GCNAT

目录

[1. 缩写词 3](#_Toc40861166)

[2. 概述 3](#_Toc40861167)

[2.1. 应用及原理 3](#_Toc40861168)

[2.2. 需求总结 5](#_Toc40861169)

[3. 总体架构 5](#_Toc40861170)

[3.1. 模块关系 5](#_Toc40861171)

[3.2. 外部模块处理流程 8](#_Toc40861172)

[3.2.1. gcnet 8](#_Toc40861173)

[3.2.2. Skbuff 8](#_Toc40861174)

[3.2.3. Bridge 8](#_Toc40861175)

[3.2.4. conntrack 9](#_Toc40861176)

[3.2.5. netdev 9](#_Toc40861177)

[4. 详细设计 10](#_Toc40861178)

[4.1. Gcnat内部模块划分 10](#_Toc40861179)

[4.1.1. CFG配置模块 10](#_Toc40861180)

[4.1.2. DB数据库模块 10](#_Toc40861181)

[4.1.3. Timer处理模块 11](#_Toc40861182)

[4.1.4. Hook处理模块 11](#_Toc40861183)

[4.1.5. Core核心处理模块 11](#_Toc40861184)

[4.2. 快转学习流程 11](#_Toc40861185)

[4.2.1. Rx Hook Proc 12](#_Toc40861186)

[4.2.2. Tx Hook Proc 12](#_Toc40861187)

[4.3. 软件快转流程 14](#_Toc40861188)

[4.4. Linux Kernel模块事件的Hook处理 15](#_Toc40861189)

[5. 数据结构 16](#_Toc40861190)

[5.1. 配置数据 16](#_Toc40861191)

[5.2. 状态数据 18](#_Toc40861192)

[5.2.1. 统计信息 18](#_Toc40861193)

[5.2.2. 表项结构 18](#_Toc40861194)

[5.2.3. 快转规则Hash链表 20](#_Toc40861195)

[5.3. 数据同步 20](#_Toc40861196)

[6. 算法描述 21](#_Toc40861197)

[6.1. 快转软表的Hash算法 21](#_Toc40861198)

[6.2. 快转规则的构建算法 21](#_Toc40861199)

[7. API接口 21](#_Toc40861200)

[7.1. 对外提供的hook处理接口 21](#_Toc40861201)

[7.1.1. Rx Proc hook 21](#_Toc40861202)

[7.1.2. Tx Proc hook 21](#_Toc40861203)

[7.1.3. 报文转发类型的处理接口 22](#_Toc40861204)

[7.1.4. 释放sk\_buff中的gcnat数据 22](#_Toc40861205)

[7.1.5. 克隆sk\_buff中的gcnat数据 22](#_Toc40861206)

[7.1.6. 根据dev删除快转表项 23](#_Toc40861207)

[7.1.7. 根据conntrack信息删除快转表项 23](#_Toc40861208)

[7.1.8. 根据MAC地址查找快转表项 23](#_Toc40861209)

[7.1.9. 根据conntrack信息查找快转表项 23](#_Toc40861210)

[7.2. 需要SDK提供的硬件表项操作接口 24](#_Toc40861211)

[7.2.1. 添加快转表项 24](#_Toc40861212)

[7.2.2. 删除快转表项 24](#_Toc40861213)

[7.2.3. 判断硬件表项是否超时 25](#_Toc40861214)

[8. 诊断调试 25](#_Toc40861215)

[8.1. 调试打印 25](#_Toc40861216)

[9. 限制条件 25](#_Toc40861217)

[10. 附录 26](#_Toc40861218)

[10.1. 待确认的问题 26](#_Toc40861219)

# 缩写词

|  |  |
| --- | --- |
| **缩写** | **全称** |
| HW | Hardware Platform |
| SDK | Software Development Kit |
| API | Application Interface |
| ct | conntrack |
| PD | Packet Description，包含接收或发送端口、设备等信息 |

# 概述

本文档描述gcnat模块，即快速转发模块的应用、原理、需求、总体架构及详细设计。

## 应用及原理

Gcnat通过监听经过网关设备的数据流转发过程，分析并建立基于流的快速转发规则，提供软件和硬件层的快速转发能力。其过程分为以下几个个阶段：

1. **学习阶段**

通过监听经过网关设备转发的数据流，基于流转发前后的信息对比建立快速转发规则。

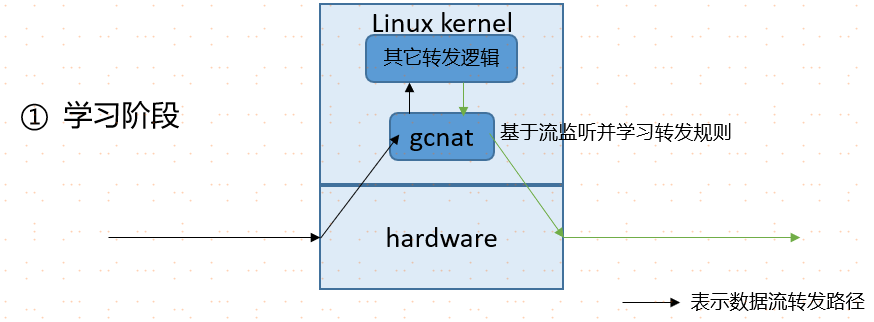


图2-1. 学习阶段流转发模型

为避免网络干扰报文等情况消耗快转资源，仅仅当转发次数达到门限的报文进行快转学习。此门限目前仅对IPv4的三层转发报文有效。其中TCP/UDP分别有独立的门限，默认都为10个报文，可配置。

1. **软件快速转发阶段**

对于已经学习并建立了转发规则的数据流，gcnat模块直接根据转发规则进行快速转发，不再经过后续其它转发逻辑的处理，以提高转发效率。

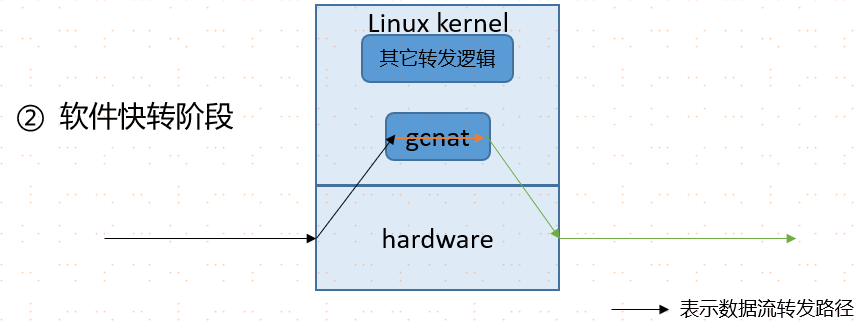


图2-2. 软件快转阶段流转发模型

1. **硬件快速转发阶段**

若硬件支持快转逻辑，开启硬件快转后，若达到硬件快转条件，则会进行硬件快转表项的设置，后续的数据流，将在硬件中直接进行转发处理，不再经由软件层转发，以达到更高的，接近线速的转发性能。

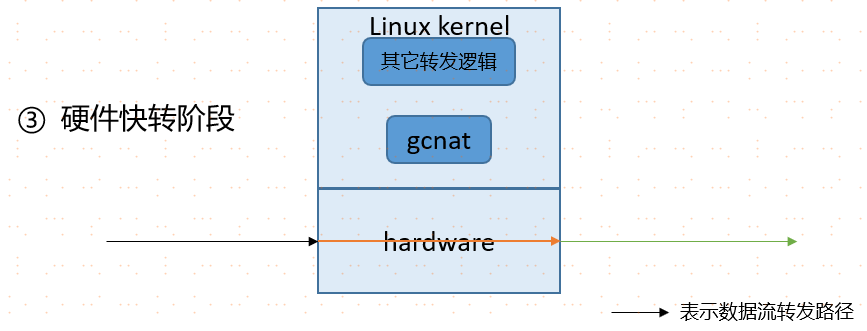


图2-3. 硬件快转阶段流转发模型

硬件快转条件为：

1. 使能了硬件快转（可配置使能或关闭）；
2. 当学习到转发规则后，若1s内该流的数据包达到指定个数（默认为10个报文，可配置）时，触发硬件快转配置表项下发；
3. **快转表项老化阶段**

本设计支持快转表项老化机制，以回收过时的资源。本设计支持硬件和软件表项的独立老化机制。当硬件表项老化后，将从硬件快转阶段回退到软件快转模式。当软件表项老化超时后，将回退到最初的未学习状态，不再进行快转，待重新学习后，再进入快转模式。

硬件表项老化时间为5s(需确认)，软件表项老化时间默认15s，可配置。只有当硬件表项老化后，软件表项才能被老化删除。

## 需求总结

结合Gcnat的应用和原理，以及从功能灵活性、易用性和可维护性方面的考虑，gcnat针对已知单播报文需要提供的细分功能如下：

1. 支持快速转发逻辑的自动监听和学习；
2. 支持软件快速转发；
3. 支持硬件快速转发，可开关配置；
4. 支持配置触发IPv4及4in6 Tunnel TCP/UDP快速转发规则学习的门限报文数量，减少干扰报文对快转资源的消耗；
5. 支持硬件快速转发规则下发的条件（即1s内此流多少个转发报文将触发硬件表项下发）配置，减少不必要的硬件资源消耗；
6. 支持全局开关配置；
7. 支持硬件和软件转发表项的定时老化机制；
8. 支持二层及三层转发规则的读取和显示；
9. 支持必要的诊断调试手段。

# 总体架构

## 模块关系

本章节描述gcnat与系统中其它有关联的功能模块之间的关系，以及各外部模块对gcnat提供的功能支持。

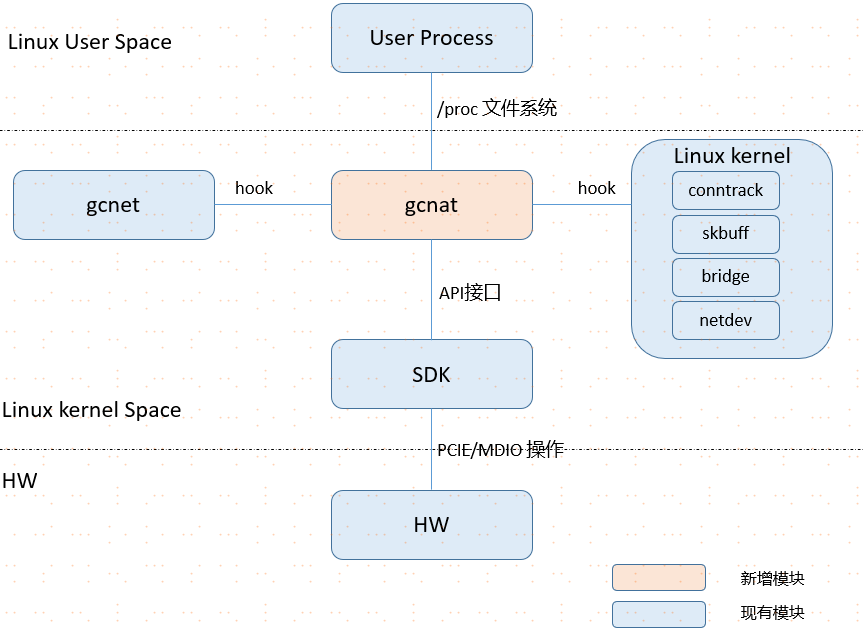


图3-1. 模块关系图

* **gcnat模块**

本设计所涉及的功能模块，提供二、三层快速转发规则的自动学习和软硬件快速转发。

gcnat以内核模块的形式存在，可动态加载于linux内核中或从linux内核中卸载。Gcnat模块通过/proc文件系统，提供用户态进程对gcnat相关参数的配置和信息的获取。

* **gcnet模块**

Gcnat的快转规则学习通过监听和分析报文在转发前后报文头和PD信息的改变，建立快转规则。所以需要在在报文接收和发送时挂载hook，在这两个点拦截和抓取报文的信息。而gcnet模块拥有报文协议栈收发的统一入和出接口，可提供挂载gcnat需要的Rx和Tx hook的处理。

* **User Process**

用户空间的进程统称，并非特定功能模块。其提供用户态程序中对gcnet的配置和状态显示。通过/proc文件操作(read/write)对gcnat进行参数配置，可配置的参数见数据结构章节配置数据部分。

* **SDK**

为硬件快转提供可操作的接口，包括如下接口需求：

* 添加快速转发硬件表项的API接口；
* 删除快速转发硬件表项的API接口；
* 判断硬件表项是否超时的API接口；
* **HW**

硬件物理芯片，与SDK衔接，SDK通过PCIE/GPIO等接口完成对芯片硬件寄存器/表项的配置。对于有硬件快转需求时，需要HW支持快转硬表的配置和根据快转硬表进行转发的能力。

* **Linux Kernel**

此处的Linux Kernel，除了指内核提供的基础内存和文件管理接口外，还包括部分未裁剪的内核网络功能模块。例如conntrack连接跟踪模块、skbuff管理模块及FDB模块。如图3-1所示，各模块为gcnat提供的功能如下：

* **Conntrack连接跟踪模块**

Conntrack为gcnat提供数据流连接状态和流量统计的获取，对于IPv4或IPv4 in IPv6的Tunnel报文，仅仅当以下条件满足时，才进行快转处理。

1. 已经建立了连接跟踪的数据流；
2. 非关联连接，即不存在helper信息的数据流；
3. TCP报文在三次握手之后，conntrack设置了‘should never be early-expired’标志，即IPS\_ASSURED\_BIT置位时；
4. conntrack连接跟踪模块统计的此流ORIGINAL方向的报文数超过配置的阈值时；

由于gcnat与conntrack存在关联，所以当conntrack entry删除时，需要通告gcnat删除相关的快转表项。

Conntrack存在老化机制，当建立了快转规则后，报文不再经过conntrack模块更新连接，所以需要在老化处理流程中，主动刷新快转规则对应的conntrack entry，避免老化。

* **skbuff模块**

skbuff模块维护报文收发过程中的sk\_buff信息。而gcnat需要将私有数据放在sk\_buff中，在收发流程中记录信息，以便建立快转规则。

Gcnat存放私有数据的数据结构为动态申请的cache内存，此数据结构同样作为快转表项记录结构，定义在数据结构章节的表项结构部分。若gcnat据此生成了快转规则，则将数据结构加入到独立的快转Hash链表中维护。否则，在sk\_buff的生命周期结束时，此内存与sk\_buff结构一并释放。

另外对于部分外部转发模块通过克隆sk\_buff用于转发的情况，同样需要申请gcnat并克隆gcnat的信息到新的sk\_buff中，避免后续用克隆的sk\_buff导致gcnat信息不一致而出错。

* **bridge模块**

bridge fdb维护系统二层MAC地址转发表项，即fdb表。当对应的fdb表项删除时，需要删除对应MAC的快转表项。不过对于fdb动态表项由于老化机制导致的表项删除时，并不立即删除fdb表，而是更新fdb表项的update时间为最新的快转表项时间，重新进入老化周期。

另外Gcnat通过bridge报文收发处理逻辑，完成对报文二层或三层转发的区分判断。区分二层或三层转发的区分主要因为：

1. 三层转发情况，需要更新对应WAN口NAT硬件配置；
2. 三层转发流量依赖conntrack模块基于流的统计，完成对快转学习触发门限的判断；

* **netdev模块**

gcnat依赖netdev的状态，仅仅当dev UP时才用于报文转发，相反，若dev link down，则需要删除对应的快转表项。Netdev模块对gcnat提供dev DOWN事件的通告和注册机制。

## 外部模块处理流程

本章节主要描述外部模块对gcnat功能支持所做的流程改动或处理。

### gcnet

如之前模块关系中gcnet提供的功能描述，此模块提供gcnat挂载Rx和Tx hook的接口。其接口如下：

* **Rx hook：**

挂载的接口函数为：gcnet\_dev\_pkt\_recv\_hook()

说明：在报文接收处理之前，调用gcnat rx hook函数，以便得到最真实的转发前报文原始信息。

* **Tx hook**

挂载的接口函数为：gcnet\_dev\_hard\_start\_xmit()

说明：在靠近dev的发送函数之前，调用gcnat tx hook函数，以便得到最终的转发后报文信息。

### Skbuff

如之前模块关系中skbuff提供的功能描述，skbuff主要处理两件事件的通告，即：

1. 克隆sk\_buff数据结构；
2. 释放sk\_buff数据结构；

对于克隆sk\_buff数据结构时，需要同时克隆gcnat的私有数据结构。在\_\_copy\_skb\_header()函数中通过调用gcnat的hook函数进行事件通告，在hook中完成gcnat私有数据结构（类型为struct gcnat\_session\_s）的克隆。

对于释放sk\_buff数据结构时，若此sk\_buff指向的gcnat私有数据结构并未添加到快转规则Hash链表中，则需要同时释放此gcnat私有数据结构。在skb\_release\_head\_state()中通过调用gcnat的hook函数进行事件通告，在hook中完成gcnat私有数据结构的释放。

### Bridge

如之前模块关系中bridge提供的功能描述，bridge fdb需要处理的事情为：

1. 在FDB老化周期中，对gcnat快转使用的MAC地址进行刷新而非直接老化；

gcnat提供判断指定MAC是否为gcnat使用，且返回此MAC在gcnat中最新的update时间，以便fdb据此update fdb entry的更新时间。FDB老化处理函数为br\_fdb\_cleanup()。

1. 当FDB表项删除时，通告gcnat删除对应的快转表项；

Fdb在删除fdb表项的统一接口函数fdb\_delete()中，调用gcnat提供的基于MAC的快转表项删除接口完成对应表项的删除。

另外，bridge报文收发逻辑中，需要处理的事情为：

1. 在bridge报文处理入口函数br\_handle\_frame()函数中，标记PACKET\_HOST时，设置报文L3转发标记。Gcnat提供设置L3标记的hook函数。
2. 在br\_handle\_frame\_finish()中，对于到本机或需要三层转发的报文，会通过br\_pass\_frame\_up()继续上送。而对于需要二层转发的报文，将查找FDB表项后，根据FDB表项进行转发，转发函数为br\_forward()，所以在调用此接口函数时，通过Gcnat提供设置的hook函数完成L2转发标记的设置。
3. 对于Router模式时的下行流量，即PON到Ethernet LAN口的流量，其RX流程不会到bridge模块，则需要通过在bridge TX流程中，即br\_dev\_xmit()中找到FDB转发表项，调用br\_forward()转发报文时，完成对三层转发标记的设置。

### conntrack

Gcnat依赖conntrack维护的连接跟踪信息。为避免建立了快转规则的conntrack连接记录老化，需要在conntrack的老化处理函数gc\_worker()中判断老化超时之前，对gcnat使用的连接记录进行刷新。此处gcnat提供基于coontrack连接记录判断是否有对应快转表项的接口。

另外，当某个conntrack连接删除时，对应的快转表项也需要清除。Conntrack entry的删除接口为destroy\_conntrack()，在此接口中，调用gcnat模块提供的基于coontrack entry删除快转表项的接口。

### netdev

当netdev接口DOWN后，需要删除该dev相关的快转表项。Netdev提供消息通告的注册机制。注册函数为register\_netdevice\_notifier(struct notifier\_block \*nb)，在其回调函数中对NETDEV\_DOWN事件进行处理，调用基于dev删除快转表项的API。

# 详细设计

## Gcnat内部模块划分

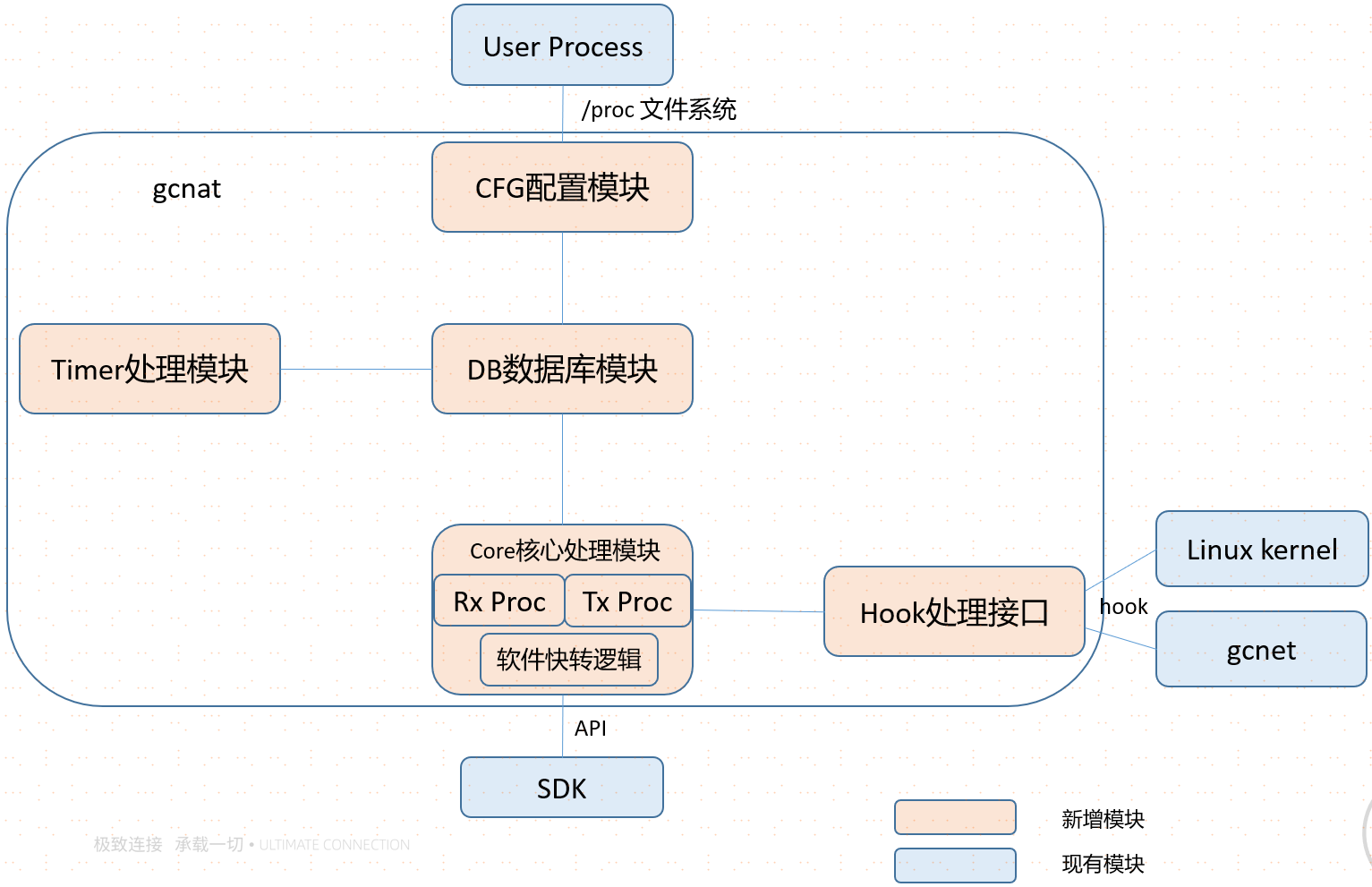


图4-1. 内部模块关系图

如图4-1，gcnat内部从逻辑层面，可划分为CFG配置模块、DB数据库模块、Timer处理模块、Hook处理接口以及Core核心处理模块（包括学习转发规则中的Rx和Tx流程处理逻辑和软件快转处理逻辑）。各模块的功能任务如下：

### CFG配置模块

CFG配置模块提供gcnat模块各种参数对外配置读取的机制和接口，例如功能开关，timer interval，tcp/udp gate threshold等参数的配置和L2/L3快转表项的读取接口等。

由于gcnat运行于内核态，CFG配置模块通过/proc机制，对外提供配置和读取操作接口。详细接口参见 数据结构章节 配置数据部分。

### DB数据库模块

DB子模块主要完成以下任务：

1. 配置参数的定义和保存；
2. Gcnat私有数据结构的定义，及该数据结构的cache内存的分配和释放接口；
3. Gcnat快转规则HASH链表的定义和维护，包括添加、删除、遍历等操作接口；

具体数据结构定义参见数据结构章节。

### Timer处理模块

gcnat模块支持快转表项的老化机制。

对于硬件表项的老化时间，固定为5s，需要确认？？？

软件表项老化周期T默认为15s，可配置。通过一个全局定时器进行老化处理。且对于存在硬件表项的软件表项将跳过当前的老化周期，所以实际的老化时间为T~3T之间。

### Hook处理模块

Gcnat对外部模块提供的操作接口，都以hook函数的方式提供，其所需要的hook处理接口在总体架构章节已经说明，归纳如下：

1. Rx Proc hook；
2. Tx Proc hook；
3. 克隆gcnat私有数据结构的hook；
4. 释放gcnat私有数据结构的hook；
5. 根据MAC删除快转表项的hook;
6. 判断是否有快转表项使用指定MAC的hook；
7. 根据conntrack entry删除快转表项的hook;
8. 根据dev删除快转表项的hook；
9. 设置二层或三层转发标记的hook；

### Core核心处理模块

此模块完成：

1. 快转规则的学习；
2. 对建立了快转规则的数据流进行快转处理；

## 快转学习流程

快转的学习通过监听和分析同一报文转发过程中在Rx和Tx阶段的报文内容和相关转发属性（转发接口等）来建立快速转发规则。

监听通过在转发流程Rx入口和Tx出口处挂载hook函数来截取报文。在hook中的处理流程如下。

### Rx Hook Proc

Rx Hook应挂载在报文接收处理的最前段。Rx Hook中完成转发前原始报文的报文头解析（包括MAC, VLAN, IP, L4头）和信息记录。其流程如下：



图4-2. Rx Proc流程图

其中，

1. gcnat enable为全局开关配置。
2. 报文解析完成MAC, VLAN, PPPOE, IP, L4 header内容的解析，以及转发需要的dev, gemport, tcont, queue, phyport等参数。
3. 为报文分配的动态数据结构，要么在后续的Tx Hook Proc中，添加到规则Hash链表中，要么在sk\_buff释放时，被一并释放。

### Tx Hook Proc

Tx Hook应挂载在报文转发处理的最末段，以便得到最终的转发时报文内容。Tx Hook中完成经过网关转发后的报文头解析（包括MAC, VLAN, IP, L4头）和发送接口信息的记录，并将转发后的报文头与转发前的原始报文头进行比较，并建立转发规则。其流程如下：



图4-3. Tx Proc流程图

其中，

1. 错误检查主要用于过滤类似本机发送的数据报文，及非转发报文。
2. 报文解析同样完成转发时报文的MAC, VLAN, PPPOE, IP, L4 header内容的解析，以及转发需要的转发接口的dev, gemport, tcont, queue, phyport等参数。
3. 快转逻辑的转发操作类型如下：

|  |  |
| --- | --- |
| ***操作类型*** | ***说明*** |
| GCNAT\_SET\_MAC | 转发时替换SMAC & DMAC |
| GCNAT\_SET\_DIP\_V4 | 对于IPv4报文，转发时替换DIP |
| GCNAT\_SET\_SIP\_V4 | 对于IPv4报文，转发时替换SIP |
| GCNAT\_SET\_DPORT\_TCP | 转发时替换TCP destination port |
| GCNAT\_SET\_SPORT\_TCP | 转发时替换TCP source port |
| GCNAT\_SET\_DPORT\_UDP | 转发时替换UDP destination port |
| GCNAT\_SET\_SPORT\_UDP | 转发时替换UDP source port |
| GCNAT\_SET\_DIP\_V6 | 对于IPv6报文，转发时替换DIP |
| GCNAT\_SET\_SIP\_V6 | 对于IPv6报文，转发时替换SIP |
| GCNAT\_SET\_TOS | 转发时替换TOS字段 |
| GCNAT\_SET\_TC | 转发时替换TC字段 |
| GCNAT\_SET\_L3\_CSUM | 转发时替换IP header checksum |
| GCNAT\_SET\_L4\_CSUM\_TCP | 转发时替换TCP checksum |
| GCNAT\_SET\_L4\_CSUM\_UDP | 转发时替换UDP checksum |
| GCNAT\_ADD\_OUTER\_VLAN | 转发时添加outer VLAN tag |
| GCNAT\_ADD\_INNER\_VLAN | 转发时添加inner VLAN tag |
| GCNAT\_DEL\_VLAN | 转发时删除VLAN tag转发，有inner tag则先消除inner tag，两次GCNAT\_DEL\_VLAN操作可达到消除双层tag的目的。 |
| GCNAT\_SET\_OUTER\_VLAN | 转发时替换outer VLAN tag |
| GCNAT\_SET\_INNER\_VLAN | 转发时替换inner VLAN tag |
| GCNAT\_ADD\_PPPOE | 转发时添加PPPOE头 |
| GCNAT\_DEL\_PPPOE | 转发时删除PPPOE头 |
| GCNAT\_ADD\_IPV6 | 转发时添加IP in IPv6的IPv6头 |
| GCNAT\_DEL\_IPV6 | 转发时删除IP in IPv6的IPv6头 |
| GCNAT\_TTL\_DEC | 转发时TTL字段减一 |

## 软件快转流程

对于已经学习到快转规则的数据流，在硬件表项未设置之前，可通过软件快转来提高转发效率。软件快转在Rx hook中进行快速转发处理，流程如下：



图4-4. 软件快转处理流程图

其中，

1. 1s内此流的数据报文转发上限检查用于 软件或硬件快转的区分，是为节省不必要的硬件资源考虑；
2. 对于硬件快转，需要硬件提供支持，SDK提供配置硬件表项的接口，当前的SDK接口需求如下：
   * 添加快速转发硬件表项的API接口；
   * 删除快速转发硬件表项的API接口；
   * 判断硬件表项是否超时的API接口；

在下发L3转发硬件快转表项时，需要同时根据转发规则更新WAN口的NAT配置。

## Linux Kernel模块事件的Hook处理

如总体架构章节描述，由于功能模块之间的依赖关系，gcnat需要关心部分Linux Kernel模块的消息事件，并通过hook调用的方式，响应和处理对应的事件。其在gcnat hook中的处理逻辑如下：

* 克隆gcnat私有数据结构的hook

无特殊逻辑，为克隆的私有结构指针分配内存后，完成信息拷贝即可。

* 释放gcnat私有数据结构的hook；

仅仅释放未成功学习的表项数据结构。学习成功的表项数据结构在Hash链表中继续维护。

* 根据MAC删除快转表项的hook;

无论源或目的MAC，只要快转表项存在使用该MAC的，都应当删除。其具体处理逻辑为：遍历所有快转表项，txInfo和rxInfo中无论dstMac或srcMac，只要与指定MAC相同即删除。

* 判断是否有快转表项使用指定MAC的hook；

此hook用于FDB update，所以仅仅判断txInfo中的dstMac或srcMac与指定MAC相同，即转发后的MAC地址必须都学习到了才能转发。

* 根据connection entry查询和删除快转表项的hook；

connection entry为conntrack模块为数据包生成的连接记录项，包含初始和应答两个方向。所以查找和删除时，需要用两个方向的信息进行操作。任何一个方向的记录存在，则表示查找到或需要删除。

任一方向的查找标准为srcIp、dstIp、L4 protocol、L3 srcPort、L3 dstPort相同。

* 根据dev删除快转表项的hook；

来自于netdev模块的通告，当dev DOWN时，删除dev关联的快转表项。在hook处理函数中，遍历所有快转表项，根据表项的rxInfo.dev和txInfo.dev判断，若其中任一一个与指定dev相同，则删除该快转表项。

* 设置二层或三层转发标记的hook；

gcnat\_session\_t中定义有l2, l3的标记位，此hook根据LOG类型设置对应的标记位，GCNAT\_LOG\_TYPE\_BRIDGE类型对应设置l2标记位，而GCNAT\_LOG\_TYPE\_ROUTE类型对应设置l3标记位。

# 数据结构

## 配置数据

1. **gcnat功能开关**

开启或关闭gcnat功能，默认为开启状态，用户态进程中，可通过/proc/gcnat/enable 进行配置。其配置变量定义如下：

static int gcnat\_enable = 1;

其在/proc/gcnat/enable 中进行读写的数据格式为不带0x前缀的8位字符宽的16进制数，例如00000001（gcnat\_enable = =1）或00000000（gcnat\_enable ==0）。

1. **gcnat硬件加速开关**

开启或关闭gcnat硬件加速功能（通过设置硬件表项实现加速转发），默认为开启状态，用户态进程中，可通过/proc/gcnat/hw\_enable 进行配置。其配置变量定义如下：

static int gcnat\_hw\_enable = 1;

其在/proc/gcnat/hw\_enable 中进行读写的数据格式为不带0x前缀的8位字符宽的16进制数，例如00000001或00000000。

1. **gcnat调试开关**

gcnat将调试打印信息区分为以下4种类型：

* 调试信息打印；
* 警告提示信息打印；
* 错误报告信息打印；
* 报文处理流程信息打印；

用户可按需配置打印其中的某一种或多种类型的调试信息。也支持关闭调试信息打印。默认为开启所有的调试打印。

其配置变量定义如下：

#define GCNAT\_DEBUG 1<<0 //调试时用的打印类型

#define GCNAT\_WARN 1<<1 //警告提示的打印类型

#define GCNAT\_ERROR 1<<2 //错误报告的打印类型

#define GCNAT\_PKT 1<<3 //报文处理流程的打印类型

static int gcnat\_debug = 0xff;

用户态进程中，可通过/proc/gcnat/debug 进行配置。其在该文件中进行读写的数据格式为不带0x前缀的8位字符宽的16进制数，例如00000001（GCNAT\_DEBUG）或00000005（both GCNAT\_DEBUG & GCNAT\_ERROR）。

1. **gcnat timer间隔时间参数配置**

gcnat timer用于清除老化的加速表项。默认timer间隔周期为15s。用户态进程中，可通过/proc/gcnat/gcnat\_timer进行配置。其配置变量定义如下：

#define GCNAT\_GC\_TIME (15\*HZ)

static unsigned int gcnat\_expire\_time = GCNAT\_GC\_TIME;

其在/proc/gcnat/gcnat\_timer 中进行读写的数据格式为不带0x前缀的8位字符宽的16进制数。

1. **gcnat快转的触发阈值配置**

为避免对个别干扰报文的处理带来性能负担和资源浪费，仅仅对转发流量达到触发阈值的报文才进行gcnat快转处理。针对TCP和UDP报文有各自独立的触发阈值，默认为10个报文，定义如下：

static unsigned int gcnat\_tcp\_gate\_cnt = 10;

static unsigned int gcnat\_udp\_gate\_cnt = 10;

用户态进程中，可通过/proc/gcnat/gcnat\_tcp\_gate\_cnt和/proc/gcnat/gcnat\_udp\_gate\_cnt进行配置。其在该文件中进行读写的数据格式为不带0x前缀的8位字符宽的16进制数。

1. **gcnat硬件快转门限配置**

为节约硬件资源，仅当1s内需要快转的报文操作指定门限时，才下发硬件快转表项。该门限默认为10个数据包，定义如下：

static unsigned int gcnat\_fps = 10;

用户态进程中，可通过/proc/gcnat/gcnat\_fps进行配置。其在该文件中进行读写的数据格式为不带0x前缀的8位字符宽的16进制数。

## 状态数据

### 统计信息

记录二层转发和三层转发的快转表项数目。

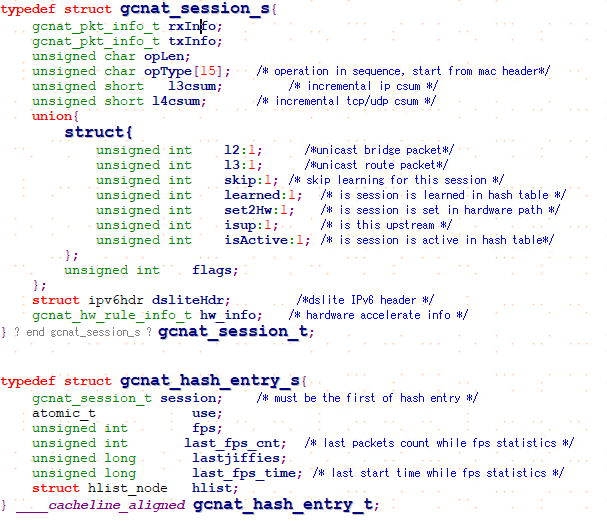
static unsigned int gcnat\_now\_l2\_uc\_entry;

static unsigned int gcnat\_now\_l3\_uc\_entry;

当添加快转表项到hash链表中，或从hash链表中删除快转表项时，进行计数更新。

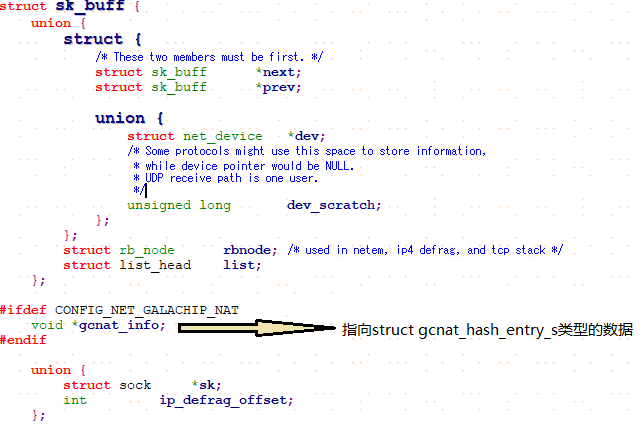
### 表项结构

快转表项信息用于记录报文转发过程中的报文头信息、PD信息、快转操作信息及其它辅助信息等。数据结构定义如下：



详细定义参见gcnat.h文件的定义。

快转表项信息结构也称为gcnat模块私有数据结构，挂载到sk\_buff中，用于存放gcnat需要的信息，参见struct sk\_buff结构体定义。



若一个报文在转发过程中，符合gcnat学习的条件，则生成快转规则，并将此快转数据结构加入快转规则链表中，避免此前分配的gcnat快转数据结构在Tx 完成后sk\_buff资源回收时被释放。

若一个报文在转发过程中，不符合gcnat学习的条件，之前生成的gcnat快转数据结构将在报文处理流程中释放sk\_buff时，一并释放（参考sk\_buff释放时的通告处理章节）。

### 快转规则Hash链表

快转规则链表为hash链表。Hash bucket深度为4096，hash key为{dstIP, srcIP, L4port}，对于IPv6，hash key为{dstIP, srcIP, L4port, L4\_Protocol}。Node为struct gcnat\_hash\_entry\_s类型的数据。如下：

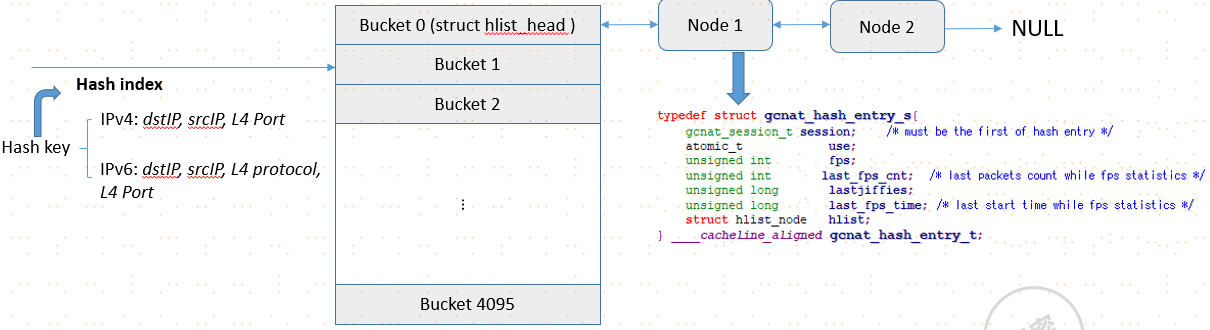


图5-1. 快转规则Hash链表图

Hash链表定义如下：

#define GCNAT\_HASH\_SIZE 4096

struct hlist\_head gcnat\_hash\_hlist[GCNAT\_HASH\_SIZE]

对于状态数据（包括统计和转发规则信息），可通过/proc/gcnat/gcnat\_l2\_entry\_list和/proc/gcnat/gcnat\_l3\_entry\_list 文件分别对二层转发规则和三层转发规则进行读取。

## 数据同步

由于外部模块调用的hook处理以及本模块timer处理分布在不同的内核线程中，对gcnat的状态数据进行读写，此处需要通过使用自旋锁对状态数据的访问进行同步。

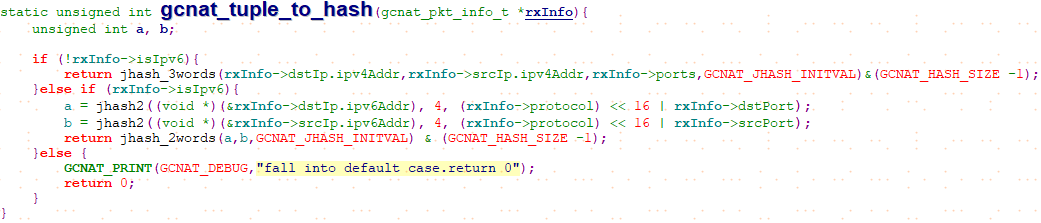
自旋锁定义如下：

static spinlock\_t gcnat\_lock;

# 算法描述

## 快转软表的Hash算法

对于IPv4，hash key为：{dip, sip, l4\_port}，对于IPv6，hash key为：{dip, sip, l4\_protocol, l4\_port}。Hash算法如下：



## 快转规则的构建算法

详见gcnat\_handle\_frame\_tx()函数中，“do learning rule”部分代码。

# API接口

## 对外提供的hook处理接口

### Rx Proc hook

/\*

功能：报文转发流程中，接收处理的钩子处理函数

参数：\*\* pskb[IN] – 报文信息，包含报文数据及Packet Description信息。

返回值：返回1为不再进行后续转发逻辑处理，0为继续后续转发逻辑处理

\*/

int gcnat\_handle\_frame\_rx(struct sk\_buff \*\*pskb)

### Tx Proc hook

/\*

功能：报文转发流程中，发送处理的钩子处理函数

参数：\*\* pskb[IN] – 报文信息，包含报文数据及Packet Description信息。

返回值：0为处理成功，否则为错误码

\*/

int gcnat\_handle\_frame\_tx(struct sk\_buff \*\*pskb)

### 报文转发类型的处理接口

/\*

功能：设置报文为二层转发或者三层转发的接口

参数：\* pskb[IN] – 报文信息，包含报文数据及Packet Description信息。

type[IN] – GCNAT\_LOG\_TYPE\_BRIDGE | GCNAT\_LOG\_TYPE\_ROUTE。

data [IN] – 对应类型的标记值，1为SET，0为UNSET。

返回值：GCNAT\_RET\_OK为处理成功，GCNAT\_RET\_FAIL为错误

\*/

int gcnat\_log\_action(struct sk\_buff \*skb, GCNAT\_LOG\_TYPE type, long data)

### 释放sk\_buff中的gcnat数据

/\*

**功能：**释放sk\_buff中的无效（未学习成功）的gcnat私有数据

**参数：**\*ptr[IN] – gcnat\_session\_t类型的私有数据指针。

**返回值：**GCNAT\_RET\_OK为处理成功，GCNAT\_RET\_FAIL为错误

**说明：**在报文处理过程中，可能因为不满足学习条件导致并为据此报文生成快转规则，则需要在该报文转发后，释放sk\_buff的时候，同时释放该报文在Rx proc中申请的私有数据内存资源。

\*/

int gcnat\_info\_free(void \*ptr)

### 克隆sk\_buff中的gcnat数据

/\*

**功能：**克隆sk\_buff中的gcnat私有数据给另一个克隆出来的sk\_buff

**参数：**\* pnew[IN] – gcnat\_session\_t类型的目的私有数据指针。

\* old[IN] – 被复制的gcnat\_session\_t类型私有数据指针。

**返回值：**GCNAT\_RET\_OK为处理成功，GCNAT\_RET\_FAIL为错误

**说明：**部分转发模块可能克隆独立的sk\_buff进行转发处理，此时需要将gcnat的信息一并拷贝过去，以便后续正确生成转发规则。

\*/

int gcnat\_info\_clone(void \*pnew, void \*old)

### 根据dev删除快转表项

/\*

**功能：**删除指定dev相关的所有快转表项

**参数：**\* dev[IN] – 指定dev信息。

**返回值：**无

**说明：**TBD

\*/

void gcnat\_del\_l3\_entry\_by\_dev(struct net\_device \*dev)

### 根据conntrack信息删除快转表项

/\*

**功能：**删除指定conntrack相关的所有快转表项

**参数：**\* ct[IN] – 指定ct信息。

**返回值：**无

**说明：**TBD

\*/

void gcnat\_del\_l3\_entry\_by\_ct(struct nf\_conn \*ct)

### 根据MAC地址查找快转表项

/\*

**功能：**判断是否有使用指定MAC地址的快转表项，并返回该使用该MAC的最近更新的时间。

**参数：**\* dmac [IN] – 指定MAC地址。

\* hitTime [OUT] – 此MAC地址对应的最近update过的快转表项时间。

**返回值：**存在返回true，否则返回false

**说明：**在FDB老化机制中，通过此接口或则最近update过的快转表项时间，并更新fdb表，以避免在快转表项老化前，FDB表项被老化掉。

\*/

bool gcnat\_l2\_entry\_use(const unsigned char \*dmac, unsigned long \*hitTime)

### 根据conntrack信息查找快转表项

/\*

**功能：**用于判断是否有conntrack连接对应的快转表项

**参数：**\* ct[IN] – 指定ct信息。

**返回值：**无

**说明：**在conntrack老化流程中，对于快转表项对应的conntrack，将通过refresh来避免老化。

\*/

bool gcnat\_ l3\_entry\_use\_by\_ct(struct nf\_conn \*ct)

## 需要SDK提供的硬件表项操作接口

### 添加快转表项

/\*

功能：添加硬件快转表项

参数：\*session[IN] – 需要添加的表项信息，包括匹配流规则和转发规则

\*index[OUT] – 添加的硬件表项对应的表项索引

返回值：0为添加成功，否则为失败错误码

说明：调用此接口时，session->hw\_info.wan\_index需要

\*/

int sdk\_nat\_hook\_1(gcnat\_session\_t \*session, int \*index)

/\*

功能：添加NAT映射规则

参数：wandev\_index [IN] – 表项索引，与sdk\_nat\_hook\_1时需要将session->hw\_info.wan\_index

\* mac [IN] – wandev的MAC地址

sip\_rep[IN] – 是否做SIP的映射

sip[IN] – wandev的IP

pppoe[IN] – 是否时pppoe连接

ppp\_session [IN] – pppoe session id

返回值：0为添加成功，否则为失败错误码

\*/

int gcnat\_ipv4\_L3\_natinfo\_set(int wandev\_index, unsigned char \*mac,

bool sip\_rep, sip, bool pppoe,

unsigned short ppp\_session)

### 删除快转表项

/\*

功能：删除指定的硬件快转表项

参数：flow\_index[IN] – 需要删除的硬件表项索引，在创建时返回

返回值：0为删除成功，否则为失败错误码

\*/

int sdk\_delete\_flow(int flow\_index)

### 判断硬件表项是否超时

/\*

功能：判断指定硬件快转表项是否超时

参数：flow\_index[IN] – 需要判断的硬件表项索引，在创建时返回

\*aged[OUT] – true超时，false为未超时

返回值：0为获取成功，否则为失败错误码

\*/

int sdk\_check\_flow\_age(int flow\_index, bool \*aged)

# 诊断调试

## 调试打印

调试打印使用以下统一的宏定义：

#define GCNAT\_PRINT(level, format, ...)

其中level为打印信息的类型，定义见数据结构章节中的 gcnat调试开关部分，可扩展。

# 限制条件

1. 当Gcnat快转规则生效期间，对应的conntrack信息不再更新，例如统计计数等。 所以其它依赖于conntrack模块且需使用此信息的模块将存在使用限制。
2. 关闭HW开关，将删除硬件表项，软件表项仍然保留，此时将进行软件快转，但重新开启HW开关，并不会立即下发硬件表项，将等待此软件表项超时后，重新学习后，才会下发硬件表项。也即开启HW开关后，仅仅重新学习的表项才会下发硬件表项，对于当前已经存在软件表项的转发规则不起效。
3. Gcnat仅对已知单播报文有效，对于组播、广播和未知单播报文不生效。
4. Gcnat软件快转逻辑在net filter之前，对于存在快转规则的流量，不再受net filter模块的影响，同样也无法被tcpdump等抓包工具获取到。

# 附录

## 待确认的问题

1. Gcnat除了与FDB表项有依赖外，是否与路由表项也有依赖。但目前没看到（ARP/router）的修改通告。是否conntrack已经覆盖了该情况？
2. Dev down是否会clear fdb？若有，则是否还需要此事件，因为clear fdb会做gcnat对应表项 的删除？
3. Gcnat\_handle\_frame\_tx()中标记ct->gcnat\_info[DIR]=1的时候，是否应同时考虑IPv4及IPv4 in IPv6。否则IPv4 in IPv6这种tunnel报文的快转规则将无法在conntrack销毁时立即被删除掉。
4. 是否应添加在FDB表项删除的时候通告gcnat 删除使用该MAC的entry。即在fdb\_delete()中进行通告，但效率如何保证？。
5. 克隆Sk\_buff时克隆gcnat info的调用是否能 统一放在 \_\_copy\_skb\_header()中，以简化代码流程。
6. br\_fdb\_cleanup()中 gcnat相关代码的 {} 未在宏内配对。
7. sdk\_nat\_hook\_1()和gcnat\_ipv4\_L3\_natinfo\_set()两个函数的功能行为定义需要再确认？