计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2017.20160823
54(6):1300-1313,2017

基于多核平台的高速网络流量实时捕获方法

令瑞林　李峻峰　李　丹
(清华大学计算机科学与技术系　北京　100084)
(lrl14@mails.tsinghua.edu.cn)

Realtime Capture of High-Speed Traffic on Multi-Core Platform

Ling Ruilin, Li Junfeng, and Li Dan
(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Abstract　With the development of Internet application and the increase of network bandwidth, security issues become increasingly serious. In addition to the spread of the virus, spams and DDoS attacks, there have been lots of strongly hidden attack methods. Network probe tools which are deployed as a bypass device at the gateway of the intranet, can collect all the traffic of the current network and analyze them. The most important module of the network probe is packet capture. In Linux network protocol stack, there are many performance bottlenecks in the procedure of packets processing which cannot meet the demand of high speed network environment. In this paper, we introduce several new packet capture engines based on zero-copy and multi-core technology. Further, we design and implement a scalable high performance packet capture framework based on Intel DPDK, which uses RSS (receiver-side scaling) to make packet capture parallelization and customize the packet processing. Additionally, this paper also discusses more effective and fair Hash function by which data packet can be deliveried to different receiving queues. In evaluation, we can see that the system can capture and process the packets in nearly line-speed and balance the load between CPU cores.

Key words　packet capture; receiver-side scaling (RSS); multi-core; DPDK platform; Hash function

摘　要　随着互联网上应用的丰富和网络带宽的增长,带来的安全问题也与日剧增,除了传统的垃圾邮件、病毒传播、DDoS攻击外,还出现了新型的隐蔽性强的攻击方式.网络探针工具是一种部署在局域网出口处的旁路设备,能够收集当前进出网关的全部流量并进行分析,而网络探针工具中最重要的模块就是数据包的捕获.传统的Linux网络协议栈在捕获数据包时有诸多性能瓶颈,无法满足高速网络环境的要求.介绍了基于零拷贝、多核并行化等技术的多种新型的数据包捕获引擎,并基于Intel DPDK平台设计并实现了一个可扩展的数据包捕获系统,它能够利用接收端扩展(receiver-side scaling, RSS)技术实现多核并行化的数据包捕获、模块化的上层处理流程.除此之外,还讨论了更有效、更公平的将数据包分发到不同的接收队列所应使用的Hash函数.经过初步的实验验证,该系统能够实现接近线速的收包并且多个CPU核心间实现负载均衡.

收稿日期:2016-11-10;**修回日期**:2017-03-09
基金项目:国家"八六三"高技术研究发展计划基金项目(2015AA01A705,2015AA016102);国家自然科学基金优秀青年科学基金项目(61522205)
This work was supported by the National High Technology Research and Development Program of China (863 Program) (2015AA01A705, 2015AA016102) and the National Natural Science Foundation of China for Excellent Young Scientists (61522205).

http://crad.ict.ac.cn

关键词 数据包捕获;接收端扩展;多核;DPDK 平台;Hash 函数

中图法分类号 TP391

随着通信技术的发展,互联网早已成为了全球资源的共享平台,实现了人与机、人与人、物与物之间的互联.互联网的繁荣发展与其标准化的基因是分不开的,任何想要接入互联网的设备都需要遵循既定的通讯协议.但也正是这种标准化和通用化导致了其安全性受到威胁.攻击者通过标准的协议与被攻击对象交互,实现其窃取信息或破坏系统等目的.除此以外,网络病毒借助电子邮件附件、网页文件下载等手段进行传播,不仅会导致网络传输效率降低、网络中断,甚至还会窃取用户的核心数据,大量的垃圾邮件也会影响个人和企业的正常生活和工作.根据调查,87%以上的病毒通过电子邮件进入企业,危害极其普遍、严重[1].

网络数据包的实时采集和分析,对于网络的服务商和管理方有十分重要的安全意义.网络的服务商能够通过分析当前网络上的流量来预测网络的流量情况,同时这些数据能够用于优化网络配置,配置出最优的路由路径和负载均衡策略[2].对于网络的管理者,能够通过采集的数据分析出入网络的流量的安全态势,及时发现可能对内网信息系统造成破坏的攻击行为以及从内向外发生的数据泄露.但网络流量的实时采集是个有挑战性的问题,尤其是在网络带宽不断增长的背景下,高性能的数据包采集系统需要在有限的时间内完整地收集和处理网络上产生的大量数据,并完成可能的管理或控制,而高速网络环境中流量分析系统的基础是能够无丢失地捕获网络数据包[2].

1 传统的报文处理

随着服务器网卡速率的提升,目前服务器中通常配有 10 Gbps 的网卡,因此,数据包捕获系统的实现不仅有传统的基于硬件方案,也有基于软件的方案.使用定制化的硬件设备进行数据包捕获的方案优点是,性能相对较好,能够支持高带宽的网络;但缺点则是系统的功能模块相对固定、扩展性差,且硬件成本很高.因此,低成本且扩展性好的软件实现就成了大多数网络服务商选择的解决方案.在高速网络环境下(如大型 IDC、云数据中心等),传输的数据量比一般传统的局域网高出几个数量级.这就要求基于软件的数据包捕获系统能够达到高效性和完整性的要求,能及时处理高速网络上的数据.

Linux 网络协议栈是处理网络数据包的典型系统,它包含了从物理层直到应用层的全过程.其处理流程为:首先驱动向上层模块发出请求,通知协议栈有数据到达;然后协议栈通过系统调用对数据包的网络层协议数据进行处理,它会将网卡驱动的接收缓冲队列中的数据包拷贝到操作系统的接收队列中;接着调用传输层的接收函数进行传输层的数据处理.如果此时系统接收队列中没有需要的数据包时,操作系统会让当前的处理进程进入休眠状态,等待系统接收队列中到达新的数据包,当休眠的处理进程被系统唤醒,它会从接收队列获得所需的数据包,处理完毕以后通过调用 read 函数,将数据包从内核缓冲区拷贝到用户态的应用缓冲区[3].在这个过程中,数据包首先从网卡驱动的缓冲队列拷贝到协议栈的接收队列,再从协议栈的接收队列拷贝到应用程序的内存空间.数据包从下向上的过程中进行了 2 次拷贝及多层系统调用处理.

从上述分析可以看出,由于 Linux 内核协议栈的主要目标是实现通用的网络服务,所以在设计上存在固有的缺陷,大量的中断和数据包的多次拷贝都是普遍认为的"病因"所在.针对这些问题,研究人员们设计了许多高性能的数据包捕获引擎,这些引擎在技术上有许多共通之处,如零拷贝、批处理等.但这些引擎的功能较为单一,不能直接用来开发复杂的网络功能设备.

1.1 Linux 网络协议栈的报文处理

Linux 内核协议栈对数据包处理过程主要分为 3 个步骤[4]:1)DMA 传输及网卡中断设置[5];2)中断处理程序拷贝数据包;3)netfilter 进行路由判决,其上的钩子函数处理数据包,并将数据包交给传输层的函数处理.

通过分析研究者们发现,Linux 内核协议栈在数据包的收发过程中进行了 2 次数据包的内存拷贝,这 2 次拷贝操作的时间开销占了整个处理过程时间开销的 65%,此外层间传递的系统调用时间也占据了 8%~10%[6].

数据拷贝会造成巨大时间开销的原因主要有 2 方面:

1) 数据拷贝过程需要占用总线,而设备中的总

http://crad.ict.ac.cn

线带宽资源是有限的,同时,数据的拷贝都是按照一定的位宽度逐字(32 位操作系统中字宽为 4 B)进行的,大量的数据拷贝操作将消耗大量的 CPU 周期;

2) cache 命中率和 CPU 访问 cache 的效率将会下降. 拷贝操作的过程中,原 cache 里缓存的 cache 行将被待拷贝的数据冲掉,而逐字拷贝会产生频繁的 cache 行替换,从而造成 cache 命中率降低,可以看出协议栈处理数据包时的拷贝操作是协议栈性能的主要瓶颈[7].

NAPI(New API)是 Linux2.6 内核之后采用的一种提高数据包处理性能的新技术,其核心就是使用中断和轮询组合收包,如图 1 所示. NAPI 的假设场景是,一旦网卡开始接收到数据包,数据包就会以高频率到达,换言之,就是针对一直有数据包到达的网络环境做了优化,主要改进[8]为:网卡在接收到第 1 个数据包时将触发硬件中断,中断处理函数会将该网卡加入到设备轮询表中,同时,为了防止后续到达的数据包触发频繁的硬件中断,需要设置该中断使之不再接收中断请求(Rx IRQ);随后,操作系统会触发一个软中断,软中断的处理函数将对轮询表上的设备进行轮询,检查是否有数据包到达并处理;直到本次处理的 CPU 时间片用尽或者数据包的处理过程结束,网卡才重新设置中断屏蔽位开中断接收中断请求.

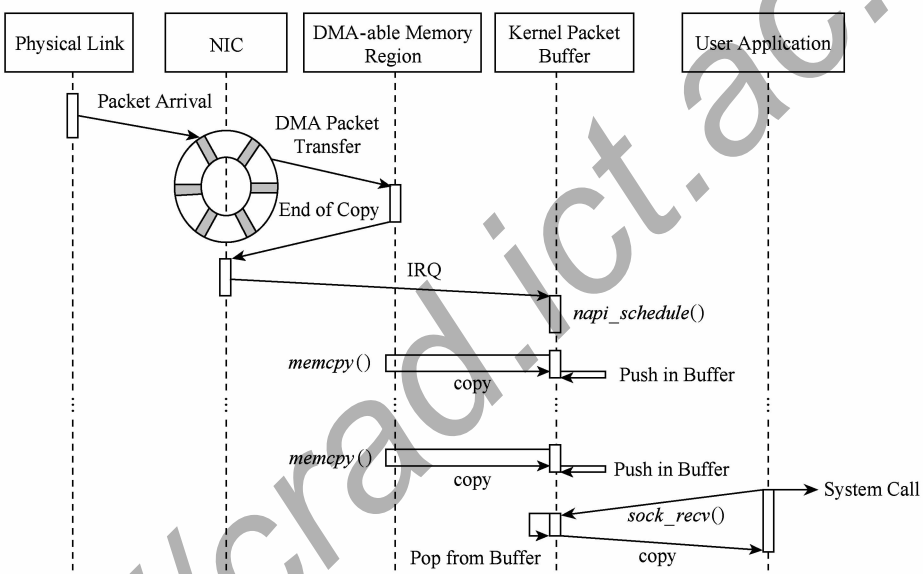

Fig. 1 Methodology of NAPI

图 1 NAPI 原理图

然而,随着网卡缓存空间和 CPU 处理能力的增强,在千兆速率的网络上使用 NAPI 技术已经能够很好地支持各类数据包的处理. 但是随着万兆网卡的普及和主干网络带宽的提升,数据包捕获及处理能力的瓶颈就转移到网卡驱动以及内核协议栈的处理能力上,频繁的软中断处理和数据包拷贝操作会大量消耗 CPU 资源,这些资源消耗在万兆网络环境下显得尤为明显. 如何高效地处理数据包内容,减少系统不必要的资源开销,优化数据包处理过程中的 CPU 利用率,成为数据包捕获技术的主要研究方向[8].

1.2 性能瓶颈识别与分析

由 1.1 节的分析可知,仅有 NAPI 技术是不足以完成在高速网络上的流量捕获这样一个具有挑战性的任务,这是因为 Linux 协议栈体系结构中的固有缺陷降低了性能. 经过大量的代码分析和性能测试,研究者们已经确定协议栈的 5 个主要问题[9-12]:

1) 针对单个数据包级别的资源分配和释放. 每当一个数据包到达网卡,系统就会分配一个分组描述符用于存储数据包的信息和头部,直到分组传送到用户态空间,其描述符才被释放. 分配和释放描述符的过程在高达 14.88 Mpps 的环境下是一个显著的时间上的开销. 此外,sk_buff 庞大的数据结构中的大部分信息对于大多数网络任务而言都是无用的. 如文献[9]中指出,每个分组的 sk_buff 的分配消耗近 1 200 个 CPU 周期,而释放需要 1 100 个周期. 事实上,对于一个大小为 64 B 的分组,sk_buff 相关操作消耗了整个接收处理的 CPU 利用率的 63%[9].

2) 流量的串行访问. 现代网卡包括多个硬件的接收端扩展(receiver-side scaling, RSS)队列可以

http://crad.ict.ac.cn

将分组按照五元组散列函数分配到不同的接收队列.使用这种技术,分组的捕获过程可以被并行化,因为每个 RSS 队列可以映射到一个特定的 CPU 核,并且可以对应相应的 NAPI 线程.这样整个捕获过程就可以做到并行化.但是问题出现在之上的层次,Linux 中的协议栈在网络层和传输层需要分析合并的所有数据包,如图 2 所示.由此引起了降低系统性能的 2 个问题:①所有流量在一个单一模块中被处理,产生性能瓶颈;②用户进程不能够从一个单一的 RSS 队列接收消息.这就造成了上层应用无法利用现代硬件的并行化处理能力,这种在用户态分配流量先后序列的过程降低了系统的性能,丢失了驱动层面所获得的加速.此外,从不同队列合并的流量可能会产生额外的乱序分组[13].

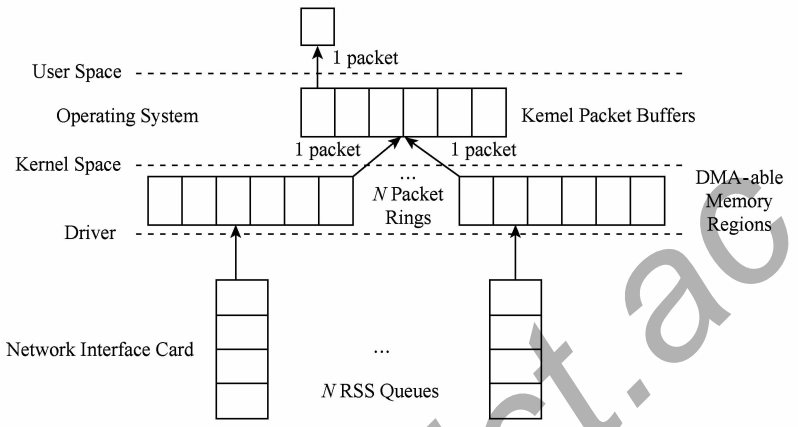

Fig. 2 Bottleneck in standard Linux network stack when using multi-queue

图 2 使用多队列的传统 Linux 协议栈中的瓶颈

3) 从驱动到用户态的数据拷贝.从网卡收到数据包到应用取走数据的一次 DMA 过程中,存在至少 2 次数据包的复制:从驱动中 DMA 的访问内存到内核的数据包缓存,以及从内核数据包缓存到应用程序.一次复制操作的消耗取决于数据包长度,一般为 500~2 000 个 CPU 周期之间[11].上述过程中更糟糕的是,当数据包很小时,逐包复制的效率更加低下,这是由于每个操作恒定的开销引起的.

4) 内核到用户空间的上下文切换.从应用程序的视角来看,它需要执行系统调用来接收每个分组.每个系统调用包含一次从用户态到内核态的上下文切换,随之而来的是大量的 CPU 时间消耗.在每个数据包上执行系统调用时产生的上下文切换可能消耗近 1 000 个 CPU 周期[11].

5) 缺少内存本地化.例如,当接收一个 64 B 分组时,cache 未命中造成了额外 13.8%的 CPU 周期的消耗[12].另外,在一个基于 NUMA 的系统中,内存访问的时间取决于访问的存储节点.因此,cache 未命中在跨内存块访问环境下会产生更大的内存访问延迟,从而导致性能下降.

由于当前的协议栈和应用程序中都不能很好地利用网卡和 CPU 硬件的新特性.我们总结了目前高性能报文捕获引擎中常用的提高捕获效率的技术,这些技术能够克服之前架构的性能限制.

1) 预分配和重用内存资源.这种技术包括:开始分组接收之前,预先分配好将要到达的数据包所需的内存空间用来存储数据和元数据(分组描述符).尤其体现在,在加载网卡驱动程序时就分配好 N 个描述符队列(每个硬件队列和设备一个).需要注意的是这会在驱动程序加载时造成一些额外的时间开销,但是减少了为每个数据包分配内存空间的开销.同样,当一个数据包被传送到用户空间,其对应的描述符也不会被释放,而是重新用于存储新到达的分组.得益于这一策略,在每个数据包分配/释放所产生的性能瓶颈得到了消除.此外,也可以通过简化 sk_buff 的数据结构来减少内存开销.

2) 数据包采用并行直接通道传递.为了解决序列化的访问流量,需要建立从 RSS 队列到应用之间的直接并行数据通道.这种技术通过特定的 RSS 队列、特定的 CPU 核和应用三者的绑定来实现性能的提升.这个架构也增加了性能的可扩展性,因为新的并行数据通道可以通过在驱动层面增加 RSS 队列的数量和 CPU 核来实现.这种技术也存在一些缺点:①数据包可能会乱序地到达用户态,从而影响某些应用的性能[13];②RSS 使用 Hash 函数在每个接收队列间分配流量[14-15].当不同核的数据包间没有相互关联时,它们可以被独立地分析,但如果同一

http://crad.ict.ac.cn

条流的往返数据包被分配到不同的 CPU 核上时，就会造成低效的跨核访问.

3) 内存映射. 使用这种方法，应用程序的内存区域可以映射到内核态的内存区域，应用能够在没有中间副本的情况下读写这片内存区域. 用这种方式我们可以使应用直接访问网卡的 DMA 内存区域，这种技术被称为零拷贝. 但零拷贝也存在潜在的安全问题，向应用暴露出网卡环形队列和寄存器会影响系统的安全性和稳定性[11].

4) 数据包的批处理. 为了避免对每个数据包的重复操作的开销，可以使用对数据包的批量处理. 这个策略将数据包划分为组，按组分配缓冲区，将它们一起复制到内核/用户内存. 运用这种技术减少了系统调用以及随之而来的上下文切换的次数；同时也减少了拷贝的次数，从而减少了平摊到处理和复制每个数据包的开销. 但由于分组必须等到一个批次已满或定时器期满才会递交给上层[16]，批处理技术的主要问题是延迟抖动以及接收报文时间戳误差的增加.

5) 亲和性与预取. 由于程序运行的局部性原理，为进程分配的内存必须与正在执行它的处理器操作的内存块一致，这种技术被称为内存的亲和性. 还有一些其他的考虑因素是 CPU 的中断亲和性. CPU 亲和性是一种技术，它允许进程或线程在指定的处理器核心上运行. 在内核与驱动层面，软件和硬件中断可以用同样的方法指定具体的 CPU 核或处理器来处理，称为中断亲和力. 每当一个线程希望访问所接收的数据，如果先前这些数据已被分配到相同 CPU 核的中断处理程序接收，则它们在本地 cache 能够更容易被访问到. 这项策略可以与先前所述的内存局部性共同优化数据的接收过程，充分利用系统的可用资源.

2 典型高性能收包引擎

本节中，我们列举了 4 个包捕获引擎，即 PF-RING[17]，PacketShader[9]，Netmap[10] 和 PFQ[18]，它们在数据包捕获方面相对于 Linux 原生方法取得了显著的性能提升. 对于每一种收包引擎，我们描述了它的系统架构、使用的何种加速技术、为开发应用提供的 API，以及提供的附加功能.

1) PF-RING

PF-RING 是一个基于 Intel 10 Gbps 网卡的数据包捕获引擎框架. 这个引擎实现了内存预分配，并在其处理数据包的整个过程中通过分配 RX 和 PF-RING 环形队列实现重用内存. PF-RING 也允许建立从硬件到用户进程并行接收队列，即它允许为每个接收队列分配一个 CPU 核，可以根据 NUMA 节点进行内存分配，允许内存亲和技术的开发. 不同于其他方案，PF-RING 实现了完全的零拷贝，它将用户内存空间映射到驱动的内存空间，使用户的应用可以直接访问网卡的寄存器和数据. 通过这样的方式，避免了在内核对数据包缓存，减少了数据包的拷贝次数. 但是，如前面所指出的，这种技术的代价是用户的应用程序可能出现的对于 PF-RING API 的不正确使用(其明确地不允许不正确的存储器存取)，并造成系统的崩溃. 在其余的方案中，直接访问网卡是有保护机制的. PF-RING 的工作过程如图 3 所示. 一些 NAPI 的步骤都被换成了零拷贝技术.

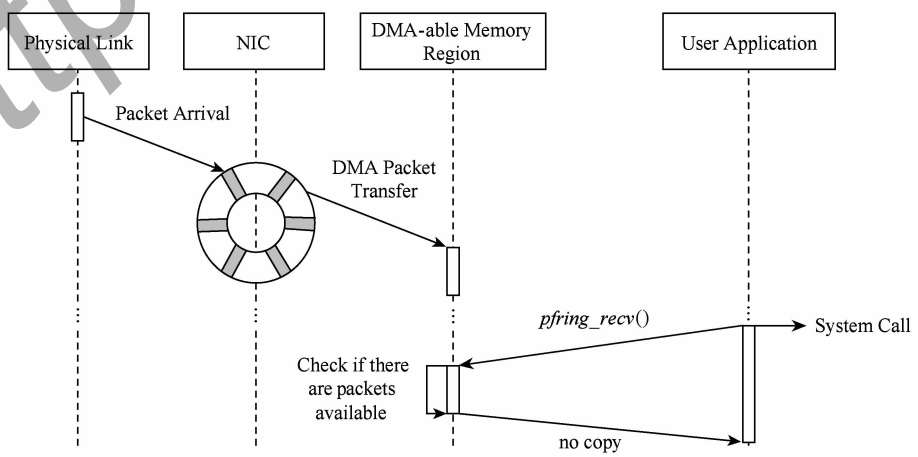

Fig. 3 PF RING scheme

图 3 PF RING 机制

http://crad.ict.ac.cn

2) PacketShader

PacketShader(PS)是开发能够线速工作的软件路由器的基础,是一套新的高度优化流量捕获的引擎.PS 也完全适用于任何涉及采集和处理数据包的一般任务.它使用了内存预分配和重用,具体而言,PS 分配了 2 个内存区域:①用于存储分组数据;②用于存储其元数据.每个缓存单元具有对应于一个包的固定大小的区域.用于存放分组数据的每个单元的大小对齐于 2 048 B,对应于与标准以太网 MTU 相邻的最小 2 的幂.元数据结构除去了对于许多网络任务不必要的字段,从 208 B 压缩至只有 8 B(96%).此外,PS 实现了内存映射,使应用直接访问内核数据包缓冲区,避免了不必要的数据拷贝.在这方面,PS 的作者强调了其引擎的性能在支持 NUMA 数据放置的优势.同样,PS 在用户层面也提供了并行化的分组处理,同时能够随着 CPU 核的数目和队列数目实现扩展性.为了减少每个包的处理开销,PS 在用户层面实现了批处理.每次批处理请求都将巨大的数据包缓冲区映射到连续内存区域,应用可以从该区域访问驱动并复制数据.为了减少 cache 未命中的次数,改进的设备驱动程序会预取当前分组的下一个分组(包括分组数据和分组描述符),如图 4 所示:

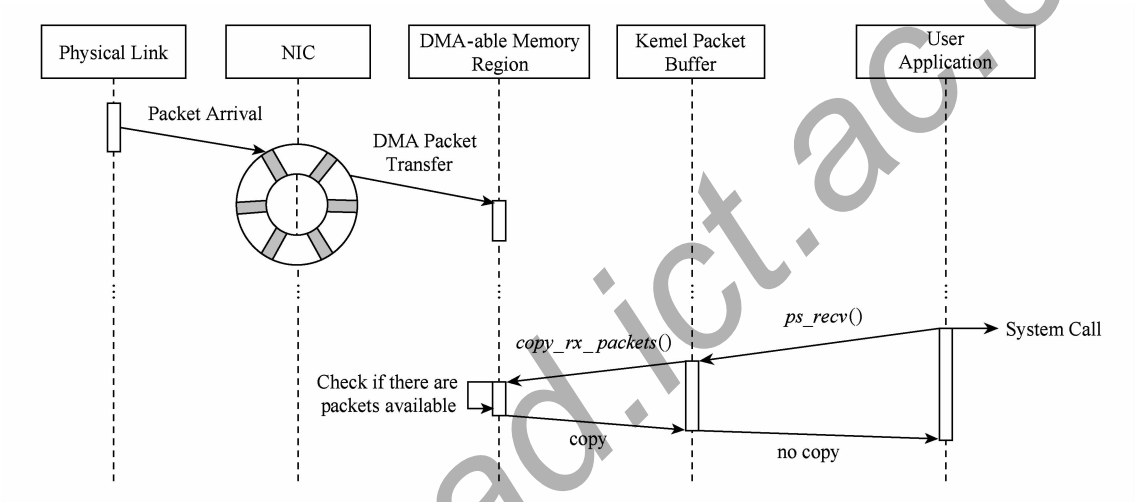

Fig. 4 PacketShader RX scheme

图 4 PacketShader 收包机制

3) Netmap

Netmap 和 PacketShader 体系结构的特征有很多相同之处.即在初始化阶段预分配大小固定的缓存空间(也为 2 048 B),批处理和并行直接数据通路.它也实现了内存映射技术,以允许用户的应用程序直接访问内核包缓存(到 NIC 直接访问被保护起来)中简化和优化过的元数据结构,这种简化的元数据叫做 Netmap ring,其结构体包含环的大小,一个指向缓存的指针(当前数据包),缓存中收到的包的数目,或缓存中可用的空槽的数目.在接收和发送时会分别设置有关状态位、内存中包缓存的偏移值以及元数据信息数组;同时还包含每个数据包的一个存储位置,包括该数据包的长度、在缓存中的索引和一些标志位.需要注意的是每个 RSS 队列,接收和发送都可以使用 Netmap ring 实现并行化的直接数据通道.

4) PFQ

PFQ 是一种新型的数据包捕获引擎,它允许在应用进程中进行并行化的包捕获.PFQ 的方法与以前的研究不同,区别于完全绕过 NAPI 的中断方案或使用 DMA 映射内存和内核包缓存到用户空间等对驱动进行重大修改的方案,PFQ 允许同时使用改进的和旧式的体系结构.它依照 NAPI 的方式抓取数据包,但实现了 2 个新的修改后的内核包缓冲器,使得数据包在进入协议栈之前先进入缓存.首先,PFQ 使用了额外的用于存放数据包副本的缓存以防止内核包缓冲器满了,这些包的拷贝操作是批处理完成的,这样降低了并发度并增强了访问内存的局部性.这种修改融合了批处理技术和内存亲和力技术.第 2 点修改,PFQ 在内核层大量地使用了并行化,所有的功能都能够在多个系统核心上并行执行.PFQ 实现了一个新的中间层,叫做数据包分流块(packet steering block),它位于用户态和批处理队列之间,向应用提供了一些有趣的功能,它在多个接收的 socket 间分配流量.这些流量分配由内存映射技术实现,避免了 socket 和用户空间之间额外的

http://crad.ict.ac.cn

拷贝. PSB 允许捕获线程将数据包分发给多个 socket,从而一个 socket 可以从不同的包捕获线程接收流量. 这个特性避免了使用并行数据通道的缺点,即不同的流或会话的数据包必须由不同的应用处理而不能共享.

3 基于 DPDK 的高速包捕获系统设计

3.1 DPDK 数据平面开发套件

Intel DPDK 是一套可用于开发数据平面应用库,定义了控制平面和数据平面功能分离的一种方式[19]. 控制平面的性能要求小于数据平面的性能要求,数据平面根据控制平面提供的配置来转发流量. Intel DPDK 允许用户空间的进程使用 DPDK 所提供的库直接访问网卡而无需经过内核. 从性能提升的角度来看,有实验表明在吞吐量方面,相比于 Linux 内核有 10～11 倍的性能提升. 在本文中,我们采用 Intel DPDK 作为底层库.

3.1.1 队列管理

为了管理任何类型的队列,librte ring 提供了 rte_ring 的环形队列结构,如图 5 所示. 这种结构具有以下性质:环形队列的大小固定,遵循先进先出的原则实现无锁,支持单/多生产者/消费者的排队场景. 此外,它能够指定进出队列的数据包的个数. 环形队列被实现为一张存储了指向存储对象的指针的一张表,每个消费者和生产者各使用一对指针指向当前队列的头部和尾部进行访问控制. 相比于普通的用长度不限的双链表实现的队列,rte_ring 有 2 个很大的优势:1)它是存取时使用无锁定机制比保护写这种结构快得多;2)由于在表中保存的是指向数据的指针,减少了由于突发操作和大量数据传输导致的 cache 未命中次数.

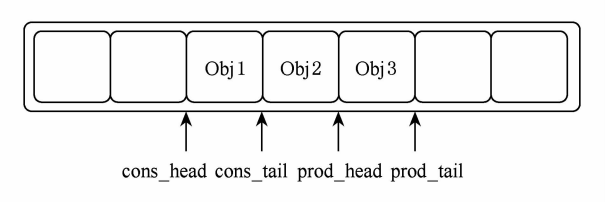

Fig. 5 The rte_ring structure

图 5 rte_ring 结构示意图[14]

3.1.2 内存管理

EAL 可以提供物理内存的映射[14],它会创建一个叫做 RTE memseg 的描述符的表来将地址上不连续的物理内存映射为可连续访问的. 然后将这些内存分成多个内存区域,如图 6 所示. 这些区域是构建于 DPDK 库之上的应用进一步使用内存的基本单元. Librte_mempool 提供的对象之一是 rte_mempool.

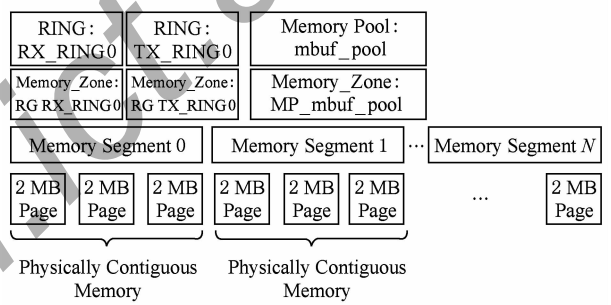

Fig. 6 Distribution of physical memory into segments and zones for further use

图 6 物理的大页内存被分为更小的单元

这种结构是用 rte_ring 来存储可通过唯一的名称确定的固定大小的对象池,如图 7 所示. 为了提高性能,可以在对象之间添加一个特定填充块,用来“只装载在内存不同的通道和等级的每个对象的起始部分”[14]. 例如,在 3 层转发和网络数据包的流分类中可以使用这种优化,因为对于数据包的处理只

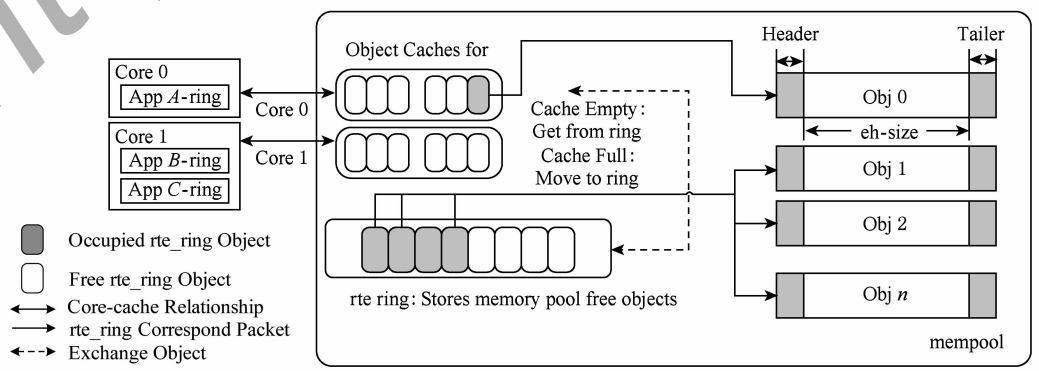

Fig. 7 A rte_mempool with associated rte_ring

图 7 rte_mempool 及其关联的 rte_ring

http://rad.ict.ac.cn

需要依赖第1个64 B即可[14].虽然无锁环状结构可以被用来实现控制访问的目的,但多个CPU核心访问ring的成本可能很高.为了避免在ring处产生瓶颈,每个CPU核心都对rte内存池保有本地高速缓存.当cache全满或者全空时,会使用批量操作与rte内存池的ring自由交换对象.以此方式,CPU核心可以对多次使用的对象进行快速访问从而减轻对ring结构的压力.

3.1.3 缓存管理

任何能够传输网络数据的应用都可以使用librte mbuf的库提供的缓冲区rte_mbuf.这些缓冲会在应用程序实际运行之前就被创建出来并存放在内存池中.在运行时,应用现在可以通过指定mbuf来指定访问某一个mempool.释放一个rte_mbuf只是将回到它来自的mempool[14].为了容纳更大的数据包的元数据,可以将多个rte_mbuf链接在一起.如图8所示,每一个rte_mbuf都包含元数据以及数据包的一部分头部信息.它还包含一个指向下一个rte_mbuf的指针以实现缓冲区链.

需要强调的是,Intel DPDK通过创建环境抽象层(EAL)提供的所有库,以及访问包括硬件和存储器空间的底层资源这些仅仅提供了基本的操作.它不提供类似3层转发等复杂的功能,使用防火墙或类似TCP或UDP协议,则需要网络协议栈的全面支持.表1将第2节所述的引擎和DPDK在各种技术方面进行了比较.

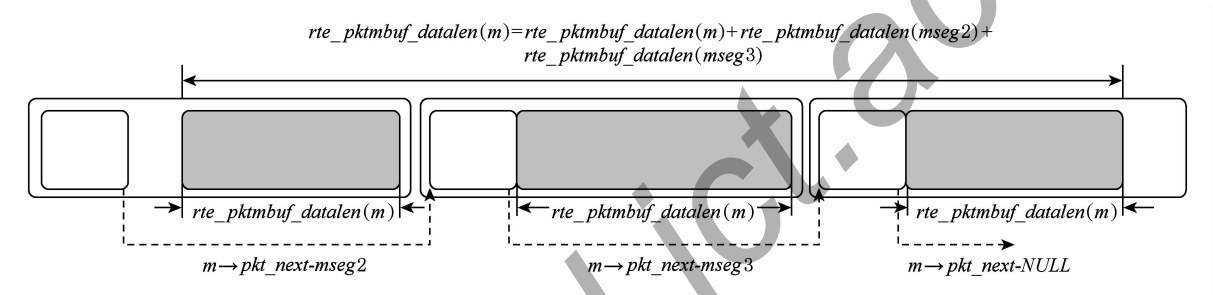

Fig. 8 One packet as chain of rte_mbufs

图8 使用多个rte_mbuf存放一个数据包

Table 1 Comparison of High-Performance Packet Capture Engine

表1 多种高性能包捕获方案的比较

Techniques	PF_RING	PacketShader	Netmap	PFQ	DPDK
Memory Pre-Allocation	√	√	√	×	√
Multi-Channel Process	√	√	√	√	√
Memory Mapping	√	√	√	√	√
Zero-Copy	√	×	×	×	√
Batching	×	√	√	√	√
CPU Interrupt Affinity	√	√	√	√	√
Memory Affinity	√	√	×	√	×
Modification	D,K,K-U	D,K-U	D,K,K-U	D,K,K-U	D,K
API	Libpcap-like	Self-defined	Libc-like	Socket-like	Self-defined

Notes: D stands for Driver; K stands for Kernel; K-U stands for Kernel-User interface; √ stands for support; × stands for unsupport.

3.2 系统总体设计

一个完整的数据包捕获系统不仅仅是包捕获的引擎,而是涉及具体场景、数据包输入、存储、处理等多方面的完整的系统框架,在框架之上有多种功能模块.一个通用的数据包捕获系统框架,我们首先按照系统的功能需求,确定系统实现所需的如下目标:

1) 可维护性.数据包捕获系统应该能够方便地修改可能存在的bug,补充新的功能模块以及对性能进行优化,维护的过程中对于原有部分的影响要尽可能的小.

2) 可靠性.数据包捕获系统是基于其上的各种安全分析模块的基础,捕获系统需要足够可靠,能够应对各种非标准格式的数据包.同时不同的功能模块的故障要进行隔离,不能因为某个模块的失效而影响正常模块的工作,扩大了故障的范围.

3) 灵活性.数据包捕获系统应在不同的网络环

http://crad.ict.ac.cn

境中使用不同的配置模式,具体的运行情况依赖于不同的配置.

4) 可重用性. 由于数据包捕获系统中不同部分对数据包格式处理的流程相似性很大,各种业务分析流程也有相似之处,所以在设计时需要尽量将常用的功能模块化.

系统输入为来自网络的数据包,包含各种不同协议和层次的数据包;系统输出为流量的统计信息和通过提取内容字段等获得的信息. 除了提取数据包的净荷之外,最原始的通信报文也需要保存下来. 系统各个部分描述如下 3 方面:

1) 接收数据包模块. 负责完成数据包的接收和分类,然后对数据包的各层头部进行基本的解封装和提取,剥离头部后剩下的各层协议的净荷传入下一级回调函数.

2) 数据包处理模块. 每一种报文解析的应用就是一种数据包处理模块,例如解析 HTTP 协议的 HTTP 模块、处理 DNS 的 DNS 模块等,各种不同的模块需要有统一设计和接口,内部解析逻辑各有区别. 本模块主要用于不同协议的内容进行分析,比如 HTTP 的请求及响应头部等.

3) 报文持久化模块. 将原始数据包及解析后的协议相关的结果写回数据库,作为分析结果供其他组件使用.

3.3 数据包接收

数据包接收使用了 Intel 的高性能数据包捕获引擎 DPDK. DPDK 的应用在初始化阶段就分配好所需使用的大页内存,这些大页内存被组织成内存池、环形缓存等多种操作单元,供应用程序使用. 除了存放数据包的大页内存,程序初始化时还需要指定程序运行所需的核数等参数. 如图 9 所示,由于 DPDK 中采用轮询代替了中断进行收包,所以收包模式为程序主动调用 *rte_eth_rx_burst*() 接口去接收一定数目的数据包,这就要求我们封装收包接口,将其封装为收到一个数据包作为一个事件,进而触发一系列挂于其上的回调函数对数据包进行处理,从而实现插件式的数据包分析.

由于 DPDK 是批量接收数据包的,所以在调用一次 *rx_burst*() 后,要循环地对每一个接收到的数据包进行处理. 我们创建一个 *local_main*() 函数作为在每个核上一直运行的主线程,在其中调用 *rx_burst*() 并循环处理每个数据包的包头,根据包头来调用不同的回调函数进行处理,在处理结束前是阻塞的.

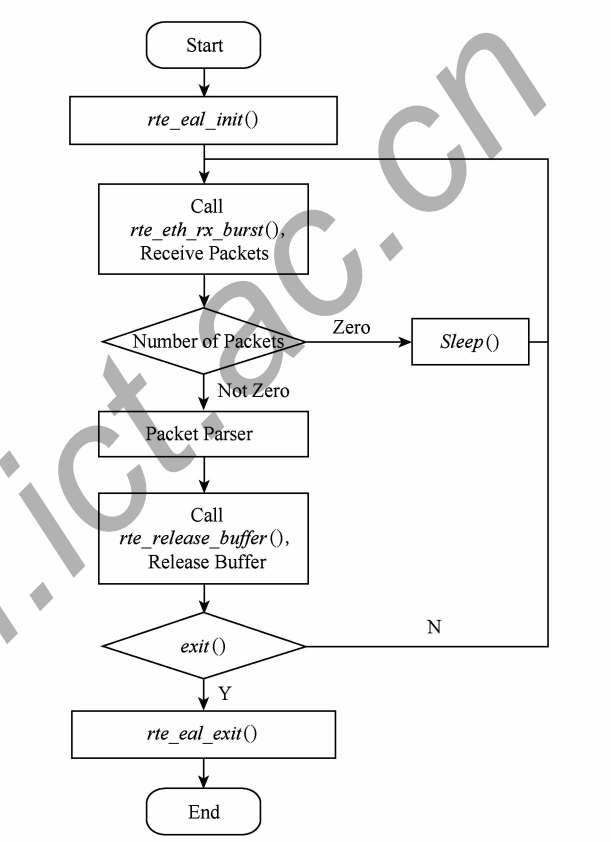

Fig. 9 Workflow of packet capture using DPDK

图 9 使用 DPDK 捕获数据包的流程

3.4 内存管理子模块

整个系统中有 2 个全局的大页内存池,分别是 TCP 流报文内存池和普通报文内存池,这里以普通报文内存池为例详述其实现. 为了减少报文复制带来的性能开销,我们将报文体存储在内存池中,而在队列间仅传递报文的地址. 报文在接收、处理到销毁的过程中涉及 3 个队列,队列传递流程如图 10 所示:

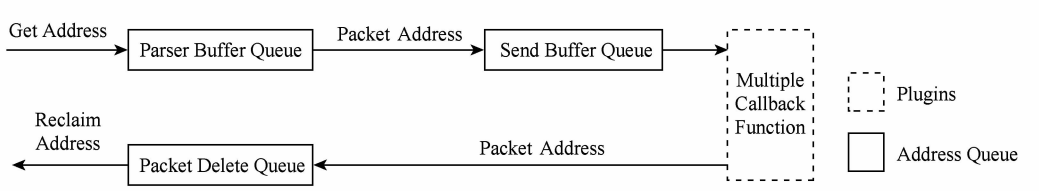

Fig. 10 Packet address transfer process in queue

图 10 包地址在队列中的传递流程

http://crad.ict.ac.cn

网卡接收到报文后,校验和正常的报文将进入内存池,其地址将进入待解析队列,插件调度模块将调用不同的插件对报文进行解析,解析结果及属性将填入报文的元数据结构中,处理结束后地址统一进入删除队列进行删除.每个队列的实现是一个环形缓冲队列,使用了 DPDK 中提供的 rte_ring 来存放地址,实现读写无锁.具体方式如图 11 所示:

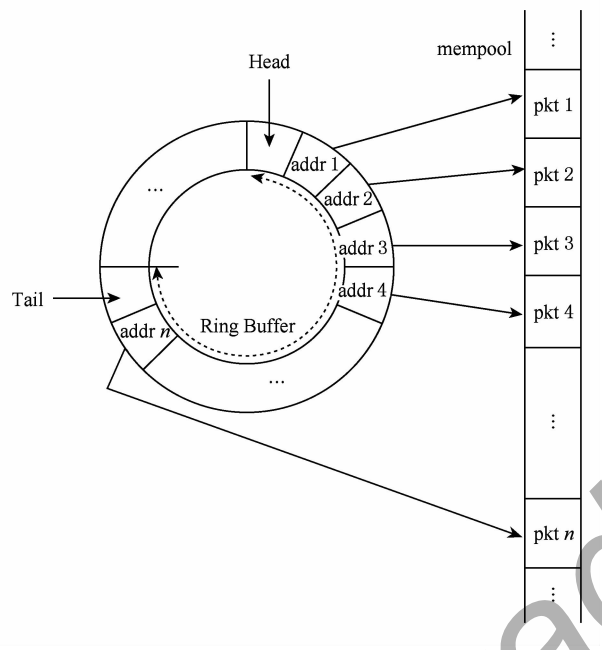

Fig. 11 Packet buffer queue

图 11 报文缓存队列

3.5 插件式数据包处理

插件式数据包处理是数据包捕获系统实现扩展能力的关键,为了解决报文格式间的差别以及不同处理逻辑间的差别,数据包的上层处理采用插件式设计.由于目前计算机网络统一采用 TCP/IP 协议,这就使得不同数据包的处理和解析有了相同的部分,可以采用相同或近似的方式处理网络报文的每一层协议封装.为了解决以上关键问题,通用的插件式数据包处理设计如图 12 所示.

插件式的报文解析模块,可以很好地解决报文解析过程中针对不同解析目标的适配灵活性,同时分发机制可以带来很好的可扩展性.插件式的数据包处理可以实现报文处理插件模块化.将不同的处理功能按照统一的结构化的模板封装成具有相同接口的插件类,这些类被编译为不同的动态链接库在需要调用时加载.增加了系统工作流程的灵活性和可维护性,程序按照配置文件的不同加载不同的插件模块,同时这种方案也可以实现数据包处理管道化.在程序启动时加载的配置文件中的 # route 项可以指定每个插件模块解析后的结果的下一级插件,或者使用 # done 表示处理当前插件已经是处理的最后一级(处理完成).例如网络层解析的下一级处理可能是传输层解析,如果传输层是 TCP 协议则下一级模块有可能是 HTTP 解析,如果是 UDP 协议则下一级有可能是 DNS 解析.

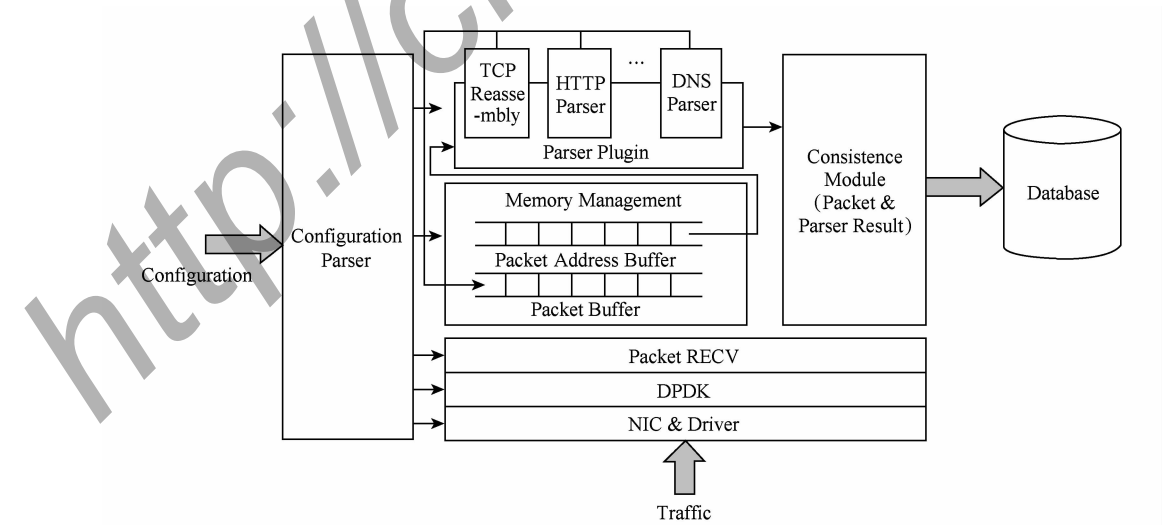

Fig. 12 Pluggable packet analysis design

图 12 插件式数据包解析设计

3.6 基于 RSS 的数据包分发

RSS 是将到达网卡的数据包分配到不同队列中的技术[14].它以可扩展的方式允许每个 CPU 核独立访问不同的网卡队列来接收分组,这样同时消除了访问网卡队列时锁的争夺,允许多个不同 CPU 核的进程并发访问不同的队列.然而,现有的 RSS

http://crad.ict.ac.cn

机制中的一个问题是它会将同一个 TCP 连接映射到不同的网卡队列,具体而言就是映射的队列取决于数据包的方向.我们在此提出一种对称 RSS 的 Hash 算法,它会将同一 TCP 连接中的数据包无论它们是上行还是下行都会映射到同一个网卡队列.

我们首先描述原始的 RSS 算法,它使用 Toeplitz 散列函数[15]基于包头计算散列值,并通过取模操作来决定哪个接收队列来存储分组.RSS 可以应用到 IP/TCP/UDP 报头中的任何字节范围,但通常而言,一般会散列{源 IP,目的 IP,源端口,目的端口,协议}组成的五元组.

算法 1. RSS Hash 值计算算法(Toeplitz)[15].

输入:数据包头部五元组 $Input[]$、随机密钥 RSK;

输出:数据包头部五元组的 Hash 值 ret.

① $ret=0$;
② for each b in $Input[]$ do
③ if $b==1$ then
④ $ret=$(left 32 bit of RSK); /* 与 RSK 高 32 位异或 */
⑤ end if
⑥ shift RSK left 1 bit; /* RSK 整体向左滚动 1 位 */
⑦ end for

算法 1 是 RSS 散列函数的伪代码.INPUT 是由网络层和传输层头部共同组成的五元组,总共 12 B.采用 40 B(320 b)的随机密钥(RSK)与输入值混合的种子值.因为 RSS 算法为了快速处理从而实现在网卡的硬件中,直接修改 RSS 算法使其支持对称映射较为困难.但是,RSK 是设备驱动程序的一部分,并在驱动程序设置网卡时先装入内存.我们专注于操纵 RSK 使得 Hash 函数允许对称映射.对称流映射需要产生相同的 RSS 散列值,即使源和目的 IP 以及端口号被颠倒.

$$Hash(srcIP,dstIP,srcPort,dstPort,RSK)=Hash(dstIP,srcIP,dstPort,srcPort,RSK),\quad(1)$$

式(1)是对称 RSS 队列映射的基本条件.我们需要找到条件满足式(1).我们观察到,不是每个位的 RSK 都用于获取散列值.例如给定一个输入位,只有 32 位的 RSK 用于异或运算.也就是说,对于 n 位 INPUT,只有 $(n-1)+32$ 位 RSK 用于生成 Hash 值.如果 INPUT 大于或等于 290 位,RSK 将循环到第 1 位.所以对于 INPUT 中的特定位,对应于 RSK 的范围是固定的.因此,我们可以导出用于计算 IPv4/TCP(或 UDP)数据包散列值的 RSK 的位范围对应关系.例如,INPUT 的第 1 个字节(前 8 位)对应 RSK 的前 39 位;第 2 个输入字节(从第 9～16 位)对应 RSK 中第 9～47 位的范围.通过这种方式,我们计算得到了表 2 中不同 INPUT 每个位所对应的 RSK 的范围.

设 $RSK[A:B]$表示 RSK 中从第 A 位到第 B 位间的密钥值.

Table 2 Associated *RSK* Range When Computing Packet Header Hash Value

表 2 计算 IP 包头 Hash 值时所用的 *RSK* 范围 b

Input	Length	Input Range	*RSK* Range
SrcIP	32	1-32	1-63
DstIP	32	33-64	33-95
SrcPort	16	65-80	65-111
DstPort	16	81-96	81-127

使用表 2 和式(1)我们可以得出式(2).式(2)需要对应于数据包中源和目的 IP 地址以及源和目标端口范围的 RSK 位采用相同的值.如果 RSK 位范围满足此条件,对于 2 个方向的 TCP 数据包的 RSS Hash 值将是相同的.

$$\begin{cases}RSK[1:63]=RSK[33:95],\\RSK[65:111]=RSK[81:127].\end{cases}\quad(2)$$

进一步将式(2)中的重叠区域拆分后可得到以下 3 个等式,即式(3):

$$\begin{aligned}&1)\ RSK[1:15]=RSK[17:31]=RSK[33:47]=RSK[49:63]=\\&\quad RSK[65:79]=RSK[81:95]=RSK[97:111]=RSK[113:127];\\&2)\ RSK[16]=RSK[48]=RSK[80]=RSK[96]=RSK[112];\\&3)\ RSK[32]=RSK[64].\end{aligned}\quad(3)$$

我们可以求得一个满足式(3)的 RSK 的一个示例,如图 13 所示:

0xDA65	0xDB65	0xDA65	0xDB65
0xDA65	0xDA65	0xDA65	0xDA65
0xDA65	0xDA65	0xDA65	0xDA65
0xDA65	0xDA65	0xDA65	0xDA65
0xDA65	0xDA65	0xDA65	0xDA65

Fig. 13 A symmetric Hash satisfied *RSK* value

图 13 一个满足对称 Hash 要求的 *RSK* 密钥值

http://crad.ict.ac.cn

4 实 验

我们在 Linux 上实现了这个高性能的包捕获系统,它由 5 000 行左右的 C 语言代码和 300 行左右的 Python 语言代码组成,使用了 DPDK 2.0 版本作为包捕获引擎,使用 MySQL 5.6 作为存储持久化结果的数据库. 我们在 CentOS 6.5 上部署了整套系统,服务器采用 Dell PowerEdge R720,CPU 为 Intel Xeon E5-2609,内存 64 GB(其中大页内存分配 16 GB),硬盘 1 TB 机械硬盘,网卡为支持 DPDK 的 e1000 和 Intel 82599 两种型号. 我们将系统部署在一个局域网的出口处,以旁路模式部署,如图 14 所示,我们在内网中部署了 10 台发送数据的服务器,在网关的另一侧部署了一台接收数据的服务器,路由器会将出入局域网的双向流量进行镜像后发送给分析系统,分析系统在收到数据包后存储在本地. 在测试中,我们使用 DPDK-Pktgen[19] 发包工具在 10 台发送服务器上向接收服务器发送 IP 报文,接收端服务器收到这些报文后会直接丢弃. Pktgen 能够产生不同长度的报文,也可以调节发送流量的大小,可以满足实验的需要. 我们还配置了普通 Linux 系统网卡驱动和 PF-RING 上基于 libpcap 的报文捕获系统,用来验证系统在数据包捕获及多核性能扩展方面的优势.

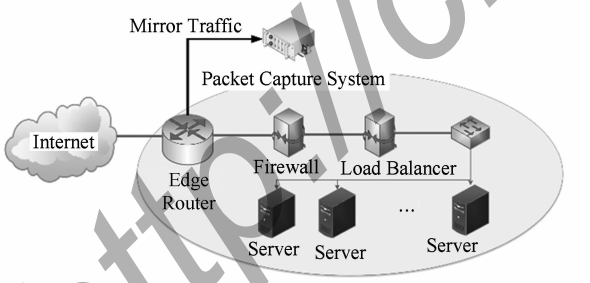

Fig. 14 Experiment setting

图 14 实验环境

我们首先测试了在不同输入流量的条件下,不同引擎的捕获性能及其多核性能扩展能力. 具体实验方法是分别使用 2 个 CPU 核和 6 个 CPU 核捕获报文,测试基于原生 Linux,PF-RING 和 DPDK 三种平台上包捕获系统的性能. 输入流量的大小从 1 Gbps 一直增长到 10 Gbps,测试的数据包长均为 250 B. 从图 15 中我们可以看到,双核环境下的原生 Linux 在输入流量为 1 Gbps 左右时就已经有 10% 左右的丢包,而 6 核环境下的原生 Linux 在输入流量为 2 Gbps 时也开始产生丢包. PF-RING 平台下的捕获情况略好于原生 Linux,但能够承受的流量负载也十分有限;而基于 DPDK 的捕获系统在双核环境下运行时就已经取得了不错的性能,直到输入流量为 3 Gbps 时才产生丢包,好于 6 核环境下的原生 Linux 与 PF-RING 引擎;而运行在 6 核环境下基于 DPDK 的包捕获引擎直到输入流量为 7 Gbps 时才开始丢包,事实上在实际测试中,该系统在 8 核环境下就可以达到线速. 同时我们还可以看到,基于 DPDK 的包捕获系统在不同 CPU 核数环境下运行时的性能有很大的差异,而原生 Linux 和 PF-RING 则不具有这样的特性,这充分说明了系统性能的可扩展性.

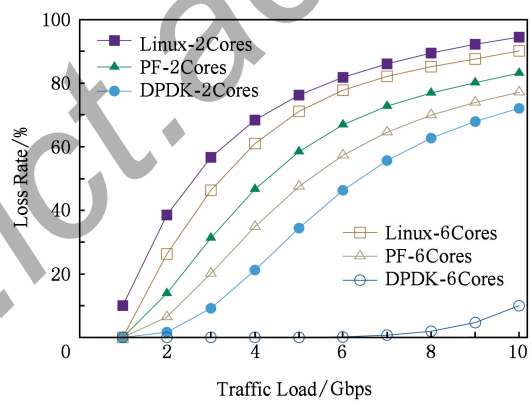

Fig. 15 Traffic loss rate with the increase of traffic load

图 15 随流量增大的丢包率

为了验证系统在不同数据包长度下的性能,我们将输入流量固定为 5 Gbps,观察当数据包长度由 64 B 向 1 500 B 变化时的丢包率. 如图 16 所示,在输入小数据包时,除 6 核环境下的 DPDK 包捕获系统外,丢包率都超过了 50%;随着数据包长度的增大,

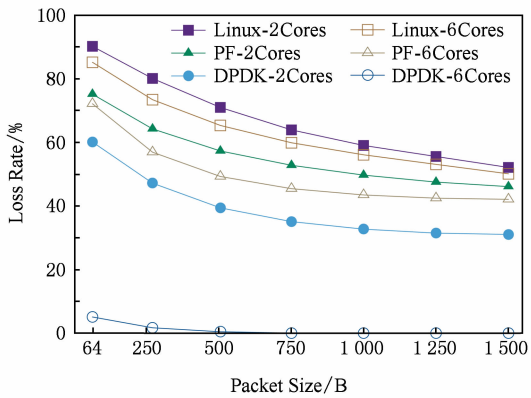

Fig. 16 Traffic loss rate with the increase of packet size

图 16 随包长增大的丢包率

http://crad.ict.ac.cn

所有平台下的丢包率都开始下降,而 6 核环境下的包捕获系统的丢包率一直为 0. 可见我们的系统在处理小包时具有明显的性能优势.

为了验证在使用 RSS Hash 进行数据包分发的有效性,我们统一开启 8 个核来处理流量,在不同流量下监控每个核心的 CPU 利用率. 图 17 是在不同流量输入下 CPU 每个核的利用率情况. 横轴代表 CPU 核的编号,纵轴代表每个核的利用率,我们采集流量分别为 1 Gbps,3 Gbps,6 Gbps 和 9 Gbps 情况下每个核的利用情况. 很显然随着流量的增大,每个核的利用率也在上升,但不同核间的利用率基本保持齐平. 需要注意的是,DPDK 采用的是轮询模式收包,在此过程中 CPU 的利用率始终为 100%,为了限制其收包线程占用的 CPU 资源,我们用 *Sleep*() 函数为收包线程加入了固定的睡眠时间,*Sleep*()中参数越大,进程所占用的 CPU 上界就越低,但过大的睡眠时间会导致丢包. 需要注意的是,图 17 中的 CPU 利用率是在不丢包的情况下测量的临界值.

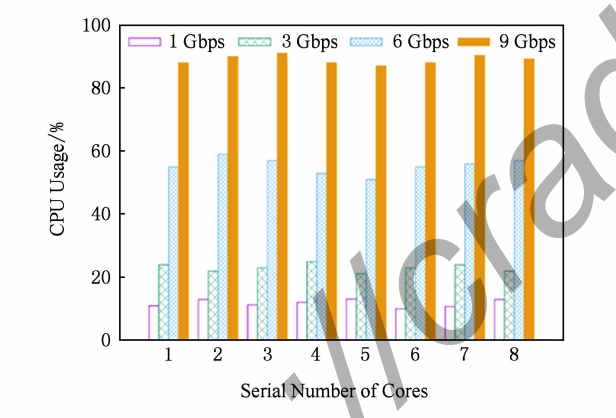

Fig. 17 CPU per-core usage under different traffic load

图 17 不同流量情况下的每核 CPU 利用率

5 结 论

本文围绕高速网络环境下的数据包捕获问题,首先分析了传统网络协议栈在数据包捕获方面存在的问题,其次围绕目前的多种捕获引擎分析了各自优劣,而后基于 DPDK 设计并实现了一个高性能可扩展的包捕获系统,最后对系统的性能做了实验验证. 数据包捕获只是网络流量分析的第 1 步,要分析网络上通信的所有会话,必须要对 TCP 连接进行恢复,以及对各种基于 TCP 的协议应用层协议进行内容分析,这将是我们下一步的工作.

参 考 文 献

[1] China Internet Network Information Center. The 34th China Internet Development Statistics Report [R]. Beijing: China Internet Network Information Center, 2014 (in Chinese)
(中国互联网络信息中心. 第 34 次中国互联网络发展状况统计报告[R]. 北京: 中国互联网络信息中心, 2014)

[2] Liu Baochen. Research and implementation of a high-performance packet-capture system [D]. Shanghai: Shanghai Jiao Tong University, 2013 (in Chinese)
(刘宝辰. 高性能数据包捕获系统的研究与实现[D]. 上海: 上海交通大学, 2013)

[3] Risso F, Degioanni L. An architecture for high performance network analysis [C] //Proc of the 6th IEEE Symp on Computers and Communications. Piscataway, NJ: IEEE, 2001: 686-693

[4] McCanne S, Jacobson V. The BSD packet Filter: A new architecture for user-level packet capture [C] //Proc of the 1993 Winter USENIX Technical Conf. Berkeley, CA: USENIX Association, 1993: 120-130

[5] Heberlein L T. A network security monitor [C] //Proc of the IEEE Symp on Research in Security and Privacy. Piscataway, NJ: IEEE, 1990: 296-304

[6] Nie Yuanming, Qiu Ping. Network Information Security Technology [M]. Beijing: Science Press, 2001 (in Chinese)
(聂元铭, 丘平. 网络信息安全技术[M]. 北京: 科学出版社, 2001)

[7] William S. Network Security Element—Application and Standard [M]. Beijing: The People's Posts and Telecommunications Press, 2000

[8] Zhang Nan. Design and implementation of general purpose data collection system based on IP network [D]. Beijing: Beijing University of Posts and Telecommunications, 2014 (in Chinese)
(张楠. 基于 IP 网络的通用数据采集系统的设计与实现[D]. 北京: 北京邮电大学, 2014)

[9] Han S, Jang K, Park K S, et al. PacketShader: A GPU-accelerated software router [J]. ACM SIGCOMM Computer Communication Review, 2010, 40(4): 195-206

[10] Rizzo L. Netmap: A novel framework for fast packet I/O [C] //Proc of the 21st USENIX Security Symp (USENIX Security'12). Berkeley, CA: USENIX Association, 2012: 101-112

[11] Liao Guangdeng, Znu X, Bnuyan L. A new server I/O architecture for high speed networks [C] //Proc of Symp on High-Performance Computer Architecture. Piscataway, NJ: IEEE, 2011: 255-265

[12] Papadogiannakis A, Vasiliadis G, Antoniades D. Improving the performance of passive network monitoring applications with memory locality enhancements [J]. Computer Communications, 2012, 35(1): 129-140

http://crad.ict.ac.cn

[13] Wu Wenji, DeMar P, Crawford M. Why can some advanced Ethernet NICs cause packet reordering? [J]. IEEE Communications Letters, 2011, 15(2): 253-255

[14] Intel. Intel DPDK: Programmers Guide [OL]. [2016-08-24]. http://dpdk.org/doc/guides/prog_guide

[15] Mircosoft. Microsoft: Receive side scaling [OL]. [2016-08-24]. http://msdn.microsoft.com/en-us/library/windows/hardware/ff567236(v=vs.85).aspx

[16] Moreno V, Santiago del Rio P M, Ramos J, et al. Batch to the future: Analyzing timestamp accuracy of high-performance packet I/O engines [J]. IEEE Communications Letters, 2012, 16(11): 1888-1891

[17] Rizzo L, Deri L, Cardigliano A. 10 Gbit/s line rate packet processing using? Commodity hardware: Survey and new proposals [OL]. [2016-09-20]. http://luca.ntop.org/10g.pdf?

[18] Bonelli N, Di Pietro A, Giordano S, et al. On multi-gigabit packet capturing with multi-core commodity hardware [C] // Proc of Int Conf on Passive and Active Network Measurement. Berlin: Springer, 2012: 64-73

[19] Intel DPDK. Data Plane Development Kit Project Page [OL]. [2016-09-12]. http://www.dpdk.org, June 2014

Ling Ruilin, born in 1992. Received his bachelor degree in Beijing University of Posts and Telecommunications. Master candidate in Tsinghua University. His main research interests include data center network, cloud computing and high-performance networking.

Li Junfeng, born in 1992. PhD candidate. Received his bachelor degree from Xian Jiao Tong University in 2015. PhD candidate in Tsinghua University. His main research interests include on network coding and network performance optimization.

Li Dan, born in 1981. Associate professor. Received his MEn degree and PhD degree from Tsinghua University in 2005 and 2007 respectively, both in computer science. His main research interests include cloud computing, datacenter network and SDN.

http://rad.ict.ac.cn

《信息安全研究》期刊简介

习近平总书记指出“没有网络安全就没有国家安全,没有信息化就没有现代化”.数字时代信息安全工具的大众化是不可阻挡的历史潮流.大众化的信息安全已经直接影响到我们每个人的利益,信息安全已成为国家、地方区域经济结构优化提升和转型发展的新机遇.在信息安全上升为国家战略、行业迎来崭新发展机遇形势下,《信息安全研究》期刊应时代而生.

《信息安全研究》是由国家发改委主管、国家信息中心主办的中文学术期刊,其宗旨是集中展示和报道国际、国内网络和信息安全研究领域研究成果及最新应用,传播信息安全基础理论和技术策略,服务国家信息安全形势发展需要.所刊登的论文均经过专家严格评审.

《信息安全研究》于2015年10月创刊发行.刊期为月刊,每期96页,由《信息安全研究》杂志社出版,国内外公开发行.

《信息安全研究》将以研究致以应用,搭建信息安全领域的学术交流平台,愿意和同行业及社会各界建立联系,友好合作,共赢美好未来.欢迎大家积极投稿、赐稿,洽谈合作.

投稿信箱:ris@cei.gov.cn

编辑部联系人:崔先生(185 0008 6481)

马先生(158 1058 2450)