测试并优化基于软件的网络分析工具

提交文章： wking
提交日期： 2004-04-10
文章属性： 翻译
文档类别： 工具介绍
相关图片： =>查看<=
浏览次数： 今6次/总6238次

===Profiling and Optimization of Software-Based Network-Analysis Applications===

Loris Degioanni, Mario Baldi, Fulvio Risso, and Gianluca Varenni
Dipartimento di Automat/ca e Informatica
Politecnico vii Torino
Corso Duca degli gbruzzi, 24- 10129 Torino, Italy
{loris, degioanni, mario, bald/, fulvio, risso, gianluca, varenni}@polito.it

======stright translated by wking===鲍博.朗曼公司===bblman@21cn.com===04.4======

综述
大量为网络操作和网络管理提供网络监视、统计、安全、流量分析和预警服务等功能的设备需要对在网络
据进行直接并且有效的实时存取。其中软件工具常因为它们低费用和高兼容性而倍受偏爱。然而,考虑到这
常在高速网络环境下表现不佳。尽管人们普遍认为，实时网络分析软件受限于网络环境因而远不能达到理
但我们将证实在有限的硬件支持下，网络分析软件性能仍能有所增进。本文将针对广泛应用于网络分析软
WinPcap库，讨论它的瓶颈，并提出增效方案。以在高速网络环境下推广基于软件的网络应用工具。

1 .介绍
伴随着带宽的增加，调制解调网络发展迅速，同时网络也变得更加复杂。想要实现象监视、问题解答、网
应用功能，需要高水准、专业化的设备。这些工具有如网络分析器、防火墙和设备监视等都是基于硬件的
件解决方案通等常是昂贵的,难以发布(例如，硬件不能被复制且不便移动)，并且它们与软件解决方案相比
高。
到目前为止,基于软件的解决方案通常作为标准操作系统的扩展得以实现，通过它们向应用程序提供原始网
据。象著名的libpcap[1]和WinPcap[3],都已应用在众多操作系统(OSes)上。这些库(library)输出一套原语，
程序在没有任何另外中介层参与的情况下实现网络应用操作。软件部件发布便利且灵活性强:一个简单的包
能提供底层数据给应用程序(例如，防火墙、NAT、嗅探器、联网监视器等等)。而且它们升级方便，花费
因为这些，许多提供网络监视和分析的专业软件都是使用这种方式来实现的。然而,性能问题是软件工具的
点,当处理高速网络时，使用硬件设备成为首选。尽管当前的CPU已很强大，但在倍速比特速率下仍然无法
软件来实现实时流量分析。
目前世界各地的研究小组(如[10],[13])正从事这方面的工作，提升网络分析工具整体性能是一个公开的话题
们把精力集中于一些特定的流量分析部件(例如包过滤器)，并通过各种途径改进这些功能部件的性能。然而
对整体流量分析性能感兴趣，而非某个单一部件。因而研究工作的成果并未能有效地被个人用户所接受。
这项工作权衡分析各种组件在网络分析工具中的比重。一套针对WinPcap库的优化方案在试验版本中配置
将以定量法测试改进后的效能。从检测数据显示，部件优化导致系统性能仅百分之几的提升，并不能改变
的观点。
本文由以下几部分组成。第2节，对相关研究成果的概述。第3节，将Winpcap作为典型的系统扩展部件，
构造。第4节，呈现对各个部件(如包截获组件、系统应用层组件)详尽性能评估的结果。第5节，介绍一些
权衡组件比重，检测它们对整体性能的影响效果。最后,在第6节做总结。

2 .领域发展

CMU/Stanford斯坦福包过滤器[14](或CSPF)可谓元老级包过滤器，它是最早发布的实现从应用层存取数据
据的可行系统。它还介绍了虚拟过滤机的概念：一个基于虚拟CPU(含有寄存器等)与精简高效指令集的包过
过滤器则是能工作在虚拟机的可执行程序。
McCanne和Van Jacobson是该领域另两名杰出贡献者，他们在1993年发布了(BPF)包过滤器[2]。它通过限
包的数量并定义了一个新的、更加高效的、基于寄存器的虚拟处理器完整指令集(即,含有load、store、con
jump等基本指令)来改进CSPF。目前BSD操作系统仍然提供BPF，作为缺省的包捕获工具;其它一些操作系
似的实现工具。BPF虚拟处理器还是libpcap库偏爱的底层实现。
Mach Packet Filter马赫包过滤器(MPF)[9],PathFinder[12],DPF[11],BPF+[10]都是一些以提升过滤处理器来
分析性能的过滤器。Packet classification[15]是另一个概念上类似于包过滤器的流量分析组件,同时因为其在
备中的应用角色得到了很多关注。
在另一方面，还有极小一些研究是针对其他形式的包捕获技术，如缓冲拷贝。NFR小组使用一个更大缓冲
BPF的版本性能，并通过检验共享缓冲区来阻止重复拷贝[13]。WinPcap[3],是一个公开代码的Windows库
记忆占有实现一个更高效的缓冲系统来改进libpcap。然而,先前的工作没有关注总体分析过程的性能,这便是
目的。试验结果将呈现为什么现有的优化措施仅能将捕获过程所花费的执行时间节省百分之几的原因。

3 .包捕获模型
本小节定义用于包截获和流量分析组件的典型结构模型。特别地,我们参照数据包从在网卡(NIC)被接受，指
递到工作站的主存，然后通过设备驱动和操作系统等中介应用于程序这条完整路径来作详细说明。其中NPF
(NetGroup Packet Filter)[3]是由BPF驱动，并已植入到WinPcap库中。本文将其作为特例经常提及，应当指
NFP同其它过滤器一样，它们的基本原理是相同的。图1和图2所展示的是WinPcap处理一个被接受的数据
传给应用程序所涉及的具体步骤及相关组件。
3.1.网卡和NIC适配驱动器
调制解调NIC板载存储器通常只有几Kbytes。独立于主工作站的存储容量，NIC存储器被用来全速接收、发
包。此外，NICs施行一些初步的检查,例如CRC校验和检测是否满足以太网短帧要求。当数据包存储在NIC
帧将被及时丢弃。

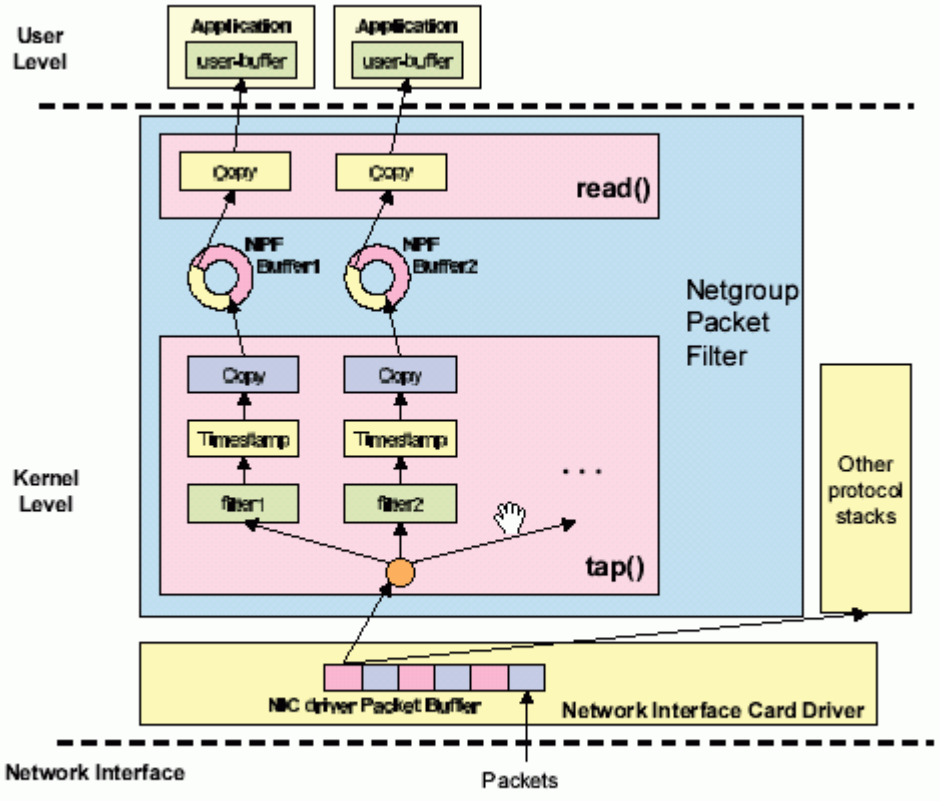


Figure 1. NPF Structure.

当NIC收到一个有效的数据包后，网卡向系统总线控制器请求总线控制权以传输数据。这时，NIC获得总线
向工作站主存传输NIC缓冲区的数据(看见图2)，释放总线后，NIC向高级程序中断控制器芯片(Advanced
Programmable Interrupt Controller_APIC)生成硬件中断。APIC接着唤醒操作系统中断处理程序，进而触发
驱动器的中断服务程序(Interrupt Service Routine_ISR)。
高效设备驱动的中断服务程序(ISR)将完成其份内职责。最基本的，如它会检测与本设备相关的中断(由于在
中允许多个设备共享同一个中断)并给予确认。然后，ISR 预置一个低优先级的函数(称作Deferred Procedu
or DPC)，它会在收到数据包后处理硬件请求并通报上层驱动(即协议层驱动、包截获驱动等)。在完成所有
中的中断请求任务后，CPU启动DPC程序。由于一次只能处理一个数据包，故NIC设备驱动处理数据包时,
NIC的中断将会被暂时禁用。另外，中断产生是一个很耗系统资源的过程，现在新型的NICs允许在一次中断
一个以上的数据包,使得上层驱动程序在被激活时有能力同时处理若干个数据包。

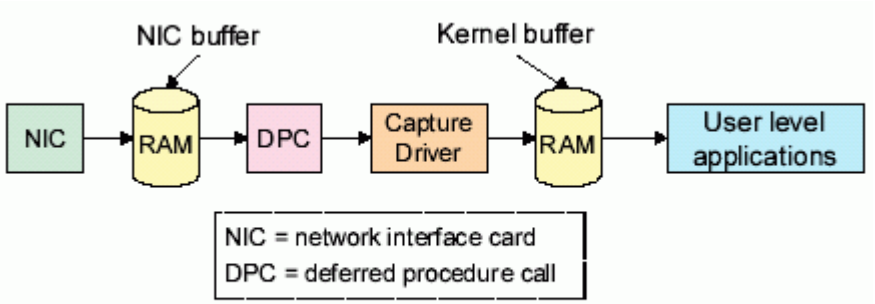


Figure 2. Path from NIC to applications.

3.2 包捕获驱动

包捕获部件类似协议栈，通常透明于其它软件模块,这将不影响标准系统的工作。它们只是在系统中插入一
以便能通过调用返回函数-tap()-及时了解数据包到达的情况。在win32系统上，包捕获部件通常以网络协议
式被实现。

tap()的第一反映便是过滤数据包,即探测它们是否为使用者所感兴趣的。NPF过滤引擎演化自BPF，是一个
单指令集的虚拟处理器，它能对通用缓冲器(数据包栈)执行一些简单字节操作。WinPcap(还有libpcap)提供
用户层API，转递高级语言表达式(例如，"提取所有UDP包")进入一套伪指令(例如，"if以太网字段头是IP且
头的IP等于17,那么返回ture")，后将它们送到过滤机,并激活该指令。上述结构体系在数据包位于NIC缓冲区
用于包的过滤，以便减少不匹配的无用拷贝。尽管如此，这仍将占用总线资源，因为它们已被送入系统内存
过滤器收到数据包的同时，也得到了一些相关的物理层信息系,例如长度和时间戳，这将有利于今后应用程
的处理。然后数据包被拷贝进一个缓冲区,通常称作核心缓冲区,在那里数据包等候被传输至用户层(看见图2
截获进程来讲，这个缓冲区的大小和体系结构设计是否合理直接影响到整体性能的表现。例如一个容量大
学的缓冲系统能弥补在突发繁杂期间用户层应用程序迟缓的速度，并能减少从包截获驱动(即，核心缓存区
程序传输调用数据的请求次数。

用户层应用程序通过类似于系统调用的方式从核心缓冲区得到数据包。当NPF就绪时，系统调用触发"钩"函
()(参见图1),它负责检查NPF核心缓存区:当缓冲区不空时，缓冲区的内容将被传输到用户程序分配指定的内
图1中被标志为"用户缓冲区")。当数据被复制到用户层时，应用程序立即被唤醒，并开始处理数据。

4 .性能评估

在这一节里，我们将呈现对网络分析系统详尽测量的结果。目的在于测定包截获进程的效能。对第3节中提
组件，我们都将统计它们具体消耗资源的准确数目。

按照常规，我们以CPU时钟周期的消耗量来统计。事实上,这样的好处是为了更加客观公正，因为这种测量
于具体系统的绝对时间和CPU速度。

4.1 试验平台

图3所展示的是试验平台:两台PC机通过高速以太网直接连接。一台PC充当通信发生器。另一台则用于实际
在它上面安装了改良版的Winpcap,里面有用于测试的扩展插件。要特别说明的是,该扩展能调用Pentium系
器的性能监视计数器[5][6]。该系列微处理器都含有内置的计数器(其类型和数量取决于微处理器的型号和版
些计数器能被用于追踪事件，比如已编译的指令数目、中断请求的数目、加载缓存的数目等。再例如,CPU
UNHALTED计数器统计的是CPU在特定时间间隔内有效的钟周期数。并且，该计数器还能识别消耗在用户
与消耗在核心级的数量。具体地，程序可通过rdpmc指令来得到该计数器的数值。

另外，扩展插件还使用特定动态链接库(DLL)，核心驱动利用它们测量特定代码需要的CPU时钟。这些库使
x86 rdtsc(Read Time- Stamp Counter)指令来确定所给定代码在执行期间所消耗的CPU时钟的准确数目。

最后,扩展插件通过Intel Vtune性能分析器[16]实现有效的采样技术。CPU在精确的时间间隔内被冻结，届时
CPU的状态来确定目前正被执行的驱动或函数。采样过程可持续相当长的时间，对截获中相关的软件模块
统计。

两台PC机装配的是来自不同厂商的网络适配器。3Com 3C996 Gigabit以太网适配器口碑级佳,它工作于10
速度下，并且它对主机系统的资源消耗相当少。还有一些细节分析过程基于Intel 85527以太网适配器，其
是由微软Driver Development Kit[7]提供的高效源程序代码。它也是少数公开代码的Windows NIC驱动程序
利于代码的进一步研究和完善。

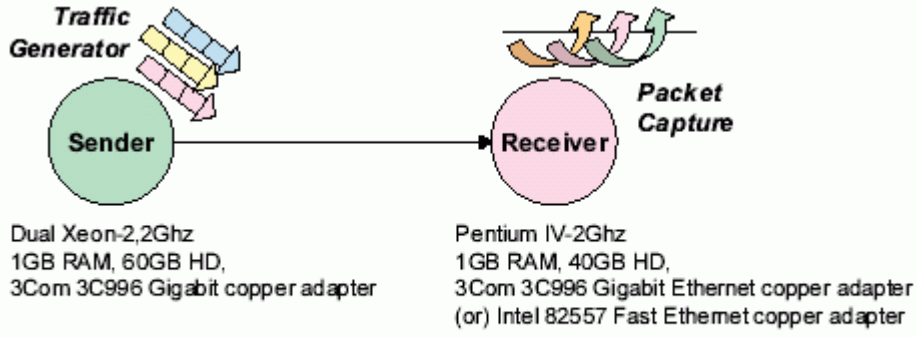


Figure 3. Testbed.

其中在一台PC上安装的通信发生器能够以精确的频率不断产生突发数据包。通信发生器指向一台在该网段

在的主机，这样一来它发出的数据包不会影响试验平台中两台主机的协议栈。两台PC电脑运用的都是Micr
Windows XP。我们使用不同的发包率来进行测试,尽管大多数以太网适配器都有最高速率的限制(148809帧
64字节)，但同时这也是考验数据包监视、分析工具最恶劣的试验环境。
流量测试模块相当简单(只是一个帧频常量)，因为我们的目的只是在长时、高负载的情况下对软件进行测试
在现实中，流量状况会好于我们所选的试验环境(例如，满足可变波型、泊松分布等)。
4.2 附加因素
许多因素都会影响到数据包处理进程，比如NIC驱动程序和OS，虽然严格地讲它们都不属于数据包捕获模
处理数据包的时候，它们与包处理过程息息相关，并显得十分重要。
4.2.1操作系统
在网卡收到数据包时，操作系统将是第一个做出反映的部件。操作系统资源耗费的多少与网络流量发包率
它主要还是与产生中断响应的多少成比例。中断是由NIC产生的，该机制以便通知系统数据包的到达，并等
理。在我们的测试平台上，每个中断大约需要2700个时钟周期。测试中，3Com NIC在148K fps的网速下每
2999个中断,这相当于每帧数据需要54个时钟周期。速度的快慢因网络适配器的不同而不同，因网络流量的
不同，同样它也取决于操作系统核心层中断处理机制的性能。例如在测试中,有三个操作系统函数是能耗大
需占用相当数量的时钟周期:HalBeginSystemInterrupt()(提升当前中断级别，标识中断控制器),KeDispatchI
(执行NIC驱动程序的DPC例程),KeInitializeInterrupt():这是非正式的(但它可能用于降低当前中断级别，撤下
制器中的标识)。
4.2.2 NIC及其设备驱动程序
尽管网卡能够自己努力工作，完成份内职责，而不要求中央CPU参与数据处理。但它的表现将影响其它部
OS和设备驱动程序。例如,请求中断的数量(它影响OS消耗)、对NIC上寄存器存取I/O操作的数量(它影响设
序)都对性能表现有重要影响。
对于后者来讲,ISR函数(它是设备驱动程序的第一个函数，每次中断都会调用它)十分简单但开销却相当高(大
850个时钟周期)，这是因为它需要对NIC进行成对的I/O操作来指明当前驱动器正在处理数据包。
可以通过增加加载的数值，即在每次响应中断时增加从NIC缓冲区提取数据包的数量来改进性能。这可减少
(更少的中断请求数目)，减少驱动程序花费(在网卡上的I/O操作次数更少了)。从试验结果的数据显示，增加
断时加载的数据包数目，将得到性能的线性增长。例如,每次中断3Com NIC平均处理49.61个帧的数据(对应
产生2999次中断)可达到每秒148k fps的速度。这意味着，已经达到适配器极限速度。故在包处理过程中，
(中断处理、NIC驱动程序)对升级整个网络分析软件性能来讲已十分有限。
根据我们的测试，当NIC驱动只结合WinPcap而不附带其它协议时(最大速率为每秒148809个数据包)，Inte
适配器需要花费2260个时钟周期，3Com 3 C996 适配器需要花费1497个时钟周期。
最后,还有一个附加因素的开销难以量化。数据包通过系统总线控制器从NIC传输到系统内存。虽然这并不
时钟，但在传输过程中总线毕竟也属于系统资源。从宏观来讲，总线是难以获得、会延迟CPU请求的一项
的因素。
4.3 包捕获程序
这节内容将分析所有捕获程序部件所需的资源开销。如,数据包从NIC设备驱动程序(还有OS)到用户应用程
径的资源开销。
4.3.1 过滤程序
过滤程序部分应受到特别的关注,因为它是唯一要处理所有流经网卡数据包的部件。显然,它的好坏不仅取决
引擎的效率,而且还与过滤器的复杂性有关，如对每个包的检查次数。图4展示的是3种过滤器耗时情况(有序
杂度):最简单的一个仅接受IP包(它要求实行3条NPF虚处理器伪指令),第二个针对5个不同的值检查TCP端口
包(需执行21条伪指令),第三个则是最复杂的，它对10个IP地址和10个TCP端口进行数据包检查(需50条伪指
虽然过滤所耗费的开销仅取决于NIC和系统总线结构，它与时钟周期并没关系。但我们在本篇文章中都将使
周期作为测量参数、评判标准。
执行所有的过滤器程序代码，完成对数据流的选择过滤。和期望的一样，过滤所花费的时钟周期与过滤指令
成线性比,如图4所示。一般典型的过滤器需要几百个时钟周期的开销。

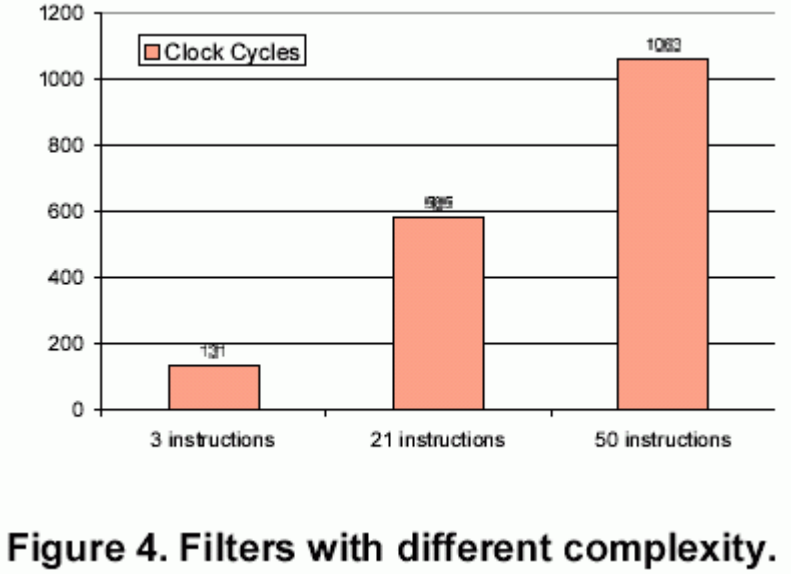


Figure 4. Filters with different complexity.

4.3.2 内存拷贝
正如第3节所解释的那样，数据包在到达用户程序前要经历两次存储器拷贝(图1):第一次，先从NIC缓冲区转
心缓存区(图2)。第二次则将它传输到用户应用程序缓存。图5展示了在测试平台的电脑上两次数据拷贝的时

及对应的数据包的大小。
根据NDIS规范，由函数NdisTransferData()实施并完成第一次传输。该函数的开销特别高，主要有两个原
1 .在拷贝过程前需要有一些附加的开销。考虑到NIC驱动程序在退出包驱动器控制权以后，整个数据包可
故根据DDK文档[7]，驱动器必须使用该功能函数。这样,该功能函数首先检查整个数据包是否已被NIC传输
果还没有,则等待直到传输完成。
2 .该功能函数所操作的数据对象并不在CPU缓存中。数据包通过控制总线从NIC板载存储器传输到主存(参
先前两点解释了图5所示的第一次拷贝高开销的原因。在处理过程中，存在有一个开销量底限基数的客观事
独立于处理数据量的大小(这就是为什么在数据包较小的情况下，处理每字节需要更高的代价)。而当数据包
时，这些开销平摊于每个字节，故每字节开销量要小一些。
第二次拷贝使用的是标准C库函数(例如memcpy())。由于待拷贝到用户区的数据可能大都不在于CPU的ca
故处理每字节的平均代价随着数据包的增大而增大，原因就是命中率下降了。另外处理每字节数据的开销
心层存储区的大小有关。例如,如果核心存储区容量较小，那么从NIC存储区拷贝数据时，会出现有部分数
在CPU cache中。
总结,第一次拷贝每个包的时钟周期开销在540到10500之间，第二次拷贝在259到8550之间变化。实际上,
个包在存入核心存储区前都要插入一个含有20个字节的帧头(包含时间戳、包长度等其它信息)，所以第二
总开销在364到8664之间。

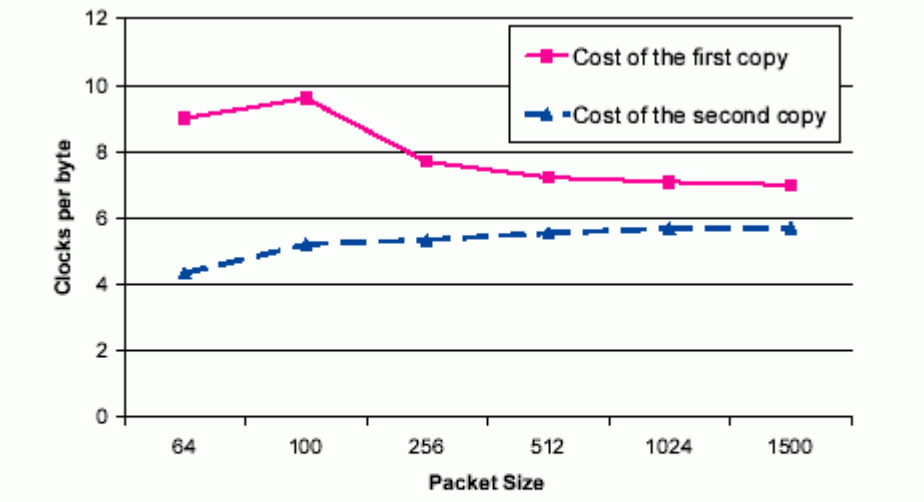


Figure 5. Cost per byte of memory copies with different packet sizes.

4.3.3 应用程序
所有应用程序与包驱动程序之间的交互作用都是通过系统调用来完成。Windows提供了 ReadFile()、Write
有DeviceIO-Control()等系统调用来完成I/O操作。所有这些系统调用都要完成两部分内容"context switches
使用context switch这个词其实并不太准确。实际上从用户层到核心层这一过程是个优先特权转闸，而非上
转闸):第一，将指令从用户层(应用程序)发送到核心层(驱动程序)。第二，返回用户应用程序的控制权。
众所周知,上下文转闸是相当复杂(它通常包含中断的产生和一些OS数据结构的初始化工作)并且开销很大的
在我们的测试平台上,类似read()这样的系统调用需要33500个时钟周期。如此高的开销！若每次系统调用只
个数据包，那可想效率多低了。因此包捕获器在每次应用程序系统调用时传输整块的数据包。每次系统调
的数据包数量的多少取决于核心层存储区的大小，并与CPU负载能力成正比。另外,它还取决于用户层应用
杂性(如果应用程序对数据包的处理时间很长，那它从核心存储区重载数据的时间间隔就大些，那么核心缓
据就没能及时“ 排干 ”)，而另一方面还受包捕获驱动器的影响(驱动器内的代码拥有较高的执行优先级,因
冲区常处于满载状态)。
由于读操作的频率，以及每次系统调用所处理的数据包的数量都是随机数(变化无常)，故无法对每个包处理
时钟开销作出明确描述。在一台过载的机器上(即CPU的使用率达到100%)，假定它接收到的是最小帧，同
动器在每次系统调用时传输256k个字节(该值为目前WinPcap上可选择的界值。)，这相当于3200个数据包
动程序后来附增加的20个字节的帧头)。在这种情况下,处理每个包的上下文转闸(context switch)近似代价为
钟周期,但这相比于我们接下来要讲的其它组件的开销，这甚至可以被忽略。
4.3.4 其它处理部件
尽管我们一般都下意识地认为：在数据包过滤和传输拷贝过程中，捕获驱动器需耗费的执行时间是最多的
为什么几乎所有文献都将注意力集中于它以达到增进性能的效果)。但通过我们的测试，揭示了其它一些关
获代价的关键因素。在它们中间，又以timestamp最为显著。
NPF驱动程序通过KeQueryPerformanceCounter()win32功能函数获得数据包的时间戳,它是唯一能提供微
的核心函数。该函数与系统晶振芯片相关联，故其开销十分巨大：在我们的测试平台上，大约需要1800个
(通过对若干单处理器机器进行测试，得到该值)。该函数要耗费若干个微秒才能返回一个带有微秒级精确
果，这不免有些荒谬，但事实正是如此。更有甚是,它必须持续运作以便在数据包到达时为之加上准确的时
额外的开销还包括与NDIS、与核心层的交互运作(交互运作大都需要使用系统调用函数，因此开销巨大)。
括管理(内存映射与解映)核心层缓冲区、在NPF中向数据包增创帧。总之，除去过滤和拷贝的开销，其它
个时钟周期的开销。
4.4 全部开销

图6以园盘切片形式形象展示了各相关组件的时钟开销。处理一个包的全部的费用是 5680 个钟周期。该测
生成的相关配置如下所示：流速为148Kfps,数据包大小为64个字节，适配器为3Com 3C996 Gigabit，过滤
令条数为21条。
显然，从图6可以看出，当数据包较小时，时间戳和NIC驱动的开销最大。又由于这两部分的开销大都取决
故进行软件优化意义不大。一些针对NIC驱动进行的小规模优化程序由于其开发商拒绝公开代码，使得它们
广泛的应用。然而在任何情况下,NIC驱动软件优化的效果远不如升级NIC卡上的芯片组。
最值得注意的是,图6说明了为什么大多数文献都将优化集中于拷贝和过滤这两个过程,以减少系统开销。但
其结果却相当有限：仅可能带来性能15%的提升。
在我们所做的性能分析中，大都使用体积较小的数据包。需要解释的是，这并不是因为受到那方面的限制,
说,众多网络分析工具(尤其像sniffers和network monitors)都只提取包的起始部分,例如前98个字节,剩下的部
弃。这正是我们在测试中考虑使用小数据包的原因。

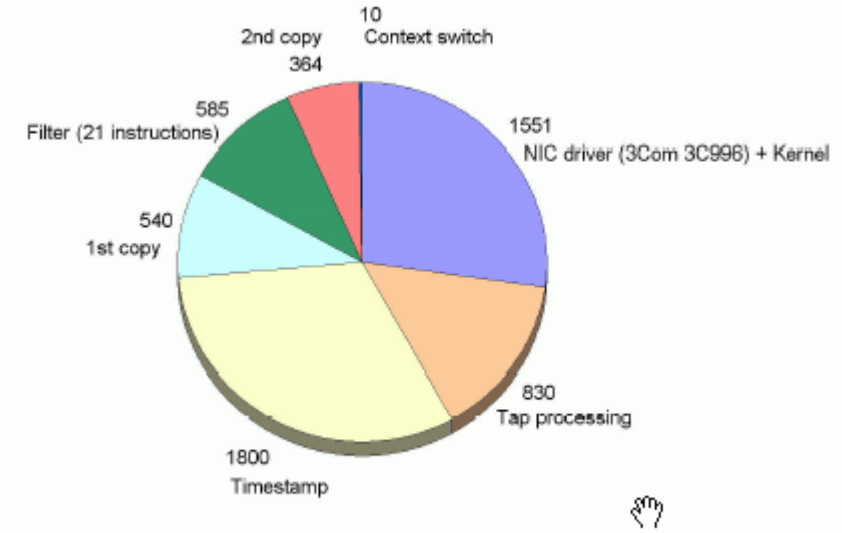


Figure 6. Details of CPU clock cycles.

4.5 对测试结果的附加说明
尽管我们的测试结果取自特定平台上(Win32)的特定工具(WinPcap),但结果还是具有普遍意义的。WinPcap
tap处理、第一及第二次拷贝、过滤)的开销类似于其它体系结构上的开销(例如,NPF与BPF相当类似)。同样
理，操作系统的相关开销也都相差不多：NIC驱动、时间戳timestamp、上下文转闸。NIC驱动的开销可以
些原先的软件功能在设计网卡时固化到芯片上的方式来减少，但这却需花费昂贵的成本。基于硬件的时间
效的优化方案之一:Endace's DAG卡的广泛使用[17]就是其中一个例子。基于Intel的硬件由于在设计x86时
芯片，故没能提供任何简单的方法来获得精确时间戳，而只能通过内部逻辑(例如，借助于CPU计数器)。更
是，精确定时是从8253/8254芯片(或其它类似芯片)中获取,然而因为它们需通过系统总线IN/OUT操作，故
相当慢的。
最后再提一点，可以忽略上下文转闸的影响，因为在不同的操作系统中其情况大都相差无几(例如,在调制解
作系统中，它被作为一项需谨慎设置的参数)。

5 .最优
针对先前几节内容讲述的这些突出瓶颈，本节将介绍并评估在NPF中所做的优化处理。
5.1 过滤部件
WinPcap使用的过滤系统是BSD包过滤器(BPF),1993[2]它被提议使用。当时文献记载的还有其它一些过滤
[10][11][12],但是在普通操作环境下，它们的性能根本无法与BPF相比。
在对BPF的优化解决方案中，动态代码的产生(即，将包过滤代码编译为CPU执行指令)确保了出色的性能[
[10]。因此,Just In Time(JIT)engine引擎被整合入NPF，通过它将BPF过滤代码编译成80x86的二进制代码
示，该优化措施带来了3.1到5的提升。这相当实现了对捕获机制总体性能8%的提升(假设为一个虚指令条数
21的过滤器)。

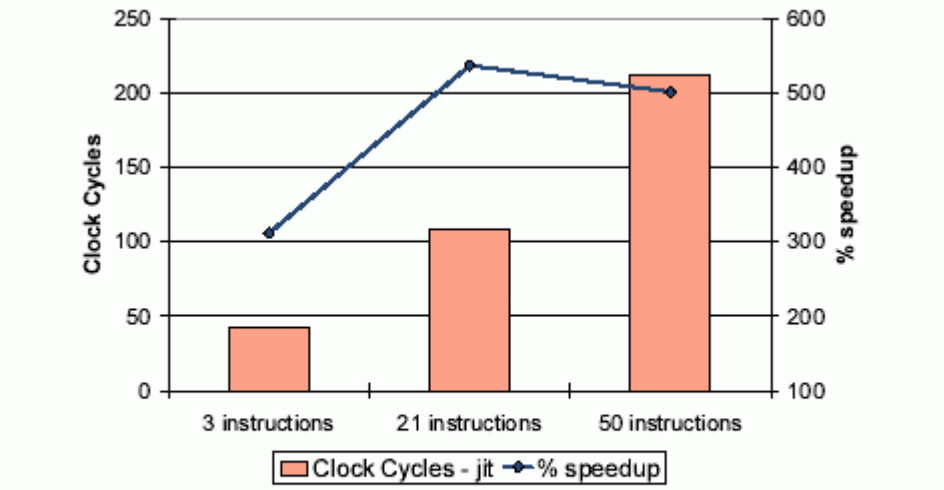


Figure 7. Cost and speedup of three filters with different complexity using JIT compilation.

5.2 存储器拷贝

先前已经说了，第一次数据包拷贝(从NIC存储器到核心缓冲区)的开销比第2次的开销来得大。原因是由于
外处理NdisTransferData()。然而,我们注意到几乎所有网络控制器(绝大多数网络适配器)在通报NIC驱动程
一个数据包完整地传输到存储器中,因此NPF驱动程序可在一个邻接的缓冲区中得到它。既然这样，我们放
法，而是通过一个标准C库函数来完成拷贝工作，结果如图8显示。由于数据包成块地传输，同时在CPU缓
较高的命中率，这样第2次拷贝的速度也有些许提高。这回两次拷贝的开销相差不多且具有相同的变化趋势

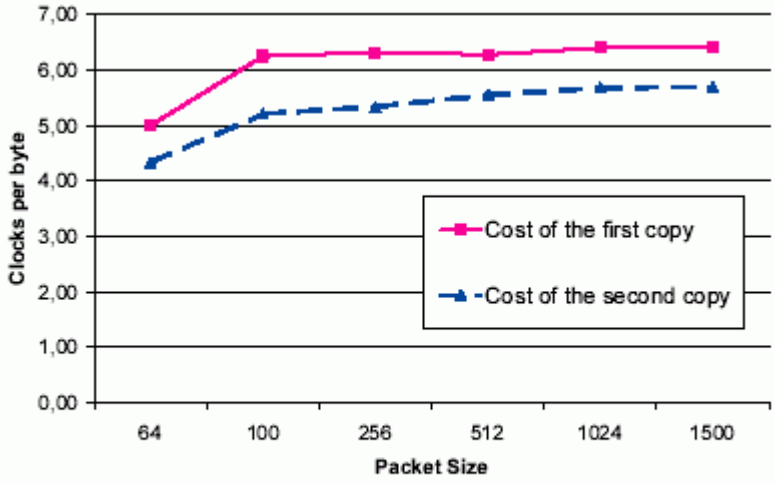


Figure 8. Cost per byte of memory copies.

感谢这一优化措施。在数据包大小为64byte的满载机器上，第1次拷贝过程的开销从540个时钟周期降到了
第二次拷贝过程的开销变化不大。这样，对于整个捕获机制带来了4%的性能提升。

5.3 时间戳

可以通过目前32位Intel处理器中的timestamp Counter(TSC)计数器来取代KeQueryPerformanceCounter()
来得到微秒级精确时间戳。该高效计数器在每个处理器时钟周期都完成一次自增,因此它的精密近乎CPU时
而且x86汇编器提供一个快速(仅一个周期)指令－rdtsc来得到精确时间戳。由于需要做两次64位的除法以便
转换到标准的struct timeval值，故使用rdtsc指令来获取timestamp需270 个时钟周期。该优化措施提速达到
应于对整个捕获机制得到27%的性能提升。然而,这项优化措施在NPF标准版上的缺省设置值是禁止，原因
特定的模式支持(rdtsc仅仅工作在Intel CPU及其兼容产品上,例如AMD速龙处理器)并且与处理器的速度(一
根据外部参数调整它们的频率,如电池强度)有关。不管怎样,这表明通过简单硬部件来获取时间戳可显著地
能。

5.4 优化tap()函数

使用标准C库存储器拷贝程序而非使用NdisTransferData()也可得到类似的tap()函数功能。如果使用
NdisTransferData()函数传输数据包需配置包含数据包的结构并在完成拷贝时预置回调函数。通过不使用
NdisTransferData()函数来避免这些步骤可显著增进性能表现。简化tap()处理可使捕获开销从830个时钟周
560个时钟周期，从而带来整个捕获机制总体性能5%的提升。

5.5 优化后的总体开销

图9描述了与图6条件相同的情况下，核心层仍然处理大小为64个字节的数据包，但实施了本解介绍的优化
各部件的开销状况。优化后的总开销为3164个时钟周期,即近似于未优化前开销的一半。

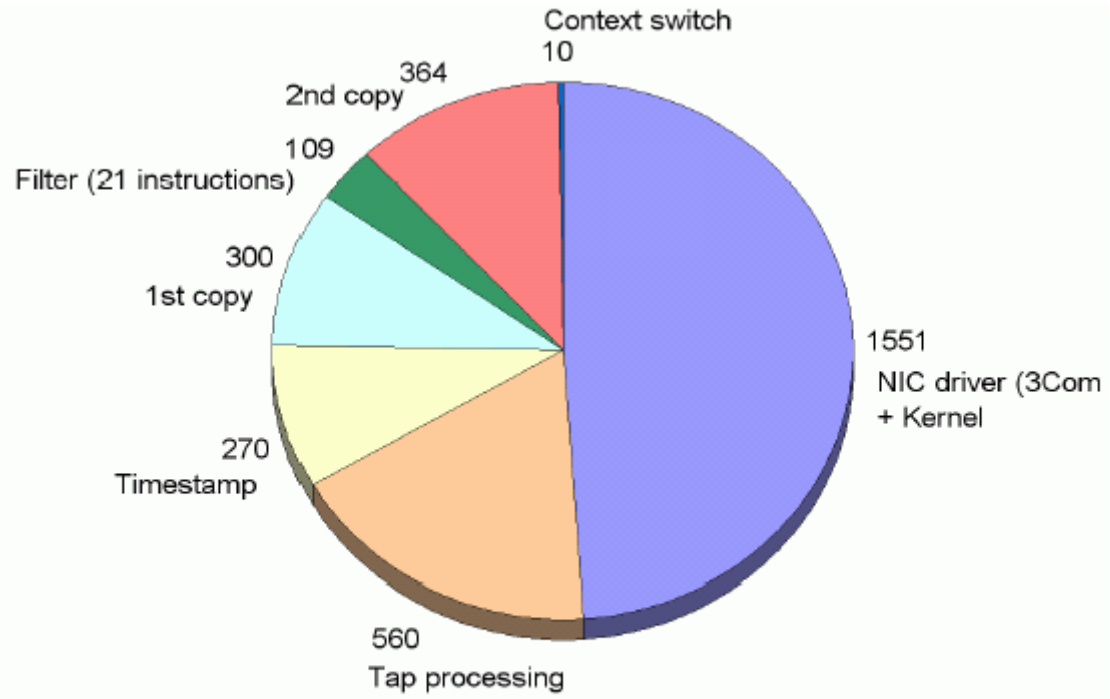


Figure 9. Details of CPU clock cycles.

值得注意的是，优化后49%的CPU时间用于NIC驱动程序和核心中断处理。事实上,这些两 部分的开销并不
措施的影响。而其它部件由于优化措施达到了2.6倍的提升。

5.6 可行的硬件加速

图9展示了大多数开销都源于包捕获组件以外的因素。一个新西兰本土公司提供了一个针对包捕获进行优化
卡:Endace[17]。他们的设备卡没有复杂的硬件优化，但却很好地解决了问题,例如，它们减少了操作系统的
销。起码地,这些设备卡为每个收到的数据包产生一个时间戳(通过硬件)，并将它们传输到系统存储器,这些
更智能的硬件来管理。得益于操作系统组织结构(例如由NIC驱动完成的大部分工作)及OS-native协议栈的开
用程序(在用户空间)绕过这些步骤可以直接获取所需的数据而无需通过其它中间层。

尽管我们没能得到任何直接的试验性数据(因这些设备卡仅仅工作在Linux上,而我们的试验平台是基于Win3
意识到那些设备卡的确减少了对包过滤处理的间接开销,但也附加了类如tap处理的额外开销(缓冲区的管理
是完全能由硬件完成，另外一些与核心层的交互工作还是比需的)。[18]证实的间接花费(在Linux上)是521个
期,通过一些优化措施可将其降至190个时钟周期(用DAG API直接代替pcap API),它与原始的系统相比得到
性能提升(29.9倍若进一步优化)；与只基于软件优化的系统相比得到6.1倍(16.7倍若进一步优化)的提升。如
的硬件支持为基于软件的包捕获及分析工具带来了光辉的前景。

6 .结论

本篇对基于软件的网络分析及监视工具(例如,嗅探器)进行测试，并描述试验结果。在试验中，我们以CPU
作为衡量包捕获机制各部分组件的执行开销。这项研究的一个颇有价值的定量结论：原先我们都普遍认为
冲这两部分性能的好坏对包捕获机制整体表现起决定性因素，然而事实并非如此。对这2个部件的优化一直
倍受重视,然而对它们的改进结果显示出这并未能带来包捕获机制整体开销的显著降低,尤其是在短包的情况
必须使用短字节映射时)。我们的测试在一个真实的系统上进行，结果显示出最大的瓶颈却隐蔽在暗处,如设
程序、应用程序和OS之间的交互作用、OS和硬件之间的交互作用。

由于包捕获机制的性能受多方面因素的影响,故优化措施因从整体考虑而非集中于单一的几个部件。本文指
些恰到好处的地方做一些力所能及的优化措施比试图重新更该系统结构的效果更加显著,并不像一些文献中
那样要将精力集中于包过滤和缓冲机制。另外本文还指出，基于软件的流量分析工具的表现能得到进一步提
管相关算法技术被认为已发展到了极限,比如借助硬件来获取时间戳、绕过操作系统间接开销。

所有有关该项测试的具体细节和数据都可以在WinPcap的主页上得到[8],同时还提供有相关文档、源代码及
文提及的大多数优化措施已在WinPcap的3.0版中实现。

参考书目

[1] V. Jacobson, C. Leres and S. McCanne, libpcap, Lawrence Berkeley Laboratory, Berkeley, CA. Initial public release June 1994. Available now at http://www.tcpdump.org/.

[2] S. McCanne and V. Jacobson, The BSD Packet Filter: A New Architecture for User-level Packet Capture. Proceedings of the 1993 Winter USENIX Technical Conference (San Diego, CA, Jan. 1993), USENIX.

[3] Fulvio Risso, Loris Degioanni, An Architecture for High Performance Network Analysis. Proceedings of the 6th IEEE Symposium on Computers and Communications (ISCC 2001), Hammamet, Tunisia, July 2001.

[4] Microsoft Corporation, 3Com Corporation, NDIS, Network Driver Interface Specification, May 1988.
[5] Intel Corporation, IA-32 Intel ~ Architecture Software Developer's Manual Volume 2: Instruction Set Reference, Order Number 245471-006.
[6] Intel Corporation, IA-32 Intel ~ Architecture Software Developer's Manual Volume 3: System Programming Guide, Order Number 245472-006.
[7] Microsoft Windows Driver Development Kits (DDKs), available at http ://www.micro so ft. com/ddk/.
[8] The NetGroup at Politecnico di Torino, WinPcap: WinPcap web site, http ://winpcap.politoit/. [9] M. Yu Bershad, C. Maeda, J.E.B. Moss, Efficient Packet Demultiplexing For Multiple Endpoints And Large Messages. In Proceedings of the 1994 Winter USENIX Technical Conference, pages 153-165, San Francisco, CA, January 1994.
[10] A. Begel, S. McCanne, Si. Graham, BPF+: Exploiting Global Data-flow Optimization in a Generalized Packet Filter Architecture, Proceedings of ACM SIGCOMM '99, pages 123-134, September 1999.
[11] Dawson R. Engler, and M. Frans Kaashoek, DPF: Fast, Flexible Packet Demultiplexing, in Proceedings of ACMSIGCOMM '96.
[12] Mary L. Bailey, Burra Gopal, Michael A. Pagels, and Larry L. Peterson. PATHFiNDER: A Pattern-Based Packet Classifier. In Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation, pages 115-123, Monterey, CA, November 1994.
[13] Marcus J. Ranum, Kent Landfield, Mike Stolarchuk, Mark Sienkiewicz, Andrew Lambeth, and Eric Wall (Network Flight Recorder, Inc.), Implementing a Generalized Tool for Network Monitoring, LISA 97, San Diego, CA, October 26-31, 1997.
[14] Jeffrey C. Mogul, Richard F. Rashid, Michael J. Accetta. The Packet Filter: An Efficient Mechanism for User-Level Network Code. In Proc. 11th Symposium on Operating Systems Principles, pages 39-51. Austin, Texas, November, 1987.
[15] Pankaj Gupta and Nick McKeown, Algorithms for Packet Classification, IEEE Network Special Issue, March/April 2001, vol. 15, no. 2, pp 24-32.
[16] Intel Vtune Performance Analyzer, Intel Corporation, 2003. Demo available at http ://developerintel.com/software/products/vtune/vpa/.
[17] Endace Measurement Systems, web site at http://www, endace.com.
[18] Luca Deri, nt op developer, private e-mail communications.

===CopyRight 2004 All Rights Reserved===博比出品 必属经典===

返回上页

Copyright © 2000-2004 **HUC** All Rights Reserved.
中国红客网络技术联盟 www.cnhonker.com
webmaster@cnhonker.net