第36卷 第10期 计 算 机 工 程 2010年5月
Vol.36 No.10 Computer Engineering May 2010
·开发研究与设计技术· 文章编号：1000—3428(2010)10—0267—03 文献标识码：A 中图分类号：TP393

基于PPC的网络行为识别系统性能优化

卢旌平，宋英雄，顾 鹏
(上海大学特种光纤与光接入网重点实验室，上海 200072)

摘 要：给出一种基于PowerPC处理器和嵌入式Linux系统的网络行为识别系统设计方案，为满足高速网络环境下的工作需求，在Linux网络子系统的基础上，对数据平面进行优化，实现基于NAPI的网络设备驱动程序，设计高效率的缓冲存储结构，缩短网络协议栈的数据处理流程。测试结果表明，优化后的系统与原系统相比，系统吞吐量得到提高。

关键词：网络行为识别；网络设备驱动程序；PowerPC处理器

Performance Optimization of Network Behavior Identification System Based on PPC

LU Jing-ping, SONG Ying-xiong, GU Peng
(Key Laboratory of Specialty Fiber Optics and Optical Access Networks, Shanghai University, Shanghai 200072)

【Abstract】This paper presents a design project of network behavior identification system based on PowerPC processor and embedded Linux. Data plane of the system is optimized to meet the needs of working under the high-speed network environment. NAPI network device driver and efficient buffer storage structure are designed. Data processing flow of Linux network protocol stack is shortened to promote the performance of Linux. Experimental result shows the throughput of the optimized system is improved compared with the original system.

【Key words】network behavior identification; network device driver; PowerPC(PPC) processor

1 概述

随着网络规模的扩大，新的网络业务不断出现，满足了人们快速、灵活获取信息的需求，但各种网络问题也随之而来。互联网络的“内容安全”问题越来越受到人们的重视。一些不合理的网络应用恶化了正常的网络服务质量，对网络运营商的当前业务造成了影响。网络行为识别系统是一种能够对网络内容进行识别和管理的系统，它满足了政府机关和网络运营商的需求。本文提出一种基于PowerPC(PPC)系列MPC8572网络处理器和Linux操作系统的网络行为识别系统设计方案，针对其数据平面进行优化，使其性能得到较大提高。

2 网络行为识别系统整体框架

传统的互联网业务识别技术主要是通过IP地址、端口和协议号来识别的。在早期的各种协议规范中，对于网络层、传输层和应用层上不同的协议由固定的协议号或端口来区分，在数据包分类时通过识别协议号或端口来识别该数据包的类型。但随着互联网上业务类型的不断丰富，基于开放端口、随机端口甚至采用加密方式进行传输的应用业务不断增多。在这种情况下，仅通过端口信息已经不能真正判断流量中的业务类型[1]。

MPC8572是一款高端的PowerPC系列网络处理器，在其内部集成了硬件查找表(TLU)、模式匹配引擎(PME)、状态规则引擎(SRE)，可分别用于数据包的包头检测、深度包检测(DPI)和状态检测[2]。将MPC8572用于网络行为识别系统充分发挥了网络处理器的硬件特性。网络行为识别系统整体框架如图1所示。在功能方面，整个系统可以分为数据平面、识别平面和管理平面。

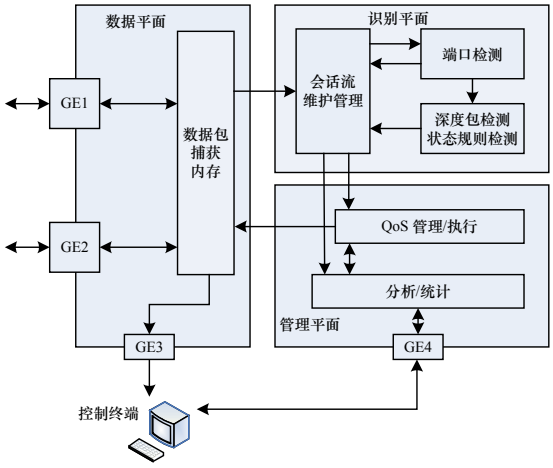

图1 网络行为识别系统整体框架

数据平面主要完成以下功能：(1)以太数据包的捕获和发送，即实现数据包在GE1与GE2之间的转发。(2)IP分片的重组。(3)根据管理平面的配置对关心的数据包通过GE3镜像输出给控制终端。(4)通知识别平面对数据包进行后处理。

识别平面主要完成以下功能：(1)会话流的跟踪、维护和管理，根据网络情况，实现会话流的添加、删除并记录每条

基金项目：上海市科委重点攻关基金资助项目(075115004)；上海市重点学科建设基金资助项目(S30108)；上海市科委重点实验室基金资助项目(08DZ2231100)

作者简介：卢旌平(1984－)，男，硕士研究生，主研方向：计算机网络，嵌入式软件设计；宋英雄，副研究员；顾 鹏，硕士研究生

收稿日期：2009-11-18 **E-mail：**lujingping@126.com

会话流的流量和持续时间。(2)数据包的端口检测，完成具有固定端口的传统业务的识别，如 http, ftp。(3)数据包的深度包检测，利用处理器内部的模式匹配引擎对数据应用层内容进行模式匹配，完成具有明确特征码业务的识别，如 BT, MSN。(4)状态检测，某些网络业务会在不同的阶段呈现出不同的特征，状态检测跟踪会话流状态的变化，以达到识别的目的，如 SIP。(5)将业务的识别信息告知分析统计模块。

管理平面实现的主要功能有：(1)将识别模块提交的业务识别信息生成报表发送至控制终端。(2)提供给用户当前网络的流量信息。(3)提供用户设置某业务或会话流 Qos 等级的接口，并执行该规则。在网络拥塞时丢弃低优先级的数据包。

数据包的捕获和转发是整个网络行为识别系统工作的基础，数据平面的性能对系统的整体性能起到了决定性的作用，如何提高数据平面的效率是系统设计中的关键一环。

3 系统数据平面优化

网络行为识别系统的数据平面是在 Linux 网络子系统基础上改进而实现的。

3.1 网络设备驱动程序优化

在传统的 Linux 网络设备驱动中，驱动程序每收到一个数据包就会产生一个硬件中断，通知中断处理程序接收和处理数据包，在低速网络环境下，这是一种有效的数据接收方式。但在千兆高速网络环境中，由下式计算：

$$pps = \frac{1000}{Packetsize \times 8 + Preamble + InterframeGap}$$

以 *Preamble*=64 bit, *InterframeGap*=96 bit, *Packetsize*=64 Byte 为例，每秒系统将会接收 148 万个数据包。对于网络行为识别系统，由于要处理双向 1 GB 的数据流量，极端情况下，每秒接收的数据包将高达 296 万个。系统无法适应如此高的中断频率，将会出现中断活锁的情况：即所有 CPU 时间全部用于处理网卡中断程序，CPU 无法响应其他用户进程。

为了适应高速网络环境，系统实现了基于 NAPI 的网络设备驱动程序，其流程如图 2 所示。

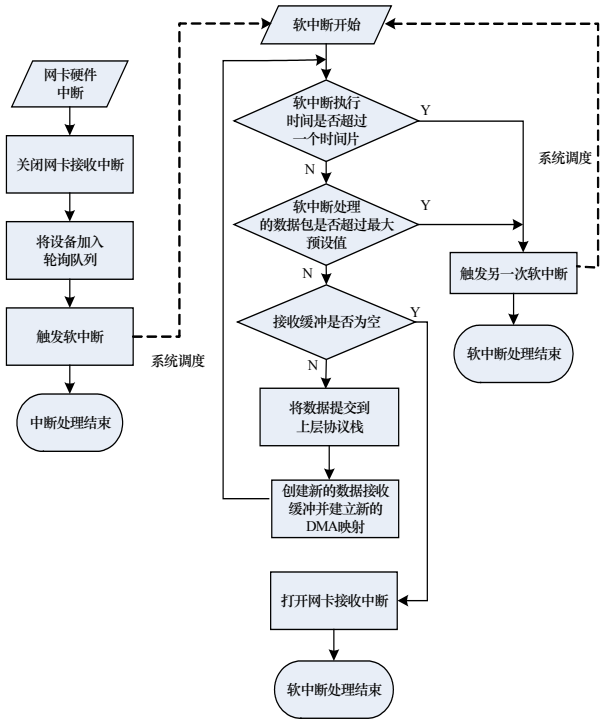

图 2 基于 NAPI 的网络设备驱动程序流程

可以看出，与传统的设备驱动程序不同，当设备接收到数据时，首先关闭硬件中断，同时发出一个软中断信号，当 CPU 处于空闲时，会运行软中断处理程序。在软中断处理程序中，采用轮询的方式，一次将多个数据包传递到上层协议栈，直到接收缓冲区中的数据全部被读出，此时会再次打开硬件接收中断。与传统的驱动程序相比，基于 NAPI 的驱动程序主要有以下的优点：

(1)接收到数据包后首先关闭中断，将耗时的数据包处理工作放在软中断中执行，避免出现中断活锁。

(2)当软中断处理程序的执行时间超过一个时间片，但缓冲区中仍有数据未处理时，软中断处理程序会将自身挂起等待下一次调度，避免长期占用处理器资源。

(3)关闭硬件中断期间，到达的数据包通过 DMA 直接放入接收缓冲，不需要 CPU 的参与。

3.2 缓冲区存储结构优化

网络设备驱动在初始化时，会建立一个缓冲区来接收数据包，当驱动将数据包传递给上层协议栈后，需要重新申请新的缓冲区。另一方面，设备将数据包发送后，会将缓冲区释放给系统。这样造成了驱动程序频繁向系统申请和释放内存的情况，降低了程序运行的效率并且容易产生内存碎片。

为了优化缓冲区的申请和释放环节，本文设计了 recycle_queue 队列，使分配好的缓冲区能得到循环使用。当数据包发送后，调用 *free_buff* 函数，将缓冲区的指针放入该队列，而不释放给系统。接收程序需要申请新的缓冲时，调用 *alloc_buff* 先从 recycle_queue 队列中获取，如果 recycle_queue 队列中没有可以获取的缓冲，再向系统申请。

3.3 Linux 内核协议栈的改进

在 Linux 系统中，每一个需要转发的数据包都要经过协议栈的桥模块或路由模块来选择数据包的流向，同时系统要耗费大量的资源更新网桥生成树或路由表[3]。对于网络行为识别系统并不负担路由的功能，数据包的入口和出口也是确定的，因此，有必要对原 Linux 协议栈进行裁剪[4]。改进前后的数据流程对比如图 3 所示，其中，实线为原数据流程；虚线为改进后的数据流程。

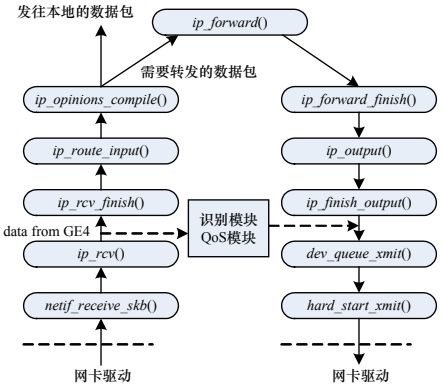

图 3 改进前后的数据流程对比

当数据包进入 ip 层后，首先判断该数据包是否从 GE4 进入，由于 GE4 是网管口，对于从网管口进入的数据按正常的数据流程提交到上层协议栈。对于从 GE1 或 GE2 进入的数据包，由钩子函数直接将数据截获并送入识别和 QoS 模块，经过处理后的数据包调用 *dev_queue_xmit* 将数据包发送至数据链路层，最后用 *hard_start_xmit* 函数将数据通过驱动程序发送出网口。经过修改后，数据处理跳过了原协议栈路由模

块，缩短了处理流程，提高了系统的转发性能。

4 系统性能比较

为了比较原 Linux 网络子系统和经过优化后网络系统的性能，本文实测了两者的吞吐量。测试环境如下：基于 MPC 8572 的网络行为识别设备 1 台，并移植有 Linux2.6.23 操作系统，Smartbit6000 网络测试仪 1 台。因为是针对数据平面的性能测试，所以在测试中没有启用识别模块，并配置所有数据包优先级相同。在测试中，由 SmartApplication 软件控制 Smartbit6000 产生双向 1 GB 的测试数据流对系统进行吞吐量的测试，包大小为 64 Byte~1 518 Byte，经过换算每秒产生的数据包数目为 1.62×10^5~2.96×10^6。性能测试拓扑结构如图 4 所示。

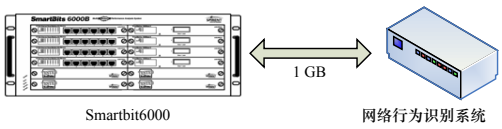

图 4 性能测试拓扑结构

先测试的是未经过优化的 Linux 系统，得到测试结果 normal，然后测试使用 NAPI 驱动时的 Linux 系统，得到测试结果 NAPI，最后测试使用 NAPI 且缓冲区、协议栈都经过优化的系统，得到测试结果 enhance，测试结果如图 5 所示。

由测试结果可知，在大数据包时，系统需要转发的数据包数目较少，由测试结果可知，在大数据包时，原系统和经过优化后的系统都能做到线速处理。在包大小为 1 024 Byte 时，转发率为 239 000 packet/s；在包大小为 1 280 Byte 时，转发率为 192 000 packet/s；在包大小为 1 518 Byte 时，转发率为 162 000 packet/s。而在小数据包时，未经过优化的系统转发性能只有 250 000 packet/s，经过优化的系统数据包转发性能大约在 500 000 packet/s，系统吞吐量得到明显提高。

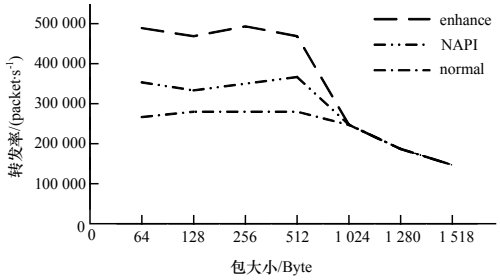

图 5 系统性能比较

5 结束语

本文提出一种基于 PowerPC 网络处理器和 Linux 操作系统的网络行为识别系统设计方案，从网络设备驱动、收发缓冲区和内核协议栈多个层面对 Linux 网络子系统进行了优化。测试结果表明，经过优化后的系统与原系统相比，系统吞吐量明显提高，可达 500 000 packet/s。但在小数据包时，系统仍未能实现线速处理。为了进一步提升系统性能，下一步将在操作系统层实现基于混合多处理的软件架构。

参考文献

[1] 王 超, 赵文杰. IP 网络带宽管理技术及应用分析[J]. 电信技术, 2007, (5): 101-103.

[2] Freescale Semiconductor Inc.. MPC8572E PowerQUICC: Integrated Host Processor Family Reference Manual[Z]. 2008.

[3] Benvenuti C. Understanding Linux Network Internals[M]. Sebastopol, CA, USA: O'Reilly Media, 2006.

[4] Accardi K, Bock T, Hady F. Network Processor Acceleration for a Linux Netfilter Firewall[C]//Proceedings of ANCS'05. Princeton, New Jersey, USA: [s. n.], 2005.

编辑 顾姣健

(上接第 266 页)

图 4(a)是系统报表模型定制界面，其中左侧的主栏/宾栏单元库是已经完成语义定制的语义元组，可以通过拖拽到右边编辑区的主栏或宾栏动态设计报表模板。基于语义的灵活性可以实现多层主栏或多层宾栏，且可以通过拖拽快速便捷地实现报表结构修改。

7 结束语

本文系统目前已成功应用于某油田公司，一年多的运行结果表明，该系统比传统报表系统具有更高的灵活性和易用性。

参考文献

[1] Ronen B, Palley M A, Lucas H C J R. Spreadsheet Analysis and Design[J]. Communications of the ACM, 1989, 32(1): 84-92.

[2] Brown P S, Gould J D. An Experimental Study of People Creating Spreadsheets[J]. ACM Transactions on Office Information Systems, 1987, 5(3): 258-272.

[3] Cragg P B, King M. Spreadsheet Modeling Abuse: An Opportunity for OR?[J]. Journal of Operational Research Society, 1993, 44(8): 743-752.

[4] Campbell-Kelly M. Number Crunching Without Programming: The Evolution of Spreadsheet Usability[J]. IEEE Annals of the History of Computing, 2007, 29(3): 6-19.

[5] Raymond D R P. Spreadsheet Errors: What We Know. What We Think We Can Do[C]//Proceedings of the Spreadsheet Risk Symposium. Greenwich, London, UK: [s. n.], 2000: 7-18.

[6] Panko R R. What We Know About Spreadsheet Errors[J]. Journal of End User Computing, 1998, 10(2): 15-21.

[7] Rajalingham K, Chadwick D, Knight B. Classification of Spreadsheet Errors[C]//Proceedings of European Spreadsheet Risks Interest Group Annual Conferenc. Greenwich, London, UK: [s. n.], 2000: 23-34.

[8] Hassinen K, Sajaniemi J, Vaisanan J. Structured Spreadsheet Calculation[C]//Proceedings of IEEE Workshop on Language for Automation Symbiotic and Intelligent Robots. Maryland, USA: [s. n.], 1988: 129-133.

编辑 陈 晖