear> <device> <rule_type> 清空加速卡中的规则，规则类型由 rule_type 指定 ，0 表示清空所有的规则。

<stat> <device> [interval]　显示当前报文分类功能的运行状态。

<enable> <device> 打开报文分类功能。

<disable> <device> 关闭报文分类功能。

4.3.6 端口采样管理工具 nf_port

主要功能：端口采样配置加载、导出、使能、状态监控等。

命令格式：nf_dev cmd

其中 cmd 以下格式之一：

<load> <device> <proto> <port_file>

<dump> <device> <proto>

<clear> <device> <proto>

<stat> <device> [interval]

<enable> <device>

<disable> <device>

参数说明:

其中 load， dump， clear， stat， enable 为子命令的关键字。

dev_name ：加速卡设备名，名称为 pag?。

proto ：协议代号，6：TCP；17：UDP；0：TCP 和 UDP。

port_file ：端口列表配置文件。

interval ：显示信息的刷新时间，单位为秒。

功能描述：

<load> <dev_name> <proto> <port_file>：加载端口列表文件。

<dump> <dev_name> <proto>：打印已经加载的端口列表。

<clear> <dev_name> <proto>：清除端口列表。

<stat> <dev_name> [interval]：显示端口采样功能的运行状态。

<enable> <dev_name>：打开端口采样功能。

<disable> <dev_name>：关闭端口采样功能。

4.3.7 TCP 连接还原管理工具 nf_tcp

主要功能：TCP 连接还原的状态显示、使能、禁用等。

命令格式：nf_tcp cmd

其中 cmd 以下格式之一：

<stat> <device>

<enable> <device>

<disable> <device> [ooo|all]

参数说明：

stat，enable，disable 为子命令的关键字。

device：加速卡设备名，名称为 pag?。

ooo: TCP 连接还原的乱序报文重排功能；

all: TCP 连接还原的会话状态管理和乱序报文重排功能。

功能描述：

<stat> < device> [interval]：显示 TCP 流还原状态。

<enable> <device>：打开 TCP 流还原功能。

<disable> <device>：关闭 TCP 流还原功能。

4.3.8 升级管理工具 nf_update

该工具用于硬件逻辑升级，可以显示逻辑版本信息，更新硬件逻辑。

命令格式： nf_update <cmd>

其中 cmd 可以包括

<list>

<update> <bus_id> <config_file>

参数说明：

list，update 为子命令的关键字。

bus_id：加速卡所在 PCI 总线的 ID。

config_file：升级使用的配置文件路径。

功能描述：

<list> 显示当前系统上的加速卡的 bus_id，设备 ID，硬件版本，逻辑版本。

<update> <bus_id> <config_file> 根据 bus_id 和 config_file 文件更新加速卡的硬件逻辑。

例如：nf_update 0700 ./update.conf

【注】升级硬件是专业性很强的工作，升级之前确保应用程序都已经关闭，升级过程中不允许其他系统操作，升级后需要重启，升级过程大约需要 3-5 分钟，升级过程一旦出错会带来不可恢复的后果，因此，强烈建议只有经过专门培训的技术支持人员，才能进行加速卡升级操作。

第五章 通用 API 接口：libpcap

NetFirm 系列智能加速卡支持标准的 libpcap 捕包 API 接口，该接口支持高效捕包，结合配置文件，还可实现多核服务器同源同宿负载均衡的分流，和对特定 tcp 或 udp 端口采用的功能。

在 libpcap 接口对应的配置文件/etc/netfirm/pag?.conf 中，可以指定把接收到的报文按一定比例分发到几个缓冲区，上层应用可以把每个缓冲区作为一个收包设备，使用方法是在设备名后面加上缓冲区号，中间用冒号隔开。

比如，设备 pag0 的配置文件中指定分流比例为 1:2，表示输入流量按 1 比 2 的比例分到 0 号和 1 号缓冲区，那么可以使用两个应用分别从缓冲区 0 和缓冲区 1 收包，第一个应用调用 libpcap 接口打开设备时，设备名为 pag0:0，第二个应用调用 libpcap 接口打开设备时，设备名为 pag0:1。

5.1 设备接口

pcap_t *pcap_open_live(char *device, int snaplen, int promisc, int to_ms, char *ebuf)

[函数说明]：

打开加速卡的收包缓冲区，获得用于捕获网络数据包的控制结构，注意：智能加速卡打开设备时会读取/etc/netfirm/下配置文件的参数。

[参数说明]：

device：打开的加速卡上的指定收包缓冲区，比如在配置文件/etc/netfirm/pag0.conf 中指定了加速卡把接收到的数据平均分到两个缓冲区，也就是说配置了 stream_ratio=1:1，如果应用要从 0 号缓冲区收取报文，那么设备名为 pag0:0。

naplen：报文的最大字节数，超过该长度的报文将被截断。

promisc：是否将网络接口置于混杂模式，注意：在智能加速卡中的非混杂模式是指只收取 IP 报文，混杂模式是收取所有的报文。

to_ms：超时时间（毫秒）。

ebuf：在函数出错返回 NULL 时用于传递错误消息。

[返回值]：

成功返回 pcap 句柄结构，失败返回 NULL.

void pcap_close(pcap_t *p)

[函数说明]：

关闭设备。

[参数说明]：

p：打开设备时返回的句柄结构指针。

5.2 收包接口

u_char *pcap_next(pcap_t *p, struct pcap_pkthdr *h)

[函数说明]：

从收包缓冲区中读取下一个报文。

[参数说明]：

p：打开设备时返回的句柄结构指针。

h：指向返回的报文的控制结构的指针。

[返回值]：

成功返回报文的报头指针；如果没有报文，则返回 NULL。

int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user)

[函数说明]：

从收包缓冲区中循环读取报文，并把每个报文交给回调函数处理。

[参数说明]：

p：打开设备时返回的句柄结构指针。

cnt: 循环读取的报文个数。

callback：应用自己实现的回调函数。

user: 指向应用数据的内存，一般设置为 NULL。

[返回值]：

成功读取指定报文个数返回 0；如果出错，则返回负数。

5.3 编程实例

在 pcap_samples 目录下提供了一系列的示例程序，这些示例程序的源码文件中有详细的注释，演示了加速卡 API 接口的使用方法，可供软件开发人员参考。

第六章 定制 API 接口：libpag

为了使用智能加速卡报文分类、流还原等定制化扩展功能，需要使用加速卡定制化的 API 接口库 libpag，这些接口在 libpag.h 文件中定义，应用代码中需要包该含头文件，编译时需要链接库文件 libpag.so。

libpag.h 定义了一系列宏定义、数据结构和 API 函数，这里只对主要的 API 函数做简单说明。

6.1 设备接口

int pag_open()

[函数说明]：

根据配置文件（当前目录中的 pag.conf 文件）打开设备，读取配置文件对加速卡进行参数配置。

[返回值]：

成功返回 1，失败返回-1。

void pag_close()

[函数说明]：

关闭设备。

6.2 收包接口

packet_t *pag_get_packet(int sid);

[函数说明]:

从加速卡的一个数据队列中获取一个报文句柄。

[参数说明]：

sid：收包数据队列的序号（0 开始的连续数字）。

[返回值]：

成功返回一个报文句柄，失败返回 NULL。

void *pag_get_ip(packet_t *)

[函数说明]：

从加速卡的一个数据队列中读取一个 IP 包。

[参数说明]：

packet_t：pag_get_packet 函数的返回值。

[返回值]：

报文的 IP 头指针；如果没有报文，则返回 NULL。

int pag_get_ip_length(packet_t *)

[函数说明]：

获取 IP 包的长度。

[参数说明]：

packet_t：pag_get_packet 函数的返回值。

[说明]：

该函数和 pag_get_ip 配合使用。

void *pag_get_nl_proto(packet_t *)

[函数说明]：

获取网络层的协议

void * pag_get_tdp(packet_t *)

[函数说明]：

获取传输层报文

int pag_get_tdp_length(packet_t *)

[函数说明]：

获取传输层报文长度

void *pag_get_payload(packet_t *)

[函数说明]：

获取负载报文头

int pag_get_payload_length(packet_t *)

[函数说明]：

获取负载长度

void *pag_get(int sid)

[函数说明]：

从制定的缓冲区中获取一个报文。

[参数说明]：

sid: 缓冲区 id 号

[返回值]：

成功返回一个报文的 ip 头，失败返回 NULL。

void *pag_get_pkt(int sid, int mode, void **pdata)

[函数说明]：

从加速卡的一个数据队列中读取一个 IP 报文，同时根据不同的 mode 值还可以再返回一个值到 pdata 中。

[参数说明]：

sid: 缓冲区 id 号

mode:等于 0,pdata 返回 NULL；等于 15，pdata 返回二层头；等于 31，pdata 返回 vlan 头；等于 47，pdata 返回三层头；等于 63，pdata 返回四层头；等于 79，pdata 返回负载位置。

pdata:根据 mode 值的不同，返回不同的值。

[函数说明]：

__u64 pag_time(void *pkt)

[函数说明]:

获取一个报文的时间戳信息;

[参数说明]：

pkt:通过 pag_get 得到的 IP 报文指针。

[返回值]:

报文的时间戳;

6.3 发包接口

void *pag_getsendbuf(int sid)

[函数说明]：

获取一个发包缓冲区。

[参数说明]：

sid：发包流 id。

[返回值]：

成功返回缓冲区首地址（开始填充报文 IP 头的位置），失败返回 NULL。

int pag_send(int pkttype， int sid， int ipdatalen)

[函数说明]：

发送一个 ip 包。

[参数说明]:

pkttype：报文类型，0:日志包，1:封堵包;2:mac 包。如果报文类型为 0 和 1 ，那么应用可以从 IP 头（pag_getsendbuf 返回的地址）开始向缓冲区内填写发包内容，加速卡硬件根据 sendlog.conf 文件中的配置自动填充以太头，并从指定网口发出; 如果报文类型为 2 ，那么应用需要从 mac 头（pag_getsendbuf 返回的地址减 14 字节）开始向缓冲区内填写发包内容，加速卡硬件根据 sendlog.conf 文件中的配置直接从指定网口发出。

sid：发包流 id。

ipdatalen ： ip 报文长度。

[返回值]:

成功返回 0，失败返回-1。

int pag_send_eth(int port， int sid， int eth_datalen)

[函数说明]:

发送一个 ethernet 包。

[参数说]明:

port ： 发包网口。

sid ： 发包流 id。

eth_datalen ： ethernet 帧长度。

[返回值]:

成功返回 0，失败返回-1。

6.4 五元组规则接口

int pag_refreshstaticrule(int ruletype， char *filename)

[函数说明]:

清空加速卡中的一类规则，重新导入一个新的规则配置文件。

[参数说明]:

ruletype：规则类型(从 1 开始)。

filename：要导入的规则文件名。

[返回值]:

成功返回 0，失败返回-1。

int pag_L4rule(int sid， unsigned int rule_num， struct filter_full_rule *pRules)

[函数说明]:

向加速卡添加或删除一系列规则

[参数说明]:

sid ：线程号（未使用）。

rule_num ：规则数。

pRules ：规则数组。

[返回值]：

成功返回添加或删除的规则数，失败返回-1

inline int pag_getfilterid(packet_t *pkt)

[函数说明]：

获取报文命中规则的 id，0 表示未命中任何规则。

[参数说明]：

pkt：pag_get_packet 函数返回值。

[返回值]：

报文命中的规则 id。

inline int pag_getfiltertype(packet_t *pkt)

[函数说明]：

获取报文命中规则的类型，0 表示未命中任何规则。

[参数说明]：

pkt：pag_get_packet 函数返回值。

[返回值]：

报文命中的规则类型。

6.5 流还原接口

void pag_setrebuild(char proto);

[函数说明]:

加速卡打开后调用一次，指定对 TCP 连接和 UDP 伪连接还原;

[参数说明]：

proto：需要进行重组的协议。0 表示不进行连接还原；6 表示只还原 tcp 连接； 17 表示只还原 udp 伪连接； 17|6 表示还原 tcp 连接和 udp 伪连接。

void pag_setlinknum(int ilinknum);

[函数说明]:

加速卡打开后调用一次，设定每个队列的最大并发连接数目

[参数说明]：

ilinknum：最大并发连接数。

struct pktinfo *pag_getstream(int sid);

[函数说明]:

从加速卡的一个数据流队列中读取接收到的数据;

[参数说明]:

sid:指定特定数据队列的序号(0 开始的数字)。

[返回值]:

指向 struct pktinfo 结构的指针;如果没有数据，则返回 NULL。

void pag_delstream(struct tcp_stream *a_tcp);

[函数说明]:

删除一个 tcp 连接，后继数据不再上传。

[参数说明]:

a_tcp: tcp 连接结构体的指针。

void pag_savedata(struct tcp_stream *a_tcp，u_char dir，int datalen);

[函数说明]:

tcp 连接数据缓存函数，要求加速卡缓冲特定连接上的一部分数据，下次读取数据时与新数据一起送给应用。

[参数说明]:

a_tcp: tcp 连接结构体的指针。

dir: 需要缓冲数据的方向。

datalen: 需要缓冲的数据长度。

6.6 编程实例

在 pag_samples 目录下提供了一系列的示例程序，这些示例程序的源码文件中有详细的注释，演示了加速卡 API 接口的使用方法，可供软件开发人员参考，其中的文件包括：

pag.conf：加速卡属性的配置文件。

rx_samples.c：从加速卡收包的示例程序。

rule_sample.c：加速卡开启报文分类后的收包示例程序。

rule.conf：五元组规则配置文件。

ruletype.conf：规则类型（优先级和生存期）配置文件。

tx_sample.c：软件发包的示例程序。

同时，在 sbin 目录下面也放置了 tcpdump 工具供用户使用。运行该 tcpdump 比较制定路径，比如，/pag/sbin/tcpdump –i pag0，要不然可能调用的是系统的 tcpdump。

另外，该 tcpdump 的所有使用的参数和标准 tcpdump 使用的参数定义是一样的。