目录

[1 零拷贝的实现方案 2](#_Toc263682907)

[1.1 ntzc背景 2](#_Toc263682908)

[1.2 ntzc概述 2](#_Toc263682909)

[1.2.1 ntzc原始代码目录层次 2](#_Toc263682910)

[1.2.2 ntzc目录层次简化 4](#_Toc263682911)

[1.2.3 ntzc的模块结构 5](#_Toc263682912)

[1.2.4 ntzc的整体实现思路 6](#_Toc263682913)

[1.2.5 内存管理模块实现 7](#_Toc263682914)

[2 零拷贝与snort的融合 8](#_Toc263682915)

[2.1 概述 8](#_Toc263682916)

[2.2 详解 8](#_Toc263682917)

[2.2.1 更改网卡驱动 8](#_Toc263682918)

[2.2.2 调整ntzc 9](#_Toc263682919)

[2.2.3 修改snort 9](#_Toc263682920)

[2.2.4 安装及使用 10](#_Toc263682921)

[3 存在的问题 10](#_Toc263682922)

零拷贝的实现方案

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](http://linux.chinaunix.net/bbs/viewthread.php?tid=1161364ntzc)

[ntzc](http://linux.chinaunix.net/bbs/viewthread.php?tid=1161364ntzc)的源码参照svn上的项目<http://code.google.com/p/ntzc/>

ntzc概述

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**

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。

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 |

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。

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

# 零拷贝与snort的融合

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：
3. 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.
4. 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
5. 将驱动中的sk\_buff的操作替换为自定义的m\_buf的操作。
6. 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。

1. 将这三个接口加入snort中，位置为snort.h/snort.c/parser.c

详解

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：

1. 增加 #include “nta.h”
2. struct sk\_buff 全部替换为 struct m\_buf
3. netdev\_alloc\_skb(PKT\_BUF\_SKB) 替换为nta\_alloc\_mbuf(NULL, PKT\_BUF\_SKB, GFP\_ATOMIC)
4. skb\_reserve替换为mbuf\_reserve:
5. dev\_kfree\_skb替换为 nta\_kfree\_mbuf:
6. dev\_kfree\_skb\_any替换为nta\_kfree\_mbuf:
7. skb\_put替换为mbuf\_put:
8. skb\_frag\_t 替换为 mbuf\_frag\_t
9. skb\_shinfo替换为mbuf\_shinfo
10. skb\_copy\_to\_linear\_data替换为mbuf\_copy\_to\_linear\_data
11. skb\_copy\_from\_linear\_data替换为mbuf\_copy\_from\_linear\_data
12. ip\_hdr换为ip\_header, tcp\_hdr换为tcp\_header
13. ip\_hdrlen换为ip\_headerlen
14. skb\_fill\_page\_desc换为mbuf\_fill\_page\_desc
15. 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
16. 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。

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。

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

1. 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();

1. Parser.c：

在正确的位置调用zc\_destroy();

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！

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。

1. 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。

1. 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。

1. 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！