Winpcap 的体系结构与性能研究*

杨 虹,陈 威
(重庆邮电大学,重庆 400065)

摘 要: Winpcap 有一系列创新(如数据包监视和数据包注入)的特点,这在以前的操作系统中是见不到的。介绍了 Winpcap 的系统结构及其功能。

关键词: Winpcap;数据包监测;数据包注入
中图分类号: TN828.6 **文献标识码**: A **文章编号**: 1674-7720(2012)09-0005-04

The structure and properties of Winpcap

Yang Hong, Chen Wei
(Chongqing University of Posts and Telecommunications, Chongqing 400065, China)

Abstract: Winpcap includes a set of innovative features (such as packet monitoring and packet injection) that are not available in previous systems. This paper presents the details of the architecture and it shows its excellent performance.

Key words: Winpcap; packet monitoring; packet injection

网络分析软件通常依赖底层调用抓取数据包,进行数据包监视或流量分析等。大多数的 Unix 系统都提供这些功能(至少是数据包抓取),而 Windows 提供的这些功能却不令人满意。Windows 提供了一些与各种内核组件相关的 API,但这些 API 存在严重的缺陷,如并不是所有的 Netmon API 都可用,而且其扩展存在很多限制。IP 过滤驱动程序(IP filter driver)只存在 Windows 2000 及以上系统中,而且它只支持 IP 协议,虽然它能够控制和丢弃数据包但却不能监测和构造数据包。PCAUSA 提供了一个商业产品,该产品含有数据包捕获和 BPF 兼容的过滤器,然而它的用户接口处于底层且没有提供过滤器构造这样的抽象方法。随着原本在 Unix 系统上的应用不断转向 Windows 系统,这些特性的缺失成了不可忽视的问题。

本文主要介绍一个强大可扩展的 Win32 平台的网络监听框架系统 Winpcap 的结构及其功能。该体系结构填补了 Unix 和 Windows 间网络监听能力的间隔,使得 Unix 应用到 Windows 的移植更简单,而且 Winpcap 把性能放在最首位,使其能满足更苛刻的需求。

*基金项目:重庆市教委科学技术研究项目(KJ100512);重庆市自然科学基金项目(2010DD2412)

1 Winpcap 的体系结构

Winpcap 的基本结构如图 1 所示,由 1 个过滤引擎、2 个缓冲区(内核层与用户层)以及一系列供开发人员使用的组件库组成。尽管 Libpcap 具有稳固的结构并且功能强大,但 Winpcap 在结构和数据捕获上仍然有自己独到的地方,而且甚至可以认为是对 Libpcap 的创新。

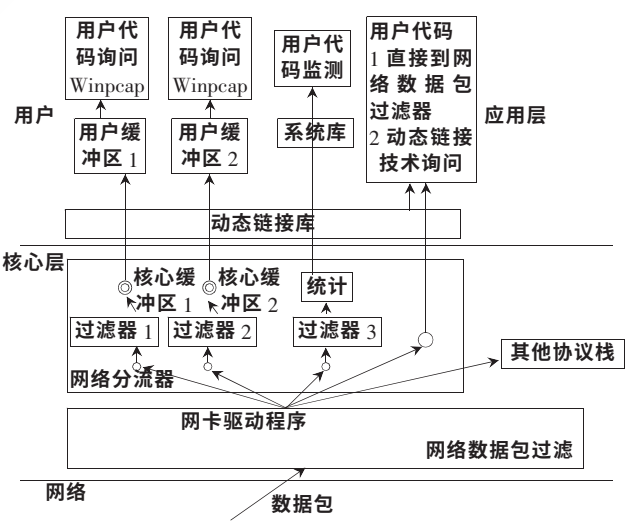

图 1 Winpacp 的体系结构

Winpcap 的过滤可从用户层开始(与 Libpcap 兼容),开发人员可以自定义包过滤条件(如 picks up all udp packets);Winpcap 会把它们编译成一些虚拟指令(如 if the packet is ip and the protocol type field is equal to 17, then return true),同时把这些虚拟指令发送到过滤引擎,并且激活这些功能。要实现这些目的,内核模块必须能够执行指令,因此需要一个"虚拟 BPF"用来在每个接收的数据包执行这些虚拟代码。其中内核层的过滤引擎是获得高性能的关键。

NPF 与 BPF 在体系上最大的不同是对循环缓冲区的使用选择[1]。Winpcap 每次复制一部分数据包,Winpcap 的缓冲区不再是固定大小(Libpcap 的内核缓冲区与用户缓冲区都是 32 KB),而且在将数据包从内核缓冲区复制到用户缓冲区的过程中对数据包进行更新,而不是复制完后更新。在复制过程中,将已经传送的数据包所占用的空间立即释放,因为尽管 Winpcap 工作在较高的优先级并且可能一直垄断 CPU,但内核层的捕获程序可能会中断复制过程。只要交换缓冲区允许占用一般的内存,Winpcap 的缓冲区就可以存放大量的数据包。

内核缓冲区仅通过一个 read 函数就可以完全复制,这可以减少大量的系统调用并且避免程序在内核层与用户层之间频繁切换。因为环境的切换之前必须保存任务状态(CPU 描述符与任务状态大约有几百个字节),频繁切换会导致 CPU 使用率下降。

Winpcap 的内核缓冲区比 BPF 的要大(为 1 MB),因为较小的内核缓存也会带来一些问题,特别是当程序无法与捕获数据包的驱动程序的速度保持一致情况下,这种情况在将数据发往硬盘或网络数据量激增的情况下很常见。相反,用户缓冲区需要小一些,通常情况下为 256 KB。内核缓冲区与用户缓冲区都可以在运行时调整。

用户缓冲区的大小至关重要,因为它决定了仅通过一个调用最多可以从内核读多少数据,同时,一次从内核中至少应该读多少数据也很重要。如果给出一个较大的值,则内核可以在数据送往用户缓冲区之前等待足够多的数据包到达,这可以减少一些系统调用。如果给出一个较小的值,则意味着内核可以将数据包尽快送往用户缓存。这对那些实时性要求比较高的应用来说很重要,一个好的捕获引擎会在两者之间做出好的平衡。NPF 是可配置的,让用户选择程序具有好的执行效率或快速的反应。而 Winpcap 提供了一组函数可以设置数据包读延时和数据包拷贝的最小值。当读延时或内核缓冲区获得的数据包数据量满足了最小值,数据都会被送往用户缓冲区。延时默认设置为 1 s,最小数据量设置为 16 KB,这种功能被称之为"延迟写入"。

1.1 统计模式

包捕获与网络分析会过多占用 CPU,因为有大量的数据需要处理和拷贝。提高执行速度最常用的方法是提高过滤引擎的速度与一次拷贝结构,这种方法是通过将内核缓冲区映射到程序内存来实现。一次拷贝的操作会减少数据的拷贝,但并不会减少内核层与用户层之间的系统调用。如果用户一次读一个数据包,环境切换的代价会抵消一次拷贝所带来的好处。

一种新的方法是监视时并不将数据包送往用户层,而是 Winpcap 将监视的功能放在内核层,这样就可以避免数据送往用户层。Winpcap 提供了一种嵌入在 NPF 过滤引擎中的可编程的统计模式,可以使过滤器成为强大的分类器而不仅仅是过滤器。应用程序可以使用这个模块监视任意的网络行为(如网络下载、两台主机间的网络流量、每秒钟的网页访问数量),并且可以在预先设定的时间间隔获得这些结果。

统计模式避免了数据拷贝,并且实行 0 拷贝模式(统计行为在数据包到达网卡驱动的存储区时执行,之后数据包被丢弃),所以不需要缓冲区。统计模式是监视网络的一种极为有效的方法,且能在高速网络中表现出良好的性能。

Winpcap 为开发人员提供了一组高层调用可以很容易地使用统计模式,对那些习惯了 Libpcap 的开发人员来说很容易掌握。

1.2 数据包注入

BPF 与 NPF 都提供了直接发送原始包的功能,可以将数据包直接发送到网络中。但是 Libpcap 并没有使用这些调用,而 BPF 也并不是用于此目的。大多数 Unix 系统都提供了直接发送原始包的功能,而 Windows 只有 2000 才提供了此功能,且功能受限。因此,Winpcap 是 Win32 平台上第一标准的而且可靠的发送原始包的类库。NPF 提供了许多新的函数可以发送多个数据包而只在内核层与用户层之间切换一次。

Winpcap 虽然提供了很多函数可以去开发这些新的功能(但并没有直接提供这些功能),它需要开发人员手工构造数据包或利用已有的工具。而用户可以使用 Libnet Packet Assembly Library(即在 Winpcap 上加了一层功能),就可以构造数据包并发送到网络中。

2 性能分析

主要是对 Winpcap 的性能进行测试,用 Winpcap 在 Windows98 与 Windows2000 下的表现与 Libpcap/BPF 在 FreeBSD 下的表现进行比较。

图 2 为两台主机直接相连,以排除外部数据包的干扰,使实验结果更精确。一台主机是 Windows2000 操作系统,使用基于 Winpcap 的工具产生大量的数据包发送到网络中,并保证高速率。数据包的大小选择,是以保证每秒所产生的数据包都能够达到一个极限值。

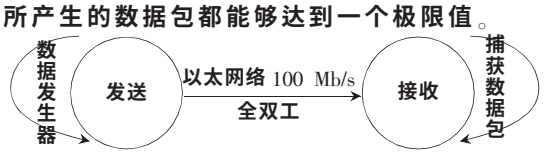

图 2 测试中使用测试平台

操作系统安装在同一台主机上的不同硬盘分区上，以避免由于硬件原因带来的差异。数据包被发送到不同的主机上，这样可以使两台主机之间不用进行交互。接收端的主机网卡设置为混杂模式，根据测试主题的不同而用不同的工具捕获数据包。根据测试需要，数据包收到后或丢弃、或传送给应用程序、或存储在硬盘中。

FreeBSD 的 CPU 负载可以在 TOP PROGRAM 查看，Windows2000 可以通过任务管理器查看，Windows98 则可以用 CPUMETER 查看（可以从网上获得）。

被测试的软件都是最新的发行版，Winpcap 的内核缓冲区为 1 MB，Libpcap 的缓冲区被设置为 2 个 512 KB 大小的空间（默认为 32 KB）。

这些实验将努力避免其他软件对实验结果的影响，但是结果并不能准确地表现出单独组件的性能，因为不同的组件不交互基本上是不可能的。

2.1 发送实验

发送实验是测试发送性能，只在 Windows2000 下进行（95/98 没有对这个功能做优化）。图 3 显示了当每个包都为 88 B 的情况下达到每秒发送的最大值（原认为在每个包为 64 B 的情况下会达到最大值），这出乎意料。因为测试并不依赖 NPF，CPU 的负载始终没有达到 100%，这也说明了 NPF 并不是瓶颈所在。当数据包为 400 B 时网络达到全速。

经实验发现发送能力更多依赖于网卡，当更换网卡、并在相同情况下进行测试时，发送速度只能达到每秒 30 000 个包。

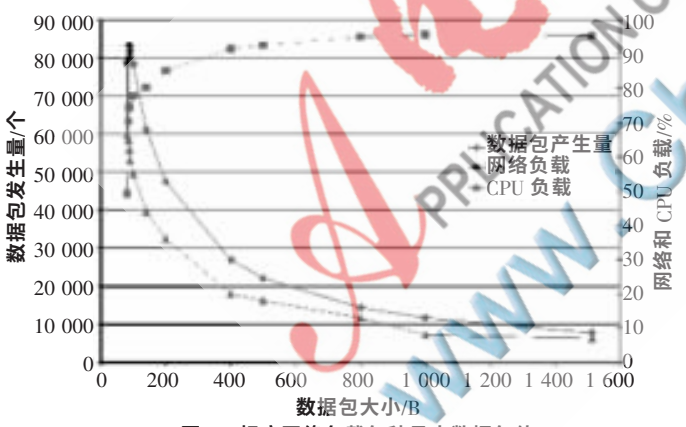

图 3 相应网络负载每秒最大数据包值

2.2 接收与过滤实验

接收与过滤能力实验测试：数据包被网络接口接收并被过滤器检查，如果所有的包都不会满足过滤条件，它们将在检查后被丢弃（不会被拷贝）。

测试在两种情况下进行：第一种使用 3 条 BPF 虚拟指令；第二种使用 13 条复杂的 BPF 虚拟指令。

图 4 为网络分离和过滤性能，几乎所有的包都被接受并没有被过滤器检查。Windows98 的 CPU 负载明显高于 Windows2000（但是也在可以接受的范围内）FreeBSD 表现很差，只捕获大约一半的数据包，而且 CPU 负载始终较高。

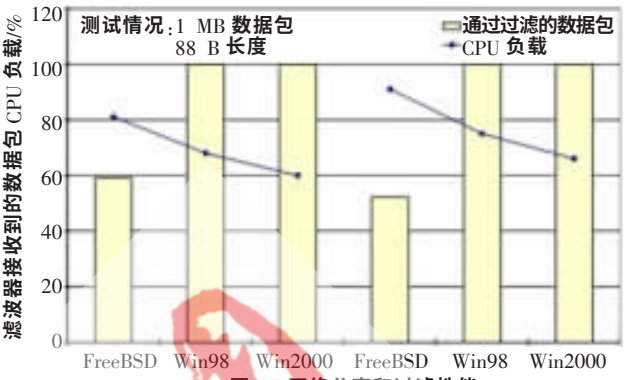

图 4 网络分离和过滤性能

2.3 向应用程序传送数据包的测试

测试应用程序从 Winpcap 接收数据包的能力（收到后直接丢弃，并不做进一步的处理），检验整个 Winpcap 体系的功能，包括数据从网卡复制到内核缓冲区再到用户缓冲区，而且将不使用过滤功能。图 5 为测试结果，Winpcap 几乎可以将所有从网络接口获得的数据全部送至应用程序，没有数据包在内核缓冲区被丢弃。而 FreeBSD 并不能做到这些，特别是在高数据率的情况下，大部分的数据包在没有到达过滤器之前已被丢弃。

无论在 Windows2000 还是 FreeBSD 中，CPU 的负载都随着数据包容量的增大而降低，Windows98 因为没有延迟写入的能力而表现一般。

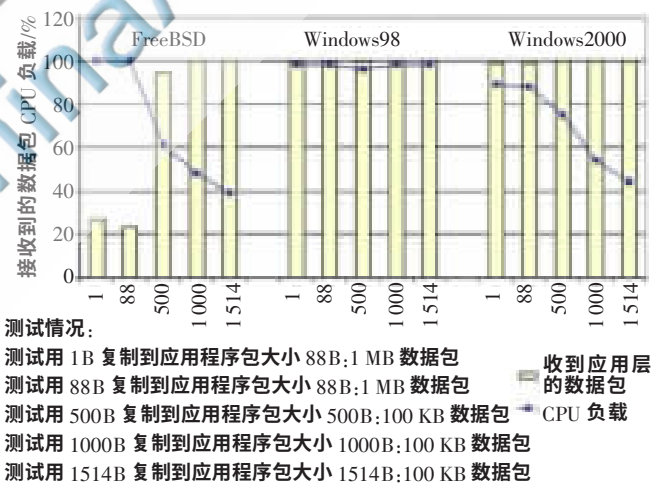

图 5 提供给应用程序性能

2.4 程序性能实验

实验将使用基于 Winpcap 的工具将获得的数据包转存在文件中，结果如图 6 所示。程序将每个数据包的 68 B 的内容存到文件中。所有的系统在高数据率的情况下都会丢失一些数据包：一部分是因为恪守 CPU 时间（当一个数据包到达时，程序却还在处理前面的包）；另一部分则因为内核缓冲区没有足够的空间去存放数据包。图 7 为整个数据包被写入文件时的结果。

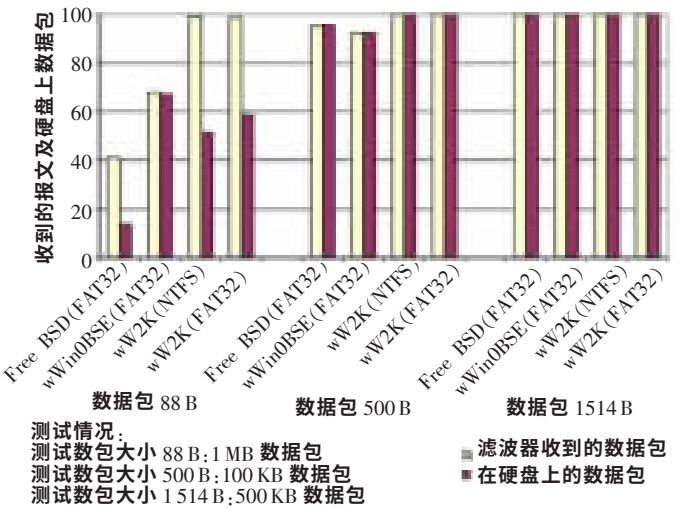

图 6 每个数据包的 68 B 的内容存到文件中

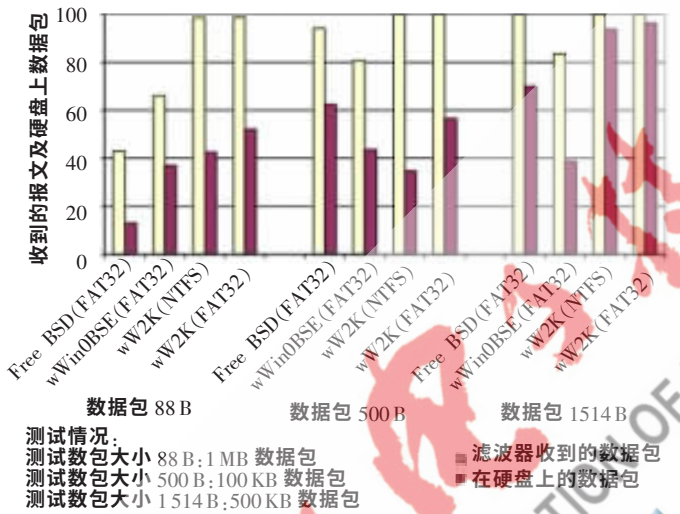

图 7 整个数据包被写入文件

2.5 监视实验

监视实验在 AD-HOC 网络上进行。实验结果表明 CPU 负载一直维持在较低的水平,而且结果与图 4 相类似。图 5 显示了在用户层监视的代价远高于内核层,这些多余的代价会给用户层监视的性能带来折扣。另外,用户层的监视需要大容量的缓冲区。

3 Winpcap 的改进意见

根据实验结果与体系结构的特点,采取了如下相应的改进措施:

3.1 过滤引擎改进

因为采用动态代码生成技术对 BPF 进行改进,使 BPF 性能取得明显效果,因此对 NPF 也进行了类似的改进,采用 JIT(Just In Time)技术,将过滤代码翻译成 X86 的二进制代码,这项改进使整体的捕获性能提高了约 8%。

3.2 内存拷贝

因数据包从网卡到程序需要进行两次拷贝过程,而第一次拷贝(从网卡到内核缓冲区)的代价明显高于第二次拷贝,一个重要的原因是 NdisTransferData()函数进行了额外的操作。但是同时,在研究过程中也注意到一些网络控制器(尤其是一些大型的网络适配器)是在通知网卡驱动之前将数据包拷贝到内存中的,因此 NPF 驱动可以在同一内存区接收数据包。在这种情况下,可以使用 C 标准库的函数来实现这个功能。

3.3 时间戳

用于产生微妙级精确度时间戳的 KeQueryPerformanceCounter 函数可以被 Inter 处理器中集成的 TSC(Time Stamp Counter)取代,TSC 的精度与 CPU 频率相同。X86 的指令集提供了一条指令来取得时间戳(rdtsc),这项改进可以使整体性能提升 27%左右。但是标准的 NPF 发现版并没有使用这项改进,因为这项指令需要硬件支持(只在 Inter 处理器或与 Inter 处理器兼容的处理器上才能运行)[2]。

3.4 Tap 函数优化

Tap 函数中同样用到了 NdisTransferData()函数,而该函数运行时要求分配一块存储区去使某个数据结构可以用来存放要传送的数据包,这个操作在数据包传送完毕时可能会导致回调函数的调用中断。同样也可以用 C 标准库的函数替代它。这项改进可以使整体性能提升 5%左右。

3.5 硬件优化

由于很多开销是在捕获过程之外产生的,这就存在硬件优化的可能性,如 NEW ZEALAND BASED 公司推出有专用于网络捕获的芯片。该芯片避免了过多地与操作系统的交互,它可以产生时间戳,而且直接将数据包传送到系统内存,用硬件来管理缓冲区。这样应用程序可以直接取得数据而不需要同其他层次打交道。因此,硬件的优化可以使基于软件的捕获程序性能得到较大的提升。

本文主要针对 Winpcap 的体系结构及其性能进行研究,并对可能的优化措施进行探讨。文中的实验主要是对 Winpcap 的各个部分的性能进行测试,与一般的看法不同,过滤和缓冲区并不是影响整体性能的最重要的部分。对这两部分的优化,引起了很多的关注,但是针对它们的改进相对整体的性能提升并没有明显的帮助,特别是在数据包比较小的情况下。经一系列实验说明,真正的瓶颈在那些隐藏的地方,如设备驱动程序、应用程序与操作系统的交互、操作系统与硬件之间的交互。

数据捕获是由很多组件配合完成的工作。所以,在对其性能进行优化这个问题上,要从全局考虑,而不应该仅仅局限在某个组件。实验说明了一些大量优化操作都集中在没什么提升潜力的地方(如过滤引擎和缓冲区优化),而真正值得关注的地方却并没有引起足够的重视。同时,即使所有涉及到的技术都很成熟,但性能提升

还是有很大的空间，希望本文能给同行提供一些新的视角。

参考文献

[1] DEGIOANNI L, BALDI M, RISSO F, et al. Profiling and optimization of software-based network-analysis applications proceedings [C]. The 15th IEEE Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2003) Sao Paulo, Brazil, November 2003:3-5.

[2] RISSO F, DEGIOANNI L. An architecture for high performance network analysis[C]. Proceedings of the 6th IEEE Symposium on Computers and Communications (ISCC 2001), Hammamet, Tunisia, July 2001:2-7.

（收稿日期：2011-12-11）

作者简介：

杨虹，男，1966年生，教授，主要研究方向：微波/毫米波集成电路设计与天线设计。

陈威，男，1985年生，硕士研究生，主要研究方向：电路与系统设计自动化。