文章编号:1001 - 9081(2005)11 - 2568 - 03

基于 IXP2400 网络处理器的高速包过滤的研究

钟 婷,刘 勇,耿 技
(电子科技大学 计算机科学与工程学院,四川 成都 610054)
(zhongting@uestc.edu.cn)

摘 要:包过滤的效率极大地影响防火墙的性能。提出了一个基于 INTEL IXP2400 网络处理器高效的包过滤方案。此方案通过动态规则表,静态规则树及哈希硬件加速单元实现对包过滤的优化,使得基于 INTEL IXP2400 的防火墙能真正达到千兆线速。

关键词:防火墙;包过滤;网络处理器;静态规则树;动态规则表
中图分类号: TP393.08 **文献标识码**:A

Study on fast packet filter under network processor IXP2400

ZHONG Ting, LIU Yong, GENG Ji
(*School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu Sichuan* 610054, *China*)

Abstract: The efficiency of packet filter is vital for firewalls. An efficient packet filter solution based on INTEL IXP2400 was talked about. This solution optimized packet filter through dynamic rule table, static rule tree and hardware hash unit. The firewall designed in this way can reach 1000M line speed.

Key words: firewall; packet filter; Network Processor(NP); static rule tree; dynamic rule table

0 引言

防火墙通过对通信数据的筛选屏蔽来防止未经授权的访问,保护网络安全。光纤技术在主干网上的应用,对防火墙的性能提出了前所未有的挑战。另一方面,网络业务的不断丰富也对防火墙的灵活性提出了更高的要求。传统的防火墙主要有两种开发模型,一种基于 X86 平台,另一种基于 ASIC 硬件平台。基于 X86 平台开发的防火墙在性能方面遭遇了瓶颈,几乎不可能支持超过 OC-12(约 655 兆)的带宽。而能够提供更高性能的基于 ASIC 的设计方案,却面临着灵活性、扩展性差,开发周期长,费用大的问题。从技术角度来讲,目前能解决性能与灵活性两方面问题的硬件平台就是网络处理器。

分组过滤是防火墙最核心的功能。而位于主干网络上的防火墙往往因为包过滤规则数过大,而造成包过滤效率低下,严重地影响了防火墙的性能。研究如何利用网络处理器平台提高包过滤的效率,有着非常重要的意义。有很多公司推出了相关的网络处理器产品,本文的设计基于 INTEL 公司的 IXP2400 网络处理器。

1 IXP2400 网络处理器

1.1 硬件结构

IXP2400 网络处理器是 Intel 公司的第二代产品。IXP2400 的 Core 单元采用 Intel 全新设计的 XScale 技术,该技术改善了内部流水线和访问存储器的性能,可提升处理各种异常包的速度。

IXP2400 内部有 8 个微引擎单元,每个微引擎可处理的线程最多也可达 8 个。在微引擎上指令集中剔除了不必要的通用处理器的特性,使得数据包的处理非常高效简洁。微引擎技术采用多微引擎并行工作及微引擎上的多线程并行工作的机制大大提升了 IXP2400 的处理能力。另外,微引擎的线程零延迟切换技术和高效的线程通信机制使得线程切换带来的消耗达到了最低。

IXP2400 采用了分级存储器组织,对数据进行分类存储以适应不同的应用目的。存储器单元支持 2 个 1.6Gbps QDR SRAM 接口和 1 个 2.4Gbps DDR DRAM 接口。其中,SRAM 的存取速度快、时延小,但价格昂贵;DRAM 的存取速度较慢,但是价格较便宜。

为了提高包处理效率,IXP2400 还支持很多硬件加速单元,比如:hash 单元,TCAM 单元。

此外, IXP2400 的 PCI 单元支持 64bit/66MHz 的 PCI 总线接口。IXP2400 支持在 PCI 接口上外接通用处理器。

1.2 软件设计

软件开发是网络处理器应用的重要一环。在 NP 平台上分为三个逻辑部分:

1) 管理层面,运行于通用处理器或 Core 之上,主要负责用户界面、设置配置、管理策略及审计。

2) 控制层面,运行于 Core 之上,主要负责协议栈的实现,以及生成、更新、管理数据层面所使用的各种表格,比如路由表。

3) 数据层面,主要负责数据包的高速处理及转发。数据层面的处理是影响性能的关键因素。数据层面又分为:快速数据层面,运行于微引擎之上,处理绝大部分的数据包,主要进行数据流输入/输出、打包/拆包、分类、快速查表、转发等实时性高的处理;慢速数据层面,运行于 Core 之上,负责处理异常数据包。异常数据包是指无法进行高速处理的数据包。包

收稿日期:2005 - 07 - 05

作者简介:钟婷(1977 -),女,四川成都人,助教,博士研究生,主要研究方向:网络安全、计算机密码学; 刘勇(1957 -),男,四川成都人,博士研究生,主要研究方向:信息安全; 耿技(1963 -),男,安徽长丰人,副教授,博士研究生,主要研究方向:网络安全.

括碎片包、带有 IP 选项的包、经过加密的数据包或错误的数据包。

基于 IXP2400 实现高效率的包过滤,应该考虑以下几个问题:

1）将各种功能合理地分配到不同的层面;

2）合理地利用不同的存储单元,尤其是数据在 DRAM 与 SRAM 的合理分布;

3）利用系统支持的硬件加速单元;

4）规则在存储器中合理的存储方式。

2　高效包过滤解决方案

包过滤是防火墙系统的核心部分。防火墙的规则通常包括以下几个部分,即:五元组、目标动作和是否记录日志。目标动作常见的有 DROP,REJECT,PASS 和 CONTINUE。五元组是指目的 IP 地址(及掩码)、源 IP 地址(及掩码)、目的端口、源端口和协议号。规则中的五元组的匹配方式包括完全匹配,前缀匹配及范围匹配。完全匹配要求规则中的字段与数据包中的对应字段完全相同。前缀匹配常常用于 IP 地址,比如 00 * 就表示从地址 000…00 到 001…11。范围匹配通常用于端口号,比如 0 ~ 1024。这三种方式的匹配带来了防火墙规则的复杂性,使得无法通过简单的哈希算法来提高规则匹配的效率。

下面将介绍一种基于 IXP2400 的高效包过滤方案。此方案通过 Core 与微引擎的合理配合,微引擎的并发工作,从动态规则表及静态规则树两个方面提高规则匹配的速度。图 1 是此方案的包处理流程。数据包通过 MSF 接口,进入网络处理器后将先后经过流监控模块、状态跟踪模块及静态包过滤模块。

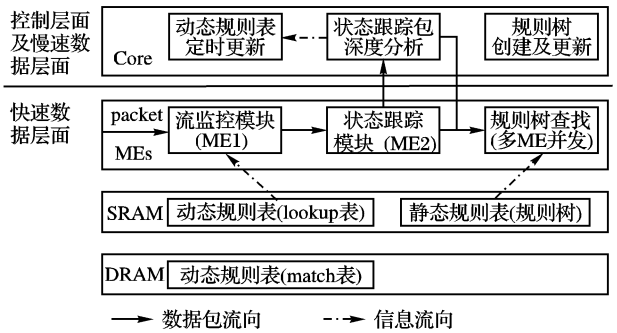

图 1　高效包过滤数据包处理流程

3　动态规则表

流监控与状态跟踪模块都是通过建立动态规则表,利用硬件哈希单元来提高包过滤性能。

3.1　基于硬件的哈希算法

IXP2400 硬件哈希单元支持对 48 位,64 位,128 位数据的哈希运算。下面以 128 位哈希运算为例。哈希单元将需要进行哈希运算的 128 位数据的每一位作为一个一元多项式的系数得到多项式 $A(x)$。硬件哈希单元自动完成下式中余式 $R(x)$ 的运算。

$$A(x)M(x) = Q(x)G(x) + R(x)$$

上式中 $M(x) = m_0 + m_1x + \cdots + m_{127}x^{127}$ 是一个在哈希单元初始化时自定义的多项式。$G(x)$ 是硬件哈希单元内置的 128 次多项式。$R(x)$ 是 $A(x)M(x)$ 除以 $G(x)$ 的余式。

对任意一个 128 位输入数据,哈希单元将返回一个 $R(x)$ 对应的 128 位的数据。再把哈希单元返回的 128 位数据分成 16 位一组做异或,从而折叠压缩得到 16 位哈希值。

考虑到 SRAM 与 DRAM 的性能与大小差异,分别在 SRAM 中建立 lookup 表,在 DRAM 中建立 match 表来完成哈希运算。如图 2 所示,首先以 16 位哈希值作为偏移值找到 lookup 表对应表项。如果表项为空,则表示无匹配项;如果表项非空,且标志位 C 为 0,则表示无冲突,Match Address 中存放的是所查数据在 DRAM 的位置;如果表项非空,标志为 C 为 1,则表示有冲突,Match Address 中存放的是冲突数据在 DRAM 中的位置。冲突数据在 DRAM 中以链表的方式存放。通过 Next Match Address 指针遍历链表,直到在 match 表中找到 Key Index 与 $R(x)$ 匹配的表项,此表项即为所查数据。

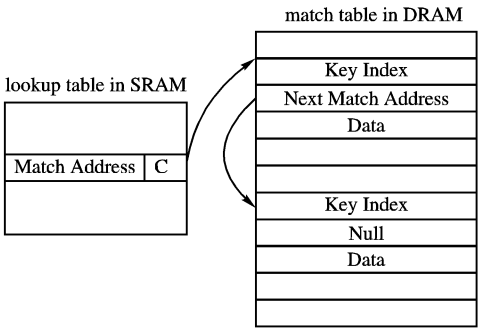

图 2　基于硬件的哈希算法

3.2　流监控

这里的流是指 TCP 数据流,一个 TCP 数据流由五元组决定。流监控的基本思想是:对于 TCP 数据流的第一个数据包,即带有 SYN 标记而不带 ACK 标记的 TCP 数据包,通过查静态规则表来获知对此数据流的动作。查到了相应规则后,需将此 TCP 数据流五元组信息及对应的目标动作存入动态规则表。此后,该 TCP 数据流的后续包到来时,只需查找动态规则表就可以获知对其应有的动作。因为动态规则表中的五元组的匹配是完全匹配,所以可以使用高效的硬件哈希算法。

动态规则表是随着新的数据流出现而建立新的表项,而随着数据流的终止而删除相应表项,所以流监控模块应用 TCP 断连表项删除机制及超时表项删除机制。由于微引擎不具有时钟,而且对表的更新效率是很低的。所以动态规则表的超时更新工作需由 Core 处理。

数据包进入网络处理器后,由一个微引擎(ME1)专门处理数据包的流监控。

3.3　状态跟踪

在应用层协议中存在着一组比较特殊的应用层协议。这些协议的连接分为控制连接与数据连接。FTP 协议就是其中比较常见的协议。这些协议的控制连接只是用于传递控制信令,而传递数据的数据连接的端口号是由控制连接在其应用层数据中通过字符串形式动态指定的。这种动态的端口号使得不可能为数据连接预先制定防火墙规则,只能为控制连接制定规则。当打开状态跟踪功能后,防火墙分析控制连接应用层数据并获知数据连接的五元组,自动地对数据连接执行与控制连接相同的动作。

状态跟踪的实现也是基于动态规则表。微引擎对应用层数据的深度分析能力很差的,所以当打开状态跟踪功能后,由 ME2 将特殊协议的控制连接数据包发往 Core 进行深度处理。Core 通过分析控制连接应用层数据获知数据连接的五元组,并将数据连接五元组与对应动作存入动态规则表。

4 静态规则树

动态规则表的引入,可大大减少但却不能避免对静态规则表的查询。静态规则表是有序的表,通常的匹配方式是低效率的顺序匹配。目前已有很多静态规则匹配优化算法,几乎所有的算法都是以空间换时间。比如 Gupta 与 McKeown 提出的 RFC[1] 及 Hicuts[2] 算法就是其中比较有影响的。Varghese 等人在 Hicuts 基础上又提出了 Hypercuts 算法[3]。Hicuts,Hypercuts 的基本思想都是试图生成一棵多元索引树,来提高规则的匹配速度。理论上来说,Hicuts,Hypercuts 都是很好的方法。但是,通过对它们的实现与分析发现,纯粹的 Hicuts 与 Hypercuts 算法并对于五元组的匹配并不实用。首先,基于五个维度上构造的规则树对空间的要求很高,当规则数较多时,SRAM 难以承受。另外,这些算法都属于复杂的尝试性算法,需要不断地回归尝试才能找到相对较好的结果。Core 更新规则树的负担将相当重,影响防火墙性能。但是规则树的思想很值得借鉴,下面将结合 IXP2400 的特点提出具体规则树的构建方案。

4.1 静态规则树算法

微引擎对前缀匹配的处理能力较弱,而对范围匹配的处理能力较强。考虑到这点,只对目的 IP 地址及源 IP 地址两个维度构造规则树。

一个数据包可能满足规则表中多个规则,但真正起作用的是所有匹配规则中最前面的一条。所以,在实施算法前必须记录每条规则的规则序号。

下面先说明算法中将用到的符号:

二元组(*Dst&Mask*,*Src&Mask*):目的地址(及掩码)、源 IP 地址(及掩码)构成的二元组。

规则组 R:将静态规则表中(*Dst&Mask*,*Src&Mask*)相同的规则放到一组,称为规则组,用 *R* 来表示。

节点(*V*,*Dst&Mask*,*Src&Mask*):规则树中的每一个节点是由一个二元组(*Dst&Mask*,*Src&Mask*)标识,故将节点表示为(*V*,*Dst&Mask*,*Src&Mask*)。

节点(*V*,*Dst&Mask*,*Src&Mask*)与规则组 *R* 相交:如果存在着某个目的 IP 地址与源 IP 地址对既匹配规则组 *R* 又匹配二元组(*Dst&Mask*,*Src&Mask*),则称节点(*V*,*Dst&Mask*,*Src&Mask*)与规则组 *R* 相交。

规则树的节点(*V*,*Dst&Mask*,*Src&Mask*)关联着所有与之相交的规则组。其中根节点为(*V*,*Dst&Mask* = *.*.*.*,*Src&Mask* = *.*.*.*),即树根对应着所有规则组。算法中预置一个量值 *MR*,以保证树的任意一个终端节点对应的规则组的数目不超过 *MR*。规则树生成算法从树根开始,按照下面的方式不断循环分支形成规则树。当规则树中每一个节点对应的规则组数都不大于 *MR* 时,算法结束。

1)生成根节点,开始进入循环分支。

2)对上一步已生成的树,找到该树的一个需要进行分支的终端节点(*V*,*Dst&Mask*,*Src&Mask*)(即其对应的规则数大于 *MR*),从此节点开始继续分支。首先要选定二元组(*Dst&Mask*,*Src&Mask*)中的一个维度进行切分。选择的原则是:计算此节点对应规则组中不同的 *Dst&Mask* 数目与不同的 *Src&Mask* 的数目,选择数目较大的一个维度进行切分。

3)假如选择 *Dst&Mask* 进行切分。由于采用掩码匹配的规则大部分是基于 *A*,*B*,*C*,*D* 类地址的掩码,所以对其每 4 位做一个切分是比较合适的。把 *Dst&Mask* 表示成二进制序列:$x_1 \cdots x_n$。例如:*Dst&Mask* = 123.*.*.*,则可表示为 01111011。接着为 *Dst&Mask* 生成 16 个分支节点,这 16 个分支节点分别关联二元组($x_1 \cdots x_n\ x_{n+1}x_{n+2}x_{n+3}x_{n+4}$,*Src&Mask*)。$x_{n+1}$,$x_{n+2}$,$x_{n+3}$,$x_{n+4}$ 分别取 0,1。删除其中不对应任何规则的分支节点。继续重复 2),直到所有的终端节点对应的规则数都小于 *MR*。

利用规则树进行规则的查询非常简单。对于一个数据包,根据其目的地址与源地址搜索规则树,找到与此数据包匹配的所有规则组。再搜索每个匹配规则组找到与此数据包五元组均匹配的所有规则。所有匹配规则中序号最小的即是最终的匹配规则。

规则树的构造与更新由 Core 完成,规则树的搜索则由微引擎完成。

4.2 规则树算法的优化及效率分析

按照这种方式生成规则树的深度不会超过 16。所以要找到对应的规则组最多只需要 16 步。但是由于只对二元组构造了规则树,一个规则组中可能还包含大量的规则,规则组内的顺序查询会造成匹配效率低下。但通过对真实规则表的分析发现,上述情况出现的概率是很小,偶尔出现也不会太影响性能。而且,对这种情况还可以做进一步的优化:当一个规则组中的规则数太大时,可以对此规则组的规则基于其他维度构造子规则树。另外,如果一个数据包的目的地址与源地址同时匹配了大量的规则组,也会使得效率低下。所以在构造规则树之前先参照文献[4]中的方法优化静态规则表。还有其他一些方法可以对规则树的构造进行优化。比如,如果规则树的两个节点对应同样的规则组,可将其中一个节点的指针直接指向另一个节点,这样可以节省大量的空间。

在仿真实验中,当 *MR* 取 4 时,采用随机生成的有 2 000 条规则的规则表进行测试,生成的树的平均深度在 7 左右,占用的空间不超过 2M,在时间与空间上都是可接受的。而由真实的规则表生成的规则树的实验结果则优于同样大小的随机规则表。

5 结语

本文基于 Intel IXP2400 网络处理器,有效利用网络处理器微引擎并行工作机制及微引擎与 Core 的协同工作,从构造静态规则树,动态规则表两方面入手,结合哈希硬件加速机制,大大地提高防火墙的性能。按照本文描述实现的基于 Intel IXP2400 网络处理器的防火墙,能真正达到千兆线速的处理能力。

参考文献:

[1] GUPTA P, MCKEOWN N. Packet classification using hierarchical intelligent cuttings[A]. Proceedings of Hot Interconnects[C], 1999.

[2] GUPTA P, MCKEOWN N. Packet Classification on Mutiple Fields[A]. SIGCOMM 99[C], 1999.

[3] GEORGE V, WANG J. Packet Classification Using Multidimensional Cuts[A]. Proceedings of SIGCOMM[C], 2003.

[4] ERONEN P, ZITTING J. An Expert System for Analyzing Firewall Rules[A]. Proceedings of 6th Nordic Workshop Secure IT-Systems (NordSec 2001)[C], 2001.

[5] QIU L, VARGHESE G, SURI S. Fast Firewall Implementations for Software and Hardware-Based Routers[A]. Proceedings of 9th International Conference on Network Protocols (ICNP 2001)[C], 2001.