第32卷 第21期 计 算 机 工 程 2006年11月
Vol.32 No.21 Computer Engineering November 2006
·安全技术· 文章编号：1000—3428(2006)21—0136—03 文献标识码：A 中图分类号：TP393.08

NP防火墙协议栈驱动模块的设计与实现

韩志耕，罗军舟
（东南大学计算机科学与工程系，南京 210096）

摘 要：彻底打通网络处理器光口到本地协议栈间通路需要协议栈驱动提供支持。针对协议栈驱动基本组成和内在驱动机制，同时确保遵循Intel IXA软件架构分层设计原则，该文提出了在Linux平台上的实现方案并进行了分析，指出了实现过程中牵涉的关键技术。Enp2611评估板上硬件光口打通测试表明设计达到了预先要求。

关键词：协议栈驱动；防火墙；网络处理器；包分类；主动式安全防范系统

Design and Implementation of Protocol Stack Driver Module in NP-based Firewall System

HAN Zhigeng, LUO Junzhou
(Department of Computer Science and Engineering, Southeast University, Nanjing 210096)

【Abstract】 Stack driver can lead to get through downright the path between the hardware interfaces and TCP/IP stack. The paper brings forward a scheme of implementing stack driver on Linux platform after having discerned its basic components and mechanism inside of the stack driver and following the principle of the hierarchy design of the Intel IXA software. It gives analysis at large and points out key technology in the course of the implementation. And the test of getting through hardware interface of light on Enp2611 evaluation board indicates that the implementation satisfies the demand in advance.

【Key words】 Stack driver; Firewall; Network processor; Packet classification; Active secure defence system

主动式安全防范系统旨在克服传统防火墙功能单一且依赖于“内部网安全”的假设，可有效防御“数据驱动”攻击方式，技术上采用可有效避免传统防火墙架构性能障碍和功能与协议快速支持障碍的网络处理器平台。但鉴于其采用多处理器硬件架构和分层体系结构，迫切要求充分发挥结构优势使核上TCP/IP协议栈能处理本地目的地报文，为彻底解决该问题，需在TCP/IP协议栈与硬件光口驱动间设计协议栈驱动，彻底打通网络处理器光口到本地协议栈间通路，也为操作系统用户态下配置光口提供方便。

1 Intel IXA架构协议栈驱动体系结构

1.1 协议栈驱动主要功能

IXA架构[1]中协议栈驱动体系结构包含一核心组件，目前仅支持Ipv4管道，不支持Ipv6管道。针对Ipv4管道，协议栈驱动核心组件兼容操作系统网络协议栈抽象。作为特殊类型组件，它支持来自/去往任何其它网络协议栈或核心组件的输入/出。从网络协议栈角度看它是底层媒体驱动，而对于IXA SDK来讲则为核心组件。协议栈驱动包含两部分：一个核心组件模块以及多个通信处理程序（本地虚拟接口设备驱动LVIDD和一个传输模块TM）。协议栈驱动[2]主要功能为：(1)发送接收自上行流核心组件提供的数据到本地网络协议栈或远程协议栈。(2)发送接收自本地网络协议栈或远程协议栈的数据到邻接下行流核心组件。需注意与协议栈驱动交互的网络协议栈可以是本地协议栈也可是远程协议栈。

1.2 内在支持和外在依赖

(1)内在支持

IXA架构核心组件基础设施允许用户为单独核心组件定义多输出。目前只支持单输出，且协议栈驱动暂只能处理IP数据包。

(2)外在依赖

协议栈驱动依赖与Intel网际交换体系结构可携带性框架可用性。其设计及实现依赖于核心组件基础设施可用性。分层结构设计可保证当基础设施不能使用或被用户自定义层所取代时大多数的代码能够被重用。

2 协议栈驱动设计

设计需平衡考虑3方面因素：提供上层应用尽量多服务，驱动运行占较少时间且尽量保持简单而避免错误丛生。且不带错误的协议栈驱动必须含下列特征：支持同步/异步操作、驱动可多次打开、能充分利用核心组件特性以及不具备“简化任务”功能或提供与策略相关操作软件层等。据协议栈驱动需对外提供的基本功能，考虑将其划分为3大功能模块，分别为协议栈驱动内部CC、LVIDD及TM，整体逻辑设计如图1所示。

内部逻辑牵涉到的逻辑功能划分兼顾Intel IXA架构中协议栈驱动体系结构分层设计思想。协议栈驱动核心组件囊括所有相对不变且与平台无关的功能，位于整个协议栈驱动层面底层，也是整个驱动模块数据交换中心，且最重要的是其内嵌实现简单包分类的功能模块。LVIDD中牵涉到些需特定内核(如Linux)参与交互的相关功能，是协议栈驱动与本地

基金项目：江苏省“网络与信息安全”重点实验室基金资助项目(BM2003201)；江苏省高技术研究基金资助项目“主动式安全防范系统的研究”(BG2004036)

作者简介：韩志耕(1976－)，男，博士生，主研方向：网络安全；罗军舟，博士、教授、博导

收稿日期：2006-02-08 **E-mail：**hanzgnit@163.com

现有协议栈交互的枢纽，但其受核心组件控制。TM 负责与远程 TCP/IP 协议栈交互，和 LVIDD 一样导出同样或相似函数集，也受核心组件控制。需注意务必采用分层结构设计，这样可以保证当核心组件基础设施不被使用或被用户自定义层所取代时多数代码可重用。

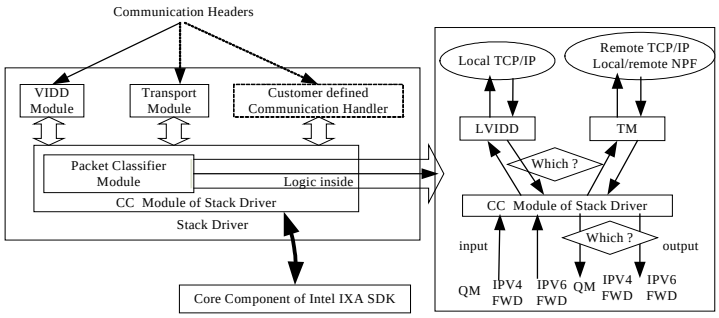


图 1 协议栈驱动内部逻辑图

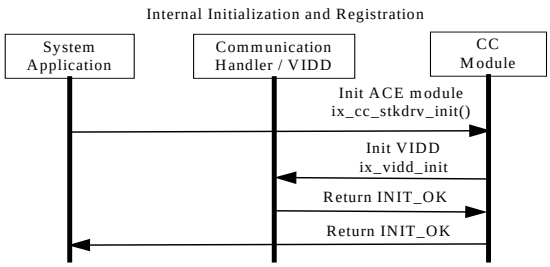


图 2 内部初始化和注册

各模块介绍如下：

(1)内部核心组件模块(CC Module)

核心组件模块属于常规核心组件（可运行在通用目标处理器上），实际上常和图 1 所示的通信处理模块(LVIDD、TM 以及用户定制通信模块)运行在相同地址空间(Xscale 层面内核空间)。依赖于 Linux 操作系统内核和 Intel 网际交换体系结构可携带框架提供的能力，核心组件模块也可与通信处理模块运行在各自不同地址空间。

(2)本地虚拟接口设备驱动(LVIDD)

LVIDD 部分遵循 Linux 操作系统驱动注册和相应应用编程接口，负责传递数据包到本地现有协议栈，需注册到核心组件模块。

(3)传输模块(Transport Module)

TM 部分负责传递数据包到远程现有协议栈或到本地/远程 NPF 协议栈，同样需注册到核心组件模块。

(4)包分类逻辑模块(Packet Classifier)

核心组件负责接收输入包并将它转交给LVIDD或TM，再由它们将该包传递给各自TCP/IP协议栈。来自TCP/IP协议栈的输出包仍由两模块接收并转交给核心组件，由核心组件模块将该包递交给IPV4 转发模块或队列管理模块。常用本地或远程协议栈，缺省使能本地协议栈。核心组件模块中包分类逻辑模块对输入包进行分类，可决定将包递交给本地VIDD或传输模块；它也可进行外出包分类，来确定将包递交给IPV4 核心组件模块或队列管理模块。当需协议栈驱动模块转发所有包到本地协议栈或远程协议栈时，包分类[3]逻辑模块尤其重要。

3 协议栈驱动实现

3.1 初始化

当系统启动时，系统应用按下列步骤初始化协议栈驱动：

(1)调用协议栈驱动初始化例程初始化核心组件，包括分配核心组件控制结构，建立转发平台链表和转发平台中物理接口链表以及相应物理接口中虚拟接口链表等。

(2)协议栈驱动初始化例程调用针对通信处理函数的初始化函数（包括 LVIDD 初始化函数；若支持远程协议栈，则还有传输模块初始化函数，它们可进行特定初始化）。接下来还需为处理数据包和消息注册回调函数，需注意注册可采用 RM 通信和 CCI 两种方法，且消息优先级高于包优先级。

(3)所有处理函数在核心组件模块中注册后，如果使能包分类，还需初始化过滤规则控制结构。但为成功向上发送数据包到本地堆栈，本地接口必须要通过外部的代码进行加载。在这里外部代码通过调用属性 API 函数为本地 VIDD 加载本地接口来添加接口。

图 2 描绘系统应用，通信处理函数（本地 VIDD）和核心组件模块之间初始化顺序。

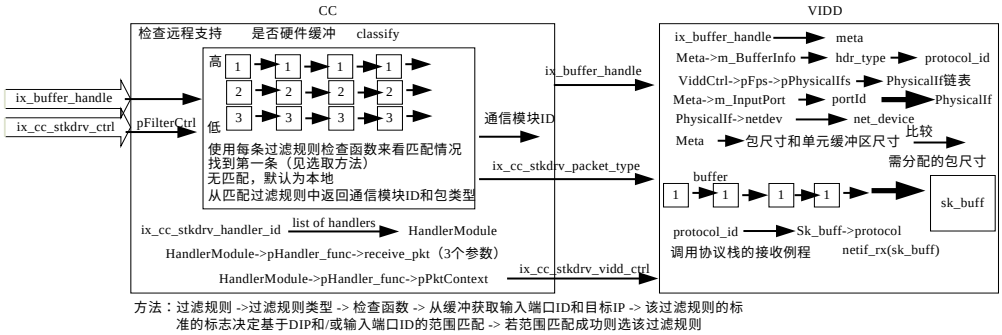
3.2 接收数据包流程

(1)核心组件边

将下层转发模块传递来的包 (buffer 结构)分类处理，解析其将去的目标协议栈，以通知相应通信处理模块来处理该包。因过滤规则相对简单且条数少，故采用 Linear Search 算法，其数据结构简单，过滤规则按优先级高低链式降序存储，形成多条链表。分类时包从最高优先级链表头开始依次和链表中各规则依次比较，直到找到一条匹配规则或到达最低优先级链表尾。需注意：包分类器绝不意味具有最大包分类能力；它唯一目的是决定将包发送到 LVIDD 或 TM。

(2)VIDD 边

从核心组件接收包并将它传递到本地 TCP/IP 协议栈。该操作需协议栈驱动核心组件模块中包处理部分驱动。包在移交给相应 TCP/IP 协议栈之前需要从众多 ix_buffer 结构缓冲区封装进一个 sk_buff 结构缓冲区中（涉及一个拷贝），最后需要驱动相关网络协议栈的接收例程(如 netif_rx)来接收该包。图 3 具体再现了协议栈驱动包接收流程。



方法：过滤规则 ->过滤规则类型 -> 检查函数 -> 从缓冲获取输入端口ID和目标IP -> 该过滤规则的标准的标志决定基于DIP和/或输入端口ID的范围匹配 -> 若范围匹配成功则选该过滤规则

图 3 协议栈驱动接收包流程

3.3 外出数据包流程

(1)LVIDD 边

内核使用网络设备驱动将包发送到相应转发组件(IPv4 转发核心组件)。调用协议栈驱动核心组件发送包例程之前需转储sk_buffer包缓冲到众多ix_buffer缓冲。注意sendto类socket函数将转化为此操作。若传输数据小到可装入单个包中，内核完全可在用户进程上下文中执行该操作。但若包很大，内核会以队列方式管理，则需多次调用ifd_tx且只能发生在后半部[4]。当该操作中断时，上下文可任意但有局限性(如

操作中禁止睡眠/等待)。

(2)核心组件边

通信处理模块调用协议栈驱动核心组件模块中发送包例程为传输入队数据包。目前在 Linux 平台上的实现也可支持下行包分类，这意味着 IPv4 或 IPv6 包能被适当路由到各自的转发模块核心组件中。可从组播包中分离出所有去往 IPv4 转发核心组件模块的 IPv4 包。针对这些组播包，会构造一个组播 L2 头并将这些包发送到队列管理器。所有 IPv6 包去往 IPv6 转发核心组件。图 4 为协议栈驱动外出包操作细节。

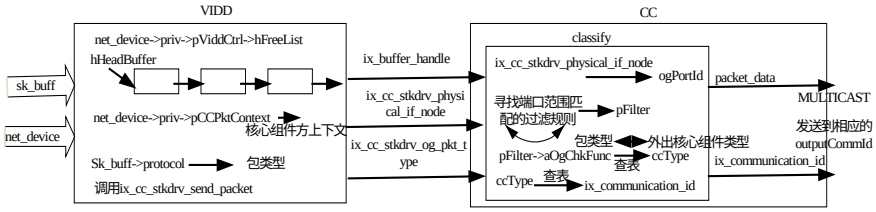


图 4　协议栈驱动外出包流程

4 协议栈驱动功能正确性测试

测试环境如图 5。ENP-2611 开发板两光纤接口(Port1 和 Port0)分别接到配置光纤接口网卡的 Win2k 和 WinXP 机器。启动需 3 脚本：make_linux_kernel_devices.sh(创建/dev/L2Config 和/dev/Rconfig 字符设备)、ipv4_enp2611_run.sh (加载 xScale 层面内核态核心组件到 linux 内核)、ethernet_rtm_config_linux.sh(配置测试路由)，路由表配置如图 6 所示，其中采用 addV4L3Info 配置 L2 表，这样可通过 ARP 协议学习端口 MAC 地址。

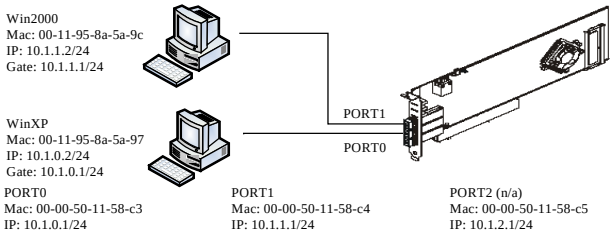


图 5 测试环境

```
# Use either addV4L3Info or addV4EthEntry. The former adds only IP address and
# learns MAC address from ARP.
./l2config addV4L3Info "1 10.1.0.2 DEFAULT"
#./l2config addV4EthEntry "1 10.1.0.2 00:11:95:8A:5A:97 00:00:50:11:58:C3 DEFAULT"
./rconfig addNextHop "1 63 1 0 1500 0 10.1.0.2 0"
./rconfig addRoute "10.1.0.2 255.255.255.255 1"

./l2config addV4L3Info "2 10.1.1.2 DEFAULT"
#./l2config addV4EthEntry "2 10.1.1.2 00:11:95:8A:5A:9C 00:00:50:11:58:C4 DEFAULT"
./rconfig addNextHop "2 63 2 1 1500 0 10.1.1.2 0"
./rconfig addRoute "10.1.1.2 255.255.255.255 2"
```

图 6 路由设置

当板载内核中无协议栈驱动时，运行路由配置脚本出现段错误，通过ifconfig命令发现不了任何光口；分析原因是该脚本中rconfig（配置路由表）以及l2config（配置L2 表[5]）需协议栈驱动输出信息解析ARP协议[6]，而此时所需协议栈驱动根本没参与工作。现在内核态核心组件加载脚本文件中写入加载协议栈驱动命令，再运行脚本，光口网络连接可正常启用，端口配置命令发现所有光口，且可为任一光口MAC地址重新绑定IP地址。现测试协议栈驱动整体（ISO：7 层；TCP/IP：5 层）功能正常性。首先NP防火墙xScale层面用户态上运行服务程序来侦听开发板（测试光口port1，IP: 10.1.0.1）端口 3490，并接收客户端（来自WinXP机器 10.1.0.2 的telnet程序）发送的字符，再将其反馈给客户端。从测试结果看，协议栈驱动从物理层到应用层工作正常。

测试过程：10.1.0.1 上启动 server，监听 3490 端口（图 7）→ 10.1.0.2 上运行 telnet 客户端程序连接 10.1.0.1：3490 → 连接成功 → 客户端发送字符 x → 服务器接收并返回同一字符 x？(图 8)。嗅包结果见图 9。

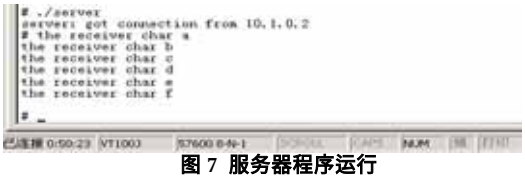


图 7 服务器程序运行

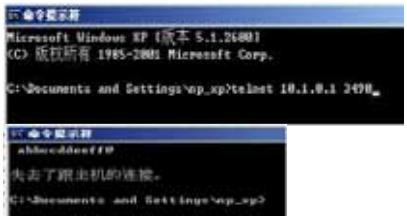


图 8　客户端连接服务器 3490 端口及服务器向客户端反馈字符

图 9 嗅包通信过程

5 结束语

网络处理器为防火墙系统设计开发提供了崭新平台，协议栈驱动为Xscale层面用户态系统应用与内核态核心组件甚至微引擎层面微块之间的交互提供了便利，同时采用层次型、模块化软件框架，兼顾了性能和功能上的需求。Enp2611 评估板上测试表明驱动运行可靠且功能正常。针对协议栈驱动的进一步研究包括丰富对远程现有协议栈以及本地/远程NPF协议栈的支持以及针对网络处理器特性的包分类算法[7]优化。

参考文献

1 IXA Portability Framework Overview. Intel® Internet Exchange Architecture Portability Framework Developer's Manual[M/CD]. Intel Inc., 2003-11: 16-21.

2 StkDrv Component. Intel® Internet Exchange Architecture Software Building Blocks Developer's Manual[M/CD]. Intel Inc., 2004-03.

3 Rashti M J, Rabiee H R, Foroutan A. A Multi-dimensional Packet Classifier for NP-based Firewalls[C]. Proceedings of the 2004 International Symposium on Applications and the Internet, 2004.

4 Rubini A, Corbet J. Linux Device Drivers (2nd Edition)[M]. O'Reilly & Associates, Inc., 2001-06.

5 Routing Table and L2 Table. Intel Internet Exchange Architecture (IXA) SDK 3.1 Software Framework, Getting Started Guide[M/CD]. Intel Inc., 2003-07: 53-58.

6 Comer D E, Stevens D L. Internet Working with TCP/IP Vol Ⅱ: Design, Implementation, and Internals[M]. Prentice-Hall, Inc., 1999.

7 Gupta P, McKeown N. Packet Classification on Multiple Fields[C]. Proceedings of ACM Sigcomm, 1999-09: 146-160.