维普资讯 http://www.cqvip.com

千兆网安全监测系统的性能分析

汪文勇　黄鹂声
（电子科技大学计算机学院　成都610054）

摘　要　网络安全监测是Internet/Intranet上的一个重要课题，如何能够在保证各种监测功能的前提下，尽量提高网络监测系统的性能，一直是一个难点。本文提出了一种基于PC机和Libpcap的纯软件网络监测系统模型，并分析其在千兆骨干网络上的实时监测性能。

关键词　千兆网，网络监测，时钟周期，流量分类，关键字检索

Performance Analyse of Gitabit Network Monitor System

WANG Wen-Yong　HUANG Li-Sheng
(School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054)

Abstract　Network Monitor is an important problem of Internet/Intranet. It's always difficult to improve the performance of a Monitor system with given functions. This paper brings forward a model of PC & Libpcap based pure software monitor system, and discusses it's performance.

Keywords　Gigabit network, Monitor, Clock cycle, Flow classification, Keyword search

1　引言

随着Internet的飞速发展，以Internet/Intranet为平台的各种应用越来越广泛和深入，网络的实时监测变得日益重要。对实时监测系统而言，人们最为关心的就是功能和性能，一般而言，网络实时监测具有如下功能和性能要求：

·支持千兆信道，峰值流量为600Mbps环境下实时采集网络报文，并且不丢包；

·具备流量统计功能，可以统计当前流量信息（pps，bps等）；

·基于统计的异常行为分析能力，使用的测度包括协议流量比重，协议平均包长、特定端口流量、特定地址流量、连接平均持续时间、连接平均持续长度、特征报文数量（如SYN报文和ACK报文数量和比重等），并且对这些测度进行计算；

·基于协议转换的异常行为分析能力，根据RFC相关标准建立应用协议的正常行为模型，发现异常的会话；

·具备协同分工能力，多个探测器可以分工完成对同一网段的监测；

·其他监测能力，如关键字实时检索等。

在千兆网上要完成以上功能和性能，目前比较流行的方案是硬件实现，采用专用硬件对网络进行实时监测和分析，该方案的优点是高性能，缺点是分析功能相对固定，难以定制。所以不少人把眼光转向了低成本、灵活性好的纯软件系统，一种有效的解决方案是采用低成本的PC来完成网络流量采集和分析。

文[2]提出了一种基于Libpcap（一种开放源码的抓包工具和开发库，可以运行于Windows、Linux等操作系统）的高速网络帧捕获系统构架，在未进行协议分析的情况下，其测试性能达到了50Mbyte/s。在此基础上，本文进一步提出了一种基于Libpcap的基本完整的网络监测系统框架，本文的重点在于分析、验证其性能。

2　纯软件网络监测系统构架设计

本文提出的网络安全监测系统是一个多点采集、集中审计的技术构架。系统由一台监测审计服务器和多台采集分析主机组成，如图1所示。

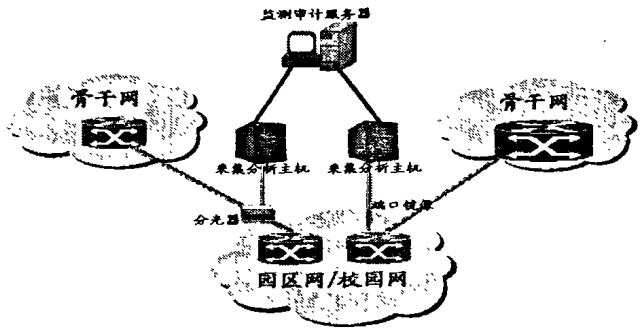

图1　网络安全监测系统构架

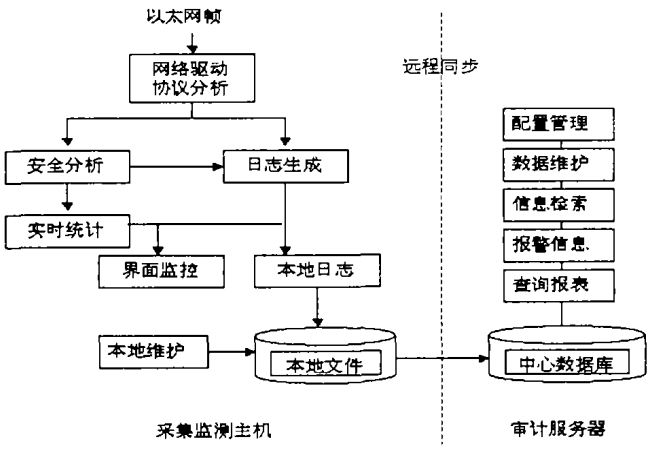

图2　网络安全监测系统软件体系结构

采集分析主机部署在网络主干或出口，可以通过分光器

维普资讯 http://www.cqvip.com

或端口镜像的方式捕获主要端口的网络流量，实时完成一系列本地协议解析和分析统计功能，记录安全监测日志，并将日志传递给监测审计服务器。监测审计服务器则只需要完成日志数据的采集、集中存储和分类汇总统计。系统的软件体系结构如图2所示。

很明显，网络监测系统的性能，完全取决于采集监测主机的硬件能力和软件性能，在固定 PC 硬件配置的情况下，则取决于该主机上运行的网络监测程序。

一般而言，基于 Libpcap 库的网络监测程序构架如图3所示。

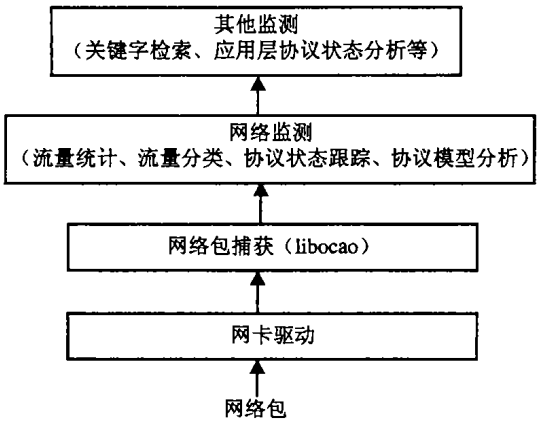

图3　网络监测程序构架

基于上述架构的监测系统，如果要达到千兆网络的实时监控性能，就必须能够处理600Mbps 的峰值流量。据文[3]的测量数据，互联网上的平均 IP 分组长度为404.5字节，这个长度还需要加上以太网帧头。为了便于计算，我们取稍坏一点的情况，假设以太网帧长度按照平均400字节计算。则监测系统对帧的处理性能需要达到187.5kfps，也就是说，每个帧的实时处理时间不能超过5.3μs。

假设每个包的处理时间为 Tpp，则计算公式为：

$$Tpp = Tcp + Tnm + Tom + Tap \quad (1)$$

其中，Tcp 为驱动捕获耗费时间，Tnm 为基本网络监测耗费时间，Tom 为其他监测耗费时间，Tap 为程序调度过程的其他开销。以上各项之和 $Tpp \leqslant 5.3\mu s$。

网络检测系统一般要求能够以相对低的成本进行多点部署，所以，我们选择普通 PC 服务器作为硬件平台来进行性能分析，试验服务器系统平台是2.4G CPU 的 PC 服务器，内存大小为2G，操作系统平台为 WindowsXP Professional，Libpcap 版本为3.0。

3　基于 Libpcap 的流量捕获性能分析

Libpcap 的抓包原理和过程，许多资料都有记载，这里不再赘述。据文[2]的描述，Libpcap 抓包过程的主要时间开销来源于几次调用过程和数据拷贝，如图4所示。

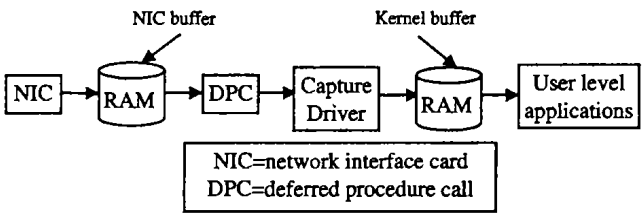

图4　Libpcap 的工作过程

据文[2]提供的试验数据，在普通的 PC 服务器和千兆网卡上，经过优化后的 Libpcap 每处理一个包大约需要3000个 CPU 时钟周期，如果采用主频为2.4G 的 CPU，可以达到800k fps 的抓包性能。平均以太网帧长取400字节，那么 Libpcap 理论上可以对2Gbps 以上的实际流量进行捕获而不丢包。每个包的平均处理时间为1.25μs。

为了验证 Libpcap 的性能，我们开发实现了一个基本的网络监测系统，仅仅完成数据包拷贝、简单协议分类和流量统计。经过24小时实际测试，该系统在网络流量为90～115MB/s 的稳定流量下工作正常，相关性能结果如表1所示。

表1　Libpcap 性能测试数据

参数指标	参数值
平均实际流量	808M bps
已验证抓包性能	135k fps
平均以太网帧长	598 byte
总 CPU 载荷	25%－30%
监测程序对 CPU 占用率	16%－20%
帧到达的平均时间间隔	7.4 μs

可以看出，实际流量没有使系统达到满负荷状态，按照 CPU 资源剩余情况，fps 还可以提高5倍，达到675k fps，单帧平均处理时间为1.48μs。

因此，可以推定：公式1中的驱动捕获耗费时间 $Tcp = 1.48\mu s$。

4　网络监测功能实现的性能分析

根据本文开始提出的网络监测功能需求，网络监测功能应包括流量统计、流量分类、协议状态跟踪、协议模型分析等功能。这些功能全部完成需要耗费的时间为 Tnm。计算公式为：

$$Tnm = Tsm + Tcl + Tpt \quad (2)$$

其中 Tsm 为流量统计耗时，Tcl 为流量分类耗时，Tpt 为协议状态跟踪和协议模型分析耗时。以下分别对这几项时间参数进行分析。

4.1　流量统计耗时 *Tsm*

由于 Libpcap 已经在内部实现了流量统计功能，因此本统计工作只需要读取 Libpcap 的数据即可，可以忽略此耗时因素。

4.2　流量分类耗时 *Tcl*

网络监测的一个重要环节就是流量分类。流量分类通常是指根据网络协议包头部的若干字段，把数据流划分为不同的类别，以便进行不同处理，如统计、跟踪等等。

目前存在不少流量分类算法，文[9]提出了一种递归流量分类(Recursive Flow Classification)算法，可以根据包头的 N 个字段，对数据包进行快速分类。E. W. Spitznagel 对该算法进行了改进，其基本思想是：根据实际需要的分类规则(如协议类型、IP 地址等)，预先建立分类索引表，通过多阶段映射，最终将网络包头部的若干关键字段组合映射为一个分类 ID，从而达到分类目的。该算法的特点是以空间开销换取时间开销，其时间复杂度为 O(1)，规则条数不会影响性能，只会增加空间开销。

根据文[9]给出的评测结果，在 PII-333的 PC 机上，采用 WindowsNT 操作系统，进行三阶段流分类，得到的平均查找耗时结果如表2所示。

维普资讯 http://www.cqvip.com

表2 流量分类算法的性能参数

Number of Rules in Classifler	Average Time per lookup(ns)
39	587
113	582
646	668
827	611
1112	733
1734	621

可以看出，该分类算法能够在0.8μs 以内实现对单个网络包的分类。

本文对上述算法进行了初步编码测试，在前述方正硬件平台上，采用500条分类规则，进行100万次分类查找，每个包的实际分类耗时略低于0.2μs。对于网络监测整体时间开销而言，该部分占的比重相当小，而且空间开销也不大(固定开销低于5MB)。

4.3 协议状态跟踪与模型分析耗时 *Tpt*

要跟踪传输层协议状态，必须对每一个 TCP 连接建立相应的数据结构(协议状态表)，用于记录其当前协议状态。一般而言，在流分类的基础上，协议状态表应当至少包括源 IP、源端口、目标 IP、目标端口、协议类型、双方序列号、协议状态等字段，每条记录应当至少占用20个字节，如果需要同时记录1000万条连接，那么需要200MB 的内存空间。

对于校园网和园区网络而言，IP 地址相对固定，假设有96个连续 C 类地址段，每个地址的每个端口都同时对外发起连接，则单机并发连接为64k，全网的对外并发连接数将达到15.98亿条，需要内存空间31G。

本文借鉴了流量分类的算法原理，在流量分类的基础上，改进了数据存储结构和查找算法，一方面避免巨大的内存浪费，另一方面尽可能提高网络连接相关数据结构的查找性能。算法的具体思想是：每个内部 IP 地址都可以通过简单的流量分类，格式化为0～32767之间的数值，用15位二进制数表示；每个 IP 地址的64k 个端口则可以通过模512运算，被格式化到0～511之间的数值，用9位二进制数表示；以上两个部分二进制拼接，每个网络包就可以通过内部 IP 和内部端口，被格式化到0～16,777,215之间的数字，通过该数字就可以直接访问哈希表对应的存储单元。当然这里存在冲突，有一部分网络包会计算得到相同的哈希值，那么需要用链表加以扩充，链表深度为64k/512＝128。也就是说，在最坏的情况下，一个网络包需要经过128次链表检索，才能找到它对应的数据结构。

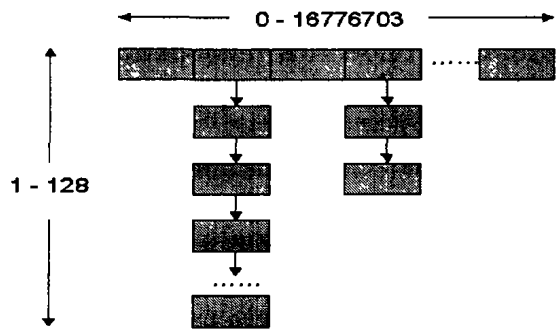

图5 协议状态跟踪的数据结构设计

因此，首先需要建立一个较大的固定存储空间，用于存放一个包含32767×512个单元的数组(哈希表)，每个单元存放一个链表的头指针，用于向下检索哈希值相同的网络连接。以上数据结构形成一个巨大的哈希链表数组，如图5所示。本算法的基本空间需求为0.3G，该空间可以管理园区网内所有主机的分别512个并发连接，最大空间需求则取决于实际活动网络连接数量。如果每台 PC 机都向外发出1000个并发连接，则算法空间需求也将低于0.7G。

该方法空间需求较大，但时间复杂度为 O(1)，查找速度很快，经过试验，时间消耗低于0.2μs。

在实际环境中，由于大部分操作系统的端口分配都是从1k 开始向上递增，因此小号端口的利用率相当高，如果单独为这些端口分配存储空间，而不格式化到512以内，则可以加快查找速度，在此本文不做更多讨论。

为了实现协议模型的分析，发现异常协议行为，还需要对上述协议状态进行实时检查，将该状态与正常协议模型进行比较。该比较过程时间复杂度也为 O(1)，时间耗费应低于0.1μs。

基于以上分析，我们可以得到公式2的计算结果：

$$Tnm = Tsm + Tcl + Tpt = 0\mu s + 0.2\mu s + 0.3\mu s = 0.5\mu s$$

5 其他应用监测功能实现的性能分析

有的情况下，为了对应用层协议进行实时监测，必须从报文数据中提取协议关键字，例如 HTTP 的“GET”请求和相应的 URL 地址。有时还需要针对部分敏感关键字，进行网络协议报文的实时文本检索。

本文采用了一种改进的 Boyer-Moore HorsPool 文本检索算法进行性能测试，该算法在文[10]中有详细描述，本文的改进在于：从尾往前检索关键字的时候，第一次比较过程同时比较两个字节，而不是仅比较一个字节，该算法的时间复杂度与 Boyer-Moore HorsPool 相同。通过对随机产生的网络报文进行多次对比检索，改进算法能够轻微改善检索性能(约0.5%～1%)。

对于一般应用层协议而言，300个字节的检索范围已经足以完成协议关键文本命令的匹配。因此，我们对300个字节的随机网络报文进行了不同关键字长度检索，检索次数为100万次，性能结果如表4所示。

表4 关键字检索性能测试结果

关键字长度 (Byte)	总消耗时间 (ms)	换算为流量 (bps)	平均每个包检索耗时(μs)
2	3255	0.73g	3.255
4	1853	1.29g	1.853
6	1112	2.15g	1.112
8	992	2.41g	0.992
10	861	2.78g	0.861
12	802	2.99g	0.802
14	751	3.19g	0.751
16	731	3.28g	0.731

可以看出，在较坏的情况下(每个报文都需要进行关键字检索)，针对8个字节关键字的平均检索时间低于1μs。其实在实际的骨干网络环境下，只需要对很少部分的数据包进行关键字检索。因此，我们可以得到公式1里面的 $Tom = 1\mu s$。

6 网络监测系统整体性能分析

根据上述实验结果，基于 PC 机和 Libpcap 的网络监测系

(下转第72页)

维普资讯 http://www.cqvip.com

评判参数，来表示该描述的优先级别。在客户端程序中增加rank授予级别函数，根据该对象所属的类别和重要程度，来授予并存储它的优先级别。对在第3节中提出的插入过程进行拓展，同样该过程可以是系统自动实现的，也可以是通过手工进行修订。在查询过程中，首先分析该查询，计算不同查询方案的优先级，选择代价最小的方案来执行。比如在上例中，考虑到作家的作品数量少于出版社出版的书籍的数量，插入该对象时，"出版社"的描述就要比"作家"的描述具有较低的优先级，从而使得前一种查询执行方案将比后一种查询执行方案好，在查询执行时应该选择前一种方案。

用户在进行数据查询时，以类似于QBE查询语言的方式给出查询请求，这个查询请求随后被转换成多个XPath查询表达式求值。

总结 现有的基于DHT的P2P系统在查询时只能对码进行精确匹配，因而它们的查询能力有限。为了弥补这一缺陷，本文提出了一个系统模型。该系统模型在应用程序和DHT之间加入一个数据管理层，由它负责以适当的方式存储数据，并根据数据描述建立DHT索引和处理查询请求。

将数据和对数据的描述区分开，极大地扩展了系统能够处理的数据类型。数据描述本身是一个XML文档，可以将这个XML文档转换成树结构表示。本文提出的索引方法是为树中的每一个叶子结点建立两个DHT索引项。如果索引项的码是对应该叶子结点的XPath查询，那么值将是存储数据的网络节点的IP地址；如果索引项的码是对应该叶子结点的父结点的XPath查询，值将是对应该叶子结点的XPath查询。

查询的时候，查询请求将被转化为多个XPath查询表达式。对于一个XPath查询表达式，先在DHT中对整个查询表达式做精确匹配，如果没有结果返回，就去掉这个查询表达式的最后一个定位步骤生成其父查询表达式，然后在DHT中对父查询表达式进行精确匹配，最后再在父查询表达式的返回结果中进行查找。多个XPath查询表达式可以通过逻辑运算构成复杂的查询，对于这种复杂的查询的执行，系统需要进行优化，以便采用最好的执行方案。

在今后进一步的研究工作中，首先要考虑的是对本文提出的方法进行实验分析，进一步验证它的性能。其次，本文提出的方法还有待进一步完善：如如何有效地进行数据的修改和删除；又如还需要进一步讨论进行复杂查询时的进行查询优化的方法，评价它们的执行效率。

参考文献

1 Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A scalable content-addressable network. In: Proc. of ACM SIGCOMM, 2001
2 Stocia I, Morris R, Karger, Kaashoek M, Balakrishnan H. Chord: A scalable peer-to-peer lookup services for internet applications. In: Proc. of the ACM SIGCOMM, 2001
3 Rowstron A, Druschel P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Proc. of Middleware, 2001
4 Zhao Y B, Kubiatowicz, Joseph A. Tapestry: An infrastructure for fault-tolerant wide-area location and routing: [Technical Report UCB/CSD-01-1141]. University of California at Berkeley, 2001
5 Harren M, Hellerstein J, Huebsch R, Loo B, Shenker S, Stoica I. Complex queries in DHT-based peer-to-peer networks. In: Proc. of IPTPS, 2002
6 Gupta A, Agrawal D, Abbadi A. Approximate range selection queries in peer-to-peer systems: [Technical Report UCSB/CSD-2002-23]. University of California at Santa Barbara, 2002

（上接第56页）

统对单个数据包需要完成从抓包、流量统计、流量分类、协议状态跟踪直到内容解码和关键字检索等一系列串行工作，耗时计算公式为 $Tpp = Tcp + Tnm + Tom + Tap$。其中 $Tcp \leqslant 1.25\mu s$，$Tnm \leqslant 0.5\mu s$，$Tom \leqslant 1\mu s$，在要求 $T \leqslant 5.3\mu s$ 的前提下，还剩余大约2.27μs可以供 Tap 消耗。而 Tap 期间内的主要工作是程序模块之间的调度和数据交互。

Tap 的一个重要时间开销在于操作系统由核心态(Kernel Mode)转为用户态(User Mode)所需要耗费的时间。由于在本实验环境下，Libpcap工作于操作系统的核心态，而其他程序模块运行在用户态，因此操作系统必须频繁地在核心态和用户态之间切换，每来回切换一次大约需要1200个CPU时钟周期，本例中换算成时间就是 $(1/2.4G) * 600 = 0.5\mu s$。因此，还剩余1.77μs可以用于监测程序的其他消耗。

系统的整体时间开销按照各个功能阶段划分如表5所示。

表5 网络监测系统的时间耗费分配表

功能	消耗时间	耗时比例
Libpcap	1.48μs	28%
基本网络监测	0.5μs	10%
应用监测	1μs	20%
调度开销和其余杂项开销	2.27μs	42%

为了验证系统的整体性能，我们将以上功能纳入统一项目，编译完成一个基本的网络监测系统，并在800Mbps的峰值流量下进行了整体连续运行和性能观察，结果程序运行正常，掉包率低于0.2%，CPU利用率稳定在75%～80%。

结论 经过分析和实验，证明了普通PC服务器完成千兆网络监测是完全可行的。当然本文提出的网络监测功能不多，也没有考虑监控和日志功能。随着网络带宽的不断增加，对实时监测的性能要求还会不断提高。本文提出的检测技术构架也存在很多可以改进的地方，例如：Libpcap本身的timestamp功能需要花费大量时间，可以进行精简；如果把监测系统程序全部放入操作系统核心态运行，也会大大改善系统性能。这些都是有待进一步研究和开发的课题。

参考文献

1 Baboescu F, Varghese G. Scalable packet classification. In: Proc. of ACM Sigcomm '01, Sep. 2001
2 Degioanni L, et al. Profiling and Optimization of Software-Based Network-Analysis Application
3 Agilent Technologies. JTC 003 Mixed Packet Size Throughput. http://advanced. comms. agilent. com/n2x/docs/journal/JTC_003. html
4 Buddhikot M M, Suri S, Waldvogel M. Space decomposition techniques for fast layer-4 switching. In: Proc. of PHSN, Aug. 1999
5 Eatherton W. Hardware-based internet protocol prefix lookups: [MS Thesis]. Washington University in St. Louis, Electrical Engineering Department, May 1999
6 Feldman A, Muthukrishnan S. Tradeofes for packet Classification. In: Proc. of Infocomm, March 2000
7 Gupta P, McKeown N. Packet Classification on Multiple Fields. In: Proc. ACM Sigcomm'99, Sept. 1999
8 Gupta P, Lin S, McKeown N. Routing lookups in hardware at memory access speeds. In: Proc. of the Conf. on Computer Communications (IEEE INFOCOMM), San Francisco, California March/April 1998, 3: 1241～1248
9 Gupta P, McKeown N. Packet Classification using hierarchical intelligent cuttings. In: Proc. of Hot Interconnects VII, Aug. 1999
10 Horspool R N. Practical fast searching in strings. Software - Practice & Experience, 1980, 10(6): 501～506
11 Baezz-Yates R A, Regnier, M. Average running time of the Boyer-Moore-Horspool algorithm. Theoretical Computer Science, 1992, 92(1): 19～31
12 Srinivasan V, Varghese G. Fast IP Lookups using Controlled Prefix Expansion. In: Proc. ACM Sigmetrics, June 1998
13 Srinivasan V, Suri S, Varghese G. Packet Classification using tuple space search. In: Proc. of ACM Sigcomm '99, Sept. 1999
14 Packet Classification Repository. http://www. cs. ucsd. edu/~baboescu/repository/
15 WinPcap Documentation. Available at: http://winpcap. polito. it/docs