786818

分类号 TP393.08　　学校代号 10561
UDC　　密级　　学　号 200235300201018

华南理工大学硕士学位论文

基于网络处理器的入侵检测系统预处理子系统的设计与实现

滑国青

指导教师：张凌
计算机科学与工程学院
申请学位级别：硕士　　专业名称：计算机系统结构
论文提交日期：2005 年 5 月　　论文答辩日期：2005 年 6 月
学位授予单位和日期：华南理工大学
答辩委员会主席：
论文评阅人：

摘 要

随着现代网络技术的飞速发展，千兆局域网解决方案已经成为一种潮流。因此，人们迫切需要速度更快、功能更好、性能更强的入侵检测系统，来适应在高速环境下的网络安全防护。

本课题研究基于 IXP2400 网络处理器的网络型入侵检测系统。整个系统被分成预处理子系统，规则检测子系统，告警子系统三大部分。作者在本课题中负责预处理子系统的设计与实现。预处理子系统由报文解码，IP 分片重组，端口扫描，TCP 协议分析，应用协议分析等模块构成。

本论文首先对 IXP2400 网络处理器技术作简要介绍；然后提出了 NP－NIDS 预处理子系统整体设计方案；接着详细探讨了报文解码模块，IP 分片重组模块，端口扫描检测模块的设计与实现方案；最后简要讨论了其它预处理功能，并对下一步工作进行了展望。

解码模块在微引擎中实现，文中对解码数据结构的设计作了详细分析，设计出的解码结构只有 5 个长字，比起 SNORT 的 packet 结构可以说有了质上的不同，非常适合微引擎处理，通过测试，基本实现了 1G 比特的报文解码功能。

IP 分片重组模块的实现分 CC 和 MB 两部分，CC 部分用于完成正常的分片重组及告警；MB 部分主要完成自保机制，两者有机配合，可以抵抗 IP 分片洪流的攻击。本文对此模块的关键点，即分片存储和重组，作了详细探讨，并提出了分区合并分片重组算法，此方法相对于 SNORT 的分片重组算法,大大降低了对内存的需求。

端口扫描模块在微引擎中实现，设计上使用了按升序排列关键字外加设标志位的办法，合理的解决了双关键字问题；采用超大 hash 表来作为连接结点数据库和源 IP 数据库，使得查寻结点在几次内就可以完成；对过时结点清除采用了多队列超时方法，使清除结点基本只与结点数量有关；对结点分配管理采用了堆栈式管理，这种管理看似浪费内存，但因为很方便缓冲到 ME 的寄存器内，所以结点分配效率极高。

关键词 网络处理器(NP)；IXP2400；入侵检测(IDS)

Abstract

With the rapid development of network technology,high speed LANs become more and more popular in enterprises.Safeguard of high speed network needs high performance NIDS.

The subject research network instruction detection system (NIDS) based on Intel network processor IXP2400. It has been divided into three subsystem to implement,which are preprocess subsystem,rule detection subsystem,alert subsystem.This paper,named my research work, fall into preprocess subsystem.The preprocess subsystem mainly include packet decode module, ip defragment module,port scan detection module,tcp analyzing module,application analyzing modules like http, telnet,rpc protocol etc.

This paper introduced the technology of IXP2400 in chapter 2. Then a whole design of the preprocess subsystem is given in chapter 3. The following three chapters, which are the main parts of the paper, described the design and implementation of packet decode module, ip defrag module and port scan detection module.

About packet decode module, we designed a special data structure for storing packet decoded information, which is only 5 longwords in size and well fit for process in ME.

We described the store and defragment of ip fragmentation in detail.The method used in this module for store and defragment can save a lot of memory compared with SNORT's method.

We solved the double key problem perfectly through ordering the connection key and using a flag in the same time.We used the super large hash tables, stack-style node management and multi-queue timeout methods to solve the store and manipulation of port scan statistic information.

Keywords network processor (NP); IXP2400; instrusion detection system (IDS)

华南理工大学

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容,本论文不包含任何其他个人或集体已经发表或撰写的成果作品。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律后果由本人承担。

作者签名：滑国青　　日期：2005 年 6 月 3 日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华南理工大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密□，在____年解密后适用本授权书。

本学位论文属于

不保密☑。

（请在以上相应方框内打“√”）

作者签名：滑国青　　日期：2005 年 6 月 3 日

导师签名：张波　　日期：2005 年 6 月 3 日

第一章 绪论

随着现代网络技术的飞速发展和核心交换技术的不断提高，局域千兆网解决方案已经成为一种潮流。用户的网络环境以及对入侵检测的安全需求发生了很大的变化，表现在以下几个方面：

- 用户的网络设备向高速化方向发展，如千兆网络交换设备、千兆网络服务器、千兆网络防火墙等等，传统的网络入侵检测产品无法适应现有的网络结构。

- 用户希望对入侵事件的发现越早越好，因此需要在主干网络上布置入侵检测系统。在高速流量环境下，现有的入侵检测系统对网络信息包捕获能力成为一个瓶颈。

- 在高速的流量环境下，用户对网络入侵识别的确定性提出了更高的要求，既不能漏过重要的攻击事件，也不能出现大量纷繁复杂的报警而使管理员要花费大量时间来从中寻找与重点防护资产有关的问题。因此对入侵检测系统的信息分析能力和策略控制能力又是一个重要的挑战。

- 近年来入侵技术无论从规模与方法上都发生了变化，入侵的手段与技术也有了“进步与发展”。只有提供高性能的入侵检测系统，才能有效降低系统的漏报率和误报率，同时提高入侵检测系统自身的抗攻击能力。

因此，人们迫切需要速度更快、功能更好、性能更强的入侵检测系统，来适应在高速环境下的网络安全防护。如何既为用户提供高性能的网络入侵检测和响应，同时又保证系统的可扩展性、可升级性、可维护性和低成本，是长期以来安全界面临的一个主要的课题和发展方向。

1.1 入侵检测系统概述

1.1.1 什么是入侵检测

入侵检测是继“防火墙”、“信息加密”等传统安全保护方法之后的新一代安全保障技术。它监视计算机系统或网络中发生的事件，并对它们进行分析，以寻

找危及机密性、完整性、可用性或绕过安全机制的入侵行为。入侵检测系统（IDS）就是自动执行这种监视和分析过程的软件或硬件产品。

1.1.2 入侵检测系统的作用

入侵检测系统作为一种积极主动的安全防护工具，提供了对内部攻击、外部攻击和误操作的实时防护，在计算机网络和系统受到危害之前进行报警、拦截和响应。它具有以下主要作用：

- 通过检测和记录网络中的安全违规行为，惩罚网络犯罪，防止网络入侵事件的发生；
- 检测其它安全措施未能阻止的攻击或安全违规行为；
- 检测黑客在攻击前的探测行为，预先给管理员发出警报；
- 报告计算机系统或网络中存在的安全威胁；
- 提供有关攻击的信息，帮助管理员诊断网络中存在的安全弱点，利于其进行修补；
- 在大型、复杂的计算机网络中布置入侵检测系统，可以显著提高网络安全管理的质量。

1.1.3 入侵检测系统的主要类型

按数据来源的不同，可以将入侵检测系统分为基于网络的入侵检测和基于主机的入侵检测。

（1） 基于主机的入侵检测系统

基于主机的入侵检测系统通常是安装在被保护的主机上，主要是对该主机的网络实时连接以及系统审计日志进行分析和检查，当发现可疑行为和安全违规事件时，系统就会向管理员报警，以便采取措施。

（2） 基于网络的入侵检测系统

基于网络的入侵检测系统一般安装在需要保护的网段中，实时监视网段中传输的各种数据包，并对这些数据包进行分析和检测。如果发现入侵行为或可疑事件，入侵检测系统就会发出警报甚至切断网络连接。基于网络的入侵检测系统如同网络中的摄像机，只要在一个网络中安放一台或多台入侵检测引擎，就可以监视整个网络的运行情况，在黑客攻击造成破坏之前，预先发出警报。基于网络的入侵检测系统自成体系，它的运行不会给原系统和网络增加负担。

1.2 高速入侵检测技术的现状与发展趋势

目前主流 NIDS 厂商推出了能够适应千兆环境的高速 IDS 系统，在技术实现路线上主要有以下几点：

1. 采用协议分析融合模式匹配的检测方式

特征模式匹配虽然是主要技术，但存在速度慢，效率低等缺点，协议分析是新一代 IDS 探测攻击手法的主要技术，它利用网络协议的高度规则性快速探测攻击的存在。协议分析技术优势在于能够详细解析各种协议，如命令字符串解析器能对不同的高层协议——如 Telnet、FTP、HTTP、SMTP、SNMP、DNS 等的用户命令进行详细的分析。探测碎片攻击和协议确认技术可以因为系统在每一层上都沿着协议栈向上解码，从而使用所有当前已知协议信息，来排除所有异常协议结构的攻击，同时大大降低传统模式匹配带来的误报问题和提高了效率，同时减少了系统资源消耗。

新一代 IDS 将协议分析和模式匹配技术融合起来，更完善得满足高速用户的应用需求，成为千兆 IDS 发展的主流趋势。

2. 负载均衡技术的使用

负载均衡是一项具备智能化的技术，通常是通过并行的几台设备，将处理流量尽量平均的分配到各个设备，使得总体的处理负荷能够“分摊”到各个设备中，从而使总体性能得到很大的提高。目前千兆 IDS 通常采用了基于中央控制下并行和分布式共存的负载均衡设计，在高速网络环境中通过多台检测器进行分析，从而能够适应更大的网络流量。然而负载均衡技术遇到的挑战是如何解决因攻击信息被分流，而造成无法识别特征造成漏报等问题。

3. 零拷贝方式的报文捕获

基于传统的处理网络数据包的方式，由于网卡驱动程序运行在内核空间，当网卡收到包以后，包会存放在内核空间内，由于上层应用运行在用户空间，无法直接访问内核空间，因此要通过系统调用将网卡中的数据包拷贝到上层应用系统中，占用系统资源造成 IDS 性能下降。而改进后的网络包处理方式，通过重写网卡驱动，使得网卡驱动与上层系统共享一块内存区域，网卡从网络上捕获到的数据包直接传递给入侵检测系统，这个过程避免了数据的内存拷贝，不需要占用 CPU 资源，最大程度的将有限的 CPU 资源让给协议分析和模式匹配等进程去利用，提高了整体性能。但零拷贝只能缓解“抓包”的瓶颈问题，千兆的实现仍要依靠 IDS 的协议分析和匹配检测等其他功能模块性能进一步加强。

4. 优化系统和充分利用硬件平台的运算优势

高速 IDS 目前大多采用通用 CPU 架构来提高性能，如文献[10]中评测的千兆 NIDS,无论是安氏 Link Trust GIGA-HA，还是福建海峡黑盾 1000e、绿盟科技冰之

眼 3.5 版、天融信 NGIDS-UF，青一色采用了多个通用 CPU 架构。采用 NP 技术，使得系统线程并行化，能同时对多个数据包进行分析和处理，此项研究在 IDS 领域正在升温，有些研究机构已经对 NP 在 IDS 应用有所成果。当然也有采用 ASIC 防火墙构架，通过优化核心运算模块，采用汇编语言对底层硬件直接编程，充分利用硬件资源的高效性，这种方法虽然很大程度上提高入侵检测系统的处理性能，但突出缺点是开发成本高，升级困难。

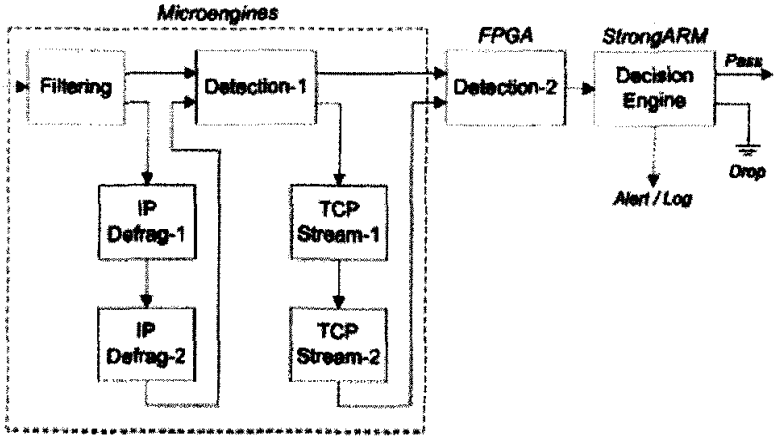
1.3 基于 NP 的 NIDS 预处理功能设计与实现的现状

商业上比较成熟的 NIDS 大都同时采用基于规则的入侵检测技术和基于异常的入侵检测技术。 SNORT[1]将入侵检测引擎分成报文解码，预处理和规则检测三部分。报文解码部分是整个检测引擎的基础，它为其它功能提供必要的报头信息；规则检测功能部分，完成基于 snort rules 的入侵检测功能；预处理部分，一方面完成基于异常的入侵检测（如端口扫描检测，ip 分片攻击检测等），另一方面要为规则检测功能提供必要的支持，如 ip 分片重组，Tcp 流重组，http 协议规整等）。SNORT 将预处理的各种功能模块以插件的形式实现，各种插件相对独立，每种插件都可以设置一些控制参数，这种方式，就象搭积木一样，对插件实现越多，系统功能越强大，值得借鉴。SNORT 软件目前支持 linux,windows 等操作系统，运行在通用 CPU 之上，这使得它的处理能力有限。文献[18]对 SNORT 的抓包及规则检测算法作了改进，并对改进后的性能作了测试，结果表明，处理流量仍无法达到 100Mb。文献[6]对 Snort2.0 预处理模块的设计与实现原理作了简要分析。

本文中所指的“预处理”，不但包括 snort 的预处理功能，还包括报文解码功能，因为作者认为，报文解码功能划入预处理更合理。

文献[16]将预处理的报文解码，ip 分片重组，tcp 流重组功能及规则检测的简单功能放在一块 ixp1200 中实现，规则检测则放在 FPGA 实现，其性能达到 100M。下图为其设计图。

图表 1-1 设计的检测引擎逻辑图



Janani Thanigachalam[11]试图在IXP2400(enp2611板)上实现Bro ids系统，从文中介绍可知，它的预处理只实现了一个报文解码功能和TCP会话状态维护功能，它将整个入侵检测功能放在一个ME上实现，采用ordered线程模型，文中没有介绍系统性能，但可以推测，这个系统根本无法达到线速处理流量，因为ordered线程模型虽然可以解决线程同步问题，但它没有发挥出ME线程的并发优势，ME的使用效率比较低，另外，一个ME，若只处理报文解码还可以达线速，但要进行TCP状态维护，就必然要不断进行内存操作，这样，一个ME根本无法达到线速，若要再加上规则检测，就更谈不上线速了。

文献[12]将snort1.9在(IXP2400+ IDT_PAX.port* 2500 CIE)硬件平台上实现，其性能是在250000pps的TCP背景流量下，加载400条规则没有发生丢包现象，文中提到，检测功能没有实现TCP会话状态维护和ip分片重组,tcp流重组功能及http应用协议分析。可以推测出，该系统针对预处理功能也只实现了报文解码。

文献[13]decsystant研究人员声称它们在IXP2400+TCAM硬件环境下实现了snort-nids,可以线速处理2.5G流量，但实际上，它们只实现了报文解码和规则检测。文中说，TCAM用于规则检测，snort功能部分只使用一个ME，但我们对它们的结果表示怀疑，因为snort在规则检测时，ME要有大量的IO操作，这会使得ME无法全速工作，更何况，背景流量报文的负载长度会严重影响性能，最长和最短情况，光报文数量就有十几倍的差别。

文献[14] Herbert Bos,Kaiming Huan在基于ixp1200基础上实现了一个简单的snort入侵检测系统，它也只实现了报文解码和规则检测，检测引擎采用了5个ME。文献[15]中分析了NP对snort模式匹配，及tcp flag,same ip的加速作用。

经过对基于 NP 的 NIDS 的文献检索，发现基于网络处理器的 NIDS 研究还是比较少的（目前高速 NIDS 多采用多个通用 CPU 架构），这些文献中介绍的系统大都只实现了报文解码和规则检测，而且都重点分析规则检测算法在 NP 上的实现，对预处理的设计与实现介绍基本没有。

1.4 论文来源和论文的工作

总得来说，高速 NIDS 必须采用高速的硬件平台和高效的检测引擎，而这两者只有有机结合起来，才能产生出高速优良的 NIDS 系统。所以我们不光要对采用的硬件平台，所用的检测引擎等核心技术进行研究，更要对两者如何有效的结合进行好好的研究。这也将是本课题的研究重心。

本课题是研究基于 IXP2400 的网络型入侵检测系统(主要是 sensor 部分，简称 NP－NIDS)，课题被分成了 4 部分，分别由林若川负责规则检测功能，由吴静波负责规则解析配置功能，由窦小龙负责告警功能，本人则完成预处理子系统的设计和实现。本课题的最终目的一方面是掌握 IXP2400 的开发技术，另一方面是对 NIDS 的关键技术作较深的研究。

本论文是对基于 NP 的 NIDS 预处理子系统研究成果的总结。文中的“线速”是指 enp2611 千兆以太口的最高处理速度，即 1G 比特。

本论文结构安排如下：

第二章对 IXP2400 网络处理器技术作简要介绍；第三章给出 NP－NIDS 预处理子系统设计方案；在接下来的第四、五、六章，重点探讨预处理子系统中三个模块的设计与实现，它们分别是解码模块，IP 分片重组模块和端口扫描检测模块；其余的模块由于模块本身的复杂性和课题时间的紧张，只在第七章中作以简单探讨，具体设计与实现将在下一阶段工作中进行。最后在第八章给出整个论文的总结，并对进一步工作作以展望。

第二章 IXP2400 网络处理器技术

随着当今网络规模和性能的迅速增长，Internet 主干网络流量的指数性增长及新业务接连的出现，这就要求网络设备具有线速和智能的处理能力。网络处理器 NP（Network Processor）便是一种新兴、有效的统一解决方案。它适用于各层网络处理，具有 ASIC 高速处理性能和可编程特性，既能保证系统灵活性，又能完成线速处理数据包所要求的高性能硬件功能。

由于网络处理器特殊的体系结构，它的软件编程模型与传统网络应用/嵌入式应用开发有较大不同，更为复杂。在一般程序设计中，可以不考虑操作系统和编译程序、线程调度的细节、寄存器的数量和容量，而在网络处理器的程序设计中，忽略这些因素就不能编写出优化的程序。

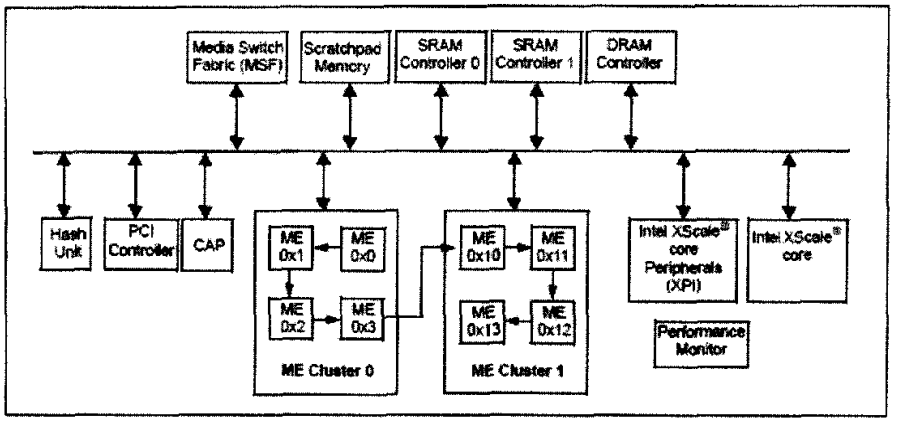
本课题是基于 intel 的 IXP2400 网络处理器进行开发的，所以下面将对 IXP2400 网络处理器技术分别从硬件系统结构，软件系统结构和开发环境作以简要介绍。

2.1 硬件体系结构简介

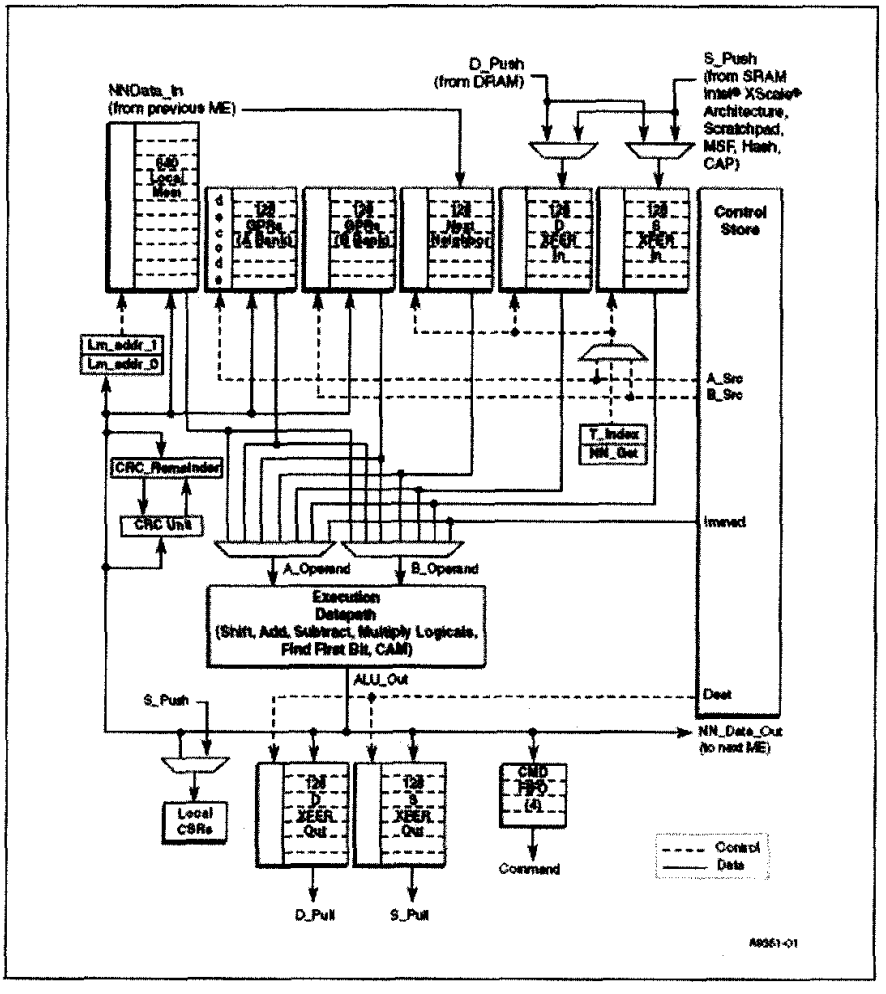
IXP2400 网络处理器是 Intel 在 2002 年推出的第二代互联网交换架构（IXA）网络处理器。它的体系结构充分体现了 SoC(System on Chip)的思想。

图表 2－1 是 IXP2400 的硬件体系结构逻辑图，图表 2－2 是 ME 的逻辑结构图 。从图中可以看出，IXP2400 共有 9 个 CPU，其中 intel Xscale core 是一个通用的 32 位 RISC 处理器 CPU，它在 IXP2400 中起控制，管理和一般数据处理作用，其上运行操作系统，如 linux、vxwork 操作系统；其余 8 个 CPU 被称为微引擎，它们被分成两个 cluster，每个 cluster 含有 4 个微引擎。微引擎执行快速通路的任务，并负责了大部分的报文处理工作。每个微引擎都可以访问所有共享的资源（譬如 SRAM，DRAM，MSF 等），还可以利用 Scratchpad 或者 Next Neighbors 进行引擎间的通信。

图表 2-1 IXP2400 硬件体系结构逻辑图



图表 2-2 Microengine 结构逻辑图



IXP2400 的内存体系是多级的，访问内存由慢到快依次是 sdram,sram,scratchpad,local memory,除了 local memory 只能被所在的 ME 访问，其它内存类型可被 xscale 和 ME 共享访问，此外 ME 内部还有多种不同功能的寄存器组，它们分别是通用寄存器组，sram 传输寄存器组，dram 传输寄存器组，next-neighbor 寄存器组。它们都对程序员可见，这种多级内存体系结构，使得程序开发人员能对 IXP2400 作非常精细的控制，为编写高效的代码提供了基础，但也正是这种灵活性，使程序开发人员比普通编程要付出更大的努力。图表 2－3 和图表 2－4 便是多级内存体系和各种寄存器的一些速度或容量等指标。

图表 2-3 多级内存技术指标

Memory	Logical width in bytes	Size in bytes – IXP2800/ IXP2400	Approx. unloaded latency in clks (IXP2800/ IXP2400)	Special Operations
Local Memory	4	2560/2560 (per-microengine)	3	Indexed addressing with post increment and decrement.
Scratchpad	4	16K/16K (on-chip)	60/60	Atomic operations including atomic subtract. 16 rings, with atomic get and put operations.
SRAM (QDR)	4	256M/128M (addressable, 64MB per-channel)	150/90	Atomic operations, excluding atomic subtract. 64-element queue array, with atomic enqueue, dequeue, get and put operations.
DRAM (Rambus/DDR)	8	2G/1G* (addressable, 1 GB per-channel)	300/120	Direct path to and from the MSF, which allows data to be moved between the two without first going through one of the processors.

图表 2-4 Microengine 各种寄存器的分配

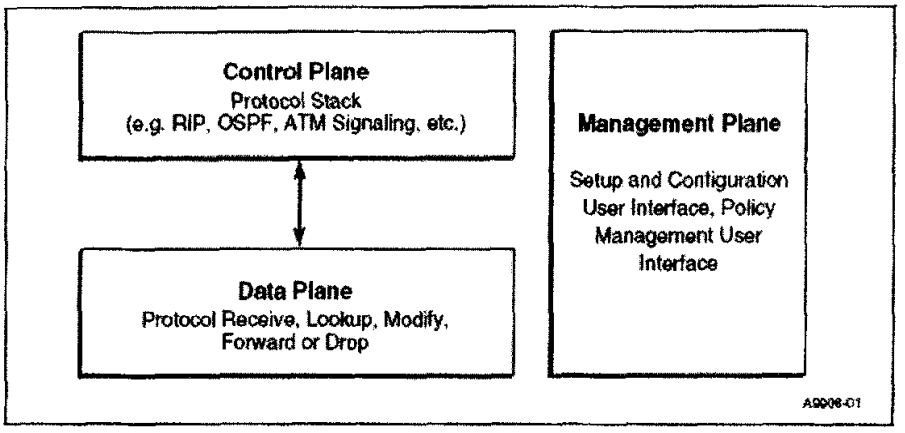
Number of Active Contexts	Active Context Number	GPR Absolute Register Numbers		S Transfer or Neighbor Index Number	D Transfer Index Number
		A Port	B Port		
8	0	0-15	0-15	0-15	0-15
	1	16-31	16-31	16-31	16-31
	2	32-47	32-47	32-47	32-47
	3	48-63	48-63	48-63	48-63
	4	64-79	64-79	64-79	64-79
	5	80-95	80-95	80-95	80-95
	6	96-111	96-111	96-111	96-111
	7	112-127	112-127	112-127	112-127
4	0	0-31	0-31	0-31	0-31
	2	32-63	32-63	32-63	32-63
	4	64-95	64-95	64-95	64-95
	6	96-127	96-127	96-127	96-127

IXP2400 还提供了丰富的专用处理单元，如硬件 hash 单元，cam 单元，CRC 单元,MSF 单元等，还有就是 sram 的原子操作，硬件 queue,ring 功能，scratchpad 的原子操作，硬件 ring 功能，都为高速的网络报文处理打下了坚实的基础。

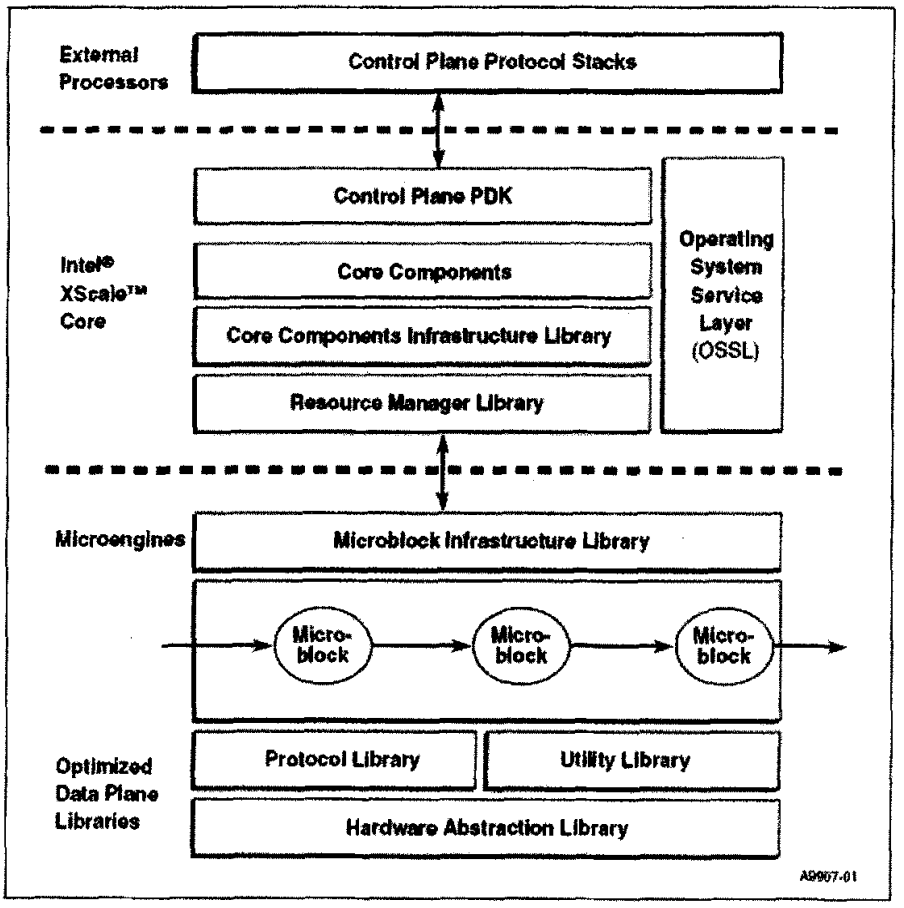
2.2 软件体系结构简介

Intel IXA 的可移植性框架（Portability Framework），它是 Intel 的互联网交换架构 IXA 的基础之一。它可以满足更快以及更具有效成本的代码开发，同时它通过软件的轻便性以及代码的重用性保护了软件的投资。一般的网络应用一般具有如图表 2－5 的体系结构，图表 2－6 是基于 IXA Portalility Framework 的软件体系结构，应用开发主要就是在 Framework 的框架下，开发运行在 xscale 上的 Core component（简称 CC）,和运行在 ME 上的 MicroBlock（简称 MB）。一般情况下，MB 都有相应的 CC，MB 完成快速处理报文的大部分工作，而 CC 则负责对 MB 进行配置管理或较复杂的慢速处理工作。应用开发，除了完成 CC 和 MB 开发外，还要完成整个应用的初始化配置及管理配置功能的开发。

图表 2-5 网络应用的平面关系图



图表 2-6 一个 IXA 应用的逻辑元构成图



2.3 开发环境简介

针对 ME 上的 MB 开发，开发工具为 workbench,它是一个图形界面开发软件，能够进行仿真开发，又能够在硬件环境下进行远程开发调试。

开发 MB 采用的编程语言有两种，一种是 microcode,相当于汇编语言，另一种是 MicroC,相当于 C 语言，但与普通 C 语言又有所不同。MicroC 语言与 C 语言相似，所以用 MicroC 可以更快的开发出 MB，但其效率比起 microcode 则逊色许多。所以可以考虑 MicroC+Microcode 开发，在对效率等方面要求严格的情况下，用 microcode 来优化代码。

针对 xscale 的 CC 开发，则相对 MB 开发容易一点，因为 xscale 是一个通用 CPU，目前支持嵌入式 linux 和 vxwork 两种操作系统，对 CC 的开发，与普通程序的开发相似，如在嵌入式 linux 上开发，主要采用 linux 内核模块编程技术；在 vxwork 上开发，则可以利用功能强大的 tornado 开发软件，其调试环境比起 linux 要棒很多。

除上面讲的开发工具外，ixa sdk 开发环境，有大量的库函数可供使用，还有全套的文档，仔细研究它们，会对应用开发很有帮助的。

2.4 小结

本章对 IXP2400 网络处理器技术分别从硬件体系结构，软件体系结构，开发环境三方面作了概要介绍，应用开发人员主要在 framework 框架的基础上从事 CC 和 MB 进行开发，开发 MB 主要使用 ixa sdk 集成开发环境 workbench，开发 CC 则依使用的操作系统不同而有所不同，在嵌入式 linux 上开发，主要采用 linux 内核模块编程技术；在 vxwork 上开发，则可以利用功能强大的 tornado 开发软件。开发人员只有从整体上掌握 IXP2400 的技术特点，才能开发出优秀的应用程序。

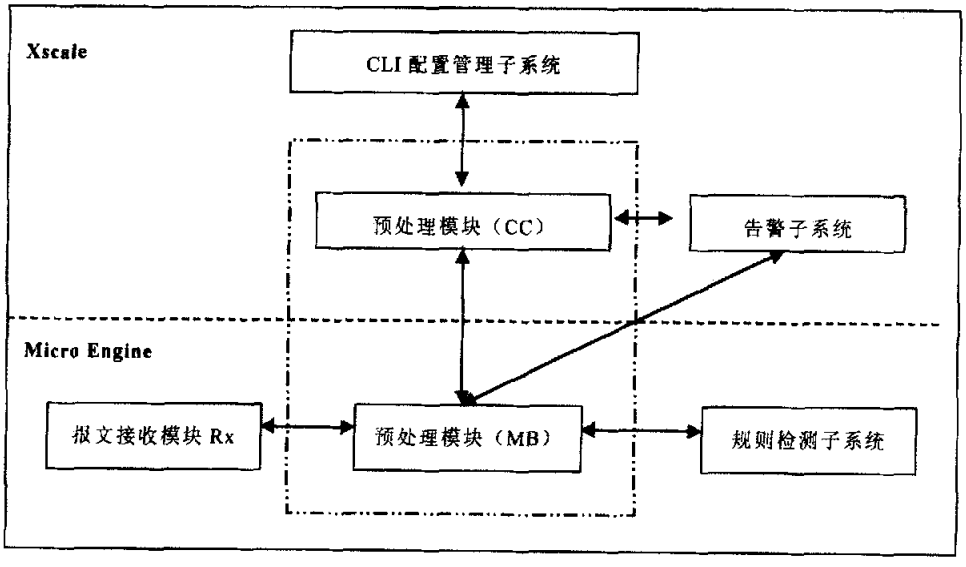
第三章 NP－NIDS 预处理子系统设计

“预处理”是相对于“规则检测”而言，规则检测子系统完成按即定的规则进行入侵检测判断，而预处理子系统则不但完成网络报文解码，ip 分片重组，tcp 流重组及应用协议正规化等操作，为规则检测子系统进一步处理报文作很好地预处理准备工作，而且承担着基于异常的检测工作，如 ip 分片攻击检测，端口扫描检测，TCP flooding 等异常检测功能。所以，预处理子系统是入侵检测引擎设计的重要组成部分，除了解码模块外，其它各模块相对比较独立，并不是必须的，但每增一个功能，都会使 NIDS 系统更加强大。

3.1 预处理子系统上下文定义

预处理子系统与报文接收模块，CLI 配置管理子系统，告警子系统，规则检测子系统都有连接。它主要接收来自报文接收模块的报文，预处理后送规则检测子系统作规则检测，若在预处理过程中发现异常行为，各 MB 会将它们的告警发送到相应的 CC，由 CC 送给告警子系统作进一步处理。CLI 配置管理子系统通过预处理子系统提供的配置接口完成预处理子系统各功能模块的配置管理工作。如图表 3－1 所示。

图表 3-1 预处理子系统上下文关联图

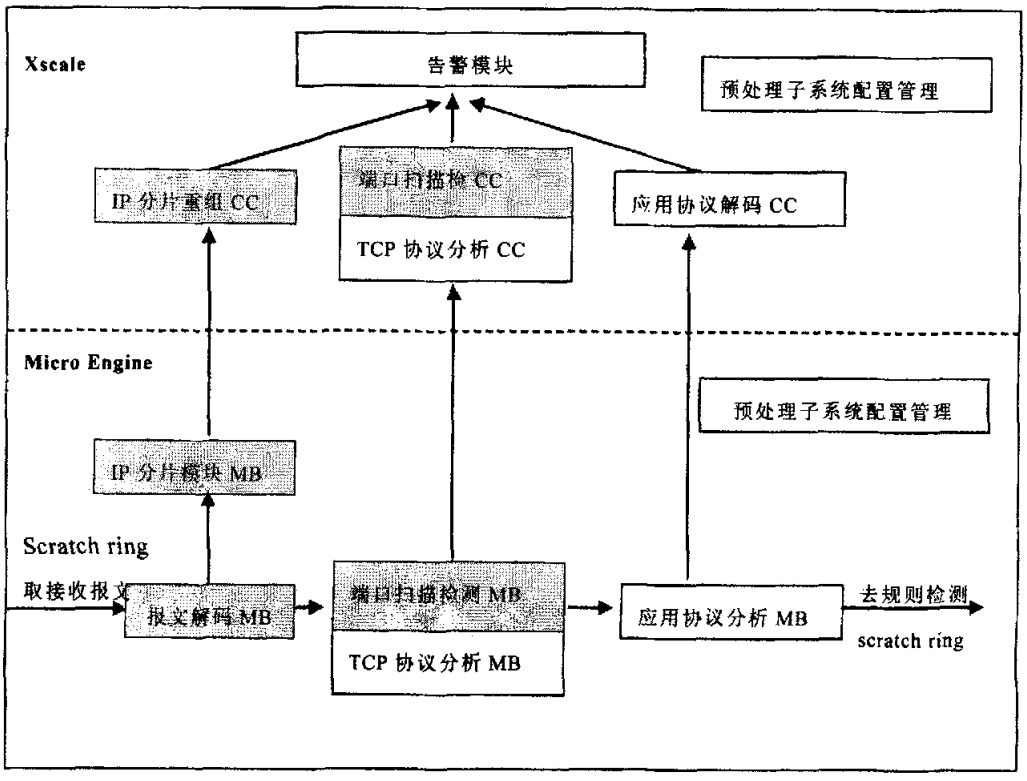


3.2 预处理子系统整体设计

预处理子系统主要由以下几部分组成,见图 3－2：

报文解码模块，IP 分片重组模块，TCP 协议分析模块（TCP 会话状态维护和 TCP 流重组），端口扫描检测模块，应用协议分析模块（包括 http,telnet,rpc 等应用协议）。

图表 3-2 预处理子系统内部设计图



本论文主要完成了预处理子系统中的报文解码模块，ip 分片重组模块及端口扫描检测模块的设计与实现(见图 3－2 灰色部分)，其它模块由于课题时间紧张和模块本身的复杂，将留在下一阶段工作中完成。

3.3 预处理子系统的接口设计

1．与报文接收模块的接口

由于 intel sdk3.5 提供了接收模块，所以与接收模块的接口就以 sdk 的接收模块接口为准，主要是报文的 meta 结构＋解码结构，通讯方式是 scratch ring

2．与规则检测子系统的接口

除了报文的 meta 结构外，还有报文对应解码结构，通讯方式是 scratch ring，与接收模块的 scratch ring 结构相同

3．与告警子系统的接口

主要是在 ME 上产生一个异常，送到各模块相应的 CC，再由各 CC 将告警送到告警子系统，通讯方式利用 sdk 的 rm 通讯机制，至于通讯的数据结构，MB 异常采用发送异常报文的标准格式，送给告警的格式见告警子系统对消息格式的定义。

4．与 CLI 模块的接口

xscale 部分主要是实现配置命令的 ioctl 功能。CC 对 MB 的配置一般通过共享 scratchpad 变量方式，ME 部分主要是设法更新 ME 上的缓存参数。

3.4 小结

预处理子系统不但完成网络报文解码，ip 分片重组，tcp 流重组及应用协议正规化等操作，为规则检测子系统进一步处理报文作很好预处理准备工作，而且承担着基于异常的入侵检测工作，如 ip 分片攻击检测，端口扫描检测，TCP flooding 等异常检测功能。

预处理子系统的每一个功能模块基本都分成 CC 和 MB 两部分，除 ip 分片的 CC 主要完成重组工作外，其它模块的 CC 主要完成配置功能。

本论文将主要分析预处理子系统的部分模块,即解码模块，IP 分片重组模块和端口扫描检测模块的设计与实现。

第四章 报文解码模块

报文解码模块的功能是对从网络上捕获的原始报文进行协议分析，尽力从各层协议头中提取重要的信息，如各种协议头的偏移，并记录在解码数据结构中，供后序的各种模块使用。它是对报文进行合法性检查的第一道关卡，如果报文本身有错，则会被丢弃，从而不必进一步分析。

4.1 snort 解码数据结构的特点分析

snort 解码数据结构 Packet[1]，通过分析观察，可以看出，它主要包含以下几部分：

- 原始报文信息，即来自网络端口的报文在内存中的基址，长度，到达时间
- 指向各层报头的地址指针，将原始报文按网络协议进行分析，获得的各个报头基本信息
- ip 分片详细信息
- ip option 解码后的信息
- tcp option 解码后的信息
- 各个协议头的重要字段信息，如 IP 地址，TCP 端口等
- 其它一些配置信息。

这个结构具有以下特点：

优点：

- 指向各层报头的基址指针十分方便后序插件访问各自所需的协议头。
- 对 ip option, tcp option 进行了解码，并以数组形式存放，方便了对 option 的检测。

缺点：

- 指向各层报头的指针变量是以列举方式给出，不符合协议分层的特点，若用 union 方式按协议分层给出，可节省不少内存。
- ip option,tcp option 是不常用字段，直接放在主控结构中，对内存是一种浪费，可以设二层结构来存储它。这样会大大缩小主控结构（即 packet 结构）的大小，从而节省大量内存。
- snort 采用单进程单线程模型，使得处理能力有限。

- snort 作为用户程序来进行，通过 libcap 库来抓包，使性能大打折扣（此条是 snort 的系统结构设计缺陷），如果放在 linux 内核空间运行，效率会大大提高。

4.2 解码模块设计与实现

4.2.1 解码数据结构设计

由于解码模块要处理千兆的流量，所以必须结合 IXP2400 网络处理器的硬件特点，对解码结构作精心的设计。下面便是解码结构的设计想法。

1) 将解码结构放在 sram 中，作为报文 Meta 的扩展

   这是权衡各种因素的结果，一方面原因是 dram 访问速度过慢，scratchpad 又过小,不足以存放大量报文的解码结构实例，local mem 不但容量过小，还不能在 ME 之间共享，所以放在 sram 中是比较合适的；另一方面原因是，IXP2400 的接收模块会给每个报文分配一个 hardware buffer(或 buffer chain),而这种 buffer 由 sram 中的 Meta 结构和 dram 中的报文原始数据两部分结成，所以把解码结构作为 Meta 结构的扩展（甚至可以去掉原 Meta 结构的部分字段），通过报文的 buffer handle 就可以很方便的访问到解码结构。

2) 解码结构提供的信息要与 snort 的 Packet 结构基本相当，这样才能方便其它模块使用。

3) 为了能够高效的访问解码结构，尽可能减少对 sram 的访问，解码结构要缓冲到 ME 的寄存器中（至少，常用字段要缓冲）。

4) 解码结构要尽可能的压缩，因为一方面 ME 内的寄存器数量有限，若为 8 线程模式,则对于每个线程，gp_reg 最多只能分得 32 个，local_mem 最多只能分得 80 个，各种 xfer 每种最多只能分得 16 个。另一方面过大的结构要多次读取内存，会极大的影响处理效率，基于这个考虑，解码结构最好 10 个长字左右(snort 的 Packet 竟然用了 200 多个长字!)。

5) 为了尽可能压缩解码结构，一方面要尽量使用 union 方式来定义字段（这对于同一协议层的各种协议非常适合）；一方面尽量用相对索引，而不用绝对的地址指针，并采用多字段合并到一个长字的方式；一方面可把非常用字段不放在解码主控结构中（如对于 ip option, tcp option,大部分报文并不含，就把它们单独存放，解码主结构只提供链接信息，能够通过链接索引访问到即可。这将会节省很多 sram 内存），

6) 适当增加统计变量，由于增加统计信息会影响效率，所以统计信息增加要适当，并能在一定条件下关闭统计信息。

基于上面的考虑，解码数据结构设计结果如下：

31 28 24	20	14	12 8	4 0
二层协议类型	三层	四层	reserved	
二层偏移	三层偏移		四层偏移	保留
数据偏移	数据长度		Ip option 偏移	Tcp option 偏移
Ip option buffer handle				
Tcp option buffer handle				

具体定义如下：

主要结构定义	结构简单说明
Decode_meta	它相当于 snort 的 packet 结构，目前结构大小为 5 个长字,内含的信息包括：网络报文含有什么协议；各协议报头的相对位置；指向 ip option,tcp option 的控制结构的句柄。 由于 ip option ,tcp option 为不常用字段，所以将它的控制信息放入二层结构中，若含有 ipotion，再从 sram 中调入。 此结构非缓冲时，同 ip option,tcp option 结构一起放在 sram 中。 typedef struct { Ipstack ipstack; /*定义现在的协议栈报头序列*/ Ipstack_hdr hdr;/*定义各协议头的偏移*/ unsigned int ipopt_handle;/*指向 ip option 数据结构*/ unsigned int tcpopt_handle; /*指向 tcp option 数据结构*/ }Ipstack_meta; typedef union {/*整个结构要尽量压缩大小，不同的协议栈可重用字段。*/ Ipstack_meta ipstack_meta; /*其它协议栈，请继续增加*/ /*这个数组大小，要根据上面的 struct 的大小，作调整*/

	unsigned int value[5]; }Decode_meta;

Ipstack	它指示出目前报文含有什么协议，它按协议栈的层次来组织,对以太网 ip 协议栈来说，二层协议占 8 位，三层占 4 位，四层占 4 位，其它位预留将来使用。通过访问各字段，并配合 Ipstack_enum 结构，可方便的判断协议栈的协议序列
	typedef union
	{
	struct
	{
	unsigned int eth_type :8;
	unsigned int net_type :4;
	unsigned int trans_type :4;
	unsigned int reserved :16;
	};
	/*基本上是用 1 个长字*/
	unsigned int value[1] ;
	}Ipstack;

Ipstack_enum	定义 ip 协议栈的各协议的编号，自定义各种协议的编号，而不采用 rfc 标准中的编号是为了压缩长度，因为标准协议长度为 16 位，太长了。通过它和 Ipstack 就可判断协议栈的协议序列 typedef enum { /*二层枚举*/ IP_MAC_ETH = 1, /*三层枚举*/ IP_NET_IP = 1, IP_NET_IPV6 = 2, IP_NET_ARP = 3,

	IP_NET_RARP = 4, /*四层枚举*/ IP_TRANS_TCP = 1, IP_TRANS_UDP = 2, IP_TRANS_ICMP = 3, IP_TRANS_IGMP = 4, IP_STACK_UNKNOWN = 0xf }Ipstack_enum;

Ipstack_hdr	它与 snort 的 packet 中的协议头指针的作用一样，只不过采用相对位置，即相对于报文数据基址。针对 tcp/ip 协议栈，由于协议头的相对位置不会超过 256 字节，所以每个协议分配 8 位就足够了。
	typedef union
	{
	struct
	{
	unsigned int eth :8;
	unsigned int net_type :8;
	unsigned int trans_type:8;
	unsigned int reserved:8;
	unsigned int data:8;
	unsigned int data_len:8;
	unsigned int ipopt:8;
	unsigned int tcpopt:8;
	};
	/*基本上是用 2 个长字*/
	unsigned int value[2] ;
	}Ipstack_hdr;

Tcpopt，Ipopt	这是 ip option,tcp option 的解码控制结构，在 packet 中为 option 保留了 40 个 entry,在这个结构里，只保留 9 个 entry,即最多可记录 9 种 option,

	这一般情况就足够了。它共占 10 个长字，
	对它的空间的分配采用一个 40 字节的硬件 buffer 来分配。它由 Decode_meta 的 ipopt_handle，tcpopt_handle 来指示。
	typedef union
	{
	struct
	{
	unsigned int opt_count:8;
	unsigned int opt_len:8;
	unsigned int reserved:16;
	Opt_entry opt_entry[IPOPT_COUNT_MAX];
	};
	unsigned int value[IPOPT_COUNT_MAX+1];
	}Ipopt;
	
	typedef union
	{
	struct
	{
	unsigned int opt_count:8;
	unsigned int opt_len:8;
	unsigned int reserved:16;
	Opt_entry opt_entry[TCPOPT_COUNT_MAX];
	};
	unsigned int value[TCPOPT_COUNT_MAX+1];
	}Tcpopt;

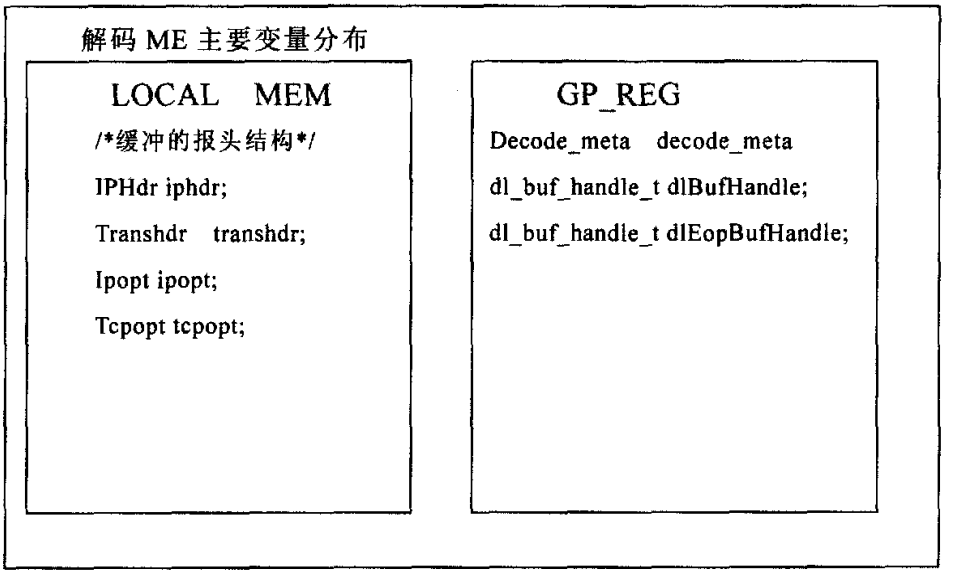
Opt_entry	描述一个 option 的基本信息。Offset 是相对于 option 数据区的 typedef union { struct { unsigned char type;

```
            unsigned char len;
            unsigned char offset;
            unsigned char reserved;
        };
        unsigned int value;
    }Opt_entry;
```

4.2.2 解码模块的实现

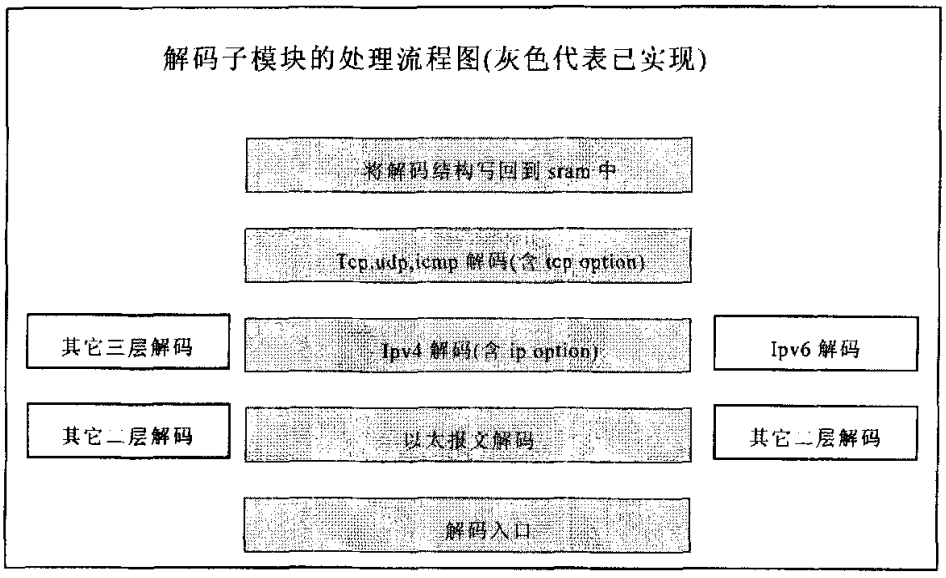
由于解码模块要对网络各层报文进行分析，所以把各层报文缓冲到 ME 的 localmem 内，作为线程内的全局变量。下面是各种重要变量在 ME 内的分布情况。

ME 内结构变量的分布



以上为解码模块的主要变量分布，放在 local_mem 的主要是各种报头,如 IPHDR 是 ip 头结构体，Transhdr 是传输层报头结构，它是 tcp,udp,icmp 等报头结构的 union 方式，放在 gp_reg 中的主要是全局控制变量，decode_meta 为报文主控制结构，dlBufHandle, dlEopBufHandle 为报文 buffer 句柄。

解码处理流程大致如下：



在对 ipv4 报头解码过程中，如果发现是 IP 分片，则程序会将解码结构写回到 sram 中，把报文发给 IP 分片处理模块继续处理；若解码过程中，发现报文有错，如校验错，长度错误等，则会将报文抛弃掉；若正常走完解码流程，则会将报文送到下一预处理子系统继续处理。

报文解码模块的调试输出结果如图 4-1；在高速流量下，应关闭调试输出功能。实测结果表明：在没有打开优化开关，解码模块在使用 2 个 ME，每个 ME 使用 4 线程模式，访问内存基本全部采用同步方式时，1G 比特流量，500 字节包长下没有发生丢包现象,这基本满足了千兆的线速处理要求。

图表 4-1 报文解码模块的调试输出截图

```
mac layer:
        ether :0 2 55 58 89 fe -> 0 0 50 11 8d aa :
net layer:
        ipv4:ipaddr 192.168.74.31  -> 192.168.74.33
    no ipopion

trans layer:
        udp: port 2049  -> 800
---->ix_cc_grasp_pkt: get pktipstack: 1->1->3
iphdr: 128->142->162
ipdata: 170
mac layer:
        ether :0 2 55 58 89 fe -> 0 0 50 11 8d aa :
net layer:
        ipv4:ipaddr 192.168.74.31  -> 192.168.74.33
    no ipopion

trans layer:
        icmp: type 8 , code 0
---->ix_cc_grasp_pkt: get pktipstack: 1->1->1
iphdr: 128->142->162
ipdata: 182
mac layer:
        ether :0 2 55 58 89 fe -> 0 0 50 11 8d aa :
net layer:
        ipv4:ipaddr 192.168.74.31  -> 192.168.74.33
    no ipopion

trans layer:
        tcp: port 63059  -> 80  , seq :-782237693
    no tcpopion
```

4.3 小结

本章首先分析了 snort 的报文解码数据结构设计的特点，然后提出了解码模块数据结构的设计思想，并给出了一个高度优化的数据结构，此解码结构只有 5 个长字，比起 snort 的 Packet 结构（大约 200 多个长字）有质的不同，非常适合 IXP2400 的 ME 硬件特点；接着简要给出解码模块的实现。实测性能结果表示：在没有打开优化开关，使用 2 个 ME，每个 ME 使用 4 线程模式，访问内存基本全部采用同步方式时，1G 比特流量，500 字节包长下没有发生丢包现象，基本满足了千兆的线速处理要求。

第五章 IP 分片重组模块

5.1 IP 分片攻击原理

5.1.1 IP 分片的由来

IP 报文要通过网络物理端口发送出去，必须经过数据链路层的封装，而数据链路层有一个 MTU 参数，即最大传输单元，它表示了数据链路层的一帧可以装载的最大数据量。当 IP 报文的长度超过了 MTU，那么 IP 层就要对报文进行分片（fragmentation）操作，使每一片的长度都小于或等于 MTU。当分片报文到达目的地时，必须对分片进行重组，以还原出完整的 IP 报文。

在 IP 报头中，有以下几个字段，提供了 IP 分片重组的必要信息：

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |R|DF|MF|    Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|<------------16------------>|<--3-->|<---------13---------->|
```

Identification：IP 分片标识，属于同一个原始 IP 报文的分片都有相同的标识值 。

R：保留未用。

DF：Don't Fragment，“不分片”位，如果将这一比特置 1 ，IP 层将不对数据报进行分片。

MF：More Fragment，“更多的片”，除了最后一片外，其他每个组成数据报的片都要把比特置 1。

Fragment Offset：该片偏移原始数据包开始处的位置。偏移的字节数是该值乘以 8。

5.1.2 IP 分片攻击

IP 分片攻击主要利用了各种系统在实现 IP 协议时的缺陷，通过恶意的使用 IP 分片字段，从而绕过检测系统或使目标系统因无法正确处理分片而产生异常。

5.1.2.1 分片拒绝服务攻击

系统内存和处理能力都是有限的，如果短时间内有大量的分片到达，则系统会因为忙于处理分片，而把系统资源耗尽。

这种攻击是比较可怕的。它不仅让系统产生拒绝服务，而且高流量的分片会占用大量的网络带宽。

例如， jolt2[50]是在一个死循环中不停的发送一个 ICMP/UDP 的 IP 碎片，就可以使 Windows 系统的机器死锁。分片重组时会预分配大量内存，若分片不断到达，则可能会因为内存被耗尽而死锁。

5.1.2.2 分片过长攻击

IP 协议用 16 位字段表示整个 IP 数据包的长度，所以 IP 报文最长为 0xffff; IP 协议用 13 位字段表示 fragment offset,由于它实际要乘 8，所以最大也能表示 0xffff。但若把 IP 头的 fragment offset 设得很大，如接近 0xffff,若再加上分片长度，则重组后的 ip 报文实际长度就超过了 0xffff,这显然无法在 ip 头中表示出来。

有缺陷的系统由于对这些特殊的情况没有很好的处理，一些攻击正是利用了这个特性，发出超长分片，从而引起受攻击系统出现死机或拒绝服务等问题。

例如，死亡之 ping(ping o' death)就是利用 ICMP 协议的一种碎片攻击。攻击者发送一个长度超过 65535 的 Echo Request 数据包，目标主机在重组分片的时候会造成事先分配的 65535 字节缓冲区溢出，系统通常会崩溃或挂起。

5.1.2.3 TEARDROP 攻击

IP 层在进行分片时，正常情况下分片与分片的数据只可能相接，但若出现分片部分相交叠或一个分片内嵌在另一分片中的现象，我们就称它为 teardrop 攻击；它实际是因为 teardrop[49]程序而得名。有的系统因为没有正确处理这种分片情况，引起缓冲区缀出，严重则导致系统崩溃，如老的 Linux 内核（1.x - 2.0.x）在处理这种分片的时候存在问题，WinNT/95 在接收到 10 至 50 个 teardrop 分片时也会崩溃。

5.1.2.4 反监测 IP 分片攻击

由于对 IP 分片进行重组需要很多系统资源，所以有些防火墙或入侵检测系统对 IP 分片重组不支持或处理能力有限，攻击者正是利用监测设备的这个弱点，通过恶意利用 IP 分片来绕过监测设备，达到攻击或入侵的目的。

如 Tiny fragment[51]攻击,它就是通过恶意操作，将 TCP 报头(通常为 20 字节)分布在 2 个分片中，这样一来，目的端口号可以包含在第二个分片中。对于包过滤设备或者入侵检测系统来说，首先通过判断目的端口号来采取允许/禁止措施。但是由于通过恶意分片使目的端口号位于第二个分片中，因此包过滤设备通过判

断第一个分片,决定后续的分片是否允许通过。但是这些分片在目标主机上进行重组之后将形成各种攻击。

再比如出现分片重复现象，snort 将它告警成 OVERLAP 攻击，当出现这种现象，不同操作系统在选择先到的重复分片还是后到的重复分片时有区别的,这就可能被攻击者所利用，从而逃避检测。

甚至，可以利用 IDS 对分片重组的有限能力，先发送大量分片，这样 IDS 就进入到自我保护状态，短暂时间内不会再对分片作监测，攻击者可以利用这个时间差，发出真正带有入侵特征的分片，从而逃避 IDS 系统的监测。

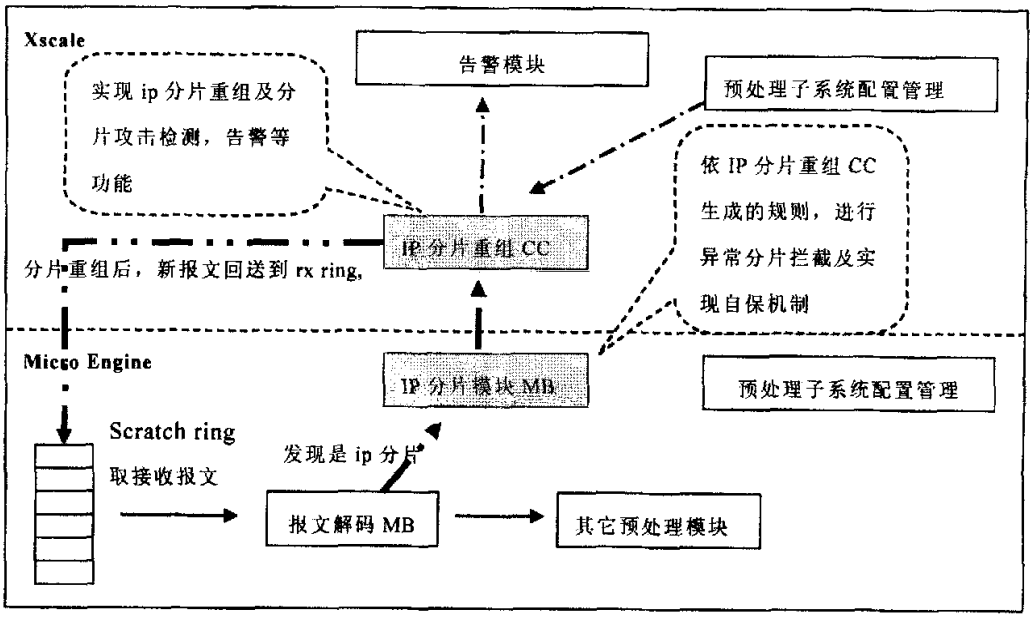
5.2 IP 分片重组设计与实现

任何能进行 tcp/ip 通讯的系统都具有 ip 分片重组功能，但是作为 IDS 系统的 IP 分片重组功能，却有它自己的特点。

1. 一般系统只对到达自身的 IP 分片进行重组，所以重组量比较小，而 IDS 系统则要对整个监测网络中的所有分片进行重组，所以重组能力必须更加强大，特别是高速 NIDS 系统，对重组能力要求更高。
2. 一般系统对 IP 分片只重组，遇到异常时不告警，而 IDS 系统必须能检测到 IP 分片报文的非正常使用,即前面讲到的 IP 分片攻击行为，并给出告警。
3. 作为 IDS 系统，首先要保证自己的 IP 分片重组功能强健，能够应付各种非正常的分片使用情况，在遇到分片洪流而自己又因处理能力有限时，能启动自我保护机制。否则，一旦出现分片洪流，便会出现“出师未捷身先死,常使英雄泪满襟”的悲惨局面。

下面在模块的设计与实现时，将兼顾重组能力，异常告警，自保机制等三方面。图表 5－1 为 ip 分片重组模块的设计逻辑图,重组成功后的分片被重新放回报文接收 scratch ring，重新进行分析。

图表 5-1 ip 分片重组模块设计逻辑图



5.2.1 异常告警

在进行分片重组过程中，必须对各种异常的 IP 分片进行告警，下表对各种告警作以总结。

告警类型	触发条件
OVERSIZE_FRAG	(1) 分片偏移大于 65515 (2) 分片偏移加分片数据长度大于 65515
TEARDROP	重组时，发现 IP 分片之间有部分重叠现象
TTL_EVASION	同组 IP 分片的 ttl 的变化超过一个阀值
OVERLAP	分片出现重复
DUPFIRST	首个分片出现重复现象
OUTOFORDER	分片到达出现乱序
IPOPTIONS	分片 IP 报头中有 ip option
TIMEOUT	在一个时间限内，分片未全部到达
SUPRESS	分片抑制，当分片过多，超过一个阀值时

5.2.2 分片重组

5.2.2.1 分片存储

在同 ID 的 IP 分片未全部到达前，必须把已到达的分片暂时保存起来，以便在所有分片到达后，进行重组，这种分片存储，是非常消耗系统内存的，所以必须对它精心设计。

snort 的分片存储方法是，在排序树中插入所有分片，但不进行分片合并。它在每个分片到达后，都是先保存下来，然后检查分片是否全部到达，若全部到达，则开始重组分片,所以 snort 的分片重组方法是等分片全部到齐才进行重组,实际上 linux2.4.18[3]中的分片重组方法也是这样的。

这种方法的不足是每次分片到达都要遍历分片树，检查分片是否全部到达，非常消耗系统资源。即使不考虑这个因素，这种方法也难以用在 IXP2400 的分片重组模块中，因为在 IXP2400 上实现时，报文的存储，并不是报文有多大，就分配多大的内存，而是为了快速进行内存分配，采用等大内存分配策略，即采用 hardware buffer 方式，每个报文至少使用一个 buffer,buffer 大小固定，一般为 2048 字节，若一个 buffer 装不下报文，则可以用多个 buffer 形成一个 buffer chain 来记录一个报文。所以即使分片数据中有一个字节，也要使用一个 2048 字节的 buffer，如果未分片报文为 60000 字节，分片数据为 8 字节，则此报文要分成 7500 个分片，则用 buffer 方式存储，则要用 7500*2048 = 15360000 字节内存，大约有 15MB 内存，这显然是无法容忍的。所以我们必须寻求新的方法来进行分片存储。

我们将采用一种分区合并分片重组算法，这种方法是按 buffer 的大小，将 IP 分片的重组空间（分片重组后的报文数据最大为 65535 字节，所以它的重组空间即为 0 至 65535）进行分区，每个分区都是一个 buffer 大小，当一个分片到达时，我们就将它拷贝到相应分区的相应位置上，若分区还未分配，则先分配一个分区，它比起一次性预分配所有分区显然节省不少内存，特别是当重组失败时，这种内存节省的效果更明显。

5.2.2.2 分片管理

在前节，我们对分片存储采用了一种分区合并分片重组算法，它实现了分片的及时合并，节省了不少内存，但我们并没有说明如何管理分片，在本小节，将给出分片管理的一种解决方案。

首先用一个 hash 表来记录每个 ip 分片的入口，关键字为四元组：id,protocol,source ip,dest ip。对同 ID 的分片管理，有两个层次，即分区层和分片层。我们用一个排序树记录所有已分配的分区。然后在分区内再用一个排序树记录分片情况，而且分片描述符是可以合并的，当一个分区被填满时，则将“分区

满”标志置 1，通过遍历分区树，并结合“分区满”标志位，就可以判断分片是否全部到齐，若分片全部到达，将所有分区按顺序用 buffer chain 方法串起来，它就自然组成了一个完整的报文。图表 5－2 为分片重组的数据结构关系图,从中可看出分片管理方法。

具体处理流程如下：

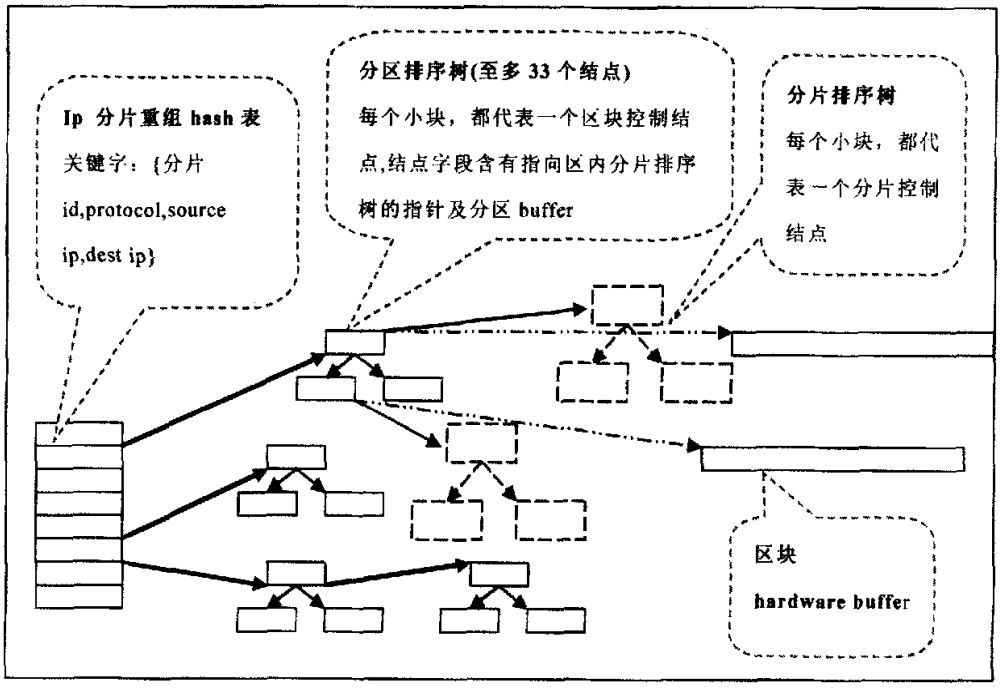
（1） 当一个分片到达后，在分区树中查找它所在的分区是否存在，若不存在，则先建一个新分区，转（2）；若存在，直接转（2）

（2） 在区内的分片树中查找分片是否存在，若存在，则告警，若不存在，则将分片拷入相应的位置，并更新分片树。

（3） 当最后一个分片到达后，检查分片报文是否全部到达，若全部到达，转（4），注意，在最后一个分片到达后，如果还有分片报文未到达，则就应该每次检查分片是否可以重组。

（4） 分片报文全部到达，进行重组，即将所有分区链接成一个 buffer chain(如果有多个分区的话)，并释放重组此分片的管理结构，重组结束

图表 5-2 ip 分片重组的数据结构关系图



5.2.2.3 分片重组超时

当属于同一个 IP 报文的分片报文有一个因网络而丢失，则重组将无法完成，所以必须有超时机制，保证无法重组的分片被及时清除掉，及时释放出系统资源。

超时方法很简单，可以设一个定时器，定期对超时分片结点进行清除即可。

5.2.3 自保机制

进行分片重组是非常消耗系统资源的，即使 IDS 系统有很强的 IP 分片重组能力，但是如果网络中存在 IP 分片洪流（如用 fragroute 对整个网络报文进行分片），它还是“吃不消的”。实际上重组能力并不是越强越好，由于 IP 分片报文源地址可以伪造，所以攻击者完全可以发送大量的 IP 分片垃圾，故意让 NIDS 系统忙于分片重组，这种情况下若还进行重组工作，反而是被攻击者“愚弄”了。自保机制就是要保证在存在大量分片，超过 NIDS 的重组能力之后，能进入自我保护状态，防止 NIDS 系统因系统资源消耗过大而无法正常工作，影响其它模块的正常工作，甚至发生崩溃。

自保机制可以一方面通过设一个内存使用阀值，使 IP 分片重组模块对内存的需求被限制在一个可控范围内；另一方面，设一个 IP 分片到达频率阀值，当超过阀值时，在一定时间内不再进行重组工作。阀值大小的设定，可根据系统的资源作调整，如果系统内存资源丰富，处理能力强大，阀值可设大点，否则小点，如在 snort 中，默认值为 1000。本模块目前也以 snort 的阀值为准。

本模块实现时，正常重组在 CC 内实现，为了能高效的进行自我保护，特意把自保机制放在 ME 内实现，作为一个 microblock 来实现，即当解码模块发现 IP 分片报文时，便将报文送给 IP 分片 microblock,此 MB 进行自保判断，若无异常，才将分片送向重组 CC 进行重组，否则，直接在解码部分进行拦丢弃即可。这样，即使用线速的 IP 分片进行“轰炸”，重组模块及其它模块也能正常工作,整个系统性能也不会受到太大的影响。

5.3 模块功能测试

本功能测试，将采用 fragroute,teardrop,jolt2 等工具时行测试，测试用例[8]如下：

test1: ordered 8 byte fragments
test2: ordered 24 byte fragments
test3: out of order 8 byte fragments

test4: ordered 8 byte fragments,duplicate first packet

test5: out of order 8 byte fragments, duplicate first packet

test6: ordered 8 byte fragments,reorder fragments in reverse

test7: ordered 16 byte fragments, fragment overlap(favour new)

test8: ordered 16 byte fragments, fragment overlap(favour old)

test9: teardrop

test10: oversize

test11: ttl evasion

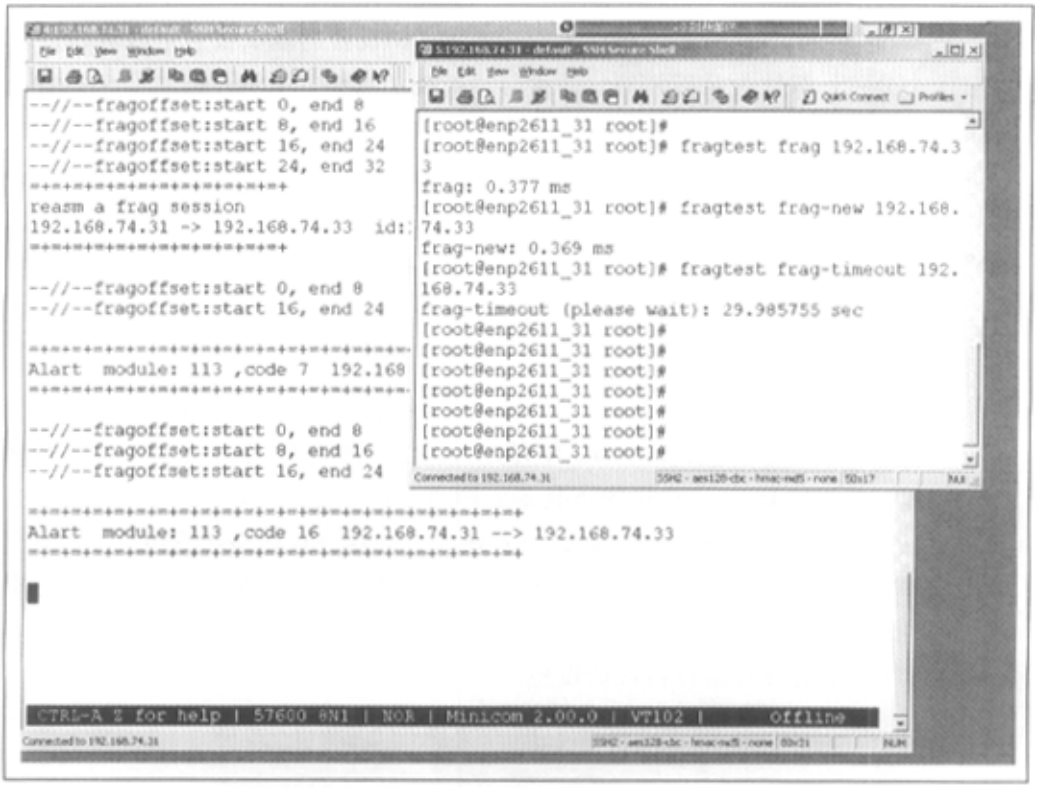
test12: ip option

测试结果如下：

测试用例	是否告警	是否重组
Test1		√
Test2		√
Test3	√	√
Test4	√	√
Test5	√	√
Test6	√	√
Test7	√	√
Test8	√	√
Test9	√	
Test10	√	
Test11	√	
Test12	√	

可以看出，达到了功能设计要求。图 5-3 为功能测试的截图，图中的 3 条告警分别是对 fragtest frag 192.168.74.33, fragtest frag-new 192.168.74.33, fragtest frag-timeout 192.168.74.33 的响应输出结果，其中 module 数值代表预处理模块编号，code 数值代表告警类型,图中正常分片重组后，也给出了告警，但在高速流量下，可以通过控制命令关闭此告警。

图表 5-3 ip 分片功能测试输出截图



5.4 进一步思考

5.4.1.1 关于 MTU 与 IP 分片拦截

MTU 是网络连接中允许在一个物理帧中传输的最大数据量，TCP/IP 驱动程序需要知道这个数值以便设置合适的数据包长度，以避免出现不必要的拆分过程。也就是说，为了保证网络传输效率，一般 MTU 不会设的太小。不同网络的 MTU 缺省值不一样，以太网的标准值为 1500，而多数 ISP 提供的拨号网络的标准值为 576，原因是 Internet 上不少路由器也为 576。

从这些数据可以看出，一般情况下 MTU 的值都大于 576，这就是说只有大于 576 的 ip 分片才可能被看成是正常的分片，小于 576 的 IP 分片很可能是恶意使用分片特性。

基于以上分析，本人认为，可以对小分片进行拦截或不进行重组处理(当然，最后一个小分片除外)。比如，在 IP 分片重组模块中，增加一个“MTU 阀值”的控制参数（如默认设成 576），当发现一个 IP 分片的载荷小于 MTU 阀值时，就发

出告警并通知防火墙等过滤系统拦截分片，则会大大减少 IP 分片对被监测网络的威协，并大大减轻 IP 分片重组对网络监测设备的压力。

从 snort 对 ip 分片的实现和众多的知名 NIDS 设备的测试报告可以看出，它们对于小分片，都认为是正常的，并给与尽可能的重组。虽然，在本项目的 IP 分片重组模块的实现中，目前也并未加入“MTU 阀值”这个控制参数，但我认为可以加入,加入之后，也就可防止攻击者用诸如 fragroute 工具对报文进行恶意的分片。

5.5 小结

本章首先分析了 IP 分片攻击的原理，并总结了常用的 IP 分片攻击技术：分片拒绝服务攻击,分片过长攻击,TEARDROP 攻击,反监测 IP 分片攻击等。接着分析了高速 NIDS 的 IP 分片模块应该具备的特点：重组能力，异常告警，自保机制，并从这三个方面解释了本模块的设计与实现，其中对分片存储和重组作了详细的分析，并提出了分区合并分片重组算法；功能测试结果，证明本模块达到了设计要求；在最后，对 MTU 和 IP 分片拦截的关系作了讨论,并提出了可以直接对小于一个 MTU 阀值的分片进行拦截的设想。

第六章 端口扫描检测模块

6.1 端口扫描技术

6.1.1 TCP 协议介绍

TCP 是一个面向连接的可靠传输协议。面向连接表示两个应用端在利用 TCP 传送数据前必须先建立 TCP 连接。 TCP 的可靠性通过校验和，定时器，数据序号和应答来提供。通过给每个发送的字节分配一个序号，接收端接收到数据后发送应答，TCP 协议保证了数据的可靠传输。数据序号用来保证数据的顺序，剔除重复的数据。

一个 TCP 头包含 6 个标志位。通过对它们进行识别，从而保证了 TCP 状态机的正确的变迁和状态的正确维持。它们的意义分别为：

- SYN:请求建立连接
- FIN:请求释放连接标志
- RST:用来复位一个连接
- URG: 为紧急数据标志。如果它为 1，表示本数据包中包含紧急数据。此时紧急数据指针有效。
- ACK: 为确认标志位。如果为 1，表示包中的确认号时有效的。否则，包中的确认号无效。
- PSH: 如果置位，接收端应该尽快把数据传送给应用层。

大部分 TCP/IP 实现遵循以下原则：

1：当一个 SYN 或者 FIN 数据包到达一个关闭的端口，TCP 丢弃数据包同时发送一个 RST 数据包。

2：当一个 RST 数据包到达一个监听端口，RST 被丢弃。

3：当一个 RST 数据包到达一个关闭的端口，RST 被丢弃。

4：当一个包含 ACK 的数据包到达一个监听端口时，数据包被丢弃，同时发送一个 RST 数据包。

5：当一个 SYN 位关闭的数据包到达一个监听端口时，数据包被丢弃。

6：当一个 SYN 数据包到达一个监听端口时，正常的三阶段握手继续，回答一个 SYN　ACK 数据包。

7：当一个 FIN 数据包到达一个监听端口时，数据包被丢弃。"FIN 行为"（关闭的端口返回 RST，监听端口丢弃包），在 URG 和 PSH 标志位置位时同样要

发生。所有的 URG，PSH 和 FIN，或者没有任何标记的 TCP 数据包都会引起"FIN 行为"。

6.1.2 常用扫描技术[45]

6.1.2.1 全 TCP 连接扫描

全 TCP 连接是长期以来 TCP 端口扫描的基础。扫描主机尝试（使用三次握手）与目的机指定端口建立正规的连接。连接由系统调用 connect()开始。对于每一个监听端口， connect()会获得成功，否则返回－1，表示端口不可访问。由于通常情况下，这不需要什么特权，所以几乎所有的用户（包括多用户环境下）都可以通过 connect 来实现这个技术。

这种扫描方法很容易检测出来（在日志文件中会有大量密集的连接和错误记录）。Courtney,Gabriel 和 TCP Wrapper 监测程序通常用来进行监测。另外，TCP Wrapper 可以对连接请求进行控制，所以它可以用来阻止来自不明主机的全连接扫描。

6.1.2.2 TCP SYN 扫描

在这种技术中，扫描主机向目标主机的选择端口发送 SYN 数据段。如果应答是 RST，那么说明端口是关闭的，按照设定就探听其它端口；如果应答中包含 SYN 和 ACK，说明目标端口处于监听状态。由于所有的扫描主机都需要知道这个信息，传送一个 RST 给目标机从而停止建立连接。由于在 SYN 扫描时，全连接尚未建立，所以这种技术通常被称为半打开扫描。SYN 扫描的优点在于即使日志中对扫描有所记录，但是尝试进行连接的记录也要比全扫描少得多。缺点是在大部分操作系统下，发送主机需要构造适用于这种扫描的 IP 包，通常情况下，构造 SYN 数据包需要超级用户或者授权用户访问专门的系统调用。

6.1.2.3 秘密扫描技术

由于这种技术不包含标准的 TCP 三次握手协议的任何部分，所以无法被记录下来，从而比 SYN 扫描隐蔽得多。另外，FIN 数据包能够通过只监测 SYN 包的包过滤器。

秘密扫描技术使用 FIN 数据包来探听端口。当一个 FIN 数据包到达一个关闭的端口，数据包会被丢掉，并且回返回一个 RST 数据包。否则，当一个 FIN 数据包到达一个打开的端口，数据包只是简单的丢掉（不返回 RST）。

Xmas 和 Null 扫描是秘密扫描的两个变种。Xmas 扫描打开 FIN，URG 和 PUSH 标记，而 Null 扫描关闭所有标记。这些组合的目的是为了通过所谓的 FIN 标记监测器的过滤。

秘密扫描通常适用于 UNIX 目标主机，除过少量的应当丢弃数据包却发送 reset 信号的操作系统（包括 CISCO，BSDI，HP/UX，MVS 和 IRIX）。在 Windows95/NT 环境下，该方法无效，因为不论目标端口是否打开，操作系统都发送 RST。

跟 SYN 扫描类似，秘密扫描也需要自己构造 IP 包。

6.1.3 扫描类型总结

根据 TCP 的 flag 字段，就大致可以判断出扫描类型，下面是对扫描类型与 flag 对应关系的汇总。黑三角代表含有对应的标志位；不标三角符号表示不含标志位，空三角也表示不含标志位，但起强调作用。

TCP flag / 正常情况	URG	ACK	PSH	RST	SYN	FIN
		▲				
		▲			▲	
					▲	
		▲				▲
		▲		▲		
		▲	▲			
	▲	▲				
	▲	▲	▲			
		▲	▲			▲
	▲	▲				
	▲	▲	▲			▲
		▲	▲	▲		

TCP flag / 扫描类型	URG	ACK	PSH	RST	SYN	FIN
FULLXMAS	▲	▲	▲	▲	▲	▲
SPAU	▲	▲	▲		▲	

SYN					▲	
FIN						▲
SYNFIN					▲	▲
NULL						
XMAS	▲		▲			▲
VECNA	▲					
			▲			
	▲					▲
			▲			▲
	▲		▲			
NMAPID	▲		▲		▲	▲
NOACK	与上面各种情况都不同					△
INVALIDACK	与上面各种情况都不同					▲

6.2 端口扫描检测模块

6.2.1 模块设计

在说明设计之前，先说明几个用语：

连接结点：记录一个 TCP 连接（或 UDP）信息的数据结构实例。

连接结点数据库：它用于存储所有连接结点信息。

源 IP 结点：记录一个 IP 地址对应的扫描统计信息的数据结构实例，因为是对一个报文的源 IP 地址作扫描统计，所以叫源 IP 结点。

源 IP 结点数据库：它用于存储所有源 IP 结点信息。

注意，上面所用“数据库”一词并不是普通意义上的数据库概念，而只是用于记录信息的某种数据结构，如 hash 表，链表，树等。

6.2.1.1 连接结点数据库

a) 连接结点的双关键字问题

一个 TCP 连接，被记录在连续数据库中，其关键字由 4 个元素组成：源端 IP，源端 port,目的端 IP,目的端 port。但是由于一个 TCP 连接有两个方向的流，这就是说其关键字有两种表示方法，如果只是简单的用当前报文的源端 IP，源端 port,

目的端 IP,目的端 port 顺序组成关键字，则会出现连接结点无法找到的可能，这时只有改变关键字元素的排列顺序，即用“目的端 IP,目的端 port, 源端 IP，源端 port”才可能找到连接结点，而 snort 的连接结点的查找正是采用这种查找失败后，通过改变关键字元素序列，然后二次查找的方法。这种方法，显然会使查找效率至少损失一半。

为了解决这个问题，在创建 TCP 连接结点时，其关键字不按“source ip, source port,dest ip, dest port”顺序排序，而是以“较小 ip, 较小 ip's port, 较大 ip, 较大 ip's port”顺序来生成关键字，则一个 TCP 连接必然只有一个关键字序列，从而解决了一次就以可以找到连接结点。

这时，会产生一个新问题，就是如何区别服务端和客户端呢？

若采用双关键字方法，由于 TCP 连接结点创建时，其报文的源 ip 是发起端，即为客户端，则若一次找到，则当前报文必为客户端的流，若第二次才找到，则当前报文必为服务端的流，通过分析是第几次查找到，就可以确定当前报文来自客户端还是服务器端。

若采用本文提出的关键字排序方法，只需在连接结点加一个标志位，通过标志位，并组合当前报文的源 ip，就可以断定出当前报文是哪个流向的。具体作法是：在创建 TCP 连接结点时，把按升序排列的关键字的“较小 ip”与“源 ip”相比较，若相同，则设标志为 0，表示升序排列关键字代表的是客户端流，即较小 ip 为客户端 ip; 反之较小 ip 代表是服务端 ip。当在以后查找结点时，当找到结点，通过标志位，就可以从连接结点中得到客户端 ip 地址，把它与当前报文的源 ip 比较，就能判断出此报文来自客户端还是服务器端。

b) 连接结点查找算法的选择

在存储连接结点时，一般有以下几种方法可供选择：

1. 一维链表方式。即用一维的链表或数组来组织结点，这种方式的查找效率会随着结点的增多而成线性增长，平均查找时间为存储的结点数的一半。
2. 排序树方式。即将结点按关键字排序，记录在排序树中，这种方式的查找效率显然比一维方式有很大的改进。snort 就是采用这种方式，而且 snort 的 portsan 模块为了提高查找效率，采用了 3 层排序树算法，即源 ip 树，目的 ip 树，源目端口树三种树，它们分别形成了查找第一二三层。具体查找是这样的：
   （1） 取当前报文的源 ip,在源 ip 树上查找相应结点，若找到，则进入第（2）步。

（2） 从源 ip 树结点中，取出目的 ip 树的根结点。用当前报文的目的 ip 在目的 ip 树中查找，若找到，则进入第（3）步。

（3） 从目的 ip 树中取出源目端口树的根结点。用当前报文的源目端口在源目端口树上查找，若找到，则可从中取出连接结点的指针，从而查找完成。

3. Trie 表方式。Linux 内核的页表组织和高速路由器上用的 trie 表就是这种方式，它的特点是查找次数基本固定，即通过若干次，必能找到最终的结点记录。所以在高速高效的系统中经常被采用。
4. 哈希方式。如果不产生冲突，它的时间复杂度是最好的。但如果产生了冲突，则对冲突域的结点组织方式，就变得格外重要了。这种方式也经常被高速高效的系统所采用。

通过对算法本身的比较，并结合 IXP2400 的硬件特点及本项目对速度的严格要求，最终，确定采用超大哈希方式。下面分析一下为什么选择哈希方式。

1. 为了能线速处理 G 比特的流量，此模块选择在 ME 中实现，并且要尽可能的减少对 sram,dram 等内存的读取，因为 ME 的效率很大程度上由对内存的访问来决定。采用一维方式和排序树，其查找次数不定，且随结点的数增加，查找次数也不断增加，而我们为了减少端口扫描模块的漏报误报问题，必须维护大量的连接，目前设计最多可支持 64K 的连接。这样多的结点，若查找次数过多，即内存访问过频，根本无法达到线速处理能力。
2. 由于此模块是在 ME 中实现，而此模块必须动态维护连接数据库，连接结点会不断的因为是新连接而创建，因为超时而消亡。所以我们必须减少连接数据库维护的代价。
3. 如果采用 trie 表方式，则 trie 表必须能查找 96 位的关键字，查找时，也基本上只能采用 16-8-8-8-8-8-8-8-8-8-8 方式，这种查找，虽然在查找次数上为恒定的，即 11 次，还可以接受，但考虑过查找表所需的内存数量，则让人生畏的，目前，就我们的硬件条件，sram 有 8MB，dram 有 196M，而言，条件要求太高，而无法实现。而且由于连接结点的不断变动，对它的维护也极其困难。
4. 若采用哈希方式，冲突域采用链表方式，如果不产生冲突，查找只需一次，如果有冲突，查找效率则与链表长度有关。但如果我们采用以空间换速度的策略，使用超大哈希表，比如采用 64K 个表项的超大表，则对于目前支持最多 64K 个结点来说，其冲突就变得很少

了。又由于 ME 支持两种硬件 hash key 生成算法，其中的硬件 hash 部件产生的 index 具有统计分布特性，则会使冲突减少到最小，可以乐观的估计，冲突一般不会超过 3 个结点。

所以说，采用超大哈希算法是综合考虑查找效率，内存需求，硬件支持等因素的结果。

在解决哈希冲突时，采用双向链好呢还是单向链好呢？

我们选择冲突采用单链方式，而不是采用双链，因为双链的维护代价较大，要修改前后的结点，用单链只要修改前驱结点，这样就减少了一次内存写操作，更重要的是，减少一个写，对线程并发同步效率是大有好处的。但单链的问题在于前驱结点的查找必须从链头开始，但似乎这种代价比双链的要小。因为采用大尺寸 hash，并结合硬件来产生 key,冲突会减少到很小，从链头查找前驱结点只是读，并不改，也很快。

6.2.1.2 堆栈式结点分配

其原理为：

假设总共可管理结点数为 64K，每个结点数据结构为 16 个字节，则结点占用内存为 64K * 16B = 1MB,可通过 xscale 的 RM 库分配得到，设其基址为 node_base_add,所以结点集就是一个大数组，我们用这个数组的索引来代表结点本身，索引用 16 位表示，则最大可表示 64K 个结点。

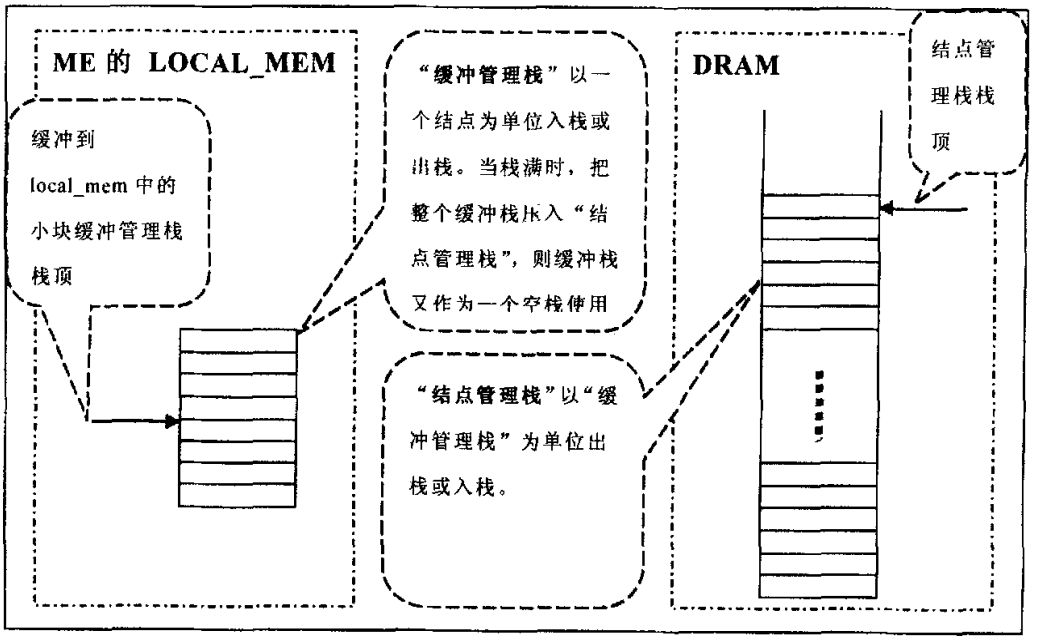
管理结构是一个堆栈，其大小为结点总数＊ 索引大小（即 2 字节），初始化堆栈为空栈，然后把从 1 至结点总数减 1 的索引一一压入栈中，最后压入 0 号索引，用于代表已达栈底，并不分配出去。

分配时，只需弹出一个索引即可，释放时，只需索引压入栈中。

有人可能奇怪，这种方法比起空闲链表法并没什么更多的好处呀？实际上，好处很大，大就大在它是一个数组方式，我们可以把栈顶的若干个索引缓冲到 ME 的寄存器中，比如，一次从内存中读入 8 个长字，则相当于 16 个结点的栈顶在 ME 的寄存器中，其分配所需时间则只需一个时钟周期，等到缓冲的结点用完时，可以从内存中再读一批，这种效果就相当于指令预取一样，从而分配结点的时间可以忽略不计。释放也一样，当缓冲块被写满时，把缓冲块回写到内存的栈顶中，然后缓冲块可以缓冲新释放的结点，堆栈式管理示意图见图表 6－1。

如果采用空闲链表法，则由于每个结点的离散性，必须顺着链一个一个分配，这样，每分配一次就要读一次内存，虽然也可以缓冲多个索引到 ME，但效率就大打折扣了。

图表 6-1 堆栈式结点管理



6.2.1.3 多队列超时

由于内存资源的有限，我们必须不断的清除已经过时的结点，释放其占用的空间，才能不断记录新到的结点。

在 snort 的端口扫描实现中，采用的方法是在每个结点中记录此结点创建或更新的时间戳，然后在每次处理报文时，先遍历整个数据库，把当前时间与每个结点一一比较，若超时，则清除。显然，这种方法极其低效。比如现在数据库中有 1K 个结点，则至少要读内存 1K 次，才能完成一次清理工作，当处理 G 比特流量时，每秒有几十万次的报文到达，则光超时清理工作就要几百万次的内存操作，显然这种方式，无法适应高速网络环境。

在此节，我们提出一种多队列超时算法，见图表 6－2，它可以把清理工作时间复杂度变成是线性的，即每秒要清理的次数只与数据库中的结点数有关。比如上面有 1K 个结点，则每秒基本上只需 1K 次操作即可。

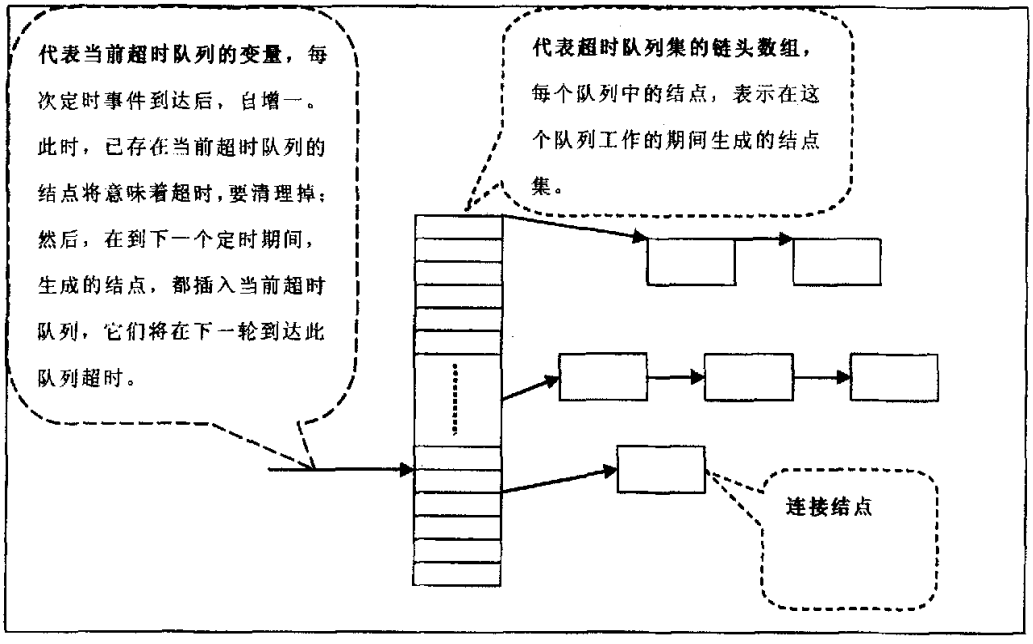
端口扫描结点的超时一般为几秒，我们假设为 2 秒，实际上它只是一个经验值，用户可以根据实际网络情况来调整。

下面我们以一个简单的例子来说明其原理。

创建 10 个超时队列，其编号为 0－9，我们设一个当前队列的指针，再设一个定时器，它每隔 T（1/队列数）秒产生一个定时事件，当产生定时事件后，我们就可以把当前队列指针下移 1，然后在新的定时事件到来时，把创建的结点都

链入当前队列。比如现在当前队列为 0 号队列，当前时间为 t,则当定时事件到来后，则当前队列变为 1，当前时间，则大约为 t+T*1；当再一个定时事件到来时，当前队列为 2,当前时间为 t+T*2;以此类推，当当前队列变为 9 时，当前时间大约为 t+T*9;当当前队列又变回 0 时，则当前时间为 t+T*10，请注意，此时当前队列中的结点就是大约在 1 秒前加入进去的。如果超时值为 1 秒，则当前队列中的结点就意味着全部超时了。所以我们只要在定时事件到来后，把当前队列加 1，然后马上对队列中的结点一一清除，不就可以清除掉相对于当前时间的所有超时结点了吗？本质上说当前队列即可以代表当前时间 t,又可代表 t-T*10,所以在定时事件刚到时，先清除掉的结点实际上大约是 t-T*10 前的结点，在新事件到达前，加入队列的结点是属于 t → t+T 之间，但我们说过，超时只是一个经验值，所以可以把此队列中的结点看作是 t 时间点的结点。当想让队列代表的时间更加精确时，可以增加队列数，比如 1K 个超时队列，则一个队列代表的时间精度就可达 1ms。

图表 6-2 多队列超时算法示意图



6.2.1.4 源 IP 数据库

端口扫描模块之所以记录所有可能的连接结点，本质上是想统计同源 IP 的结点个数，若统计数超过一个阀值，我们就认为此源 ip 对应的主机有扫描行为。

所以我们必须再建一个以源 ip 为关链字的数据库，当一个报文到达时，我们若找到它的连接结点，通过连接结点的源 IP 结点指针，就可以找到源 IP 结点，若为新连接结点，则不但要创建一个连接结点，同时还要查源 IP 数据库，看是否已经存在源 IP 结点，若存在，则对源 IP 结点的统计变量增一，表示连接数增加一个。其源 IP 数据库组织方法和结点分配方法与连接结点数据库的管理方法一样，即超大哈希算法＋堆栈式结点分配算法,这里就不再赘述。

6.2.1.5 多线程同步问题

由于此模块采用多个线程同时处理，所以多个线程在修改数据库时，会产生棘手的线程同步问题。在 ME 程序设计里，有很多种同步方法,请参见[7]，不同方法有不同的特点，所以在具体实现时，要选择较好的同步算法，并尽量缩小同步锁的粒度，以达到较好的并发度。

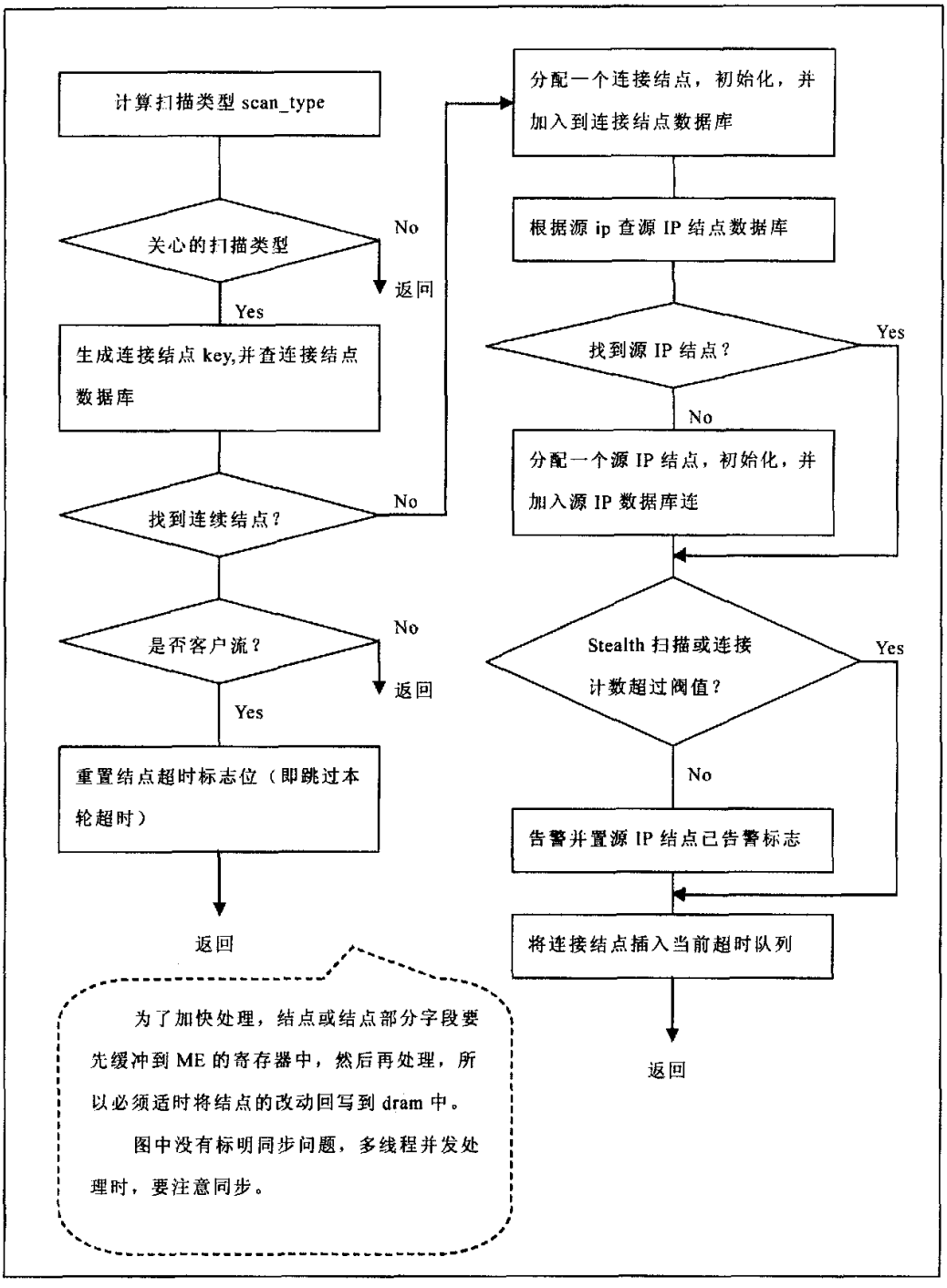
6.2.1.6 告警类型

告警类型	说明
PORTSCAN_SCAN_DETECT	发现 stealth 扫描或某 IP 的端口连接超过阀值，发出告警信息
PORTSCAN_INTER_INFO	对被告警的 IP 的端口连接情况作连续记录
PORTSCAN_SCAN_END	某源 IP 的扫描统计计数变为 0 时，给出告警，意味着此源 IP 的扫描暂时结束。

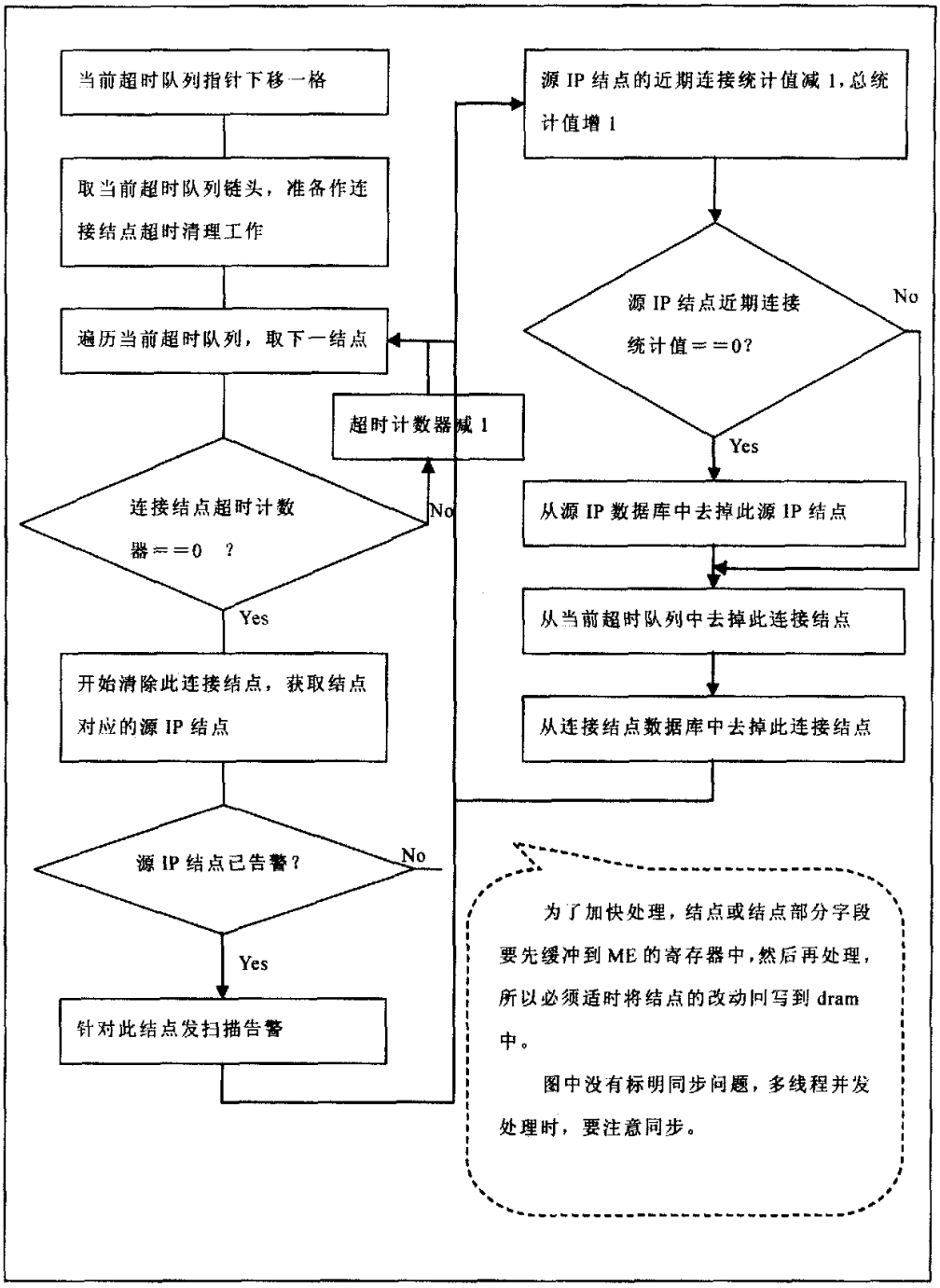
6.2.2 处理流程设计

主要分成两支进行处理，一支正常处理 TCP（或 UDP）报文时，更新连接结点数据库和源 IP 数据库；另一支为定时器激活时，作超时结点清理工作。下面分别从这两支入手来简要说明一下处理流程。

6.2.2.1 更新连接结点数据库处理流程图



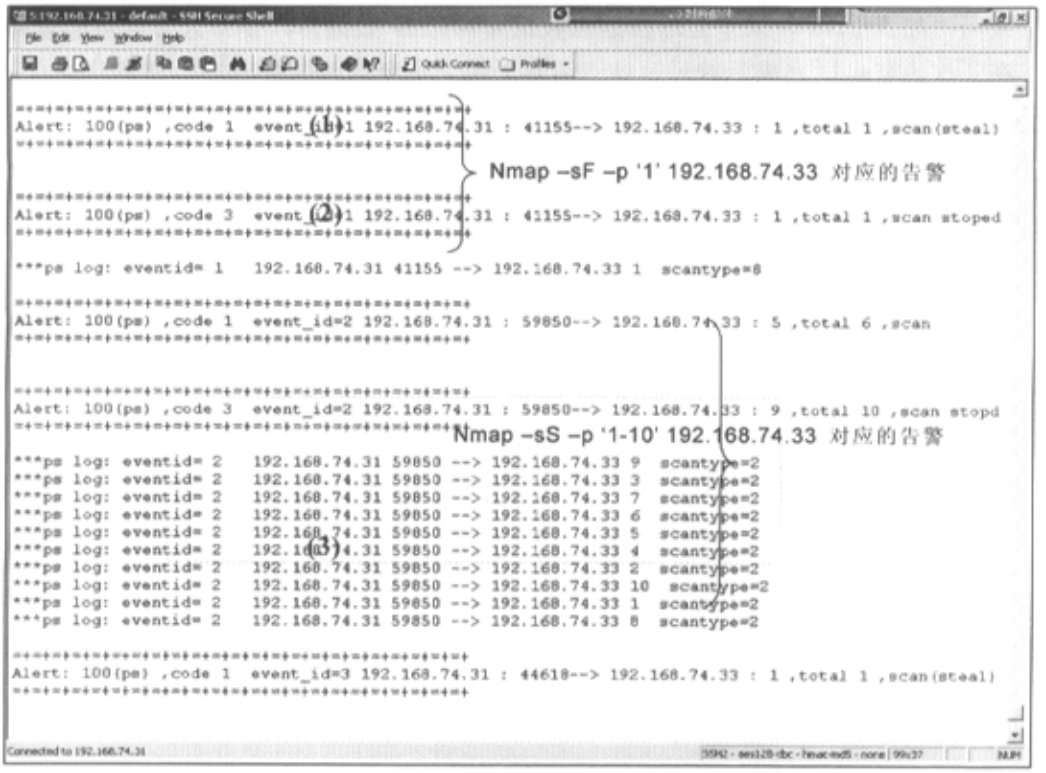
6.2.2.2 结点超时处理流程图



此模块以单线程方式工作时,超时时限设为 3 秒,扫描阀值设为 6(参数借鉴了 snort 的端口扫描经验值)，通过 nmap[48]对它进行全 TCP 连接扫描，半连接扫描，

FIN 扫描,及其它各种 strealth 扫描，模块都正确给出了告警。图 6-3 是端口扫描检测模块的告警截图，图中显示的是两种扫描情况下的告警信息。每当检测到扫描行为后，一般会输出 3 种类型的告警，一种表示发现扫描，如图中(1)所标识；一种表示扫描暂停，如图中(2)所示，一种表示所有发现的扫描连接记录，如图中(3)所示。

图表 6-3 端口扫描检测模块告警截图



6.3 进一步思考

6.3.1 正常 TCP 连接与端口扫描误告警

从 snort 的端口扫描原理来看，是通过统计源 IP 的连接数量是否超过一个阀值参数来判断是否有扫描行为。但这种方法，必然在一定条件上引起误告警，比如：如果当前一个源 IP 正在与一台或多台主机的多个端口进行繁忙的 TCP 通讯，则端口扫描模块必然会记录下这些连接的，但由于它们都处于繁忙状态，所以，不可能通过超时机制把它们从记录中清除掉，如果这样的连接数量超过端口告警阀值时，则会引起 snort 误告警。目前，还没有看到有文献对这种情况进行很好

的解决，一般只是通过调整端口告警阀值来减少误告警。作者以为，如果把已经完成 TCP 连接和未完成 TCP 连接的情况分开考虑，并分别设置阀值，比如将未完成 TCP 连接数的阀值设小点，把已连接的 TCP 阀值设大点，则可减少误报。

6.4 小结

本章首先介绍了各种常用的端口扫描技术，接着详细介绍了端口扫描检测模块的设计，采用了按升序排序外加设标志位的办法合理的解决了双关键字问题，采用的超大 hash 表来作为连接结点数据库和源 IP 数据库，使得查寻结点的效率在几次内就可以完成，对过时结点清除采用了多个超时队列方法，使清除结点基本只与结点数量有关，对结点分配管理采用了堆栈式管理，这种管理看似浪费内存，但它因为可以方便的缓冲到 ME 的寄存器内，所以分配效率极高。最后本文对正常 TCP 连接与端口扫描误告警问题作了简要的讨论。

第七章 其它预处理模块介绍

由于预处理子系统的功能十分丰富，在有限的课题研究时间内不可能把它们在 IXP2400 上一一实现，所以预处理子系统中，本项目课题目前只重点设计和实现了前几章讲的几个模块，本章中的功能，只是作一般程度上的探讨。

7.1 TCP 协议分析

对 TCP 协议进行分析，主要进行两方面的分析，一个是 TCP 连接状态维护，一个是 TCP 流重组。

进行 TCP 协议分析，实际上是 TCP 协议的部分实现，即 TCP 状态机和 TCP 窗口的整流功能的简单实现，参见[4][5]，但 NIDS 系统 TCP 协议分析的实现比起一般主机的 TCP 协议的实现，具有更大的挑战。其挑战就在于因为它要对整个网络的 TCP 流量进行分析，它所需的系统资源是巨大的，这对 NIDS 的计算能力，内存容量，内存带宽都是巨大的压力。

下面将从 TCP 连接状态维护和 TCP 流重组两方面对 TCP 协议分析作以简单探讨。

7.1.1 TCP 连接状态维护

NIDS 系统之所以需要 TCP 连接状态维护，主要是对付无连接攻击行为(stateless attack)。一个正常的 TCP 会话，需要通过 3 次握手来建立连接，然后才能进行正常的 TCP 通讯，在 TCP 会话结束时，也要拆除连接。显然，如果不进行 3 次握手，则 TCP 通讯会被认为是无效的，会被处理机丢弃。可以把这种 TCP 报文称为 TCP 垃圾。这些 TCP 垃圾报文除了占用带宽，增加目标机的处理负担外，对一个 TCP 实现正确的主机，不会产生其它影响，当然，那种通过大量的发送连接请求但不完成连接的攻击，则可能会使目标机产生拒绝服务。

但如果一个 NIDS 系统没有实现 TCP 连接状态维护的话（stateless device），只是按一个个报文进行规则检测的话，则 TCP 垃圾报文对此 NIDS 系统作会产生重要影响。一方面影响是，这些 TCP 垃圾会被无状态 NIDS 系统当成正常 TCP 流量进行规则检测，会浪费消耗 NIDS 大量的系统资源；更可怕的是诸如 stick 工具的攻击，这种工具专门用来与无状态的 NIDS 系统作对，它会按照 NIDS 的内容规则库生成符合规则的 TCP 垃圾，这些垃圾会被无状态 NIDS 系统的规则检测

引擎区配上，并发出告警，如果大量的发送这种 TCP 垃圾，无状态 NIDS 则会产生大量的误告警，这相当于对 NIDS 系统是一种致命的打击。

但当 NIDS 系统引入 TCP 连接状态维护的话，即实现一个简单的 TCP 状态机，则这些 TCP 垃圾会因未建连接而被作为垃圾丢弃，从而不再受 TCP 垃圾的困挠。

7.1.2 TCP 流重组

TCP 流重组对所有的 TCP 协议栈都有实现，即 TCP 的滑窗协议，它保证了送给应用层的数据是一个和发送端一致的有序的数据序列，这便是流的概念。显然，流是连续的，但流数据要通过网络传送，必须被分割成一个个片段来发送，这便是 TCP 分片的内含，由于 TCP 分片在网上传送可能乱序，所以需要在目的端进行整流，这便是滑窗协议的任务。

NIDS 设备一般并不实现一个完整的 TCP 协议栈，它只是对 TCP 报文作简单的检查，如果它没有实现流重组功能，则只能以 TCP 分片作为检测单元进行内容匹配，则如果攻击者故意把攻击特征串通过特殊的工具分散在多个分片中，则这种分片仍会被目的端的主机正确重组，不会影响攻击效果，但对 NIDS 设备，则每个分片都不会再被规则检测引擎匹配上。比如说含“cmd.exe”的字符串的 TCP 报文被认为是攻击行为，但攻击者把它分成'c','m','d','.exe'几个分片来传送，则 NIDS 的规则检测因为无法匹配上任何一个分片报文，从而不会产生告警。

7.2 应用协议分析

应用层协议十分丰富，如 http,telnet,rpc 等等，它们各有自已的特点和特殊性，通过对不同的应用协议作不同的分析，一方面可以去掉不可能含攻击的流量，从而大大减少规则检测引擎的负担；另一方面，对应用协议数据作规整，减少漏报问题。下面仅就 HTTP, TELNET, RPC 协议分析作以简单介绍。

7.2.1 HTTP 协议分析

针对 NIDS 的逃避技术，目前有两种基于 http 协议的逃避技术[43]，它们分别是无效协议解析和无效协议段解码。无效协议解析就是让 IDS 根本找不到 URL，比如如果 IDS 的 HTTP 解码器假设每个请求包只有一个 URL，那么一个包里包含两个 URL，IDS 就不能对第二个 URL 正确解析；无效协议段解码，就是利用 URL 编码方式的多样性或目录表达方式的多样性，故意组合各种方法，如果 IDS 对 URL 编码方法支持度不够或被复杂的目录表达不能理解的话，就不能正确解码出本来正确的 URL。

http 协议分析就是针对上面的逃避技术而设计产生的，目前 snort 的 http 预处理器就是通过对各种编码的 URL 进行规范的表达，来让规则检测引擎识别出攻击行为，从而减少漏报率。

7.2.2 TELNET 协议分析

TELNET 协议是非常简单的一种协议，它是在正常的 TCP 流中插入以 255 开头的协议命令，来进行协议协商的。由于这种插入的协议命令可以在流中任何位置，所以攻击者通常用它故意把一个连续的攻击特征字符串分隔开，从而使 IDS 的规则检测失配，所以这也是一种攻击逃避技术。

TELNET 协议分析，简单的说，就是通过去掉插入在流中的协议命令，使攻击特征串暴露出来，从而被 IDS 系统检测到。

7.2.3 RPC 协议分析

一些应用如 NFS, NIS 都是基于 sun 的 RPC 协议，RPC 协议即可以使用 TCP, 也可以用 UDP 作为载体，为了能通过 TCP 或 UDP 以一致的方式传递消息，RPC 协议定义了一个“record”结构来表达一条消息，而且 record 可以被分成多个 record fragment，即 rpc 分片，即使多个分片仍位于同一个报文中，但有由分片之间被分片控制字段分隔，从而使攻击特征串变得不连续，产生了象基于 IP 分片或 TCP 分片的逃避技术一样的 rpc 分片逃避技术。Rpc 协议分析就是要将 rpc 分片重组起来，形成一条条完整的消息，从而检测出潜在的攻击行为。

7.3 小结

俗话说“道高一尺，魔高一丈”，攻击者越来越“精心”设计攻击行为,借以逃避 IDS 系统的监测。

TCP 连接状态维护使得“TCP 垃圾”被真正作为垃圾而丢弃，TCP 流重组让攻击者的“分身术”失效。应用协议分析，就是通过对协议作仔细的分析，从而使逃避技术失灵。目前，基于 http 的逃避技术主要有无效协议解析和无效协议段解码两种技术；基于 telnet 协议的主要是通过插入协议命令打断攻击特征串的方法来达到逃避目的；基于 rpc 协议的主要是通过对 rpc 请求进行分片来达到逃避检测的目的。

第八章 结论

8.1 研究总结

本论文主要研究了 NIDS sensor 的预处理子系统。预处理子系统功能相当的丰富，每个子功能都具有相对的独立性。解码模块主要完成对捕获的网络流量按网络协议进行解码；IP 分片重组模块完成对 IP 分片报文的重组及对 IP 分片攻击的检测；端口扫描模块主要完成对网络连接的统计，分析潜在的扫描行为，并给出告警；流重组模块主要完成对 TCP 协议的状态维护和 TCP 流数据的重整工作，如果没有流重组，基于签名的规则检测模块就可能有大量的漏检；基于应用协议的分析，一方面可以大大减少规则检测模块的匹配报文的数量，从而提高整个 NIDS 的检测效率，另一方面对应用协议进行解码和规整工作，会使规则检测更加准确。

本论文主要对预处理子系统中的解码，ip 分片重组，端口扫描 3 个子功能作了详细的研究和总结，并给出了针对网络处理器的特点作了精心优化的解决方案，下面对这 3 个子模块的创新点作简单的总结。

解码模块作为 microblock 实现，文中对解码数据结构的设计作了详细分析，设计出了解码结构只有 5 个长字，比起 snort 的 packet 结构可以说有了质的不同，非常适合网络处理器处理，通过测试，实现了线速的报文解码功能。

Ip 分片重组模块的实现分 CC 和 MB 两部分，CC 部分用于完成正常的分片重组及告警；MB 部分主要完成自保机制，两者有机配合，可以抵抗 IP 分片洪流的攻击。文中对此模块的关键点，即分片存储和重组作了详细探讨，并提出了分区合并分片重组算法，使得分片对内存的要求大大减少（比起 snort 的算法）。

端口扫描模块的设计，采用了按升序排序外加设标志位的办法合理的解决了双关键字问题；采用的超大 hash 表来作为连接结点数据库和源 IP 数据库，使得查寻结点的效率在几次内就可以完成；对过时结点清除采用了多个超时队列方法，使清除结点基本只与结点数量有关；对结点分配管理采用了堆栈式管理，这种管理看似浪费内存，但它因为可以方便的缓冲到 ME 的寄存器内，所以分配效率极高。

由于课题被分成多个子系统，分别由不同的人来设计和实现，所以若对规则检测子系统有兴趣，请参见文献[52]；规则解析模块及系统测试请参见文献[53]。

8.2 进一步工作展望

一个完整的NIDS系统极其复杂，往往需要数人数年的开发才能完成。由于本项目课题研究的时间和实验条件的限制，所以只研究实现了其中的一部分功能，代码运行还不太稳定，还需要不断优化，提高性能。所以，在以后的研究中，还有大量的工作要做。

预处理子系统已实现的模块主要有以下几方面需要改进：

- 报文解码模块要加强代码优化，如访问dram,sram改成异常IO方式，数据结构在ME中的缓冲做更精细的设计，尽量缩小数据结构变量的作用域，减少寄存器压力。
- IP分片重组模块可以考虑把重组功能放在ME中实现，以求进一步提高重组能力,但是否有必要，还有待进一步论证。
- 端口扫描模块多线程同步问题还没有很好解决，目前还暂时采用单线程处理，以后要考虑改成多线程处理。
- 各模块的可调参数目前还基本采用snort软件使用的默认值，应该通过实验找出本系统的较佳参数。
- 已实现的模块目前还不太稳定，仍需要不断的调试和测试。

预处理子系统还有很多功能，也要在下一阶段逐步实现，如：

- 增加对各种二层协议的支持，以适应不同的网络类型。
- 对流重组及各种应用协议分析作深入的研究，并给出适合网络处理器架构的高效的设计与实现方案。
- 在不断增加预处理模块功能时，一个IXP2400能否胜任，还要进一步验证
- ipv6网络将越来越流行，针对IPV6网络的网络安全将会受到关注。目前，本课题还只是对ipv4报文进行了预处理工作，若有可能，应增加对ipv6网络的支持。

参考文献

[1] snort2.0源码，http://www.snort.org

[2] zebra-0.94源码，http://www.zebra.org

[3] linux2.4.18源码，http://www.linux.org

[4] Andrew S.Tanenbaum著，熊桂喜 王小虎译，计算机网络（第3版），清华大学出版社，1998

[5] W.Richard Stevens,TCP/IP协议详解1,2卷,机械工业出版社，2002.1

[6] Jay Beale, James C. Foster, Jeffrey Posluns, Brian Caswell, snort2.0 instrusion detection, Syngress Publishing, 2003

[7] Erik J. Johnson, Aaron R. Kunze, ixp 2400/2800 programming-the complete microengine codeing guide, 2003, intel press

[8] NSS-Public_Test_Report-880 Gigabit instrusion detection system Group Test(Edition 2), NSS Group, 2003

[9] NSS-public_test_report-501 IDS Group Test (Edition 3) ,NSS Group ,2002

[10] 2003-2004 年度中国市场主流IDS产品评测技术报告, www.ccident.com

[11] Janani Thanigachalam, Working on the IXP2400, http://wail.cs.wisc.edu/ixpdev/ixp.html

[12] Intel corp, Case Study: IDT* PAX.port* 2500 content inspection engine (CIE) and Intel® IXP2400 network processor --New Ways to Alleviate the Content Inspection Bottleneck in Security Systems, http://developer.intel.com

[13] Consystant corp, The Snort Network Intrusion Detection System on the Intel® IXP2400 Network Processor, http://snort.deceng.com/, 2003

[14] Herbert Bos, Kaiming Huang, A network intrusion detection system on IXP1200 network processors with support for large rule sets, 2004

[15] Gokhan Memik , William H. Mangione-Smith, Design and Evaluation of a Network Processor Accelerator for Layer Seven Applications

[16] Chris Clark, Wenke Lee, David Schimmel, et al. A Hardware Platform for Network Intrusion Detection and Prevention.

[17] Intel, Intel IXP2400 NetworkProcessor Hardware Reference Manual, November 2003

[18] Intel, Intel Internet Exchange Architecture (IXA) Software Building Blocks Applications: Building Blocks, July 2003.
[19] Intel, Intel IXP2400/IXP2800 Network Processor: Development Tools User' s Guide, July 2003.
[20] Intel, Intel IXP2400/IXP2800 Network Processor: Microengine C Compiler Language Support Reference Manual, June 2003.
[21] Intel, Intel IXP2400/IXP2800 Network Processor: Programmer' s Reference Manual, July 2003.
[22] http://www.cisco.com/global/CN/products/sc/ids/cisco_research.shtml
[23] 王自亮,罗守山,杨义先，入侵检测系统的测试与评估，中宽网 http://www.cntele.com
[24] 张 杰，戴英侠，入侵检测系统技术现状及其发展趋势，中宽网 http://www.cntele.com
[25] 黑客攻击与反攻击(2)----黑客攻击行为特征分析与反攻击技术，http://www.infosec365.com.cn
[26] 千兆防火墙的两种技术实现：NP 和 ASIC，http://www.infosec365.com.cn
[27] 张少波，高速 IDS 面临的技术瓶颈与发展趋势，http://network.ccidnet.com/pub/article/c238_a66358_p1.html
[28] Wu Yanga, Bin-Xing Fanga, Bo Liub, Hong-Li Zhang, Intrusion detection system for high-speed network. Computer Communications 27 (2004) 1288 - 1294
[29] 唐洪英，付国瑜，入侵检测的原理与方法，重庆工学院学报，2002.4：71－73
[30] 刘伟娜，高性能网络入侵检测研究，西安交通大学硕士毕业论文，2003.1
[31] 联想网御入侵检测系统产品白皮书电子版
[32] 启明星辰天阗黑客入侵检测产品白皮书电子版
[33] 清华紫光比威入侵检测系统技术白皮书电子版
[34] 天融信网络卫士入侵检测系统技术白皮书电子版
[35] 网警网警分布式入侵检测系统技术白皮书电子版
[36] 中科安胜入侵检测系统技术白皮书电子版
[37] Chris Clark, Wenke Lee, David Schimmel, Didier Contis, Mohamed Kone, Ashley Thomas, A Hardware Platform for Network Intrusion Detection and Prevention, Center for Experimental Research in Computer Systems (CERCS), Georgia Institute of Technology, Atlanta, GA, USA

[38] Herbert Bos, Kaiming Huang, A network intrusion detection system on IXP1200 network processors with support for large rule sets, Technical Report 2004-02

[39] Hyeyoung Cho, Daeyoung Kim, Juhong Kim, Yoonmee Doh, Jongsoo Jang, Network Processor Based Network Intrusion Detection System

[40] 常见 IP 碎片攻击详解, http://linuxeden.com

[41] 如何突破各种防火墙的防护, http://www.linuxbyte.net

[42] Modem 优化大全,http://www.pconline.com

[43] 两种基于 HTTP 的通用 IDS 躲避技术, http://www.pcdog.com

[44] 反 NIDS 技术介绍, http://www.xfocus.net

[45] 端口扫描技术全攻略, http://www.hacker.com.cn

[46] 高级扫描技术及原理介绍, http://www.enet.com.cn

[47] 网络扫描技术的原理、比较、建议, http://www.enet.com.cn

[48] 扫描之王——Nmap 详解,http://www.enet.com.cn

[49] http://packetstormsecurity.org/Exploit_Code_Archive/teardrop.c

[50] http://packetstormsecurity.org/0005-exploits/jolt2.c

[51] 想躲避 IP 分片攻击？先了解-基于 IP 分片的攻击方法, 瑞星反病毒资讯网

[52] 林若川，基于网络处理器的入侵检测引擎的设计与实现，华南理工大学硕士论文，2005

[53] 吴静波，基于网络处理器的入侵检测系统规则解析模块的设计与实现，华南理工大学硕士论文，2005

在学期间已发表（包括已接受待发表）的论文，以及已投稿、或已成文打算投稿、或拟成文投稿的论文情况（只填写与学位论文内容相关的部分）：

序号	作者（全体作者，按顺序排列）	题　　目	发表或投稿刊物名称、级别	发表的卷期、年月、页码	相当于学位论文的哪一部分（章、节）	被索引收录情况
1	滑国青	一种 CLI 命令解析器的改进设计	2004年度华南理工大学电子与信息学科学科研究生学术研讨会	2004/12 P425-P428		

注：在“发表的卷期、年月、页码”栏：

1 如果论文已发表，请填写发表的卷期、年月、页码；

2 如果论文已被接受，填写将要发表的卷期、年月；

3 以上都不是，请据实填写“已投稿”，“拟投稿”。

不够请另加页。

致谢

感谢我的导师张凌教授为我们提供了优良的科研环境和氛围以及先进的实验设备，提供了研究最新科研项目的机会，使我们接触到潮流的脉搏，把握了计算机学科研究性和实用性结合的发展趋势。

感谢网络中心的董守斌，袁华,周杰，陈启愉等老师对我们在生活上、学习上无微不至地关怀和指导。

感谢 NP－NIDS 项目组的李家春指导老师，她为我们项目能够紧张而有序地进行做出了巨大的努力，对我们的论文提出了许多宝贵的建议。感谢项目组的林若川，吴静波，窦小龙同学对我的研究工作的积极配合；感谢肖红，许宪成，洪雪玉，陈宝刚等博士对我学习和生活上的帮助和鼓励。

感谢中兴通讯“高性能 IPV6 路由器”项目组的所有同仁，他们使我学习到了许多先进的项目研发经验，也让我感受到了协作研发的无穷乐趣，那一年的研发生活，至今让我记忆犹新。

特别感谢我的父母对我的养育之恩，她们是我工作，生活，学习的最坚强的后盾。特别感谢我的妻子，她在生活上对我晰心照料，学习上对我积极鼓励。

最后，我由衷地感谢“开源社区”，感谢 GPL 协议，正是“开源人”的无私胸怀，将研究成果，特别是源代码贡献出来，才使我们能够快速的进入到某个研究领域并有所成就，在本项目中，也正是通过对 snort,zebra，linux 等开源软件的研究、学习、借鉴，我们才能如此快速对本课题有所成果。

滑国青

二零零五年六月

于华南理工大学网络中心