目录

1 零拷贝的实现方案

1.1 ntzc 背景

到现在为止零拷贝有两个比较成熟的实现，一个为 PF_RING，一个为 nta,据了解，PF_RING 在网卡到内核之间仍然存在一次拷贝，而 nta 年代久远，存在很多的问题。

我们的项目所使用的零拷贝源码为 ntzc， ntzc 是网络上的开源项目，其本身也是根据 nta 改造而成，详细参考网站：http://linux.chinaunix.net/bbs/viewthread.php?tid=1161364

ntzc 的源码参照 svn 上的项目 http://code.google.com/p/ntzc/

1.2 ntzc 概述

1.2.1 ntzc 原始代码目录层次

```
ntzc
|-- nta
|      |-- Makefile
|      |-- control.c
|      |-- control.h
|      |-- send.c
|      |-- sniff.c
`-- zc
       |-- Makefile
       |-- README
       |-- bnx2.c
       |-- bnx2.h
       |-- bnx2_fw.h
       |-- bnx2_fw2.h
       |-- bvl.c
       |-- bvl.h
       |-- igb
       |      |-- Makefile
       |      |-- e1000_82575.c
       |      |-- e1000_82575.h
       |      |-- e1000_api.c
       |      |-- e1000_api.h
       |      |-- e1000_defines.h
       |      |-- e1000_hw.h
       |      |-- e1000_mac.c
       |      |-- e1000_mac.h
```

```
|   |-- e1000_manage.c
|   |-- e1000_manage.h
|   |-- e1000_mbx.c
|   |-- e1000_mbx.h
|   |-- e1000_nvm.c
|   |-- e1000_nvm.h
|   |-- e1000_osdep.h
|   |-- e1000_phy.c
|   |-- e1000_phy.h
|   |-- e1000_regs.h
|   |-- igb.h
|   |-- igb_ethtool.c
|   |-- igb_main.c
|   |-- igb_param.c
|   |-- igb_regtest.h
|   |-- kcompat.c
|   |-- kcompat.h
|   `-- kcompat_ethtool.c
|-- ixgbe
|   |-- Makefile
|   |-- Module.supported
|   |-- ixgbe.h
|   |-- ixgbe_82598.c
|   |-- ixgbe_82599.c
|   |-- ixgbe_api.c
|   |-- ixgbe_api.h
|   |-- ixgbe_common.c
|   |-- ixgbe_common.h
|   |-- ixgbe_dcb.c
|   |-- ixgbe_dcb.h
|   |-- ixgbe_dcb_82598.c
|   |-- ixgbe_dcb_82598.h
|   |-- ixgbe_dcb_82599.c
|   |-- ixgbe_dcb_82599.h
|   |-- ixgbe_dcb_nl.c
|   |-- ixgbe_ethtool.c
|   |-- ixgbe_fcoe.c
|   |-- ixgbe_fcoe.h
|   |-- ixgbe_main.c
|   |-- ixgbe_mbx.c
|   |-- ixgbe_mbx.h
|   |-- ixgbe_osdep.h
|   |-- ixgbe_param.c
|   |-- ixgbe_phy.c
```

```
|      |-- ixgbe_phy.h
|      |-- ixgbe_sriov.c
|      |-- ixgbe_sriov.h
|      |-- ixgbe_sysfs.c
|      |-- ixgbe_type.h
|      |-- kcompat.c
|      |-- kcompat.h
|      |-- kcompat_ethtool.c
|      |-- modules.order
|      `-- set_irq_affinity.sh
|-- nta.c
|-- nta.h
|-- pcnet32.c
|-- sky2.c
|-- sky2.h
|-- zc.c
`-- zc_comm.h
```

其中，包含了大量的修改后的驱动示例，包括如下

igb, for Intel 82575
ixgbe, for Intel 82598
bnx2, for BCM Server GE
pcnet32.c, for [AMD] 79c970, used by VmWare virtual machine
sky2.h sky2.c, for Marvell NIC

1.2.2 ntzc 目录层次简化

此处，为了方便说明，我们将简化其目录结构，方便阅读。简化后如下：

```
ntzc
|-- nta
|      |-- Makefile
|      |-- control.c
|      |-- control.h
|      |-- send.c
|      |-- sniff.c
`-- zc
       |-- Makefile
       |-- bvl.c
       |-- bvl.h
       |-- nta.c
       |-- nta.h
       |-- zc.c
       `-- zc_comm.h
```

其中我们去掉了大量的网卡驱动文件，暂时不做说明。

1.2.3 ntzc 的模块结构

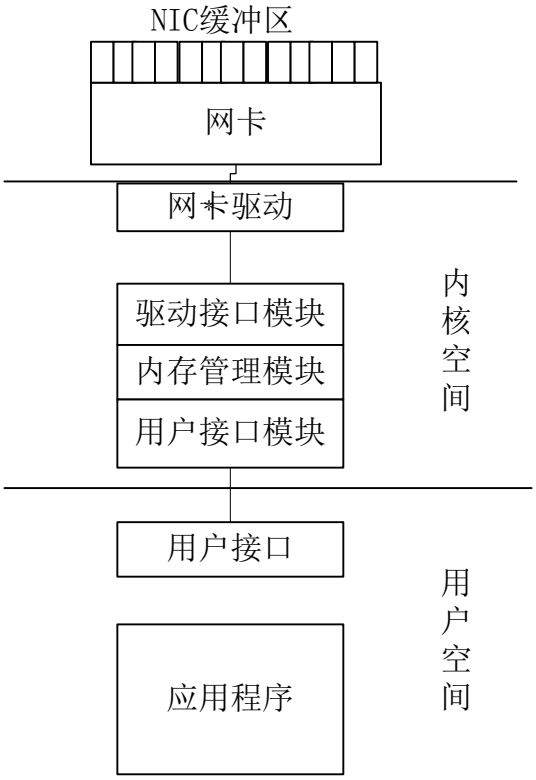

图 1

表 1 Ntzc 的文件说明

文件	所在文件夹	所属模块	功能
Bvl.h bvl.c	Zc	内存管理模块	Ntzc 的核心模块，为用户自定义的内存缓冲区提供管理，包括分配，回收，计数等等功能
Nta.h Nta.c	Zc	驱动接口模块	为网卡驱动提供了内存管理接口，网卡驱动将利用这些接口将数据包转移到自定义的缓冲区，而忽略协议栈缓冲区
Zc.c Zc_comm.h	Zc	用户接口模块	为内核缓冲区与用户的交互提供管理
Control.h Control.c	Nta	用户接口	为用户提供零拷贝接口，用户将利用零拷贝接口完成自己的功能
Send.c Sniff.c	nta	应用程序	零拷贝使用实例应用程序，可以不研究

1.2.4 ntzc 的整体实现思路

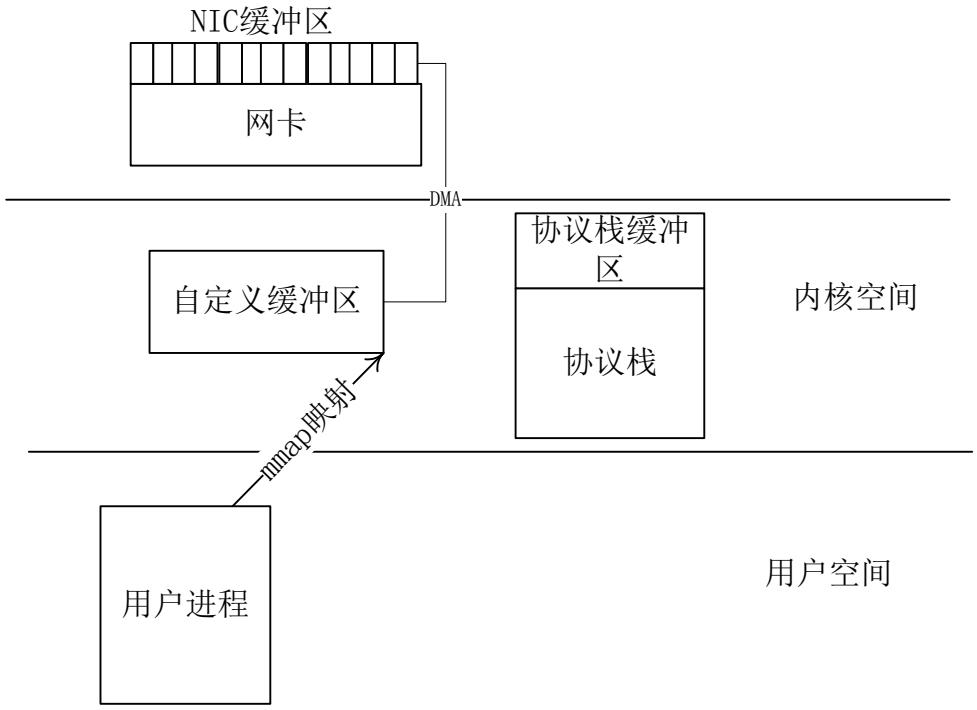

图 2

Ntzc 通过修改过的驱动将数据包劫持到自定义的缓冲区（DMA 传输），用户进程 mmap 到此内存区域，从而达到 零拷贝的目的，在此过程中，显而易见，网卡数据将无视内核协议栈缓冲区的存在，TCP/IP 协议栈完全作废，故使用零拷贝期间，无法正常上网。

NTZC 的设计中，认为既然用户空间可以以足够低的成本收发报文了，因此取消了网卡驱动和 Linux 协议栈之间的交互，当然，从 NTZC 管理的报文内存中拷贝一份出来交给标准协议栈处理，也是很容易的事情。

由于优先考虑到不对内核打补丁，因此没有修改 sk_buff 的内存管理机制，相反的，实现了和 sk_buff 接口语义几乎相同，但内存管理机制发生变化的报文数据结构（m_buf，只是借用了 BSD 里面的数据结构名称，骨子里就是对 sk_buff 的复制）。使用这个 m_buf 数据结构及 API，需要对网卡驱动进行修改。

正因为是个通用的零拷贝支持模块，网卡驱动的修改很容易，全局替换 sk_buff 的操作到 m_buf 的操作即可，因此，理论上任何在 Linux 中已经有源代码支持的 NIC Driver，都很容易被改造成 NTZC 的 NIC 驱动，只是，这块网卡将不能在被 Linux 协议栈使用。NTZC 的代码中也给出了 Intel 82575 改造后的驱动作为例子。

最后，提供一个配套的用户空间 API 代码，以帮助应用程序方便的访问零拷贝的编程接口。对应的收包和发包示例程序也包含在内。

现在用户空间 API 的定义还是一个非标准的私有接口，未来可能会考虑发展成和 libpcap

接口一致。

1.2.5 内存管理模块实现

对于 ntzc 的内存管理模块，由于此模块占据了极其重要的地位，在此做出特别说明（详细参见 bvl.h, bvl.c）。由于此模块为 nta 中自带模块，无详细设计文档，下图为本人自我总结，供参考，若有错误，望指正。

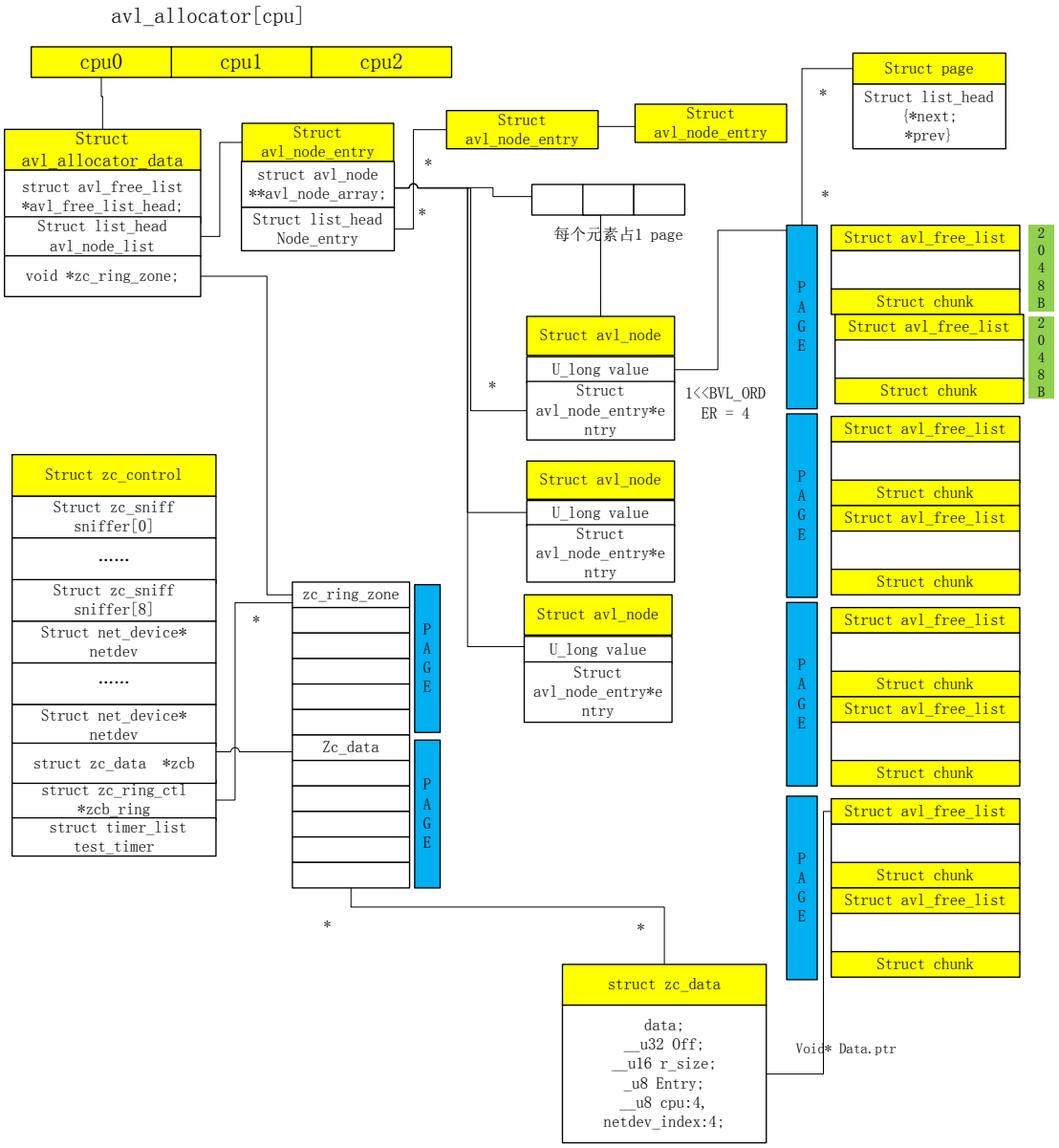

图 3

2 零拷贝与 snort 的融合

2.1 概述

若使用 ntzc 的原始程序，与 snort 融合必然存在重大问题，而且，ntzc 本身并不完善，在此基础上我做了一定程度的修改，修正了一些小的 bug，及增加了一些接口文件。
改正方案如下：

1. Debug 过程不详述。
2. Nta.h 中增添了若干个接口，增添的接口是在本人修改 sky2 网卡驱动的过程中所需要但是 nta.h 并未提供（作为接口文件，驱动中必然需要增加头文件，#include “nta.h”）。开发人员若在使用 ntzc 的过程中发现缺少更改驱动的必要接口，可按如下方案更改：
   a) 找到对应版本的内核源码中此函数的定义，推荐一个网址：http://lxr.oss.org.cn/ident，网络上很多与此类似的提供交叉引用，标识符查找的网站.
   b) 从中找到源定义，将 sk_buff 的全部操作替换为 ntzc 的 m_buf 操作,将此函数添加到 nta.h 中
   c) 将驱动中的 sk_buff 的操作替换为自定义的 m_buf 的操作。

3. 添加了 3 个用户接口，位置为 control.h control.c 中，声明如下：
   - zc_t* zc_open_live(const char*dev, int snaplen, char* errbuf);
   - void zc_destroy(zc_t* zc_ctl);
   - int zc_loop(zc_t *zc_ctl, int cnt, void (*zc_handler)(unsigned char* user, const struct pcap_pkthdr *h, const unsigned char *bytes), unsigned char* user);

   这三个函数是仿照 libpcap 的函数原型编制而成，用户可参考 pcap_open_live，zc_destroy, zc_loop 的用法使用此三个接口。
4. 将这三个接口加入 snort 中，位置为 snort.h/snort.c/parser.c

2.2 详解

2.2.1 更改网卡驱动

查找自己的网卡型号及对应的网卡驱动，将 sk_buff 的全部操作替换为 m_buf 的操作。
以 Marvell 的网卡驱动 sky2 为例：
步骤：

a) 增加 #include “nta.h”

b) struct sk_buff 全部替换为 struct m_buf

c) netdev_alloc_skb(PKT_BUF_SKB) 替换为nta_alloc_mbuf(NULL, PKT_BUF_SKB,

GFP_ATOMIC)

d) skb_reserve替换为mbuf_reserve:

e) dev_kfree_skb替换为 nta_kfree_mbuf:

f) dev_kfree_skb_any替换为nta_kfree_mbuf:

g) skb_put替换为mbuf_put:

h) skb_frag_t 替换为 mbuf_frag_t

i) skb_shinfo替换为mbuf_shinfo

j) skb_copy_to_linear_data替换为mbuf_copy_to_linear_data

k) skb_copy_from_linear_data替换为mbuf_copy_from_linear_data

l) ip_hdr换为ip_header, tcp_hdr换为tcp_header

m) ip_hdrlen换为ip_headerlen

n) skb_fill_page_desc 换为 mbuf_fill_page_desc

o) sky2_xmit_frame 拆解为两个函数 sky2_xmit_frame 和 sky2_xmit_frame_fake，并且.ndo_start_xmit = sky2_xmit_frame_fake

p) 添加 nta_register_zc(dev, pcnet32_start_xmit);

基本修改方案即如上所述，再次基础上编译，若出现错误，只要按顺序解决即可，不会存在不可解决的障碍，此时，网卡驱动已经准备好了。

2.2.2 调整 ntzc

在修改网卡驱动过程中，有可能添加了一些网卡驱动缺少的接口函数，此时，需要重新编 ntzc。

2.2.3 修改 snort

由于 snort 多处用到了 libpcap 的函数，故全局替换 libpcap 的函数势必引入新的错误，

而且由于 ntzc 还未开发出一整套类似 libpcap 的接口，故采取 libpcap 与零拷贝共存的方案。Snort 的修改方法如下：

1) 将 zc_comm.h 与 control.h/control.c 添加进入 snort 源码中
2) Snort.h
   #define ZeroCopy 1　//此宏即是零拷贝的开关，定义为 1 时使用零拷贝，反之不使用
3) Snort.c:
   在 pcap_open_live()处同时使 zc_open_live()
   将 pcap_loop()替换为 zc_loop()
   在正确的位置调用 zc_destroy();
4) Parser.c：
   在正确的位置调用 zc_destroy();

2.2.4 安装及使用

1. 进入 ntzc/zc 文件夹，make
2. sudo insmod ntzc.ko　　//加载 ntzc
   sudo insmod sky2.ko　　//加载驱动
3. 进入修改过的 snort 目录，./configure, make, make install

Snort 的零拷贝版本就可以正常运行了！

3 存在的问题

1. Ntzc 不稳定，insmod 过程中经常出现 kernel panic，原因是内存溢出，因为 ntzc 需要从内核中分配出大量的内存，易出现内存不足的问题。
   这个问题尚无好的办法解决，只能等待下一代版本修正这个 bug，若出现这个问题，目前的解决办法是重启电脑，重新加载。并且在 insmod 的时候最好在终端模式下运行，不要在 X Window 模式下运行，成功率会高很多。
2. 修改过的网卡驱动无法上网，B/S 架构受影响。
   解决办法是：使用双网卡，目前本人机器已经实现有线网卡 eth0 抓包，无线网卡同时上网。
3. Eth0 不上网无法嗅探局域网的包，这个貌似是因为交换机的过滤功能使得只有发到本机的包会被抓到，与程序无关
   解决方法是：可以使用 ARP 欺骗或其他硬件方面的手段，强制使用交换机将所有包输出到本机。Hub 下将无此类问题。
4. 零拷贝无法上网，本机无 IP，测试困难。
   这个问题才是最大的问题，本来已经下载了一个 snort 的测试工具—stick，能够读取 snort 规则并生成会导致警报的包，但由于协议栈无法使用，snort 无法探测数据包，无法测试零拷贝性能，尚未解决！