[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
H04L 12/56 (2006.01)
H04L 29/06 (2006.01)

[12] 发明专利说明书

专利号 ZL 200410045613.3

[45] 授权公告日 2008 年 7 月 23 日

[11] 授权公告号 CN 100405784C

[22] 申请日 2000.6.30

[21] 申请号 200410045613.3

分案原申请号 00813675.0

[30] 优先权

[32] 1999.6.30 [33] US [31] 60/141903

[73] 专利权人 倾向探测公司

地址 美国加利福尼亚州

[72] 发明人 R·S·迪茨 J·R·迈克斯纳
A·A·科彭哈维 W·H·巴雷斯
H·A·萨尔基相
J·F·托尔格森

[56] 参考文献

US 5787253 A 1998.7.28

US 5414704 A 1995.5.9

计算机网络. Andrew S. Tanenbaum，21.28，清华大学出版社. 1998

审查员 赵 晨

[74] 专利代理机构 中国专利代理(香港)有限公司

代理人 罗 朋

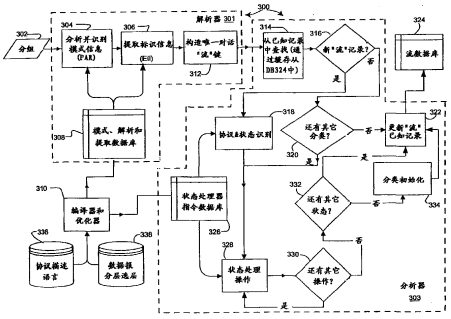
权利要求书 3 页 说明书 137 页 附图 28 页

[54] 发明名称

用于监控网络流量的方法和设备

[57] 摘要

一种监控器和一个方法，用于检查通过计算机网络上一个连接点的分组。 每个分组符合一个或多个协议。 该方法包括从分组拦截设备接收分组并在分组上执行一个或多个解析/提取操作以创建包括该分组的选中部分的函数在内的解析器记录。 解析/提取操作取决于该分组所符合的一个或多个协议。该方法还包括查找包含以前遇到的对话流的流－入口的流－入口数据库。 该查找使用选中的分组部分并确定该分组是否是现有的流。 如果该分组是现有的流，该方法把该分组归类为属于找到的现有的流，并且如果该分组是新流，该方法在流－入口数据库中为该新流存储一个新的流－入口。 包括用该新流－入口识别的将来分组的标识信息。 若该分组是现有的流，该方法更新该现有的流的流－入口。这样的更新可以包括存储一个或多个统计测量。 在流的任意阶段，都维护它的状态，并且该方法执行已经识别出的状态的任意状态操作以促进识别该流的过程。 该方法就这样实时检查通过该连接点的每个分组直到确定与该流相关的应用程序为止。



1. 一种方法，包括：检查通过计算机网络上一个连接点的分组，此步骤包括：

对于一个分组，

(a)接收该分组；

(b)识别该分组中使用的协议，包括识别该分组的基本协议和该分组中所用协议的子协议，所述识别根据所接收到的相应于多个符合分层模型的协议的协议描述集，

其中，在所述协议描述集中，位于一个特定层上的一个特定协议的协议描述包括：

(i)该特定协议的零个或多个子协议，对于该特定协议的任意特定子协议，该分组在其本身中的一个或多个位置上包括与该特定子协议有关的信息，

(ii)该分组中存储与该特定协议的任意子协议有关的信息的所述一个或多个位置，和

(iii)对于所述特定层上的所述特定协议要对该分组执行的零个或多个协议特定操作；以及

(c)根据该分组的所述基本协议和该分组中所用协议的子协议对该分组执行由所述协议描述集指定的所述协议特定操作。

2. 根据权利要求 1 的方法，其中执行协议特定操作的(c)步可以对子协议的任意子协议递归执行。

3. 根据权利要求 1 的方法，其中(c)步中执行哪些协议特定操作取决于分组的内容，这样该方法能够根据分组的内容适应不同协议。

4. 根据权利要求 1 的方法，还包括：

在存储器中存储一个数据库，该数据库从协议描述集产生并包括包含与可能的协议有关的信息的数据结构，为了定位任意协议的子协议有关信息而对该数据结构进行了组织，由一组一个或多个索引指向该数据结构内容，由包括有效指示的索引值的特定集合指向数据库记录，

其中子协议有关信息包括子识别模式，

其中执行协议特定操作的(c)步包括，在从基层开始的任意协议层上，在特定协议的分组中查找子协议字段，该查找包括检索该数据结

构直到找到有效记录，并

借此为了快速查找而用索引集对该数据结构进行配置。

5．根据权利要求 4 的方法，其中协议描述以协议描述语言提供，该方法还包括:

编译 PDL 描述以产生该数据库。

6．根据权利要求 4 的方法，其中该数据结构包括一组数组，每个数组由一个第一索引标识，对每个协议至少有一个数组，每个数组还由一个第二索引进行指示，第二索引是分组中存储子协议有关信息的位置，这样在该数据结构中找到一个有效记录就为找到一个已识别协议的子识别模式提供了分组中的位置。

7．根据权利要求 6 的方法，其中每个数组进一步由一个第三索引进行索引，第三索引是存储了子协议有关信息的分组中的区域的大小，这样在数据结构中找到一个有效记录就为找到所述子识别模式提供了分组中区域的位置和大小。

8．根据权利要求 7 的方法，其中该数据结构被按照压缩方案进行了压缩，该压缩方案利用了数据结构中有效记录的稀疏性。

9．根据权利要求 8 的方法，其中压缩方案把两个或更多没有冲突的公共记录的数组合并起来。

10．根据权利要求 4 的方法，其中该数据结构包括一组表格，每个表格由一个第一索引标识，对每个协议至少有一个表格，每个表格还由一个第二索引进行指示，第二索引是子识别模式，该数据结构还包括一个表格，它为每个协议提供分组中存储子协议有关信息的位置，这样在该数据结构中找到一个有效记录就为找到一个已识别协议的子识别模式提供了分组中的位置。

11．根据权利要求 10 的方法，其中该数据结构被按照压缩方案进行了压缩，该压缩方案利用了表格集中有效记录的稀疏性。

12．根据权利要求 11 的方法，其中压缩方案把两个或更多没有冲突的公共记录的表格合并起来。

13．根据权利要求 1 的方法，其中协议特定操作包括分组上的一个或多个解析和提取操作以提取出分组的选中部分，进而为标识该分组为属于一个对话流而形成选中部分的函数。

14．根据权利要求 1 的方法，其中协议描述用协议描述语言提供。

15．根据权利要求 14 的方法，还包括:

编译 PDL 描述以产生一个数据库并把该数据库存储在存储器中，该数据库从协议描述集产生并包括一个包含与可能的协议有关的信息

的数据结构，为了对定位任意协议的子协议有关信息而对该数据结构进行组织，由一组一个或多个索引指向该数据结构内容，由包含有效指示的索引值的特定集合指向数据库记录，

其中子协议有关信息包括子识别模式，

其中执行协议特定操作的步骤包括，在从基层开始的任意协议层上，在特定协议的分组中查找子协议字段，该查找包括检索该数据结构直到找到有效记录，并

从而为了快速查找而用索引集对该数据结构进行配置。

16. 根据权利要求 13 的方法，还包括:

查找包含零个或多个流－入口的流－入口数据库，对每个以前遇到的对话流至少有一个流－入口，该查找使用至少若干选中的分组部分并确定分组是否匹配现有的流－入口:

如果分组是现有流，就把它归类为属于找到的现有的流；并且

如果分组是新流，就在流－入口数据库中为这个新流存储一个新的流－入口，包括用该新流－入口识别的将来分组的标识信息;

其中解析和提取操作取决于零个或多个分组报头的内容。

17. 根据权利要求 13 的方法，其中协议特定操作还包括一个或多个状态处理操作，它们是该分组的流的状态的函数。

18. 根据权利要求 1 的方法，其中协议特定操作包括一个或多个状态处理操作，它们是该分组的流的状态的函数。

用于监控网络流量的方法和设备

技术领域

本发明涉及计算机网络，尤其涉及在数据网络中所传递的分组的实时说明有关，包括按照协议和应用程序进行的分类。

对有关申请的交叉引用

本申请对美国临时专利申请序列号 60/141，903 题为“用于监控网络流量的方法和设备”中的利益提出权利要求，所要求的权利由发明人 Dietz 等人所有，他在 1999 年 6 月 30 日提交了该申请并转让给 APPTITUDE 有限公司，该公司是本发明的受让人。

版权公告

背景技术

很久以来就一直需要有一个网络活动状况监控器。但在最近 Internet 和其它 internets 大量流行的情况下这种需要变得更加迫切，“internet”是任意一个对大量网络进行互联而形成的更大的单一网络。随着用作客户机端集合—这些客户机端从网络上的一个或多个服务器获取服务—的网络的增长，能够对这些服务的使用进行监控并对它们做出相应的评定就显得愈加重要。例如，像哪些服务(即应用程序)正在被使用、谁在使用它们、它们多长时间被访问一次以及每次访问多长时间这样的客观信息在这些网络的维护和后继操作中非常有用。尤其重要的是被选中的用户能够远程访问网络以便实时产生网络使用报告。同样地，也需要一个实时网络监控器能够提供警报通知选中的用户网络或站点上可能出现了问题。

现有技术中的一种监控方法使用日志文件。在这种方法中，可以通过查看日志文件对选中的网络活动进行回顾分析，日志文件由网络服务器和网关共同维持。日志文件监控器必须访问这个数据并分析(“我的”)它的内容以确定关于服务器或网关的统计信息。但这

种方法存在几个问题。首先，日志文件信息并未提供实时使用图；第二，日志文件库不支持全部信息。这个方法依赖于由大量网络设备和服务器所维持的日志，这就需要对信息进行提炼和对比。并且有些时候信息不能简单地被任意网关或服务器直接用来产生日志文件条目。

例如，关于 NetMeeting®(Microsoft 公司，雷蒙德，华盛顿州)会话的信息就是一个这样的情况，在 NetMeeting 会话中两台计算机在网络上直接连接，而服务器和网关根本看不到在它们之间所传送的数据。

创建日志文件的另一个缺点是这个方法要求使能网络元素的数据记录特性，给设备增加了实实在在的负载，这导致网络性能随之下降。另外，日志文件增长得很快，没有能够存储它们的标准存储装置，并且它们需要大量的维护开销。

通过 Netflow®(Cisco 系统有限公司，圣何塞，加利福尼亚州)、RMON2 以及其它可用于实际监控网络的网络监控器，它们降低了应用内容的可见度，而且一般在提供网络层信息上也有限制。

模式匹配解析器技术已经是众所周知，在该技术中对分组进行解析并且应用了模式过滤器，但这些在进入协议栈检查分组的进入深度上也受限制。

一些现有技术中的分组监控器把分组分类成连接流。术语“连接流”一般用于描述单个连接所涉及到的所有分组。另一方面，对话流是作为一个活动—例如，因为客户机的请求而在服务器上运行一个应用程序—的结果在任意方向上所交换的分组序列。希望能够对对话流而不只是连接流进行识别和分类。这样做的原因是一些对话流涉及不止一个连接流，有些甚至涉及到客户机和服务器之间的多个分组交换。这在使用像 RPC、DCOMP 和 SAP 这样的客户机/服务器协议时尤其正确，这些协议使得能够在对服务有任何使用之前对其进行设置或定义。

这种情况的一个实例是 SAP(服务广告协议)，一个用来识别服务和隶属于网络的服务器地址的 NetWare(Novell 系统，Provo，犹他州)协议。在初始交换中，客户机可以发送一个 SAP 请求给服务器以要求打印服务。服务器随后将发送一个 SAP 应答，该应答把一特定

地址—例如，SAP#5—标识为该服务器上的打印服务。这样的响应可以用来更新路由器中的表格，例如通常所说的服务器信息表。已经无意中看到过这个响应或者访问该表格(通过有服务信息表格的路由器)的客户机便会知道特定服务器的 SAP#5 是一个打印服务。因此，要想在服务器上打印数据，这样的客户机无需产生打印服务请求，只要简单地把数据发送出去以在指定的 SAP#5 上进行打印。像前一个交换那样，传输要打印的数据也涉及客户机和服务器之间的交换，但要求第二个连接利并因此独立于初始交换。为了降低对话交换断开的可能性，希望网络分组监控器能够把第一和第二个交换“虚拟连接”—也就是链接—在一起。如果是同一客户机，两个分组交换将被正确地标识为同一对话流的一部分。

其它可能产生断开的流的协议包括 RPC(远程过程调用)；DCOM(分布式组件对象模型)，以前称作网络 OLE(Microsoft 公司，雷蒙德，华盛顿州)；和 CORBA(公共对象请求代理结构)。RPC 是来自 Sun 微系统(Paloalto，加利福尼亚州)的一个编程接口，它允许一个程序使用远程机器中的另一个程序的服务。DCOM 是 Microsoft 与 CORBA 相对应的协议，它定义了允许通过网络远程运行那些对象—自主式软件模块对象—的远程过程调用。CORBA 是来自 OMG(对象管理组织)的标准，用于在分布式对象之间进行通信，它提供了一种方式能够执行用不同语言所写运行在不同平台上的程序(对象)，而不管它们是否存在于网络中。

因此，需要一个能够在流量很大的网络上连续不断地分析所有用户会话的网络监控器。这样的监控器应该能够对通过网络上任意点的所有信息(也就是通过网络中任意位置的所有分组和分组流)进行非插入、远程探测、特性表示、分析和捕捉。网络监控器应该不仅能探测和分析所有的分组，而且能够针对这些分组中的每一个确定协议(例如，http、ftp、H.323、VPN 等)、协议中的应用/用途(例如，声音、视频、数据、实时数据等)以及每个应用或应用环境(例如，所选的选项、所投递的服务、持续时间、日时、所请求的数据等)中终端用户的使用模式。而且，网络监控器不应该依赖于像日志文件这样的服务器驻留信息。相反，它应该给像网络管理员或 Internet 服务提供商(ISP)这样的用户提供装置以对网络活动进行客观的测

量和分析；自定义要收集并分析的数据的类型；承担实时分析；并接收网络问题的及时通知。

再次考虑前面的 SAP 实例，因为本发明的一个特性是正确标识第二个交换为与服务器的打印服务相关联，甚至在客户机不相同时也能够识别这样的交换。区别本发明和现有技术中的网络监控器的特性是本发明能够识别出属于同一个对话流的已断开的流。

很多发明人已经认识到了监控网络通信中的数据值。Chiu 等人在题为“用于网络会话的实时监控和局域网的设备和方法”的美国专利 5，101，402(“402 专利”)中描述了一个用于在计算机网络的会话层上收集信息的方法。402 专利为几种特定类型的分组指定了提取信息从而识别分组会话的固定位置。例如，如果出现了一个 DECnet 分组，402 专利就查看分组中的六个字段(在六个位置)以识别该分组的会话。另一方面，如果出现了一个 IP 分组，就为 IP 分组指定包含六个不同位置的一个不同集合。随着协议的增殖，明确指定所有可能的位置并确定会话变得越来越难。同样地，添加一个新协议或应用也很困难。在本发明中，对特定类型的分组从该分组中的信息适应地确定要检验的位置以及要从任意分组中提取信息。对于寻找什么和在哪里寻找它们以形成标识签名没有固定的定义。例如，本发明的监控器实现适合于对来自老一些的以太网类型 2(或版本 2)DIX(DIGITAL－Intel－Xerox)分组的 IEEE 802.3 分组进行不同的处理。

402 专利系统能够一直识别到会话层。在本发明中，对于任意特定协议要检验的层数也有所变化。此外，本发明能够一直检验到足够唯一标识到一个必需层的任何层，甚至所有通往应用层(在 OSI 模型中)的路线。

还知道其它一些现有技术中的系统。Phael 在题为“网络监控设备和系统”的美国专利 5，315，580 中描述了一个只处理随机选中的分组的网络活动监控器。Nkamura 在题为“网络监控系统”的美国专利 4，891，639 中讲授了一个网络监控系统。Ross 等人在题为“用于网络分析的方法和设备”的美国专利 5，247，517 中讲授了用于分析和监控网络活动的方法和设备。McCreery 等人在题为“分析 Internet 活动的设备和方法”的美国专利 5,787,253 中描述了一个

Internet 活动监控器，该监控器在 Internet 协议层上对分组数据进行解码。McCreery 方法对 IP 分组进行解码。它对每个分组进行解码操作，并因此耗费了对已识别流和未识别流进行处理的总开销。在本发明的监控器实现中，为每个流构造一个签名，这样就很容易识别该流的将来分组。当该流中的一个新的分组到达时，识别方法能够从它最后离开的地方和构造的新签名开始去识别流中的新分组。

网络分析员应该能够分析多种不同的协议。在底层，有很多用在数字通信中的标准，包括以太网、HDLC、ISDN、Lap B、ATM、X.25、帧中继、数字数据服务、FDDI(光纤分布式数据接口)、T1 以及其它标准。这些标准中有很多都采用了不同的分组和/或帧格式。例如，数据在 ATM 和帧中继系统中是以 53 个八位位组(也就是字节)的固定长度分组(称为“单元”)进行传输的。需要几个这样的单元来组成可能包含在其它一些协议为相同的有效负载而使用的分组的信息—例如在使用帧中继标准或以太网协议的对话流中。

为了使网络监控器能够分析不同的分组或帧格式，监控器需要能够在每个分组上执行协议特定操作，每个分组都带有遵守不同协议的信息并与不同的应用有关。例如，监控器需要能够把不同格式的分组解析成字段以便理解封装在不同字段中的数据。随着可能的分组格式或类型数量的增加，需要用来解析这些不同分组格式的逻辑数量也随之增加。

还有一些现有技术中的网络监控器对单个分组进行解析并在不同的字段上寻找信息以便用它们构造一个用于标识分组的签名。Chiu 等人在题为“用于网络会话的实时监控和局域网的设备和方法”的美国专利 5,101,402 中描述了一个用于在计算机网络的会话层上收集信息的方法。在这个专利中，为特定类型的分组指定了固定的位置。例如，如果出现了一个 DECnet 分组，Chiu 系统就查看分组中的六个特定字段(在六个位置)以便识别该分组的会话。另一方面，如果出现了一个 IP 分组，就检验一个包含六个位置的不同集合。该系统只从最低层查看到协议层。随着协议的增殖，明确指定所有可能的位置以确定会话变得越来越困难。同样的，添加一个新协议或应用也很困难。

希望对特定类型的分组能够适应地确定位置以及要从任意分组

中提取的信息。用这种方式，可以用协议－有关和分组－内容－有关定义来定义一个最佳签名，分组－内容－有关定义规定了找什么和到哪里去找它以形成签名。

这样还需要网络监控器能够适合不同的协议和不同的应用程序或为它们进行特制。这样还需要网络监控器能够容纳新协议和新的应用程序。还需要用于指定新协议和新级别，包括新应用，的装置。还需要一种机制来描述协议特定操作，例如包括什么信息与分组有关、什么分组需要被解码以及指定解析操作和提取操作。还需要一种机制来描述在处于流的一种特定识别状态的分组上执行的状态操作以便促进识别该流。

收集通过网络中一点的分组上的统计信息比简单地对每个分组进行计数要有利。通过在与对话流有关的流－入口中保持统计测量，本发明的实施方案能够实时收集特定的度量，否则将做不到这一点。例如，希望为对话中的每个交换保持与基于整个流的双向对话有关的度量。通过保持流的状态，本发明的实施方案还能够确定与流的状态有关的特定度量。

现有技术中使用统计度量的多数流量监控器只收集终点和会话终点的相关统计信息。这种通用度量的实例包括分组计数、字节计数、会话连接时间、会话超时、会话和传输响应以及其它。所有这些都是对可以直接与单个分组中的事件有关的事件进行处理。这些现有技术中的系统不能收集与网络中流的完整分组序列或相同流的几个已断开的序列有关的性能度量。

应用数据分组上基于时间的度量是很重要的。如果能够为后来的分析存储并转发所有的时间戳和相关数据，就可以确定这样的度量。但是，在面对每秒钟有数千或数百万个对话甚至更快的网络时，存储所有这些数据，即使是压缩过的数据，也将占用过多的处理时间、存储器以及管理器下载时间，因而是不可行的。

这样就需要从那些从流中的分组积累起来的统计测量中保持并汇报基于时间的度量。

网络数据被适当地模拟成对象总体而不是样本。这样，所有数据都需要进行处理。因为应用协议的本性，只对一些分组进行采样无法给出关于流的好的测量。只要漏掉了一个重要分组，例如数据将

要在其上进行传输或者应用程序将要在其上运行的指定的一个附加端口，就会导致有效数据的丢失。

这样还需要从那些从流中的每个分组积累起来的统计测量中保持并汇报基于时间的度量。

还需要确定与事件序列有关的度量。一个好的例子是相对振动。测量从一个方向上一个分组的末端到达同一方向上带有相同签名的另一个分组的时间就收集了与正常振动有关的数据。这种类型的振动度量对于在分组网络中测量广泛的信号质量是很有用的。然而，它对正在一个分组集群中传输的有效负载或数据项来说却不够明确。

由于分组进入系统的高速度，本发明的实时方案还包括一个高速缓存器。希望能够使高速缓存器系统中的命中率达到最大。

现有技术中典型的高速缓存器系统用来加速进出微处理器系统的存储器访问。在现有技术中这样的系统有几种机制可以用来预测查找，这样就可以使命中率达到最大。例如，现有技术中的高速缓存器系统可以使用前视机制来预测指令高速缓存器查找和数据高速缓存器查找。这样的前视技术无法用于分组监控应用程序的高速缓存器子系统。当一个新的分组进入监控器时，下一个高速缓存器访问，例如来自查找引擎，可能是与上一个高速缓存器查找完全不同的对话流的，这样就无法提前知道下一个分组属于什么流。

这样在本领域中就需要一种适合用在分组监控器中的高速缓存器子系统。希望这样的高速缓存器系统能有的一种属性是最近最少使用(LRU)替换策略，该策略在需要高速缓存器替换的时候替换 LRU 流－入口。替换最近最少使用的流－入口是优选的，因为跟在最近的分组后面的分组很可能和它属于同一个流。这样，新分组的签名很可能匹配最近使用的流记录。反之，与最近最少使用的流－入口相关的分组最不可能马上到达。

经常使用散列来加快查找。散列把记录随机散布在数据库中。这种情况下，就希望能有一个联合高速缓存器。

这样就需要联合高速缓存器子系统也包括 LRU 替换策略。

希望分组监控器能够维持流的状态以执行必要的任意状态处理从而促进确定与流相关的应用程序的过程。这样就需要一个状态处

理器来分析新的和现有的流以便按照应用对它们进行分类。

状态处理器可能需要的一种通用操作是在分组内容中查找一组已知字符串中的一个的存在情况。这样的识别法对于促进识别对话流的应用内容的方法非常有用。例如，可能希望查找与 http 协议相关的分组的统一资源定位符(URL)，或者需要查找标识协议或协议特征的特定字符串，例如字符串“port”、“get”、“post”等。这些字符串中的任意一个都可能在分组中，并且哪个字符串存在于分组中以及存在于分组中的什么地方通常是未知的。

在多数通用处理系统中，实现的指令集基本上是通用的。所有处理系统都有一个有关对指令和程序计数器的分析和操作的典型指令集。这些指令包括跳转、调用和返回。另外，这些相同的处理系统包含合适的指令来分析和操作寄存器和存储器区域。这些指令包括加、减、移动、比较和逻辑操作。

虽然状态处理器能够包括这样一个基本的标准指令集，但用这样的标准指令集实现在目标数据流中查找一个或多个已知的字符串可能会占用时间过长以致无法适应分组到达的高速度。因此希望有一个能够执行一些特定的查找功能的处理器，需要用它来对网络上分组中的数据以及分组中数据的内容极快地进行计算。

尤其需要一个查找设备，它可以是状态处理器的一部分并能够在目标数据流中快速查找指定的引用字符串。此外，需要有一个可编程处理器，它包含有调用查找设备执行这样的查找的指令。

在网络监控器中使用这样的处理器就使得监控器能够度量并满足任意的网络速度需求。

发明内容

本发明在它的几个不同的实施方案中提供了一个网络监控器，它能够实现一个或多个下列目标和优势：

- 识别在客户机和服务器之间进行交换的所有分组并把它们归类成各自的客户机/服务器应用。
- 在所有协议层上识别并归类在任一方向上通过网络中一点的对话流。
- 根据网络上交换的单个分组确定客户机和服务器之间的连接和流进展。

- 用来根据需要网络资源的客户机/服务器应用的当前混合帮助调节网络性能。
- 保持与使用网络资源的客户机/服务器应用的混合有关的统计信息。
- 报告由客户机/服务器网络对话流的特定应用所用的特殊分组序列的出现。

本发明实施方案的其它方面:

- 适当分析在客户机和服务器之间交换的每个分组并保持与这些对话流中每一个的当前状态有关的信息。
- 提供一个灵活的处理系统，当新应用进入客户机/服务器市场时可以对它进行修改和调整。
- 保持与客户机/服务器网络中的对话流有关的统计信息按照单个应用进行分类。
- 报告一特定标识符，它可被其它面向网络的设备用来和特定客户机/服务器网络对话流的特定应用一起识别分组串。

本发明的实施方案大体上克服了现有技术中的问题和缺点。

像这里所描述的那样，一种实施方案分析了在任意方向上通过网络中任意点的每个分组，以便能够得到用来在客户机和服务器之间通信的实际应用。注意有几个同步且重叠的应用程序通过独立且异步的网络执行。

本发明的监控器实施方案成功地对每个在网络上看到的单个分组进行了归类。对分组内容进行解析并把选中的部分装配成一个签名(也称为密钥)，它随后被用于识别同一对话流的更多分组，例如进一步分析该流并最终识别出应用程序。这样该密钥就是选中部分的一个函数，并且在优选实施方案中，该函数是对选中部分的串联。优选实施方案形成并记住任意对话流的状态，这个状态由网络上的单个分组和整个对话流之间的关系来决定。通过以这种方式记住流的状态，实施方案就可以确定对话流的环境，包括它所相关的应用程序以及像时间、对话流长度、数据传输速率这样的参数。

监控器很灵活，能够适应为客户机/服务器网络所开发的将来应用程序。可以通过编译用高级协议描述语言所写的文件来引入新的协议和协议组合。

本发明的监控器实施方案优先用特定应用集成电路(ASIC)或场可编程门阵列(FPGA)来实现。在一种实施方案中，监控器包括解析器子系统，它从分组形成签名。此外监控器还包括分析器子系统，它从解析器子系统接收签名。

像介质访问控制(MAC)或分割重组模块这样的分组拦截设备用来向监控器的解析器子系统提供分组。

在硬件实现中，解析子系统包括两个子部分，模式分析与识别引擎(PRE)和提取引擎(限选器 slicer)。PRE 根据模式数据库解释每个分组，尤其解释每个分组中的单个字段。

可以存在于不同层上的不同协议可被看作是由结点链接而成的一个或多个树上的结点。分组类型是树的根。每个协议是一个父亲结点或者终端结点。父亲结点把协议链接到可以在更高层的其它协议(子协议)上。例如，以太网分组(根结点)可以是以太类型的分组—也称作以太网类型/版本 2 和 DIX(DIGITAL－Intel－Xerox 分组)—或是 IEEE 802.3 分组。IEEE 802.3 类型分组继续下去，它的一个孩子结点可以是 IP 协议，IP 协议的一个孩子结点可以是 TCP 协议。

模式数据库包括对分组的不同报头和它们的内容，以及这些怎样与树中的不同结点发生关系的描述。PRE 遍历该树直到它能到达的地方。如果一个结点不包括到更深层的链接，就宣告了模式的完全匹配。注意协议可以是几个模式的孩子。如果为每个可能的父亲/孩子树产生一个唯一的结点，模式数据库将会变得非常庞大。替代方法是在多个父亲之间共享孩子结点，这样能够使模式数据库变得紧凑一些。

最后当只要求协议识别时 PRE 可以被用在它自己上面。

对识别出的每个协议，限选器从分组中提取重要的分组元素。这些形成分组的签名(也就是密钥)。限选器还为快速识别流优先产生散列，该流可能拥有来自已知流的数据库的这个签名。

分组的流签名、散列和至少一些有效负载被传递给分析器子系统。在硬件实施方案中，分析器子系统包括统一流键缓冲区(UFKB)、查找/更新引擎(LUE)、状态处理器(SP)、流插入与删除引擎(FIDE)、存储器、和高速缓存器。UFKB 用于接收来自解析器子系统的部分分组并存储方法中的签名。LUE 在流记录数据库中查找以前遇到过的对

话流以确定签名是否来自一个现有的流。SP 用于执行状态处理。FIDE 用于往流数据库中插入一个新的流。存储器用于存储流数据库。高速缓存器用于加速对包含流数据库的存储器的访问。LUE、SP 和 FIDE 都连接到 UFKB 和高速缓存器上。

这样统一流键缓冲区就包含了分组的流签名，散列和至少一些有效负载用于分析器子系统中的分析。尽管分组签名存在于统一流键缓冲区中，仍然可以执行许多操作以进一步阐明客户机/服务器对话流所涉及到的分组的应用程序内容的同一性。在分析器子系统的特定硬件实施方案中可以并行处理多个流，并且来自正在被并行分析的所有分组的多个流签名也可被保存在 UFKB 中。

来自解析器的分组的分组分析方法的第一步是在已知分组流签名的数据库中查找实例。查找/更新引擎(LUE)首先用散列，然后用流签名，来实现这个任务。查找在高速缓存器中进行，并且如果高速缓存器中没有带有匹配签名的流，查找引擎就试图从存储器中的流数据库获取流。以前遇到过的流的流－入口最好包括状态信息，状态信息被用在状态处理器中以执行为状态而定义的任意操作并确定下一个状态。一个典型的状态操作是在存储在 UFKB 中的分组的有效负载中查找一个或多个已知的引用字符串。

一旦 LUE 完成了查找处理，无论它找到了相应的流或是发现这是一个新的流，都要在统一流键缓冲区结构中为这个分组流签名设置一个标志声明。对现有的流，由 LUE 的计算器部件对流－入口进行更新，它给流－入口数据库中的计数器增加一个值，流－入口数据库用来存储流的一个或多个统计测量。计数器用于确定流上的网络使用度量。

在完成对分组流签名的查找后，并且当前流签名的内容在数据库中时，状态处理器可以开始分析分组有效负载以进一步阐明这个分组的应用程序组件的同一性。状态处理器的提取操作以及由它执行的功能根据对话流(flow)的流(stream)中的当前分组序列而有所变化。状态处理器从与同一个流签名一起被看到的前一分组移动到存储的下一个逻辑操作。如果在这个分组上需要任何处理，状态处理器将针对这个状态执行来自状态指令数据库的指令，直到没有指令剩下或者遇到表示处理的指令为止。

在优选实施方案中，状态处理器功能是可编程的以便为分析新应用程序、新的分组序列和能够由使用这样的应用引起的状态作准备。

如果在查找这个特定分组流签名的过程中需要把流插入到活动数据库，就对流插入和删除引擎(FIDE)进行初始化。状态处理器也可以创建新的流签名并且这样就可以命令流插入和删除引擎把一个新的流作为一个新项插入到数据库中。

在优选硬件实施方案中，LUE、状态处理器以及 FIDE 中的每一个都独立于其它两个引擎进行操作。

本发明的另一个方面是收集与流有关的度量。使用这里描述的状态处理，因为状态处理器能够查找特定的数据有效负载，可以对这里描述的所发明的监控器的实施方案编程以便为流中与特定数据有效负载有关的一群分组收集相同的振动度量。这就使得所发明的系统能够提供更集中在与一组分组有关的质量类型上的度量。通常在评价网络中一个系统的性能时更希望有与单个分组相关的度量。

尤其可以对监控系统编程以保持在对话流的任意状态上任意类型的度量。监控器还可以把实际的统计信息编程写进任意点的状态中。这使得监控器系统的实施方案能够收集与网络使用和性能有关的度量，以及与特定状态或分组序列有关的度量。

可以只和状态一起收集的一些特定度量是与在一个方向上的一组流量有关的事件，与在一个或两个方向上通信序列的状况有关的事件以及与特定序列中的特定应用的分组交换有关的事件。这仅是需要引擎的度量的一个小样本，该引擎能够把流的状态和一组度量联系起来。

另外，因为监控器提供对对话或流中的特定应用更高的可见性，因而可以对监控器编程以收集特别针对那种类型的应用程序或服务的度量。换句话说，如果一个流是 Oracle 数据库服务器的，监控器的实施方案可以收集完成一次事务所要求的分组数量。只有与状态和应用分类一起才能从网络得到这种类型的度量。

因为可以对监控器编程以收集不同组的度量，对那些在多种环境中都要求的度量来说可以把本系统作为数据源。尤其是那些用来监控并分析与特定应用组有关的性能和传输流量的质量的度量。其它

实现可以包括与特定传输流量及其所带事件的记帐和返还(charge-back)有关的度量。然而也可对其它实现进行编程以提供对故障检修和容量设计有用并直接与焦点应用和服务有关的度量。

本发明的另一个方面是根据每个分组确定服务度量的质量。

这样就在这里就公开了监控设备的一个方法,用于分析通过计算机网络上一个连接点的分组的流。该方法包括从分组拦截设备接收分组，并在包含以前遇到的对话流的流-入口的流-入口数据库中进行查找。查找是为了确定收到的分组是否是一个现有的流。对每个分组都进行了处理。如果该分组是一个现有的流，该方法就更新这个现有的流的流-入口，包括存储一个或多个保存在流-入口中的统计测量。如果该分组是一个新流，该方法就在流-入口数据库中为这个新流存储一个新的流-入口，包括存储一个或多个保存在流-入口中的统计测量。这些统计测量用来确定与流有关的度量。这些度量可以是来自服务度量被确定的质量的基本度量，或者是服务度量的质量。

这里还描述了一个联合高速缓冲存储器系统,用于查找外部存储器的一个或多个单元。高速缓冲存储器系统包括连结到外部存储器上的一组高速缓冲存储器存储单元，一组相联存储器单元(CAMs)，每个 CAM 包含一个地址和指向一个高速缓冲存储器单元的指针，以及一个带有输入的匹配电路，这样 CAM 就在输入和 CAM 单元中的地址相同时确认一个匹配输出。特定的 CAM 指向哪个高速缓冲存储器单元是随着时间变化的。在优选实现中，CAMs 被按照自顶至底的顺序连接在一起，底部的 CAM 指向最近最少使用的高速缓冲存储器单元。

这里还描述了一个处理器,用于处理通过计算机网络上一个连接点的分组的内容。该处理器包括一个有一个或多个比较器的查找设备，比较器用于在分组内容中查找引用字符串。处理器实时处理通过该连接点的所有分组的内容。一种实现是，处理器是可编程的并且拥有一个指令集，指令集中包含调用查找设备从分组范围内的某个未知位置开始查找指定的引用字符串的指令。

这里还公开了可以用在该处理器中的查找设备的实施方案。该查找设备可配置用来从目标数据中的一组起始位置中的任一个开始在

目标数据中查找 N_R 个单元的引用字符串。查找设备包括一个可配置用来接收包含引用字符串的 N_R 个单元的引用寄存器；一个或多个连结成组以接收目标数据的目标数据寄存器；以及多个比较器集，每个比较器集对应于一个起始位置。特定起始位置的比较器集连接到引用寄存器的每个单元以及从该特定起始位置开始的目标数据寄存器的 N_R 单元上，并从该特定起始位置开始比较引用寄存器内容和目标数据寄存器的 N_R 个连续单元的对应内容。如果从一个比较器集的相应不同起始位置开始的目标数据中有第一个引用字符串的匹配，那么该比较器就对此做出指示。如果在任意起始位置开始的目标数据寄存器中包含有第一个引用字符串，那么就由相应的比较器集对此做出指示。

附图说明

虽然通过参考详细的优选实施方案可以更好地理解本发明，但不应该用这些来把本发明限制在任意特定的实施方案上面，因为这样的实施方案只是用于说明目的。还要借助下列附图依次说明这些实施方案。

图 1 是本发明的网络实施方案的功能框图，其中的监控器被连接用来分析通过一个连接点的分组。

图 2 中的图表把一些起始时可能交换的分组及其格式的实例描绘成说明性实例，并描绘了在被监控和分析的网络上的客户机和服务器之间的对话流。还描绘了这个实例和本发明的实施方案所特有的一对流签名。描绘了一些可能的流签名，在分析分组和识别产生离散应用分组交换的特定服务器应用的方法中可以产生并使用这些流签名。

图 3 是本发明的一种方法实施方案的功能框图，它能够像图 1 中所示的分组监控器那样操作。这个方法可以用软件或硬件来实现。

图 4 是一种高级协议语言编译及优化方法的流程图，它在一种实施方案中可用来根据本发明的版本为监控分组产生数据。

图 5 是分组解析方法的流程图，该方法被用作所发明的分组监控器的实施方案中解析器的一部分。

图 6 是分组元素提取方法的流程图，该方法被用作所发明的分组

监控器的实施方案中解析器的一部分。

图 7 是流签名构造方法的流程图，该方法被用作所发明的分组监控器的实施方案中解析器的一部分。

图 8 是监控器查找和更新方法的流程图，该方法被用作发明的分组监控器的实施方案中分析器的一部分。

图 9 是有代表性的 Sun 微系统远程过程调用应用的流程图，它可以被所发明的分组监控器识别。

图 10 是硬件解析器子系统的功能框图，包括模式识别器和提取器，它们构成了所发明的分组监控器的实施方案中解析器模块的一部分。

图 11 是包括状态处理器在内的硬件分析器的功能框图，它构成了所发明的分组监控器的实施方案的一部分。

图 12 是流插入和删除引擎方法的功能框图，它构成了所发明的分组监控器的实施方案中分析器的一部分。

图 13 是状态处理方法的流程图，它构成了所发明的分组监控器的实施方案中分析器的一部分。

图 14 是本发明的方法实施方案的简单功能框图，它能够像图 1 中所示的分组监控器那样进行操作。这个方法可以用软件实现。

图 15 是图 3(以及图 10 和图 11)的分组监控器怎样用像微处理器这样的处理器在网络上进行操作的功能框图。

图 16 是以太网分组的顶(MAC)层以及可以根据本发明的一个方面被提取出来形成签名的一些元素的实例。

图 17A 是图 16 的以太网分组的以太网类型报头以及可以根据本发明的一个方面被提取出来形成签名的一些元素的实例。

图 17B 是 IP 分组，例如图 16 中所示的以太网分组，以及可以根据本发明的一个方面被提取出来形成签名的一些元素的实例。

图 18A 是用来存储模式元素、解析和提取数据库元素的三维结构，该数据库根据本发明的一个实施方案可被解析器子系统所使用。

图 18B 是存储模式元素、解析和提取数据库的一种替代形式，该数据库根据本发明的一个实施方案可被解析器子系统使用。

图 19 是图 11 的分析器子系统中状态处理器部件的框图。

图 20 是图 11 的分析器子系统中查找引擎部件的框图。

图 21 是一个数据流框图，显示了查找引擎的四个单独的查找模块。

图 22A 是查找引擎核心的框图；图 22B 显示了核心中的比较器部件，它对一组输入和另一组输入进行比较。

图 23A 更详细地显示了输入核心的实现；图 23B 显示了核心的比较器部件。

图 24 显示了不同的 PDL 文件模块，它们都由图 20 中作为实例描述的编译方法按照本发明的编译方面进行编译。

图 25 是根据本发明的一个方面编译高级语言文件的方法的流程图。

图 26 是图 11 的分析器子系统中高速缓存器子系统 1115 中的高速缓冲存储器存储部分。

图 27 是高速缓存器子系统中高速缓冲存储器存储控制器和 CAM 控制器的框图。

图 28 是高速缓存器子系统 1115 中 CAM 阵列的一种实现的框图。

具体实施方式

注意本文档包括可能包括信号名的硬件图表及描述。多数情况下，这些名字就可以充分描述，但其它情况下并不需要用它们来理解本发明的操作和实践。

网络中的操作

图 1 描绘了本发明的一个系统实施方案，这里用通用标号 100 来指代它。系统 100 有一个在不同计算机之间传递分组(例如，IP 数据报)的计算机网络 102，例如在客户机 104－107 和服务器 110 和 112 之间。该网络被以示意图的形式显示为带有几个网络结点和链接的一片云，网络结点和链接显示在云的内部。监控器 108 检验在任一方向上通过它的连接点 121 的分组，并且能够根据本发明的一个方面阐明与每个分组相联的是什么应用程序。图中所示监控器 108 正在检验在服务器 110 的网络接口 116 和网络之间的分组(即数据报)。监控器还可以放在网络中的其它点上，例如网络 102 和客户机 104 的接口 118 之间的连接点 123，或者像网络 102 中某处的连接点 125 所指示的其它位置。没有显示出来的是在网络的 123 位置上的网

络分组拦截设备，它把网络上的物理信息转换成能够输入到监控器108中的分组。这样的分组拦截设备是通用的。

网络可以采用多种协议来建立并维持必需的通信，例如可以用TCP/IP协议等。任意网络活动—例如由客户机104(客户机1)运行的一个应用程序和在服务器110(服务器2)上运行的另一个应用程序进行通信—都将产生在网络 102 上的分组序列交换，这正是各自的应用程序和网络协议的特征。这样的特征在单个分组级上毫无启迪作用。可以要求由监控器 108 对许多分组进行分析以拥有足够的所需信息来识别特定的应用程序。可能需要在不同协议的环境中对分组进行解析和分析，例如，符合 ISO 分层网络模型的一种类型的分组通过应用会话层进行传输。

通信协议是分层的，它也被称作协议栈。ISO(国际标准化组织)已经定义了一个通用模型，它为通信协议层的设计提供了一个框架。下面的表格中显示了这个模型，充当理解现有通信协议功能的基本参考。

ISO模型

层	功能	实例
7	应用	Telnet、NFS、Novell、NCP、HTTP、H.323
6	表示	XDR
5	会话	RPC、NETBIOS、SNMP等
4	传输	TCP、Novell SPX、UDP等
3	网络	IP、Novell IPX、IPX、VIP、AppleTalk等
2	数据链路	网络接口卡(硬件接口)MAC层
1	物理	以太网、令牌环网、帧中继、ATM、T1(硬件连接)

不同的通信协议采用ISO模型的不同层，或者使用与ISO模型类似但并不严格遵守它的分层模型。特定层中的协议对其它层上所用的协议可以是不可见的。例如，一个应用(第 7 级)可能无法识别一个通信尝试(2－3层)的源计算机。

在一些通信技术中，术语“帧”通常指在OSI第二层上封装的数据，包括目的地址、流控制的控制位、数据或有效负载以及用于错误校验的CRC(循环冗余校验)数据。术语“分组”通常指在OSI第三

层上封装的数据。在 TCP/IP 世界中，还使用了术语“数据报”。在本规范中，术语“分组”意指包括分组、数据报、帧和单元。通常，分组格式或帧格式指如何用不同的字段和报头封装数据以便通过网络进行传输。例如，数据分组通常包括目的地址字段、长度字段、纠错码(ECC)字段或循环冗余校验(CRC)字段以及标识分组的开始和结束的报头和报尾。术语“分组格式”和“帧格式”是同义的，也称作“单元格式”。

监控器 108 查看通过连接点 121 的每个分组以便分析。但是，并非每个分组都带有对识别协议的所有层次有用的相同信息。例如，在和一个特定应用相联的对话流中，应用将使服务器发送一个 A 类型的分组，但其它应用也将如此。然而，如果特定的应用程序总是在 A 类型的分组之后紧跟着发送一个 B 类型的分组，而其它的应用程序并不这样做，那么为了识别那个应用的对话流的分组，可用监控器识别匹配与 A 类型分组相联的 B 类型分组的分组。如果在一个 A 类型分组之后识别出了这样的分组，那么该特定应用程序的对话流就已经开始向监控器 108 揭示它自己。

在识别出对话流与某应用程序相联之前还需要检验更多的分组。通常，监控器 108 只能同时部分完成对其它应用相关对话流中的分组交换的识别。监控器 108 的一个方面是它能够保持流的状态。流的状态是对流中能够导致识别所有协议层内容—例如 ISO 模型协议级别—的所有以前事件的指示。本发明的另一个方面形成提取出的分组特征部分的签名，它可以用来快速识别属于相同流的分组。

在对监控器 108 的实际使用中，在网络 102 上通过监控器 108 的连接点的分组数可达每秒百万个之多。因此，监控器只有很少的时间可以用来对每个分组进行分析并测定类型以及识别并维护通过连接点的流的状态。监控器 108 因此屏蔽每个分组中所有那些对它的分类没有贡献的不重要的部分。但是，每个分组中要屏蔽的部分根据分组所属的流以及流的状态不同而变化。

根据应用程序的执行所产生的分组识别出分组类型并最终识别出相联的应用程序在监控器 108 中是一个多步方法。例如，第一步，几个应用程序都将产生第一种分组。从分组的选中部分产生的第一个“签名”将使得监控器 108 能够有效地识别属于同一流的任何分

组。某些情况下，仅分组类型就能让监控器识别出在对话流中产生这样的分组的应用。随后该签名可用来有效识别在与该应用有关的流量中产生的所有将来分组。

在其它情况下，第一个分组仅仅启动分析对话流的方法，还需要更多分组来识别相关的应用程序。在这种情况下，要用签名识别第二种类型的后继分组—但它也可能属于同一个对话流。随后在第二步那些应用程序中只有少数拥有能够产生第二种分组类型的对话流。在分类方法的这一步中，所有不能产生这样的分组类型序列的应用程序都被排除在对包含这两种分组的对话流进行分类的方法之外。根据用协议和可能的应用的已知模式产生一个签名，它能够识别跟随在对话流中的任何将来分组。

现在可能已经识别出了该应用，或者识别还需要用第二步中的签名来继续进行第三步分析。因此，对每个分组，监控器都要对分组进行解析并产生一个签名，以确定这个签名是否和一个以前遇到过的流相一致，或者用来识别属于同一流的将来分组。在以前遇到过的分组序列环境(状态)以及过去的序列在与不同应用相关的对话流中可能产生的将来序列的环境中进一步实时分析分组。还可以产生用于识别将来分组的新签名。分析过程继续进行直到识别出应用为止。最后产生的签名随后可以用来有效识别与同一对话流相关的将来分组。这样的装置使得监控器 108 能够应付每秒成百万个必须被检查的分组。

本发明的另一个方面是增加窃听(Eavesdropping)。在本发明的能够窃听的实施方案中，一旦监控器 108 识别出通过网络 102 中某一点正在执行的应用程序(例如，因为客户机 105 或服务器 110 所执行的应用)，监控器向网络上的一些通用处理器发送一个消息，这些处理器能够输入来自网络上同一位置的分组，并随后装入属于它自己的该应用程序的可执行副本并用它来读取网络上正在交换的内容。换句话说，一旦监控器 108 完成了对应用程序的识别，窃听就开始启动了。

网络监控器

图 3 显示了一种网络分组监控器 300，在本发明的实施方案中可以用计算机硬件和/或软件来实现它。系统 300 与图 1 中的监控器 108

相似。检验分组 302，例如，分组可以来自网络 102 中 121 位置上(图 1)的分组拦截设备，对分组进行估计以试图确定它的特征，例如多层模型中的所有协议信息以及什么应用产生了该分组。

分组拦截设备是一个通用接口，它对物理信号进行转换并随后把它们解码成位，并根据特定的网络(以太网、帧中继、ATM 等)把它们解码成分组。分组拦截设备向监控器 108 指示拦截到的一个或多个分组的网络类型。

这里所显示的方面包括：(1)对监控器进行初始化以产生需要在不同类型的分组上执行的操作—由编译器和优化器完成，(2)对分组进行处理—解析并提取选中部分—以产生一个识别签名—由解析器子系统 301 完成，和(3)分析分组—由分析器 303 实现。

编译器和优化器 310 的目的是提供协议特定信息给解析器子系统 301 和分析器子系统 303。在监控器的操作之前要对其进行初始化，并且只在有新协议加入时才需要重复初始化动作。

流是正在网络中的任意两个地址之间交换的分组序列。对每个协议都有几个已知的字段，像目的(接收者)、源(发送者)等等，在监控器 300 中用这些和其它字段来识别流。还有其它一些字段对识别流不太重要，像校验和以及那些不用于识别的部分。

解析器子系统 301 用模式识别方法 304 检验分组，该方法对分组进行解析并确定协议类型以及分组 302 中存在的每个协议层的相关报头。解析器子系统中的提取方法 306 从分组 302 中提取出特征部分(签名信息)。用于解析和有关提取操作的模式信息，例如提取掩码，都由解析模式结构和提取操作数据库(解析/提取数据库)308 提供，由编译器和优化器 310 对这个数据库进行填充。

协议描述语言(PDL)文件 336 描述了可以在任意层上出现的模式和所有协议的状态，包括怎样解释报头信息，怎样从分组报头信息确定下一层上的协议，以及为了识别流并最终识别出应用和服务要提取什么信息。层次选择数据库 338 描述了由监控器处理的特定分层操作。也就是说，在任意层的协议之上运行的是什么协议。这样 336 和 338 就联合描述了怎样解码、分析并理解分组中的信息，以及另外怎样对信息分层。这个信息被输入到编译器和优化器 310 中。

当编译器和优化器 310 执行时，它产生两组内部数据结构。第一

组是解析/提取操作的集合 308。模式结构包括解析信息并描述在分组的报头中将要识别什么；提取操作是根据已经匹配的模式将要提取出分组中的什么元素。这样，解析/提取操作数据库 308 包括描述怎样从分组的数据确定一组一个或多个协议有关提取操作的信息，这些操作指示分组中所用的协议。

由编译器 310 构造的其它内部数据结构是状态模式和方法的集合 326。这些是发生在不同对话流中的不同状态和状态转换，以及在对话流的任意状态中需要执行以促进分析话流任务的状态操作(例如，需要检验的模式和需要构造的新签名)。

这样，编译 PDL 和层次选择向监控器 300 提供了它需要用来开始处理分组的信息。在替代实施方案中，可以手动或自动产生数据库 308 和 326 中一个或多个结构的内容。注意，在一些实施方案中层次选择信息是内在的而没有明确描述。例如，既然协议的 PDL 文件包括了子协议，那么父协议也可以被确定。

在优选实施方案中，来自拦截设备的分组 302 被输入到分组缓冲区中。由模式分析与识别引擎(PAR)执行模式识别方法 304，PAR 分析并识别分组中的模式。尤其是 PAR 定位报头中的下一协议字段并确定报头的长度，并为特定类型的协议报头执行其它特定任务。这种操作的一个实例是对进行类型和长度比较以区分 IEEE 802.3(以太网)分组和较老的类型 2(或版本 2)以太网分组，也称为 DIX(DIGITAL - Intel - Xerox)分组。PAR 还用模式结构和提取操作数据库 308 来识别下一个协议以及与该协议相关联的参数，这些使 PAR 能够分析下一个协议。一旦识别出一个或一组模式，它/它们将与一组零个或多个提取操作相联。这些提取操作(以命令及相关参数形式)被传递给由提取和信息识别(EII)引擎实现的提取方法 306，EII 引擎从分组中提取选中的部分，包括为标识该分组为某个流的一部分而需要的标识信息。提取出的信息被依次放置并随后在 312 框中对它们进行处理以构造这个流的唯一流签名(也称为“键”)。流签名依赖于分组中所用的协议。对于某些协议，提取出的成分可能包括源和目的地址。例如，以太网帧里的终端地址对构造更好的流签名十分有用。这样，签名通常包括客户机和服务器地址对。签名用来识别更多是这个流或可能是这个流的一部分的分组。

在优选实施方案中，构造流键包括用散列函数产生签名的散列。目的—如果使用这样的散列很方便的话—是把由签名标识的流－入口散布在数据库中以便能够有效查找。所产生的散列最好是根据散列算法并且这样的散列生成也为本领域中的技术人员所知。

在一种实施方案中，解析器把来自分组并包括签名在内的数据—解析器记录—(即分组中选中的部分)、散列及分组本身传递下去以能够执行需要来自分组的更多数据的任意状态处理。解析器子系统的改进实施方案可以产生解析器记录，其中有一些预定义的结构并且包括签名、散列、与解析器记录中的一些字段有关的一些标志以及解析器子系统已经确定的部分分组有效负载，对分组进行进一步的处理，例如状态处理，可能需要这个解析器记录。

注意替代实施方案可以使用除了把分组的选中部分连接起来之外的一些函数来构造标识签名。例如，可以使用串联的选中部分的一些“摘要函数”。

解析器记录被传递到查找方法 314 上，它观察系统已经遇到过的已知流的内部记录数据存储中，并判断(在 316 中)这个特定的分组是否属于一个已知流，由数据库 324 中是否存在匹配这个流的流－入口指示判断结果。数据库 324 中的每个记录与一个遇到过的流相关。

解析器记录进入被称为 统一流键缓冲区(UFKB)的缓冲区中。UFKB 在与解析器记录相似的数据结构中存储流上的数据，但它还包含一个可以修改的字段。尤其是有一个 UFKB 记录字段存储分组序号，另一个字段以状态处理器的程序计数器形式用状态信息进行填充，状态处理器实现状态处理 328。

判定带有相同签名的记录是否已经存在由查找引擎(LUE)执行，LUE 获得新的 UFKB 记录并用其中的散列地址去查找是否有匹配的已知流。在特定的实施方案中，已知流数据库 324 是在外部存储器中。有一个高速缓存器与数据库 324 相联。LUE 通过使用散列地址访问高速缓存器以查找已知记录，如果该入口不在高速缓存器中，就到外部存储器中查找该入口。

流－入口数据库 324 所存储的流－入口包括唯一流签名、状态信息、从分组中提取用于更新流的信息以及关于流的一个或多个统计

信息。每个入口完整地描述了一个流。数据库 324 被组织成包含 N 个流－入口(也称为存储桶，每个对应一个流－入口)的存储柜，在优选实施方案中 N 等于 4。通过来自解析器子系统 301 的分组中的散列(即 UFKB 记录中的散列)对存储桶(即流－入口)进行访问。散列提供更浅的存储桶把流散布在数据库中以便能够快速查找入口。设计者根据监控器所带的存储器的数量选择存储桶深度 N 以及所用散列数据值的位数。例如，在一种实施方案中，每个流－入口都是 128 个字节，同时有 128K 个流－入口，因而需要 16M 字节的存储器。使用 16 位的散列可以在每个存储桶中给出两个流－入口。经验显示这在大多数情况下都是足够的。注意别的实施方案使用了 256 字节长的流－入口。

这里描述了任何时候对数据库 324 的访问，可以理解该访问是通过高速缓存器进行的，除非它被从环境中重置或清除。

如果没有找到区配新流的签名的流－入口，那么协议和状态识别方法 318 就进一步确定状态和协议。也就是说，方法 318 确定协议以及在这个协议的这个分组所属的流的状态序列中的什么地方。识别方法 318 使用提取出的信息并参考状态模式和方法数据库 326。方法 318 后面跟的是需要由状态处理器 328 在这个分组上执行的状态操作。

如果在数据库 324(例如，在高速缓存器中)中找到了与该分组匹配的流－入口，那么方法 320 就从查找到的流－入口确定是否需要由流签名的状态进行更多分类。如果不需要，方法 322 就更新流－入口数据库中的流－入口(例如，通过高速缓存器)。更新包括更新存储在流－入口中的一个或多个统计测量。在我们的实施方案中，统计测量存储在流－入口中的计数器里。

如果需要进行状态处理，就开始状态方法 328。状态处理器 328 执行为该流的状态指定的任意状态操作并更根据从状态模式和方法数据库 326 获得的一组状态指令把状态更新到下一个状态。

状态处理器 328 对新的和现有的流进行分析以分析协议栈的所有层，最终按照应用(ISO 模型中的第七层)对流进行分类。它通过根据预定义的状态转换规和状态处理器指令数据库 326 中指定的指令从状态－到－状态进行来做到这一点。状态转换规则通常包含一个

测试以及紧随在其后的在测试结果为真时进行的下一状态。操作是在状态处理器处于特定状态中时要执行的操作—例如，为了估计需要应用的状态转换规则的数量。状态处理器通过每个状态和每个状态方法直到测试为真，或者没有要执行的测试为止。

通常，状态处理操作集可以是分组上的零个或多个操作，执行这些操作可能会把一个操流留在一个导致在完成识别之前就退出系统的状态中，但可能知道更多关于接下来—即遇到这个流的下一个分组时—需要执行的状态和状态方法的信息。

通过维护流的状态并知道可以用来自以前遇到的流的信息对新流进行设置，网络流量监控器 300 提供了(a)流的单一分组协议识别和(b)流的多分组协议识别。监控器 300 甚至能够从出现在服务器通告类型的流中的一个或多个断开的子流识别应用程序。一些看起来与现有技术中的监控器不相关的流，本发明中的监控器也可用流签名识别出它们是与以前遇到过的子流相关的子流。

这样，状态处理器 328 为这个特定流－入口的分组应用第一个状态操作。方法 330 判断是否还需要为这个状态执行更多操作。如果是，分析器就在框 330 和 328 之间继续循环，对这个特定的分组应用额外的状态操作直到所有那些操作都被完成—也就是说，直到在这个状态中没有这个分组的其它操作为止。方法 332 根据流的状态和协议判断对这种类型的流是否还有其它状态要分析，以便完全表征该流。如果没有，现在就已经被完全表征了该对话流并且由方法 334 结束该流的对话流的分类。

在特定的实施方案中，状态处理器 328 通过把解识析器识别出的最后一个协议用作跳转表(跳转向量)中的位移来启动状态处理。跳转表在状态模式和方法数据库 326 中找到用于该协议的状态处理器指令，或者在已知流数据库 324 中找到相应的流－入口，如果该入口存在的话。状态处理器必须测试位，进行比较、加或减操作以执行测试。例如，一个由状态状态处理器执行的通用操作是在 UFKB 的有效负载部分中查找一个或多个模式。

这样，在分类里的 332 中，分析器判断流是否处于一个结束状态。如果不在结束状态，就在方法 332 中对这个流－入口进行更新(如果是新流就进行创建)。

此外，如果该流是已知的并且在 332 中确定它还有更多需要用后来的分组进行处理的状态，就在方法 322 中更新流－入口。

在识别完成之后还要更新流－入口以使能够从属于这个完全分析过的对话流的签名识别出属于这个流的任意将来分组。

在更新之后，数据库 324 包括所有已经出现的对话流的集合。

这样，图 3 中所示的本发明的实施方案自动保存流－入口，它在一个方面中包括存储状态。图 3 的监控器还产生分组的特征部分—签名，可用来识别流。可以用流－入口的签名对它们进行识别和访问。一旦识别出分组来自某个已知的流，该流的状态是已知的并且这个知识使得能够对每个不同的协议和应用实时执行状态转换分析。在复杂分析中，要遍历的状态转换和要检验的分组一样越来越多。那些属于同一对话流的将来分组从以前到达的状态继续它们的状态分析。当已经处理了足够多的与感兴趣的应用有关的分组时，终于达到了一个最终识别状态，即已经由状态分析遍历了一组状态以完全表征该对话流。那个最终状态的签名使得同一对话流中每个新到达的分组能够被实时单独识别。

用这种方式可以实现本发明的一个重大优点。一旦为第一次和以最终状态结束遍历了状态转换的一个特定集合，就可以产生快捷识别模式—签名—它将在每个与该对话流有关的新来的分组上标上键值。检验签名涉及到简单操作，使得能够在网络上成功监控高分组速率。

在改进的实施方案中，几个状态分析器并行运行，因此能够检验大量协议和应用。每个已知的协议和应用都将有至少一个唯一的状态转换集合，并因此可以通过观察这样的转换而唯一识别它们。

当每个新的对话流开始时，就立即自动产生标识该流的签名，并随着遇到的该对话流中的将来分组对签名进行更新，并根据该流的状态转换规则进一步遍历任意潜在应用的状态转换集中的状态。该流的新状态—与一个或多个潜在应用的状态转换集合相关的那些状态—被加入到以前遇到的状态的记录中，以在遇到流中的一个新分组时能够轻松识别和检索。

详细操作

图 4 图示了一种包括编译方法在内的初始化系统 400。也就是

说，初始化的一部分产生模式结构和提取操作数据库 308 以及状态指令数据库 328。这样的初始化可以脱机或者发生从中心位置发生。

能够存在于不同层中的不同协议可以看作是由结点链接而成的一个或多个树中的结点。分组协议是树的根(称为第 0 层)。每个协议是一个父亲结点或终端结点。一个父亲结点可以把一个协议链接到更高层上的其它协议(子协议)上。这样协议就可以有零或多个孩子。例如，以太网分组有几种变体，每一种都有基本上保持相同的基本格式。以太网分组(根或第 0 层结点)可以是以太类型的分组—也称为以太网类型/版本 2 和 DIX(DIGITAL－Intel－Xerox 分组)—或 IEEE803.2 分组。IEEE 802.3 分组继续下去，它的一个孩子结点可以是 IP 协议，IP 协议的一个孩子结点可以是 TCP 协议。

图 16 显示了完整以太网帧(即分组)的报头 1600(底层 1)，包括目的介质访问控制地址(Dst MAC 1602)和源介质访问控制地址(Src MAC 1604)上的信息。图 16 中还显示了 PDL 文件中为提取签名指定的一些(并非全部)信息。

图 17A 显示了以太类型分组 1700 的下一层(第 2 层)报头信息。对以太类型分组 1700 来说，来自分组指示下一层的有关信息是一个两字节类型字段 1702，其中包含下一层的子识别模式。剩下的信息 1704 在图中显示为阴影，因为它与这一层无关。列表 1712 显示了以太类型分组的可能的孩子，由在位移 12 发现的子识别模式所指示。图 17B 显示了一个可能的下一层的报头结构，也就是 IP 协议的报头结构。表 1752 中显示了 IP 协议的可能的孩子。

模式、解析和提取数据库(模式识别数据库，或 PRD)308 由编译方法 310 产生，在一种实施方案中，它是三维结构形式，提供对下一协议的分组报头的快速查找。图 18A 显示了这样一个 3 维表示 1800(可以看作是 2 维表示的编入索引的集合)。3 维结构的压缩形式是优选的。

图 18B 中描绘了数据库 308 中所用的数据结构的一种替代实施方案。这样，像图 18A 的 3 维结构那样，该数据结构允许由模式识别方法 304 通过存储器中的变址定位而不是执行地址链接计算来执行快速查找。在这个替代实施方案中，PRD308 包括两个部分，单一协议表 1850(PT)，对每个监控器已知的协议它都有一个记录，和一组

查找表 1870(LTU's)，用来识别已知的协议和它们的孩子。协议表包括模式分析和识别方法 304(由 PRE1006 实现)需要用来估计与那个协议相关的分组报头信息的参数，以及提取方法 306(由限选器 1007 实现)需要用来处理该分组报头的参数。当有孩子时，PT 描述对报头中的哪些字节进行计算以确定子协议。尤其每个 PT 记录包含报头长、到孩子的位移、限选器命令和一些标志。

在报头字段中找到特定的“子识别代码”就执行了模式匹配，并用这些代码来检索一个或多个 LUT's。每个 LUT 记录有一个结点代码，它可以有下列四个值之一，指示该协议已经被识别出来，指示该协议已被部分识别(需要更多 LUT 查找)的代码，指示这是一个终端结点的代码，和指示空记录的空结点。要查找的下一个 LUT 也从 LUT 查找返回。

图 4 中描述了编码方法。以协议描述语言形式的源代码信息显示为 402。在特定的实施方案中，高级解码描述包括一组协议描述文件 336，每个文件针对一个协议，以及一组分组层次选择 338，它们描述了监控器能够处理的特定分层(多组协议树)。

编译器 403 编译这些描述。产生(404)分组解析和提取操作集合 406，以状态处理器的指令形式产生一组分组状态指令和操作 407，状态处理器实现状态处理方法 328。每种类型的应用和协议的数据文件由分析器识别并从模式、解析和提取数据库 406 下载到解析器和提取引擎的存储器系统中。(参见解析方法 500 的描述及图 5；提取方法 600 的描述及图 6；解析子系统硬件描述及图 10)。每种类型的应用和协议的数据文件由分析器进行识别并从状处理器指令数据库 407 下载到状态处理器中。(参见状态处理器 1108 的描述及图 11)。

注意产生分组解析和提取操作构造并链接了三维结构(一种实施方案)或 PRD 的所有查找表。

因为可能的协议树和子树的数量巨大，编译器方法 400 中包括比较树和子树以了解哪些孩子共享公共父亲的优化过程。当以 LUT's 形式实现时，这个方法可以从多个 LUT's 产生单一的 LUT。优化方法中还包括减少需要用来存储 PRD 数据的空间的压缩方法。

作为压缩的一个实例，考虑图 18A 的 3－D 结构，它可以看作是一组 2－D 结构，每个 2－D 结构表示一个协议。为了能够对那些有

几个父亲的协议每个只用一个数组以节省空间，在一种实施方案中，模式分析子方法保持一个“当前报头”指针。3－D 结构中每个协议 2－D 数组的位置(位移)索引是从该特定协议的报头起始处开始的一个相对位置。此外，每个二维数组都是稀疏数组。优化的下一步是对每个 2－D 数组和所有其它 2－D 数组进行核对以找出哪些共享存储器。这些 2－D 数组中有很多都是被稀疏填充的，每个数组里只有很少量的有效记录。因此接下来用一个“折叠”方法把两个或更多 2－D 数组结合成一个物理 2－D 数组而且不丢失任意一个原始 2－D 数组的同一性(即所有的 2－D 数组逻辑上连续存在)。折叠可以在任意 2－D 数组之间进行，而不管它们在树中的位置，只要满足特定的条件即可。只要单个记录间不相互冲突，就可以把多个数组结合成一个单一数组。然后用一个折叠号使每个元素和它的原始数组相联。在图 18B 的替代实施方案中为 LUTs 集合 1850 使用了类似的折叠方法。

在 410，分析器已经完成了初始化并准备执行识别。

图 5 显示了实际解析器子系统 301 如何活动的流程图。从 501 开始，在 502 步把分组 302 输入到分组缓冲区中。503 步从分组 302 装入下一个(在初始时是第一个)分组成分。分组成分是从每个分组 302 每次提取的一个元素就是。504 执行检查以确定装入分组成分操作 503 是否成功，如果判断结果为真则表明分组中还有其它成分要处理。如果判断结果为假，表明所有成分都已装入，解析器系统 301 构造分组签名(512)—下一阶段(图 6)。

如果在 503 中成功装入一个成分，就从模式、解析和提取数据库 308 中取出(505)结点和方法以为该结点提供一组模式和方法以应用到装入的分组成分上。解析器子系统 301 执行检查(506)以确定取出模式结点操作 505 是否成功完成，判断结果为真表明在 505 步装入了一个模式结点。如果判断结果为假，则在 511 步移动到下一个分组成分。如果判断结果为真，就在 507 步中将模式匹配方法应用到 503 中提取出的组件。在 507 中获得的模式匹配(由测试 508 指示)意味着解析器子系统已经在解析元素中发现了一个结点；解析器子系统 301 继续 509 步提取元素。

如果将结点方法应用到该成分上并未产生匹配(测试 508)，解析

器子系统 301 就从模式数据库 308 转移到下一个模式结点(510)，再到 505 步以取出下一个结点和方法。这样，在 508 和 505 之间就有了一个“应用模式”循环。一旦解析器子系统 301 完成了所有模式，不管是否有匹配，解析器子系统 301 都转移到下一个分组成分(511)。

一旦从输入分组 302 装入了所有分组成分并对它们进行了处理，再装入分组就将失败(由测试 504 指示)，解析器子系统转而构造分组签名，在图 6 中对此有所描述。

图 6 是从分组中提取信息以构造分组签名的流程图。流程在 601 开始，它是图 5 中的退出点 513。在这一点上解析器子系统 301 在缓冲区中已经有了可用的全部分组成分和模式结点(602)。603 步从图 5 的模式分析方法装入可用的分组成分。如果装入完成(测试 604)，表明实际上还有其它分组组件，解析器子系统 301 在 605 步取出在 602 步接收自模式结点成分的提取和方法元素。如果成功取出(测试 606)，表明有要应用的提取元素，解析器子系统 301 在 607 步根据接收自模式结点的提取指令把提取方法应用到分组成分上。这就从分组成分中去掉并节省了一个元素。

在 608 步，解析器子系统 301 是否还有其它要从这个成分提取的东西，如果没有，解析器子系统 301 就返回到 603 装入手边的下一个分组成分并重复这个过程。如果判断结果为真，那么解析器子系统 301 就转到下一个分组组件棘轮。随后在 603 中装入新的分组成分。当解析器子系统 301 通过 608 和 603 之间的循环时，如果还有更多要提取的分组成分可以将额外的提取方法应用到同一分组成分上，或者如果没有其它要提取的可以把额外的提取方法应用到不同的分组成分上。

这样提取方法根据模式和提取数据库 308 中的信息针对特定分组提取越来越多的成分就构造出了签名。一旦装入一下一个分组成分操作失败(测试 604)，表明已经提取出了所有成分。把构造出的签名装入到签名缓冲区中(610)，解析器子系统 301 继续进行到图 7 以完成签名产生方法。

现在参考图 7，该方法从 701 开始。签名缓冲和模式结点元素可用(702)。解析器子系统 301 装入下一个模式结点元素。如果装入成

功(测试 704)则表明还有更多结点，解析器子系统 301 在 705 中根据在元素数据库中的模式结点中发现的散列元素对签名缓冲区元素进行散列。由此产生的签名和散列在 706 中被打包在一起。在 707 中解析器子系统 301 移到在 703 中装入的下一个分组组件。

703 到 707 循环继续执行直到其它没有元素模式剩下为止(测试 704)。一旦所有的元素模式都被进行了散列，解析器子系统的方法 304、306 和 312 也就完成了。解析器子系统 301 已经产生了由分析器子系统 303 所用的签名。

解析器记录被装入到分析器中，尤其要以 UFKB 记录形式装入到 UFKB 中，UFKB 记录与解析器记录相似，但有一个或多个不同的字段。

图 8 是描述实现查找操作 314 的查找/更新引擎(LUE)的操作的流程图。该方法在 801 从带着包括签名、散列和至少部分有效负载的解析器记录的图 7 开始。在 802 中它们都以缓冲区中的 UFKB－记录形式显示。LUE，即查找引擎从流－入口的散列信息计算“记录存储柜编号”。这里的存储柜可以有一个或多个“存储桶”，每个存储桶包含一个流－入口。优选实施方案中每个存储柜有四个存储桶。

既然优选的硬件实施方案包括高速缓存器，就规定图 8 的流程图中对记录的所有数据访问都是通过高速缓冲存储器进行。

这样，在 804 中，系统用散列在高速缓存器中查找来自那个存储柜的存储桶。如果高速缓存器从存储柜编号成功返回一个存储桶，表明该存储柜中还有更多存储桶，查找/更新引擎比较(807)当前签名(UFKB－记录的签名)和存储桶中的签名(即流－入口签名)。如果签名相匹配(测试 808)，就在 810 步把记录(在高速缓存器中)标记为“进行中”并加上时间戳。811 步向 UFKB 指示 802 中的 UFKB－记录有“找到”状态。“找到”标记允许状态处理开始处理这个 UFKB 元素。优选的硬件实施方案包括一个或多个状态处理器，这些处理器都可以和查找/更新引擎并行操作。

在优选实施方案中，由计算器为被分析的每个分组执行一组统计操作。统计操作可以包括对与这个流相关的分组进行多次计数；确定与流的分组大小有关的统计信息；编译关于每个方向上分组之间的差异的统计信息，例如使用时间戳；并确定同一方向上分组的时间戳的统计关系。统计测量结果保存在流－入口中。也可以编译其

它的统计测量。这些统计信息可以由统计处理器部件单独或者结合使用以分析流的多个不同方面。包括从统计测量结果中确定网络使用度量，例如确定网络为这个应用传输信息的能力。这样的分析为测量对话服务的质量、测量应用怎样在网络中执行及测量应用所消耗的资源等作好了准备。

为了提供这样的分析，查找/更新引擎在 812 步更新流－入口(在高速缓存器官)中的一个或多个计数器。方法在 813 退出。在我们的实施方案中，计数器包括流的总分组数、时间及从上一个时间戳到当前时间戳的微分时间。

可能该存储柜的存储桶并未产生匹配(测试 808)。这种情况下，分析器在 809 中移动到这个存储柜中的下一个存储桶。804 步再次在高速缓存器中查找来自那个存储柜的另一个存储桶。查找/更新引擎就这样继续查找该存储柜中的存储桶直到在 808 中有一个匹配或者操作 804 失败(测试 805)为止，表明在这个存储柜中已经没有其它的存储桶而且没有发现匹配的存储桶。

如果没有发现匹配，那么该分组就属于一个新(以前未曾遇到过的)的流。在 806 中系统指示统一流键缓冲区中这个分组的记录为新，并在 812 中通过更新高速缓存器中的流－入口为这个分组执行统计更新操作。更新操作在 813 结束。流插入/删除引擎(FIDE)为这个流创建一个新的记录(再次通过高速缓存器)。

这样，更新/查找引擎以该分组的 UFKB－记录带着“新”或“找到”状态结束。

注意上述系统使用多个流－入口都能够匹配的散列。可以使用更长的散列对应于单一流－入口。在这样一个实施方案中简化了图 8 的流程图，对那些本领域的技术人员来说应该清楚这一点。

硬件系统

现在根据图 10 和图 11 描述系统中数据流所经过的每个单独的硬件单元。注意，尽管我们正在描述图 3 的发明实施方案的特定硬件实现，但那些本领域的技术人员应该理解图 3 中的流程也可用运行在一个或多个通用处理器上的软件实现，或者只用硬件实现一部分。本发明的实现可以在图 14 所示的软件中操作。虽然图 14 的软件系统适用于低速网络，但硬件实施方案(图 10 和 11)可以在每秒超

过一千万个分组的网络上进行操作。本领域的技术人员应该明白随着处理器变得越来越快系统中可以有越来越多的部分用软件实现。

图 10 是对用硬件实现的解析器子系统(301，这里显示为子系统 1000)的描述。存储器 1001 是模式识别数据库存储器，其中存储了将要分析的模式。存储器 1002 是提取－操作数据库，其中存储了提取指令。1001 和 1002 都符合图 3 中的内部数据结构 308。通常，从微处理器(未显示)初始化该系统并同时经由内部总线 1003 和 1004 通过主机接口多路复用器装载这些存储器。注意 1001 和 1002 的内容优先由图 3 中的编译方法 310 获得。

分组使用控制信号 1021 和 1023 经由 1012 进入解析系统并到达解析器输入缓冲区存储器中，1021 和 1023 控制输入缓冲区接口控制器 1022。缓冲区 1008 和接口控制 1022 连接到一个分组拦截设备(未显示)。缓冲区拦截设备产生分组开始信号 1021 且接口控制 1022 产生下一分组(即准备接收分组)信号 1023 以控制数据流进入解析器输入缓冲区存储器 1008。一旦分组开始装入缓冲区存储器 1008，模式识别引擎(PRE)1006 就在图 3 的 304 框中描述的输入缓冲区存储器上执行操作。也就是确定分组中存在的每个协议层的协议类型和相关报头信息。

PRE 查找数据库 1001 和缓冲区 1008 中的分组以识别分组所包含的协议。在一种实现中，数据库 1001 包括一组链接在一起的查找表。每个查找表用八位寻址。第一个查找表总在地址零上。模式识别引擎使用来自控制寄存器的基本分组位移以启动比较过程。它把这个值装入当前位移指针(COP)中。随后从解析器输入缓冲区读取基本分组位移处的字节并用它作为进入第一个查找表的地址。

每个查找表返回一个链接到其它查找表的字或者返回一个结束标志。如果查找产生了识别事件那么数据库也返回一个限选器命令。最后返回该值并把它添加到 COP 上。

PRE 1006 包括一个比较引擎。比较引擎第一步检查协议类型字段以确定是否是一个 802.3 分组并且该字段应该被作为长度对待。如果它不是长度，就在第二步中检查协议。第一步只是协议层，而协议层是不可编程的。第二步有两个为将来的协议增加而定义的完整的 16 位相联存储器(CAMs)。

这样，无论何时只要 PRE 识别出了一个模式，它都要产生一个提取引擎(也称为“限选器”)1007 命令。识别出的模式和命令被送往提取引擎 1007，它从分组中提取信息以构造解析器记录。这样，提取引擎的操作就是在图 3 的框 306 和 312 中执行的操作。命令以提取指令指针的形式从 PRE1006 送往限选器 1007，命令提取指针告诉提取引擎 1007 在提取操作数据库存储器(即限选器指令数据库)中的什么地方找到指令。

这样，当 PRE1006 识别出一个协议时它把协议标识符和方法代码都输出给提取器。协议标识符被添加到流签名上，方法代码用于从指令数据库 1002 中取出第一个指令。指令包括一个操作码，通常还包括源和目的地址以及长度。位移和长度都以字节表示。典型的操作是 MOVE 指令。这个指令告诉限选器 1007 把 n 个字节的数据不加任何更改从输入缓冲区 1008 复制到输出缓冲区 1010。提取器包含一个按字节移动装置，因此能够把被移动的字节打包进流签名中。提取器还包含另一个称为 HASH 的指令。这个指令告诉提取器从输入缓冲区复制数据到 HASH 发生器。

这样这些指令就用于提取输入缓冲区存储器中的分组的选中单元并把数据传送到解析器输出缓冲区存储器 1010。一些指令还产生散列。

提取引擎 1007 和 PRE 以流水方式进行操作。也就是说，提取引擎 1007 在输入缓冲区 1008 中已经由 PRE1006 处理过的的数据上执行提取操作，而 PRE1006 同时正在解析其它(即后来到达的)分组。这种操作方式所提供的高处理速度足以应付分组到达的高速率。

一旦分组中所有用来形成分组签名的选中部分都被提取出来，就把散列装载到解析器输出缓冲区存储器 1010 中。还包括进一步的分析所要求的来自分组的任意附加有效负载。解析器输出存储器 1010 通过分析器接口控制 1011 与分析器子系统相接。一旦分组中的所有信息都在解析器输出缓冲区存储器 1010 中，就由分析器接口控制确认数据就绪信号 1025。当分析器就绪信号 1027 被确认时就通过 1013 把来自解析器子系统 1000 的数据移动到分析器子系统中。

图 11 显示了硬件部件和分析器子系统的数据流，分析器子系统执行图 3 中的分析器子系统 303 的功能。在操作之前要对分析器进

行初始化，初始化包括把由编译方法 310 产生的状态处理信息装载到数据库存储器中用于状态处理，也称为状态处理器指令数据库(SPID)存储器 1109。

分析器子系统 1100 包括使用分析器主机接口控制器 118 的主机总线接口 1122，它们依次访问高速缓存器系统 1115。高速缓存器系统对系统 1108 的状态处理器进行双向访问。状态处理器负责从主机总线接口 1122 上给出的信息初始化状态处理器指令数据库存储器 1109。

随着 SPID 1109 的装载，分析器子系统 1100 把来自解析器并包括分组签名和有效负载的解析器记录接收到统一流键缓冲区(UFKB)1103 中。UFKB 由可设置用来保存 UFKB 记录的存储器组成。UFKB 记录基本上就是解析器记录；UFKB 保存将要处理或正在处理的分组记录。此外，UFKB 还提供一个或多个字段用作可修改状态标志以允许不同的方法并发运行。

三个处理引擎并发运行并访问 UFKB 1103 中的记录：查找/更新引擎(LUE)1107、状态处理器(SP)1108 和流插入和删除引擎(FIDE)1110。这些引擎中的每一个都可以用一个或多个有限状态机(FSM's)实现。在每个有限状态机和统一流键缓冲区 1103 之间都有双向访问。UFKB 记录包括存储分组序号的字段，和另一个以状态处理器 1108 的程序计数器形式用状态信息进行填充的字段，状态处理器实现状态处理 328。任意记录的 UFKB 中的状态标志包括 LUE 完成标志和标记 LUE 正在向状态处理器传送对记录的处理的标志。LUE 完成指示器还用来指示 LUE 的下一个记录。还提供了一个标志来指示状态处理器完成了对当前流的操作并指示用于状态处理器的下一个记录是什么。还提供了一个标志来指示状态处理器正在传送 UFKB－记录的处理到流插入和删除引擎。

新的 UFKB 记录由 LUE 1107 处理。已经由 LUE 1107 处理过的记录可以由状态处理器 1108 进行处理，UFKB 记录数据可以在状态处理器 1108 处理之后由流插入/删除引擎 1110 进行处理或者只由 LUE 进行处理。特定引擎是否已经被应用到任意统一流键缓冲区记录上由该引擎在操作完成时设置的状态字段来确定。在一种实施方案中，UFKB－记录中的一个状态标志指示记录是新的或找到的。在其它实

施方案中，LUE 产生一个标志以把记录传递到状态处理器进行处理，并且新记录所要求的操作也包含在 SP 指令中。

注意每个 UFKB－记录可能需要由全部三个引擎进行处理。此外，一些 UFKB 记录可能需要由某个特定引擎进行多次处理。

这三个引擎中的每一个都对包括超高速缓存引擎的高速缓存器子系统 1115 进行双向访问。高速缓存器 1115 被设计用来使信息能够在系统中的五个不同点之间流入流出，这五个点是：三个引擎、经由统一存储器控制(UMC)1119 和存储器接口 1123 的外部存储器、以及经由分析器主机接口和控制单元(ACIC)1118 和主机接口总线(HIB)1122 的微处理器。这样分析器微处理器(或者专用逻辑处理器)就能够在高速缓存器中直接插入或修改数据。

高速缓存器子系统 1115 是一个相联存储器，包括一组相联存储器单元(CAMs)，每个相联存储器单元包括地址部分和指向高速缓冲存储器(例如 RAM)的指针部分，高速缓冲存储器中包含高速缓存的流－入口。CAMs 被按照自顶至底的顺序排列成栈。底部 CAM 的指针指向最近最少使用(LRU)的高速缓冲存储器记录。无论任何时候只要发生了高速缓冲存储器缺失，底部 CAM 所指向的高速缓冲存储器的内容就被来自流－入口数据库 324 的流－入口所取代。这个记录现在变成了最近使用过的记录，因此底部 CAM 的内容被移到顶部并且所有的 CAM 内容均向下移动。这样，高速缓存器就成为使用真正的 LRU 替换策略的相联高缓冲存器。

LUE 1107 首先处理 UFKB－记录，并基本上执行图 3 的 314 和 316 框中的操作。提供一个信号给 LUE 以指示一个“新”的 UFKB－记录可用。LUE 用 UFKB－记录中的散列从高速缓存器读取一个相当于四个存储桶的匹配存储柜。如果匹配存储柜不在高速缓存器中，高速缓存器 1115 就向 UMC 1119 发出请求以从外部存储器引入相匹配的存储柜。

当用散列方法找到一个流－入口时，LUE 1107 查看每个存储桶并用它的签名和 UFKB－记录的签名进行比较直到找到一个匹配或者没有其它存储桶剩下为止。

如果没有匹配，或者从高速缓冲存储器提供流－入口的存储柜失败，就在 UFKB 记录的流键中设置一个时间戳，用编译方法 310 在初

始化期间装载的一个表格进行协议识别和状态确定，该记录的状态被设置为指示 LUE 已经处理过该记录，并做出该 UFKB－记录准备好起动状态处理的指示。识别和状态确定产生一个协议标识符，它在优选实施方案中是状态处理器的“跳转向量”，由 UFKB 为这个 UFKB－记录保存它，并由状态处理器用来起动针对特定协议的状态处理。例如，跳转向量跳转到用于处理该状态的子程序。

如果有匹配，表明该 UFKB－记录中的分组是以前遇到过的流的，然后计算器部件输入存储在流－入口中的一个或多个统计测量，包括时间戳。另外，可以存储自上次存储的时间戳以来的时间差并更新分组计数。通过查看存储在数据库 324 的流－入口中的协议标识符对流－入口对从该流－入口获得的流状态进行检查。如果该值指示不需要其它分类，那么该记录的状态就被设置成指示 LUE 已经处理过该记录。在优选实施方案中，协议标识符是状态处理器到一个对该协议进行状态处理的子程序的跳转向量，并且在优选实施方案中跳转向量为零指示没有其它分类。如果协议标识符指示更多处理，那么就做出该 UFKB－记录准备起动状态处理的指示并把该记录的状态设置成指示 LUE 已经处理过了该记录。

状态处理器 1108 在 LUE 完成之后根据 UFKB－记录处理高速缓存器中的信息。状态处理器 1108 包括状态处理器程序计数器 SPPC，它产生由编译方法 310 在初始化期间装载的状态处理器指令数据库 1109 中的地址。它 8 包含产生 SPID 地址的指令指针(SPIP)。指令指针可以增加或者从推动条件转移的跳转向量多路复用器装载。SPIP 可以从下列三个来源装载：(1)来自 UFKB 的协议标识符，(2)来自当前解码后的指令的立即跳转向量，或者(3)由状态处理器中包含的算术逻辑单元(SPALU)所提供的值。

这样，在 LUE 把流键和已知协议标识符放在 UFKB 中之后，由解析器用识别出的最后一个协议对程序计数器进行初始化。这第一个指令是跳转到一个对解码出的协议进行分析的子程序。

状态处理器 ALU(SPALU)包含所有必要的算术、逻辑和字符串比较功能以实现状态处理器指令。SPALU 的主要部分是：A 和 B 寄存器、指令解码&状态机、字符串引用存储器查找引擎、输出数据寄存器和输出控制寄存器。

查找引擎依次包含目标查找寄存器集、引用查找寄存器集和通过对两个操作数进行异或操作以对它们进行比较的比较块。

这样，在 UFKB 设置了程序计数器之后，就在状态处理器中执行一序列一个或多个状态操作以为这个特定分组进一步分析流键缓冲区记录中的分组。

图 13 描述了状态处理器 1108 的操作。在 1301 带着要处理的统一流键缓冲区记录进入状态处理器。UFKB－记录是新的或者与对应于一个找到的流－入口。这个 UFKB－记录是在 1301 步从统一流键缓冲区 1103 得到的。在 1303，该 UFKB－记录的协议标识符被用来设置状态处理器的指令计数器。状态处理器 1108 通过把解析器子系统 301 识别出的最后一个协议用作跳转表中的位移来启动该方法。跳转表把我们带到用于那个协议的指令。多数指令测试某物是否在统一流键缓冲区或者流－入口中，如果流－入口存在的话。状态处理器 1108 必须测试位、进行比较、加或法操作以执行测试。

第一个状态处理器指令是在 1304 步从状态处理器指令数据库存储器 1109 中取出的。状态处理器执行一个或多个取出的操作(1304)。在我们的实施方案中，每个单个的状态处理器指令是非常简单的(例如，移动、比较等)，因此需要在每个统一流键缓冲区记录上执行许多这样的指令。状态处理器的一个方面是它能在 UFKB 记录的有效负载部分中查找一个或多个(最高达四个)引用字符串。这由状态处理器中引擎部件响应特殊查找指令实现。

在 1307 中执行检查以确定对该分组是否还有任意其它的指令要执行。如果有，就在 1308 中由系统设置状态处理器指令指针(SPIP)以得到下一个指令。SPIP 可以由当前已经解码的指令中的立即跳转设置，或者在处理过程中由 SPALU 提供一个值。

下一个要执行的指令现在已经取出来(1304)用于执行了。13047 和 1307 之间的这个循环继续进行直到没有其它要执行的指令为止。

在这一步，在 1309 中检查在这个特定分组上所做的处理是否已经导致了一个终结状态。如果是，表明分析器不仅已经对这个分组而且对这个分组所属的整个流都完成了处理，并且该流已被完全确认。如果实际上对这个流没有其它状态要处理，就在 1311 中由处理器终止处理。一些终结状态可能需要把状态放置在适当的位置告诉

系统移除一个流—例如，如果连接从更低层的连接标识符消失。假如那样的话就在 1311 中设置流移除状态并把它保存在流－入口中。流移除状态可以是 NOP(无操作)指令，意味着没有移除指令。

一旦设置并保存了为这个流指定的适当的流移除指令(NOP 指令或相反)，该方法就在 1313 结束。状态处理器 1108 现在可以得到另一个统一流键缓冲区记录进行处理。

如果在 1319 确定对这个流所做的处理还未完成，那么系统就在 1310 步把状态处理器指令指针存储在当前流－入口中。这将是 LRE 1107 在下一次执行的下一个操作，以在 UFKB 中找到区配这个流的分组。现在处理器在 1313 退出对这个特定的统一流键缓冲区记录进行的处理。

注意状态处理更新了统一流键缓冲区 1103 中的信息和高速缓存器中的流－入口。一旦状态处理器执行完毕，就在 UFKB 中为该记录设置一个标志，指示状态处理器已经完成了操作。此外，如果该流需要从流数据库插入或删除，就为这个流签名和分组记录把控制传递到流插入/删除引擎。这由状态处理器完成，状态处理器在 UFKB 中为这个 UFKB－记录设置另一个标志，指示状态处理器正在把对这个记录的处理传递给流插入和删除引擎。

流插入和删除引擎 1110 负责维护流－入口数据库。尤其负责在流数据库中创建新流，以及从数据库中删除流以使它们能够被重新使用。

现在借助图 12 描述流插入方法。用散列值把流组合成由存储桶组成的存储柜。该引擎处理可能是新的或者状态处理器已经另外指示其为需要创建的 UFKB－记录。图 12 显示了正在被创建的新记录的情况。在 1203 步得到一个对话记录存储柜(优选包含 4 个存储桶，用于 4 个记录)。这是一个与 UFKB 中的散列值匹配的存储桶，因此这个存储桶可能已经由 LUE 为该 UFKB－记录找到了。在 1204 步 FIDE 1110 请求保存在高速缓存器系统 1115 中的记录存储柜/存储桶。如果在 1205 步中高速缓存器系统 1115 指示该存储柜/存储桶是空的，就在 1207 把该流签名(用散列)插入到存储桶中并在高速缓存器 1115 的高速缓存器引擎中用一个时间戳把该存储桶标记为“用过”，该时间戳在这个方法中自始至终都被维护起来。在 1209 中，FIDE 1110

比较存储柜和存储桶记录流签名和分组以核实所有的元素都在合适的位置以完成该记录。在 1211 中系统把记录存储柜和存储桶标记为“处理中”并在高速缓存器中（并因此在外部存储器中）标记为“新”。在 1212 中，在高速缓存器系统中设置该流－记录的初始统计测量，并且可以为在特定的流中所见到的第一个分组执行分析器所要求的用于统计操作的其它程序。

回到 1205 步，如果该存储桶非空，FIDE 110 就请求高速缓存器系统中这个特定的存储柜的下一个存储桶。如果成功，就为这下一个存储桶反复执行 1207、1209、1211 和 1212 中的过程。在 1208，如果没有有效存储桶，就把该分组的统一流键缓冲区记录设置为“丢失”，表明系统无法处理该特定分组，因为系统中没有剩余的存储桶。该方法在 1213 退出。FIDE 1110 向 UFKB 指示对这个 UFKB－记录的流插入和删除操作已经完成。这还导致 UFKB 向 FIDE 提供下一个 UFKB 记录。

一旦需要用来访问并控制特定分组和它的流签名的所有引擎在一个统一流键缓冲区记录上完成了一组操作，该统一流键缓冲区记录就被标记为“完成”。随后该元素将被解析器接口用于从解析和提取系统到达的下一个分组和流签名。

所有的流－入口都保存在外部存储器中，有一些还保存在高速缓存器 1115 中。高速缓冲存储器系统 1115。高速缓冲存储器系统 1115 的智能足以访问流数据库并理解存在于存储器接口另一端上的数据结构。查找/更新引擎 1107 能够请求高速缓存器系统把特定的流或流的“存储桶”从统一存储控器控制 1119 取到高速缓存器系统中以便进一步处理。状态处理器 1108 能够操作依靠查找/更新引擎请求在高速缓存器系统中找到的信息，并且如果请求是根据统一流键缓冲区 1103 中的信息流插入/删除引擎 1110 能够在高速缓存器系统中创建新记录。高速缓存器通过存储器接口 1123 和统一存储器控制 1119 从存储器获得所请求的信息，并通过存储控制器 1119 更新所请求的信息。

对特定的硬件实现有几个通往图 11 的模块之外的系统部件的接口。这些接口包括主机总线接口 1122，它被设计成通用接口，可以和像微处理器或多路复用器(MUX)系统这样任意类型的外部处理系

统一起操作。因此，可以把图 11 和 12 中的整个流量分类系统连接到一些其它处理系统之中以管理分类系统并提取由系统产生的数据。

存储器接口 1123 被设计用来和想用来存储流－入口的任意存储器相接。可以使用不同类型的存储器系统，像随机访问存储器(DRAM)、同步 DRAM、同步图形 DRAM、静态访问存储器(SRAM)等等。

图 10 还包括一些“通用”接口。有一个分组输入接口 1012—和输入缓冲区接口控制 1022 的信号一起工作的通用接口。这些接口被设计成能够和任意类型的通用系统一起使用，然后这些通用系统可以向解析器提供分组信息。另一个通用接口各自进出主机接口多路复用器和控制寄存器 1005 的流水线接口 1031 和 1033。这使得解析系统可以由外部系统进行管理，例如微处理器或其它种类的外部逻辑，并且使外部系统可以对解析器编程或者进行控制。

本发明的这个方面的优选实施方案是用像 VHDL 或 Verilog 这样的硬件描述语言(HDL)进行描述的。用 HDL 对它进行设计和创建以使它可被用作单一芯片系统或者被集成到另一个通用系统之中，该通用系统被设计用于与在网络中创建并分析流量有关的目的。Verilog 或其它 HDL 实现只是描述硬件的一个方法。

根据硬件实现，图 10 中所示的单元用一组六字段可编程逻辑阵列(FPGA's)实现。这些 FPGA's 的分界如下。图 10 中的解析子系统实现为两个 FPGAS；一个 FPGAS 包括块 1006、1008、1012、部分 1005 以及存储器 1001；第二个 FPGA 包括 1002、1007、1013、1011 和部分 1005。参考图 11，统一查找缓冲区 1103 实现为一个单独的 FPGA。状态处理器和部分状态处理器指令数据库存储器 1109 是另一个 FPGA。部分状态处理器指令数据库存储器 1109 保存在外部 SRAM's 中。查找/更新引擎 1107 和流插入/删除引擎 1110 在另一个 FPGA 中。第六个 FPGA 包括高速缓存器系统 1115、统一存储器控制 1119 和分析器主机接口及控制 1118。

注意可以把系统实现成一个或多个 VSLI 设备，而不是一组像 FPGA's 这样的特定用途集成电路(ASIC's)。希望将来设备集成度继续提高，以致于整个系统能够最终构成一个更大的单一芯片单元上的一个子单元(“核心”)。

发明操作

图 15 显示了网络监控器 300 的实施方案怎样被用来分析网络 102 中的流量。分组拦截设备 1502 拦截来自网络 102 上的连接点 121 的所有分组以使在任意方向上通过点 121 的分组都可以提供给监控器 300。监控器 300 包括解析器子系统 301，它确定流签名；和分析器子系统 303，它分析每个分组的流签名。存储器 324 用来存储由监控器 300 确定并更新的流数据库。主机 1504 用来分析存储器 324 中的流，它可以是任意处理器，例如一个通用计算机。通常主机 1504 包括一个存储器，称为 RAM，图中显示为主机存储器 1506。另外，主机可以包含磁盘。在一种应用中，主机可以从 RMON 探测开始，这种情况下主机连接到网络接口卡 1510 上，网络接口卡 1510 又连接到网络 102。

本发明的优选实施方案由可选的简单网络管理协议(SNMP)实现所支持。图 15 描述了怎样实现 RMON 探测，其中网络接口卡用来向网络发送 RMON 信息。商业 SNMP 实现也可以使用，并且使用这样的实现可以简化把本发明的优选实施方案移植到任意平台上的方法。

另外，MIB 编译器也可以使用。MIB 编译器是一个能够大大简化对专用 MIB 扩展的创建和维护的工具。

分组说明实例

监控器 300，尤其是分析器 303 能够对通常被称为“服务器通告”类型交换的分组交换执行状态分析。服务器通告是用来简化带有多个应用的服务器之间的通信的方法，这些应用都可以从多个客户机同时访问。许多应用采用服务器通告作为把单一端口或套接字多路复用到许多应用和服务的手段。采用这种类型的交换，可以在网络上用广播或多播途径发送消息来宣告一个服务器或应用，并且网络中的所有站点都可以接收到这些消息并对它们进行解码。该消息能够使站点获得一个合适的连接点以对特定应用和特定服务器通信。采用服务器通告方法，特定应用用服务通道以 IP 协议组中的 TCP 或 UDP 套接字或端口的形式进行通信，或者用 Novell IPX 协议组中的 SAP 进行通信。

分析器 303 还能够执行分组交换的“流内分析”。“流内分析”方法可以用做主识别方法或第二识别方法。作为主方法，流内分析

帮助提取用来进一步识别特定应用和应用组件的详细信息。流内分析的一个很好的实例是任意基于 Web 的应用。例如，通用点播 Web 信息应用就可以这种方法进行识别；在点播服务器和客户机之间的初始连接过程中，数据交换中存在特定的键标记将导致产生签名以识别点播。

流内分析方法也可以和服务器通告方法结合起来。很多情况下流内分析将增强其它识别方法。在像 SAP 和 BAAN 这样的商业应用中可以找到把流内分析和服务器通告结合起来的实例。

“会话追踪”也是已知追踪客户机/服务器分组交换中的应用的一种主要方法。追踪会话方法需要一个到预定义套接字和端口号的初始连接。这个通信方法用在多种传输层协议中。在 IP 协议的 TCP 和 UDP 传输协议中最为常见。

在会话追踪过程中，客户机用特定端口号和套接字向服务器发出请求。这个初始请求导致服务器创建一个 TCP 或 UDP 端口以在客户机和服务器之间交换剩下的数据。随后服务器用这个新创建的端口对客户机的请求做出应答。客户机用来向服务器建立连接的原始套接字在这个数据交换过程中将不再使用。

会话追踪的一个实例是 TFTP(一般文件传输协议)，TCP/IP FTP 协议没有目录和口令功能的版本。在 TFTP 的客户机/服务器交换过程中，特定端口(端口号 69)总是用来初始化分组交换。这样，当客户机开始通信过程时，就向 UDP 端口 69 发送一个请求。一旦服务器接收到这个请求，就在服务器上创建一个新端口号。随后服务器用这个新端口向客户机发送应答。在这个例子中，非常清楚的是，为了识别 TFTP 网络监控器分析来自客户机的初始请求并为它产生一个签名。监控器 300 使用这个签名来识别应答。监控器 300 还分析来自服务器带有关键端口信息的应答，并送它创建签名以监控这个数据交换的其余分组。

网络监控器 300 还能够理解网络中特定连接的当前状态。面向连接的交换通常受益于状态追踪以正确地识别应用。一个实例是通用 TCP 传输协议，它提供了一个在客户机和服务器之间发送信息的可靠方法。当初始化数据交换时，发送一个同步 TCP 请求。这个消息包含用来追踪来自服务器的应答的特定序号。一旦服务器确认了同步

请求，就可以在客户机和服务器之间交换数据。当不再需要通信时，客户机向服务器发送一个结束或完成消息，服务器用包含了来自请求的序号的应答确认这个结束请求。这样一个面向对象的交换的状态与多种类型的连接和维护消息有关。

服务器通告实例

服务器通告协议的单个方法有所不同。但基本的底层方法是相同的。向网络中的一个或多个客户机发送典型的服务器通告消息。这种类型的服务器消息有特定内容，在本发明的另一方面中得到了应用并且保存在系统的流－入口数据库中。因为该通告被送往一个或多个站点，将来可能与服务器有分组交换的客户机将假定所通告的信息是已知的，并且所发明的监控器的一个方面是它也做同样的假定。

Sun－RPC 是由 Sun 微系统公司(帕洛阿尔托，加利福尼亚州)实现的远程过程调用(RPC)，是一个允许一个程序使用远程机器上另一程序的服务的编程接口。现在用一个 Sun－RPC 实例来解释监控器 300 怎样捕获服务器通告。

希望使用服务器或程序的远程程序或客户必须建立一个连接，可以用 RPC 协议做到这一点。

每个运行 Sun－RPC 协议的服务器必须维护一个方法和称为端口映射表的数据库。端口映射表在 Sun－RPC 程序或应用和 TCP 或 UDP 套接字或端口(对 TCP 或 UDP 实现)之间创建一个直接连接。应用或程序编号是由 ICANN()分配的一个 32 位唯一标识符，ICANN 管理与 Internet 协议有关的大量参数(端口号、路由器协议、多播地址等)。Sun－RPC 服务器上的每个端口映射表通过使用特殊请求或直接通告给出在唯一程序编号和特定传输套接字之间的映射。根据 ICANN，端口号 111 与 Sun RPC 相连。

作为一个实例，考虑客户机(例如，图 1 中显示为 106 的 CLENT 3)在预定义的 UDP 或 TCP 套接字上向服务器(例如，图 1 中显示为 110 的 SERVER 2)发出一个特定请求。一旦 Sun RPC 服务器上的端口映射表方法接收到该请求，就在对客户机的直接应答中返回该特定映射。

1. 客户机(图 1 中的 CLIENT 3,106)在端口 111 上向 SERVER 2(图

1 中的 110)发送一个 TCP 分组，带有 RPC 绑定查找请求(rpcBindLookup)。TCP 或 UDP 端口 111 总是与 Sun RPC 相连。这个请求指定程序(以程序标识符)、版本，并且可以指定协议(UDP 或 TCP)。

2. 服务器 SERVER 2(图 1 中的 110)从该请求中提取出程序标识符和版本标识符。服务器还利用这样的事实：这个分组使用 TCP 传输进来并且没有指定协议。这样对它的应答将使用 TCP 协议。

3. 服务器 110 向端口号 111 发送一个 TCP 分组，带有 RPC 绑定查找应答。该应答包含在其上接收特定 RPC 程序标识符(例如，程序'program')和协议(UDP 或 TCP)的将来事务的特定端口号(例如，端口号'port')。

希望从现在开始每次都使用端口号'port'，分组与应用程序'program'相连直到端口号'port'不再与程序'program'相连为止。监控器 300 通过创建流-入口和签名而拥有一个记忆交换的机制，使得它能够把使用端口号'port'的将来分组都与应用程序'program'相连。

除了 Sun RPC 绑定查找请求和应答之外，还有其它一些特定程序—比如'program'—可以与特定端口号相连的方式。一种是在应用服务和端口号之间的特定连接的广播通告，称为 Sun RPC 端口映射表通告。另一种是当一些服务器—比如同一 SERVER 2—向一些客户机—比如 CLIENT 1—做出应答要求一些带有 RPC 端口映射表应答的端口映射表分配时。其它一些客户机—比如 CLIENT 2—可能无意中看到了这个请求，这样就知道了对特定的服务器 SERVER 2 端口号'port'与应用服务'program'相连。希望网络监控器 300 能够把使用用端口号'port'的 SERVER 2 的任意分组与应用程序'program'相连。

图 9 为 Sun 远程过程调用描绘了在图 3 的监控器 300 中一些操作的数据流 900。假设客户机 106(例如，图 1 中的 CLIENT 3)正在通过它到网络的接口 118 和服务器 110(例如，图 1 中的 SERVER 2)通过该服务器到网络的接口 116 进行通信。进一步假定远程过程调用被用来和服务器 110 进行通信。数据流 900 中的一条路径是从 910 步由客户机 106 发出的远程过程调用绑定查找请求开始并以服务器

状态创建步骤 904 结束。这样的 RPC 绑定查找请求包括要用的‘program’、‘version’和‘protocol’—如 TCP 或 UDP—的值。网络监控器 300 中 Sun RPC 分析的方法包括下列方面:

- 方法 909: 提取‘program’、‘version’和‘protocol’(UDP 或 TCP)。提取 TCP 或 UDP 端口(方法 909),该端口是 111 指示 Sun RPC。
- 方法 908: 对 Sun RPC 分组进行解码。为 ID 检查 RPC 类型字段。如果该值是端口映射表,就保存成对套接字(即用于目的地址的 dest,用于源地址的 src)。对端口和映射进行解码,保存带有套接字/地址键的端口。对每个映射表分组可以有不只一对。来自签名(例如,键)。在数据库 324 中创建一个流-入口。请求保存现在完成。

在后来的某个时间,服务器(方法 907)发出一个 RPC 绑定查找应答。分组监控器将从分组提取签名并从以前存储的流中识别它。监控器将得到协议端口号(906)和查找请求(905)。将创建一个新签名(即一个键)并在流-入口数据库中把服务器状态(904)的创建保存成由新签名标识的一个记录。这个签名现在可以用来识别与服务器相连的分组。

不仅可以从绑定查找请求/应答对到达服务器状态创建步骤 904,而且可以从显示为 901 的 RPC 应答端口映射表分组或显示为 902 的 RPC 通告端口映射表到达它。远程过程调用协议可以宣告它能够提供特定应用服务。当客户机和服务器之间发生交换时本发明的实施方案最好能够分析并追踪网络中那些已经接收到服务通告的站点。

RPC 通告端口映射表通告 902 是一个广播。这导致不同的客户机执行一组相似的操作,例如保存从通告获得的信息。RPC 应答端口映射步骤 901 对端口映射器请求做出应答,并且应答也是一个广播。它包括所有服务参数。

这样监控器 300 就创建并保存了用于后来与特定服务‘program’有关的流的分类的所有这样的状态。

图 2 显示了 Sun RPC 的实例中的监控器 300 怎样构造签名和流状态。例如,在示例 Sun 微系统远程过程调用协议中交换的多个分组

206－209。本发明的一种方法实施方案能够从在分组 206 和 207 中找到的信息产生一对流签名，“签名－1”210 和“签名－2”212，分组 206 和 207 在这个实例中分别相当于 Sun RPC 绑定查找请求和应答。

考虑第一个 Sun RPC 绑定查找请求。假设分组 206 相当于从 CLIENT 3 发往 SERVER 2 的这样一个请求。这个分组包含在根据本发明的一个方面构造签名中要用到的重要信息。源和目的网络地址占据了每个分组的头两个字段，并且依照模式数据库 308 中的模式，流签名(在图 2 中显示为 KEY－1 230)也将包含这两个字段，因此解析器子系统 301 将在签名 KEY－1(230)中包含这两个字段。注意在图 2 中，如果一个地址标识客户机 106(也显示为 202)，那么图中所用的标号是“C_1”。如果这样的地址标识服务器 110(也显示为服务器 204)，图中所用的标号是“S_1”。分组 206 中的头两个字段是 214 和 215 是“S_1”和“C_1”，因为分组 206 是由服务器 110 提供去往客户机 106 的。假定对这个例子，“S_1”在数值上小于“C_1”。第三个字段“p^1”216 标识正在使用的协议，例如 TCP、UDP 等等。

在分组 206 中，第四个字段 217 和第五个字段 218 用来传递要用的端口。对话方向确定端口字段在哪里。字段 217 中的斜线图案用来标识源端口模式，字段 218 中的交叉图案用来标识目的端口模式。它们之间的次序指示客户机－服务器消息方向。由“i^1”219 所指示的第六个字段是客户机正在向服务器请求的一个元素。由“s^1a”220 所指示的第七个字段是客户机向服务器 110 请求的服务。随后的第八个字段“QA”221(用于问题标志)指示客户机 106 想要知道用什么来访问应用“s^1a”。第十个字段“QP”223 用来指示客户机想让服务器指示对特定的应用使用了什么协议。

分组 206 初始化分组交换的顺序，例如发往 SERVER 2 的 RPC 绑定查找请求。它采用明确定义的格式，所有分组都是样这样，并在众所周知的服务连接标识符(端口 111 指示 Sun RPC)上被传送到服务器 110。

分组 207 是在从服务器到客户机 106 的应答中第一个被发送的分组。它是作为请求分组 206 的结果的 RPC 绑定查找应答。

分组 207 包括十个字段 224－233。目的和源地址在字段 224 和

225，例如，分别由“C_1”和“S_1”指示。注意现在顺序是颠倒的，因客户机－服务器消息方向是从服务器 110 到客户机 106。所用协议“p^1”在字段 226 中。请求“i_1”在字段 229 中。已经为应用端口号填充的值在字段 233 中，并且协议“p^2”也在字段 233 中。

现在描述作为这个交换的结果而构造的流签名和流状态。当分组监控器 300 看到来自客户机的请求分组 206 时，就在解析器子系统 301 中根据模式和提取操作数据库 308 构造第一个流签名 210。这个签名 210 包括目的地址 240 和源地址 241。本发明的一个方面是按照一个特定的顺序构造流键而不管对话方向。有几种机制可以用来实现这一点。在特定实施方案中，数值较小的地址总是被放在数值较大的地址前面。这样从低到高的顺序用来为查找操作获得签名和散列的最好分布。因此在这种情况下，既然我们假定“S_1”＜“C_1”，那么顺序就是地址 S_1”后跟客户地址“C_1”。用来构造签名的下一个字段是从分组 206 的字段 216 中提取出的协议字段 242，这样就是协议“p^1”。用于签名的下一个字段是字段 243，它包含目的源端口号，在图中显示为来自分组 206 的字段 218 的交叉阴影图案。可以在分组的有效负载中识别这个字段进行以得出这个分组或分组序列怎样作为一个流而存在。在实践中，这些可以是 TCP 端口号或 TCP 端口号的组合。就该 Sun RPC 实例来说，阴影表示用于 p^1 的一组 UDS 端口号，将被用来识别这个流(例如，端口 111)。端口 111 指示这是 Sun RPC。一些像 Sun RPC 绑定查找这样的应用在解析器级是可以直接确定的(“已知”)，并且状态处理器应该为其它复杂的识别工作而继续进行的次态，表示为“st_D”，被放置在流－入口的字段 245 中。

当拦截到 Sun RPC 绑定查找应答时，由解析器再次构造流签名。这个流签名和 KEY－1 等同。因此，当签名从解析器子系统 301 进入分析器子系统 303 时，就得到了完整的流－入口，并且在这个流－入口中指示状态“st_D”。状态处理器指令数据库 326 中对状态“st_D”的操作命令状态处理器构造并存储一个新的流签名，在图 2 中显示为 KEY－2(212)。由状态处理器构造的这个流签名还包括目的地址 250 和源地址 251，分别针对服务器“S^1”和后跟(数值较大的地址)的客户机“C^1”。协议字段 252 定义要用的协议，例如“p^2”，它是

从应答分组得到的。字段 253 包含也是从应答分组得到的一个识别模式。这样，应用是 Sun RPC，而且字段 254 指示这个应用“a^2”。次态字段 255 定义状态处理器应该为其它复杂的识别工作而继续进行的下一个状态，例如状态“st^1”。在这个特定实例中，这是一个终结状态。这样，KEY－2 现在可以用来识别以任意方式与应用“a^2”相连的分组。图中显示了两个这样的分组 208 和 209，每个方向上一个。它们使用在原始绑定查找请求中所请求的特定应用服务，并且每一个都将被识别出来，因为将要在每种情况下构造签名 KEY－2。

两个流签名 210 和 212 总是把目的地址和源地址排列成“S_1”后跟“C_1”。这样的值是在第一次在特定流签名中创建地址时自动填充的。优选地，大量流签名被按照从低到高的顺序保存在查找表中以便最好地散布流签名和散列值。

其后，客户机和服务器交换大量分组，例如由请求分组 208 和应答分组 209 所表示。客户机 106 发送拥有目的地址 S_1 和源地址 C_1 的分组 208，这两个地址在一对地址字段 260 和 261 中。字段 262 定义协议为“p^2”，字段 263 定义目的端口号。

一些网络服务器应用识别工作非常简单，只要发生一个状态转换就能够查明产生分组的应用。其它的需要发生一系列的状态转换以便匹配已知和预定义的状态到状态的转变(climb)。

这样通过预定义当一个相对简单的 Sun 微系统远程过程调用绑定查找请求指令执行时对这个例子发生什么样的分组交换序列就自动设置了用于识别应用“a^2”的流签名。比这个更复杂的交换可以产生两个以上的流签名和它们的相应状态。每个识别可以包括设置一个复杂的状态转换表以在到达像字段 255 中的“st_1”这样的“终结”静止状态前对它们进行遍历。所有这些都用来为在将来识别特定应用构造流签名的最后集合。

状态处理器细节

状态处理器 1108 对新的和现有的流进行分析以便按照应用对它们进行分类。它通过根据由工程师定义的规则从状态到状态地进行来做到这一点。规则是测试后跟测试为真时要进行的下一个状态。状态处理器通过每个规则直到测试为真或者没有要执行的测试为止。状态处理器 1108 通过把解析器子系统 1000 识别出的上一个协

议用作进入跳转表(跳转向量)的位移来起启动这个过程。跳转表把我们带到用于那个协议的指令。多数指令测试某物是否是否在统一流键缓冲区 1103 或流－入口中，如果该流－入口存在的话。状态处理器必须进行测试位、比较、加或减以执行测试。

在多数通用处理系统中，实现的指令集实际上是通用的。所有的处理系统都有一个典型的有关对指令和程序计数器的分析和操作的指令集。这些指令包括跳转、调用和返回。另外，这些相同的处理系统包含适合的指令对寄存器和存储器位置进行分析和操作。这些指令包括加、减、移动、比较和逻辑操作。

优选实施方案的状态处理器 1108 还包括这样一个基本的标准指令集。但是，优选实施方案状态处理器 1108 还包括一些特殊的功能需要用来估计网络上分组中的数据内容。有四个由优选实施方案状态处理器执行的特殊功能来实现这些目标。其中两个是专用的转换指令，设计用来解释特定格式的文本元素并把它们转换成数学或数字形式。这些指令是 AH2B(ASCII 十六进制到二进制转换)和 AD2B(ASCII 十进制到二进制转换)。这些指令实际都是单周期指令。这些指令很奇特并被用来提供由优选实施方案状态处理器所执行的功能的时间敏感特性。

为了让系统速度加快并实现分类目标。状态处理器中还提供了几个其它特殊功能。这些功能主要处理查找、定位、分析和估计字符串序列。这些字符式可以是格式化的或无格式的。

初高级指令是 In_Find 和 In_Find_CONTINUE 指令，它们由状态处理器中的查找设备(查找引擎)实现。这些功能和查找设备已经设计用来使状态处理器能够同时查找来自发送到监控器 300 中的分组的有效负载内容。这使得监控器能够测量并满足任意网络速度需求。

状态处理器在图 19 中显示为处理器 1108。它执行它的来自状态处理器指令数据库(SPID)1109 的指令，SPID 由主机 CPU 作为编译方法 310 的一部分对其进行填充。SP 1108 包含几个子块，包括程序计数器 1903(SPPC)、控制块 1905(SPCB)、算术逻辑单元 1907(SPALU)、能够把数据从不同的源地址移动到不同的目标地址的地址发生器和数据总线多路复用器(复用器)。两个地址发生器是指向 UFKB 的 SP

流键地址发生器1911(SPFKAG)和指向高速缓存器子系统的SP流－入口地址发生器1913(SPFEAG)。SP 1108还包括四个数据复用器:SP ALU数据复用器 A 1919、SP ALU 数据复用器 B 1921、SP UFKB 数据 复用器 1915和SP 高速缓冲存储器数据复用器 1917。这些复用器促进在状态处理器1108的不同块中以及进出UFKB 1103和高速缓存器子系统1115的数据移动。

SP控制块1905对出自SPID 1109的指令进行解码并把它们分成不同的字段以控制状态处理器1108。SPCB1905的主要功能是指令解码和控制信号产生。有两类指令。一类完全由SPCB执行，一类被传递给SPALU 1907以部分或全部执行。下面描述了一些SP指令。

当一个指令需要被传递给SPALU 1907时，SPCB 1905对指令解码，在总线上给SPALU 1907提供指令码并确认“开始”信号。

当一个指令，例如移动或跳转指令，可以完全由SPCB 1905执行时，SPCB产生适当的控制信号给SP程序计数器1903、SP地址发生器1911和/或1913和SP 复用器以便实现特定的移动或跳转指令。

SPID 1109中的字是40位长并由SPCB 1905根据指令码分成不同的字段。一个字段是指令码。SPID 字中剩下的位被根据附随的指令分成不同字段。例如，SP 1108能够实现跳转、调用、等待和后面跟条件码和跳转地址的WaitRJ指令。SP 1108还能够执行后跟常量值的移动立即指令。此外，SP 1008能够执行装载地址发生器指令，它后跟要装载的地址。在对指令字段进行解码之后，SPCB 从所包括的解码 PAL 产生组合控制信号。这些控制信号选择不同的复用器加速数据移动并产生装载数值到不同寄存器的选通信号。

程序计数器SPPC 1903产生到状态处理器指令数据库中的地址。它包含一个产生 SPID 1109 地址的指令指针。指令指针可以增加或从跳转向量多路复用器装载，跳转向量多路复用器促进条件转移。指令指针可以从三个来源之一装载：(1)来自UFKB的协议指针，(2)来自当前已解码的指令的立即跳转向量或(3)由 SPALU 1907 提供的值。

在由LUE把流签名和已知协议标识符一起放到UFKB中之后，程序计数器 1903 被用解析器子系统识别出的最后一个协议进行初始化。第一个指令是跳转到对解码出的协议进行分析的子程序。

为了实现 JUMP 立即指令，程序计数器从带有跳转向量的 SPCB 1905 取出一个输入字段并用跳转向量装载指令指针。

状态处理器 ALU 1907 包含所有必要的算术、逻辑和字符串比较功能以实现状态处理器指令。SP ALU 1907 的主要块是：A 寄存器和 B 寄存器、指令解码&状态机、引用字符串存储器、查找引擎 1930、输出数据寄存器和输出控制寄存器。

状态处理器 ALU 查找引擎 1930(SPALU_SE)依次包含目标查找寄存器集、引用查找寄存器集和比较块。查找引擎 1930 能够在目标中的任何地方查找多达几个(在我们的实施方案中是四个)的引用字符串，并且如果找到了其中一个引用字符串，就返回它和它在目标中的位置。

流键地址发生器产生状态处理器正在统一流键缓冲区中访问的地址。SPFKAG 的主要单元是流键地址指针寄存器和产生地址的 ROM 解码。

流－入口地址发生器 1913 提供状态处理器正在高速缓存器子系统 1115 中访问流－入口的地址。如果流－入口存在，来自散列的高位地址用来在流数据库 324 中查找存储桶。中间位来自找到的存储桶记录。低位来自状态处理器 1108 正在使用的位移。

SPFKAG 的主要块是流键地址指针寄存器和产生地址的 ROM 解码。

状态处理器 UFKB 数据复用器 1915 选择去往 UFKB 的数据源。它把三个数据源中的一个送往 UFKB。这三个数据源是 ALU 输出数据总线、高速缓存器输出数据总线和 SPCB 数据。选择信号是一个 2－位信号。

状态处理器高速缓存器数据复用器 1917 从进入高速缓存器子系统的四个数据源中选出去往高速缓冲存储器子系统的数据源。这四个数据源是：ALU 输出数据总线、UFKB 数据总线的低位、UFKB 数据总线的高位和 SPCB 数据。选择信号是一个 2－位信号。为了能够进行 16 位移动，SPMUXCA 引入两个 16 位复用器向高速缓存器子系统的低 16 位和高 16 位提供信息。

状态处理器 ALU 数据复用器 A 1919 选择去往 UFKB 的数据源并把三个 32 位数据源之一送往 ALU 的 A 端。这三个数据源是：高速缓存

器子系统数据总线、UFKB 数据总线的低 32 位和 UFKB 数据总线的高 32 位。选择信号是一个 2－位信号。

状态处理器 ALU 数据复用器 B 1919 选择去往 SP ALU B 端的数据源并把两个 32 位数据源之一送往 ALU 的 B 端。这两个数据源是高速缓存器子系统数据总线和 SPCB 数据字。选择信号是一个 1－位信号。

状态处理器指令定义

下面几节描述一些在状态处理器 1108 中可用的指令。应当指出通常并未向状态处理器 1108 提供汇编程序。这是因为工程师一般不需要为这个处理器编写代码。编译器编写代码并从协议列表(PDL 文件)中定义的协议把它装载到状态处理器指令数据库中。

下面的表格被分成两个实施方案：实施方案 1 和实施方案 2。实施方案 2 更复杂一些，它包括实施方案 1 指令的更复杂的版本以及附加指令。

状态处理器指令定义	
实施方案 1 指令(简单实施方案)	
指令	说明
In_Noop	无操作
In_Wait	等待条件发生，根据条件执行绝对跳转
In_Call	调用子程序
In_Return	从子程序返回
In_WaitJR	等待一个条件发生，根据条件执行相对跳转
In_Jump	根据条件跳转到一个立即跳转向量
In_Move	把数据从 X 位置移到 Y
In_Load_FKAG	装载 FK 地址发生器 1911
In_Inc_FKAG	增加 FK 地址发生器 1911
In_Dec_FKAG	减小 FK 地址发生器 1911
In_Load_FEAG	装载 FE 地址发生器 1913
In_Inc_FEAG	增加 FE 地址发生器 1913
In_Dec_FEAG	减小 FE 地址发生器 1913
In_Set_SPDone	设置 SP 完成位，指示 SP1108 完成的标志

实施方案 1 ALU 指令	

指令	说明
In_INC	增加寄存器 A 的值
In_DEC	减小寄存器 A 的值
In_ADD	寄存器 A+寄存器 B
In_SUB	寄存器 A－寄存器 B
In_AND	寄存器 A 和寄存器 B 按位与
In_OR	寄存器 A 和寄存器 B 按位或
In_XOR	寄存器 A 和寄存器 B 按位异或
In_COM	寄存器 A 按位求补
In_Simple_Compare	比较寄存器 A 和寄存器 B，如果相等返回 SPALU_MATCH

实施方案 2 ALU 指令(更复杂的实现)	
指令	说明
In_Compare	看固定位置上的字符串是否和引用字符串数组中的一个相匹配
In_Find	在一个范围内查找一个(或一组)字符串，起始位置未知
In_FindContinue	从找到上一个字符串的位置开始执行查找操作
In_AD2B	把 ASCII 十进制字符转换为二进制
In_AD2BContinue	把 ASCII 十进制字符转换为二进制
In_AD2B	把 ASCII 十六进制字符转换为二进制
In_AH2BContinue	把 ASCII 十六进制字符转换器二进制

现在详细描述其中一些指令。

Move

移动指令集包括处理不同大小的字从源地址移动到目的地址的特殊移动指令。移动指令集已经开发出来以确保字的大小总是匹配的，包括 32 位和 16 位移动指令。

移动指令从以下位置移动数据：立即数到 SP ALU B 寄存器、立

即数到高速缓存器子系统、立即数到 UFKB、SP ALU 输出到 UFKB、SP ALU 输出到高速缓存器子系统、高速缓存器到 SP ALU A 寄存器、高速缓存器到 SP ALU B 寄存器、UFKB 到高速缓存器子系统以及 UFKB 到 SP ALU A 寄存器。

In_Compare

In_Compare 指令命令 ALU 1907 执行比较操作并和匹配的字符串一起返回一个 MATCH 信号。比较操作对第一个字符在 UFKB 中一个未知位置的目标数据和引用字符串存储器中的已知引用字符串进行比较。在执行这个指令之前，要用指向目标字符的地址装载 SP UFKB 地址发生器 1911。ALU 引用字符串中的一个位置保存一串引用字符以进行比较。

ASCII 十进制转二进制

这个指令传入表示十进制值的 ASCII 码字符串的位置。结果是等价的二进制值。这个指令在一个周期内执行完毕。

ASCII 十六进制转二进制

这个指令传入表示十六进制值的 ASCII 码字符串的位置。结果是等价的二进制值。这个指令在一个周期内执行完毕。

In_Find

In_Find_Continue

这些指令在下面描述查找引擎 1930 一节中有详细描述。

查找引擎和调用查找引擎的 SP 指令

监控器 300 的一个方面是它能够实时分析每个分组。网络流量可以在非常快的速率上移动。状态处理器 1107 的一个任务是查找数据中一个或多个已知的字符串。这样的查找在 UFKB 记录上执行，例如在记录的有效负载部分中执行。查找可以是记录的已知部分或未知部分，例如 UFKB 记录的有效负载部分中的任意地方。此外，查找可能需要以非常高的速率执行。

状态处理器 ALU 指令包括执行这样查找的查找引擎 SPALU_SE 1930。查找引擎 1930 能够在 UFKB 中的目标数据里查找最高达四个引用字符串，并且并行指示(1)是否在目标中的任意地方找到了四个字符串中的任意一个，(2)找到了哪一个字符串，以及(3)在目标中的什么地方找到了该字符串。

查找引擎为执行下列状态处理器指令做准备。

In_Find

In_Find 指令向 ALU－查找引擎提供信息以执行查找操作并和匹配的字符串及在目标中找到该字符串的位置一起返回匹配信号。

指令格式如下:

In_Find[引用字符串数组地址], [UFKB 字节位移], [范围]

指令字定义	
位	说明
In_Find	操作码
N(A 总线宽度)	ALU 引用存储器中引用字符串数组地址。ALU 引用存储能够存储一组一到四个要查找的引用字符串。每个引用字符串是 N_R 个单元长。在我们的实现中，单元是字节，N_R 是 16。
位移(2:0)	UFKB 字节位移 这是指向选中的 UFKB 记录中的一个字节的位移地址。 位移用来确定选中的 UFKB 记录中的哪个字节是开始查找操作的第一个字节位置。如果 UFKB 是 64 位(8 字节)，这个字段就是 3 个字节宽并且指向第一个目标字节以开始查找操作。
范围(7:0)	范围，用字节数表示，在要查找的 UFKB 区域中。这意味着要查找的字节数。它通常指定要在特定 UFKB 记录中查找多少字节。 如果在比较这个范围后没有产生完全匹配，查找操作结束。

In_Find 操作的引用字符串存储器数据结构

位字段	说明
字符串#(8 位)	数组中的字符串#指示这个数组中的字符串总数。对 1、2、3 或 4 个字符来说有效数字是 0、1、2、3。给它分配 8 位是便于将来的扩展并且简化了实现。
1st 字符串的大小(4 位)	这个参数以字节指示 1st 字符串的大小。放在这里的值是 $N_{R1}-1$。对一个小到 1 个大到 0xF 个字符的字符串来说有效数字是 0－F。
2nd 字符串的大小(4 位)	这个参数以字节指示 2nd 字符串的大小。放在这里的值是 $N_{R2}-1$。对一个小到 1 个大到 0xF 个字符的字符串来说有效数字是 0－F。
3rd 字符串的大小(4 位)	这个参数以字节指示 3rd 字符串的大小。放在这里的值是 $N_{R3}-1$。对一个小到 1 个大到 0xF 个字符的字符串来说有效数字是 0－F。
4th 字符串的大小(4 位)	这个参数以字节指示 4th 字符串的大小。放在这里的值是 $N_{R4}-1$。对一个小到 1 个大到 0xF 个字符的字符串来说有效数字是 0－F。
字符串 1	字符串 1 的 1 到 16(=N_R)个字符
字符串 2	字符串 2 的 1 到 16(=N_R)个字符
字符串 3	字符串 3 的 1 到 16(=N_R)个字符
字符串 4	字符串 4 的 1 到 16(=N_R)个字符
向量(16 位)	这是返回到程序计数器的一个 16 位(即 N_R 位)向量以指向 SPID 中处理 In_Find 结果的一个区域。

当查找完成时，确认查找完成位。根据查找结果确认或重置 MATCH 位。ALU 中的总线，称 ALU_DATA 总线，将保持下列信息:

- 跳转向量[15:0]—这是存储在引用字符串数组中的一个向量并指示当找到一个引用字符串时状态处理器跳转到什么指令(例如，子程序)。
- 字符串代号[1:0]—这是指示查找到哪个字符串的字符串代号，对四个字符串实现来说就是 0、1、2、3。

保存流键缓冲区中找到字符串的位置。这是UFKB字地址和目标中找到的字符串的第一个字符的字节位置。

如果找到了任意引用字符串的第一次出现或者在整个查找范围内都没有匹配，查找就完成了。

考虑下面的例子。假设我们希望在UFKB的有效负载区域查找一个引用字符串而且从有效负载的第5字节开始查找，在第100字节结束查找。假定该引用字符串位于0050h处。这个实例的指令格式如下：

In_Load_FKAG，有效负载地址

In_Find，0050_{16}，5，60_{16}

范围是$100-9+1=60_{16}$

考虑第二个例子，查找UFKB中的12_{16}到$2A_{16}$。下面的状态处理器指令将实现这一目标。

In_Load_FKAG 02_{16}

In_Find [引用字符串地址]，2，19_{16}

注意$2A_{16}-12_{16}+1=19_{16}$。

In_Find_Continue

这个指令跟随在In_Find指令后面并告诉ALU-查找引擎1930从发现上一个字符串的位置开始执行查找操作并与匹配的字符串信息和找到目标字符串的位置一起返回MTCH信号。这个指令的目的是帮助从以前查找结束的地方开始查找一个新的引用字符串。因此，就不向查找引擎提供位移因为查找引擎将记住它结束它以前的查找的位置。

指令格式如下：

In_Fide_Continue[引用字符串数组地址]，[0]，[范围]

指令字定义	
位	说明
In_Fid	操作码
N(A 总线的宽度)	ALU 引用存储器中止引用字符串数组地址。 在这个位置，有要查找的一组一到四个引用字符串。 下面在引用存储器数据结构一节中定义了数组中的 引用字符串数据结构。(缺省 N=16)
位移(2:0)	UFKB 字节位移，总为零。
范围(7:0)	范围用字节数表示，在要查找的 UFKB 区域中。这 意味着要查找的字节数。 如果在比较这个范围后没有产生完全匹配，查找操作结束。

作为例子，假定我们希望在 UFKB 的有效负载区域 In_Find 一个字符串(字符串 A)，并且查找从有效负载的第 5 个字节开始，在第 100 个字节结束。假定引用字符串(字符串 A)位于 0050_{16}。在找到第一个引用字符串之后，假定我们希望在随后的 30_{16} 个字节中继续查找一个新的字符串(字符串 B)。假定字符串 B 位于 0080h。

用于这个例子的指令格式如下：

In_Load_FKAG，有效负载地址

In_Find，0050_{16}，5，60_{16}

……

In_Find_Continue，0080_{16}，5，30_{16}

范围是 100－5+1=96=60_{16}

图 20 是查找引擎(SPALU_SE)1930 的总框图，它是 ALU 1907 的一部分并且执行提交给 ALU 1907 的 In_Find 和 In_Find_Continue 指令。In_Find 指令查找 UFKB 的一个区域并在目标(UFKB)区域中查找最多达四个可能的引用字符串。引用字符串存储在 ALU 引用字符

串存储器中。

如图 20 中所示，查找引擎和下列部件相连：

(a) ALU 字符串引用存储器，存储引用字符串。

(b) SPALU 数据复用器 A 1919，通过它提供用于以匹配引用字符串的目标数据。在 In_Find 指令使用 SP_UFKB 数据复用器 1915 的操作中连接到 UFKB。

(c) SPALU 数据复用器 1921，通过它提供指令码，包括启动查找的“开始”信号 SPALUGO 2005。

(d) 状态处理器流键地址发生器 1911，用来增加及减小 UFKB 地址。

(e) 状态处理器指令计数器 1903，在这里报告查找结果。

该系统通过时钟信号 CLK 2001 运转并可以由 RESET 信号进行重启。指令译码块 SE_INST 2009 对 In_Find 和 In_Find_Continue 的指令码进行解码并在 SPALUGO 信号 2005 激活时启动查找引擎。查找引擎连续不断地监控 SPMuxB 1921 输出总线 2007 和 SPALUGO 信号 2005 以检测 In_Find 和 In_Find_Continue 指令。在查找引擎 1930 操作期间该引擎通过 SPMUXA 1919 从 UFKB 1103 以字长接收目标数据 2011。同样地，来自引用字符串存储器 2003 的适当地址的引用字符串以 SP_Data_RMB 2013 到达。

一旦提交了 In_Find 或 In_Find_Continue 指令，查找引擎引用装载(SE_LOAD)模块 2015 负责“清理”引用字符串寄存器。它从引用字符串存储器 2003 取出一个引用字符串数组对其进行解释并用该信息装载引用字符串寄存器。

在处理状态中，SE_LOAD 模块 2015 首先从引用存储器 2003 的起始位置装载第一个字。假定在提交该指令之前起始位置就被设置在正确的位置。一旦装载了字符串数量和字符串大小，装载方法继续装载所有的引用字符串。增加引用信号 2025 增加正在装载引用字符串的引用存储器地址。在字符串的装载过程中未确认 LOAD_KEY_DONE 信号 2017。当正在装载最后一个引用字符串的最后一个字时用脉冲传送 LOAD_KEY 信号 2019，指示查找引擎模块 2030 从下一个时钟周期开始查找。在下一个时钟周期中确认 LOAD_KEY_DONE 信号 2017 并同时从引用存储器 2003 装载跳转向量 2021。

查找引擎增加/控制模块(SE_INC)2023 负责增加流键地址发生器 1911 以便从 UFKB 向查找引擎提供新的字。它监控出自查找引擎模块的找到信号并报告结果。SE_INC 2023 还负责计算真正的结束地址并根据 In_Find 指令中提供的范围确定最后一个字中要检查的最后一个字节。

SE－4SEARCH 2030 模块包括四个查找引擎以便同时查找四个字符串。该引擎对四个引用字符串的每一个输出找到信号 2031 和位置信号 2033 以指示是否找到该字符串及在哪里找到了该字符串。

由 SE－INC 2023 确认的 SPALU_Done 信号指示查找完成。如果同时确认了 SPALU_Match 信号 2037 那么表示这是一个成功查找。成功查找还导致 SPALU_Data 总线 2039 携带跳转向量和找到引用字符串的查找引擎编号。

本发明的一个方面是查找快速。从提交指令以在 UFKB 中的 N 个字中的任意位置开始查找到确认 SPALU_Done 信号 2035 的最长时间是 N 个时钟周期加上用于预装入和成功查找情况下的指针调整的附加时钟周期。在我们的实施方案中，这个附加负担是 11 个时钟周期。因此，每个额外的字仅占用一个时钟周期。

图 21 显示了包括四个单一查找模块的 SE_4SEARCH 模块 2030。替代实施方案可以包括更多的单一查找模块以便能够同时查找多于四个的引用字符串。四个查找引擎模块中的每一个都是相同的，因此只描述一个这样地模块 2103。

每个单一查找模块 2103 执行一个单独的引用字符串查找。使用这个模块的多个副本可以在公共源缓冲区(UFKB)中查找多个不同的引用字符串。该模块包括核心比较器矩阵块 2105(查找引擎核心)和状态机 SE_SM 2107。查找引擎核心 2105 能够对长达 N_R 个单元(在我们的实施方案中是 16 字节)的引用字符串和三个 8－字节字(在三个连续时钟周期中装入，每次一个字)的目标字符串进行比较。在每个时钟周期中，单一查找模块 2103 从第一个字的 8 个字节中的任意一个开始在目标中查找引用字符串。每一个引用字符串字节都附有一个校验位，指示是否检查这个字节。如果确认了该校验位，就禁止相应的字节检验。随着 64－位(8 字节)字被以流水方式装入到三个寄存器中，在它们被取出之后两个时钟周期比较开始。

如果查找成功需要调整源(UFKB)地址指针。如果查找成功，匹配信号 2111 成为活动的并且引用字符串的第一个字节的位置被放在位置总线 3113 上。SE_SM 状态机 2107 每个时钟周期执行若干任务。它有三个状态：复位、空闲和进行。当处于空闲状态时，状态机 2107 等待来自 SE_LOAD 模块 2105 的信号以切换到进行状态。在进行状态的第一个时钟周期中，如果匹配发生就核对位置和字节位移 2115。如果字节位移大于位置，就忽略掉它，即没有确认找到。同样，如果它是要检查的最后一个字，就检验终点位移字节和位置，如果位置大于范围 2117 中要检查的最后一个字节就把它忽略掉。否则，就在查找引擎核心 2105 发现匹配时确认找到信号并锁住该位置把它转发到 SE_INC 模块 2023。

现在看图 22A，查找引擎核心 2105 是查找引擎模块的核心比较器矩阵。它在目标字符串中查找 N_R－单元的引用字符串。它包括一个引用轴和一个目标轴。引用轴包括保存引用字符串的寄存器，通常字符串是 N_R 个单元。在优选实施方案中单元是字节，N_R 是 16，并且很容易进行更改以查找其它引用字符串大小。目标数据被组织成字。目标数据被排列成一组单字寄存器，每个单字寄存器保存目标数据的一个字。在一个时钟周期内打入寄存器一个字。这样，目标数据的第一个字就在三个时钟周期内打入目标轴。

查找引擎核心 2105 包括至少一个 N_R－单元比较器，每一个有 N_R 对输入和一个输出，输出指示 N_R－对输入中每一对的匹配。图 22B 中显示了一个这样的比较器 2203。输入对显示为(2207－1，2209－1)、(2207－2，22－09－2)、…(2207－N_R，2209－N_R)。输出是 2211。图 22A 显示了多个(比如 N_{start} 个)比较器，标为 2203－1、2203－2、…、2203－N_{start}。考虑这些比较器中的任意一个，查找引擎核心还包括一个连接矩阵，其中包括沿着引用轴指示引用字符串值的 N_R 个连接和沿着目标轴指示目标数据值的 N_R 个连接，从目标数据的第一个起始位置开始并在结束位置结束的目标数据连接。比较器是连在一起的，这样当引用轴和目标轴相互垂直面向时，任意比较器都是沿着矩阵的对角线定向，这样目标数据的 N_R 个连接就可以和引用字符串进行比较。每个比较器在目标轴中的一个不同位置开始。在我们的实现中，每个字是 8 字节长，比较器数量 N_{start} 等于字长，并且比较

器从目标轴上的连续位置开始。这样，在一个时钟周期内查找引擎核心 2105 能够找到从第一个字中的任意位置开始的引用字符串。注意如果目标字符串正巧跨越了一个字边界，查找引擎核心仍将自动找到该字。

比较器的输出是 N_{start} 输入优先级编码器的输入，它指示如果找到了一个字符串的话是 N_{start} 个比较器中的哪一个找到了该字符串。这提供了位置。

在装入数据之后的操作中，状态机 2107 在第一个时钟周期将忽略可能在第一组少量比较器中找到的任意字符串，比较器数量由位移指示。在每个后继时钟周期中，查找引擎 2105 将找到从第一组 N_{start} 个位置中的任意地方开始的引用字符串。这样，在多个时钟周期中查找引擎核心 2105 将找到目标数据中任意位置的引用字符串。

图 23 更详细地显示了输入核心的一种实现。这个实现用于找到在一组起始位置的任意一个中的引用字符串。该实现包括用于接收引用字符串之一的 N_R 个单元的引用寄存器 2203、连接成组以接收目标数据的一组目标数据寄存器 2205、多个比较器集、对应每个起始位置的一个比较器集、连接到引用寄存器的每个单元—在这个实现中指是每个字节—和从特定起始位置开始的目标数据寄存器的 N_R 个单元—这种情况下也是字节—的比较器集，并比较第一引用寄存器内容和从特定起始位置开始的目标寄存器的 N_R 个单元。每个比较器集指示从它相应的不同起始位置开始的目标数据中是否有第一引用字符串的匹配。可能的起始位置集可以包括 N_{start} 个不同位置。这些位置可以是连续的或不连续的，如果连续，就把一个或多个目标数据寄存器连接成组以接收目标数据的至少 $N_R+N_{start}-1$ 个单元。这样就有了 N_{start} 个比较器集，对 N_{start} 个起始位置中的每一个都有一个比较器集。

每个比较器集包括 N_R 个连续的比较器。图 23B 中显示了一个这样的比较器 2313，并且包括引用输入、目标输入、使能输入和一个指示匹配的输出，这样就在引用和目标输入匹配并且使能输入被确认时确认比较器 2313 的输出。对针对特定起始位置的特定比较器集来说，连续比较器的引用输入连接到引用寄存器的连续单元上，连续比较器的目标输入连接到从该特定起始位置开始的目标数据寄存

器的连续单元上，使能集合中的第一个比较器，并且每个比较器的使能输入连接到前一个比较器的输出上，这样当引用字符串的 N_R 个单元和目标数据的 N_R 个单元一致时确认最后一个比较器的输出。

用这种方式，状态处理器能够以极高的速率在分组(存储在 UFKB 中)区域中的未知位置上定位字符串。

高速缓冲存储器子系统

再参考图 11，高速缓存器子系统 1115 连接到查找更新引擎(LUE)1107、状态处理器(SP)1108 和流插入/删除引擎(FIDE)上。高速缓冲存储器 1115 保存存储在存储器 1123 中的流－入口数据库中的一组流－入口，因而它通过统一存储器控制 1119 连接到存储器 1123。根据本发明的一个方面，高速缓存器中的这些记录是那些下次很可能被访问的记录。

希望能够使高速缓存器系统中的命中率达到最大。典型的现有技术高速缓存器系统用来加速从微处理器系统到存储器和从存储器到微处理器系统的访问速度。在这样的现有技术系统中有几种机制可用来预测查找这样就能使命中率达到最大。例如现有技术的高速缓存器可以使用前视机制来预测指令高速缓存器查找和数据高速缓存器查找。这样的前视机制无法用于高速缓存器子系统 1115 的监控器应用。当新分组进入监控器 300 时，下一个高速缓存器访问，例如来自 LUE 1107，可能是针对与上次高速缓存器查找完全不同的流，而且无法提前知道下一个分组将属于哪一个流。

本发明的一个方面是当需要高速缓存器替换时替换掉最近最少使用(LRU)的流－入口的高速缓存器系统。替换最近最少使用的流－入口是首选的，因为跟随当前分组的下一个分组很可能属于同一个流。这样，新分组的签名很可能将匹配最近使用过的流－入口。反之，与最近最少使用的流－入口相连的分组最不可能很快到达。

此外，在对流－入口进行操作的其中一个引擎，例如 LUE1107，完成了在流－入口上的操作之后，同一引擎或其它引擎很可能将很快用到同一个流－入口。这样就希望确保最近使用过的流－入口保留在高速缓冲存储器中。

本发明的高速缓存器的一个特性是无论何时只要有可能最近使用最多(MLU)的流－入口都被保存在高速缓存器中。既然通常同一流

的分组总是大量到达并且 MRU 流－入口很可能为分析子系统中的其它引擎所需要，尽最大可能把 MRU 流－入口保留在高速缓存器中就能提高在高速缓存器中找到流记录的可能性，这样就提高的高速缓存器命中率。

然而本高速缓存器发明的另一方面是它包括一个使用一组相联存储单元(CAMs)的相联存储器。CAM 包含地址，在我们的实现中它是与包括存储器单元的高速缓冲存储器(例如，数据 RAM)中对应的流－入口相联的散列值。在一种实施方案中，每个存储器单元是一个页。每个 CAM 还包括指向高速缓冲存储器页的指针。这样，CAM 内容包括地址和指向高速缓冲存储器的指针。通常，每个 CAM 单元包括带有输入的匹配电路。把散列值送进 CAM 的匹配电路输入中，如果该散列值匹配 CAM 中的散列值，就确认匹配输出，指示有了一个命中。CAM 指针指向流－入口在高速缓冲存储器中的页号(即地址)。

每个 CAM 还包括高速缓存器地址输入、高速缓存器指针输入和高速缓存器内容输出以输入和输出 CAM 的地址部分和指针部分。

特定实施方案高速缓存器存储一个存储桶的页中的流－入口，即它能够存储单独的流－入口。这样，指针就是高速缓冲存储器中的页号。在一种版本当中，每个散列值对应 N 个流－入口(例如，在这个版本的优选实施方案中是 4 个存储桶)的存储柜。在另一种实现中，每个散列值指向一个单一地流－入口，即存储柜和存储桶大小相当。为简单起见，在描述高速缓冲存储器 1115 时假定采用第二种实现。

此外，按照常规，提供匹配输出信号给高速缓存器中的相应位置，因此可以在高速缓冲存储器中的该位置进行读写操作。

本发明的一个方面实现了相联和真正的 LRU 替换策略的结合。为了这一点，高速缓存器系统 1115 的 CAMs 被按照一种顺序组织成我们称为 CAM 栈(也叫 CAM 阵列)的形式，带有顶部 CAM 和底部 CAM。从顶部 CAM 开始每个 CAM 的地址和指针输出都连接到下一个 CAM 的地址和指针输入上，直到底部。

在我们的实现中，散列用来给高速缓存器定址。散列是 CAM 阵列的输入，所拥有的地址与输入散列相匹配的任意 CAM 都确认它的指示命中的区配输出。当产生高速缓冲存储器命中时，产生命中的 CAM

的内容(包括地址和指向高速缓冲存储器的指针)被放入栈顶的 CAM 中。在产生命中的 CAM 之上的那些 CAM 内容(高速缓冲存储器地址和高速缓冲存储器指针)下移以填充空隙。

如果产生未命中,就把任意新的流记录放在由底部 CAM 所指的高速缓冲存储器单元中。底部之上的所有 CAM 内容均下移一格,并把新的散列值和指向新的流-入口的高速缓冲存储器的指针放在 CAM 栈的最顶部的 CAM 中。

用这种方式,CAMs 被按照使用的远近顺序进行排列,底部 CAM 指向最近最少使用的高速缓存器内容,顶部 CAM 指向最近使用过的高速缓存器内容。

此外,和通常基于 CAM 的高速缓存器不同的是,在 CAM 中的地址和它所指向的高速缓冲存储器的哪个单元之间没有固定的关系。CAM 和高速缓冲存储器页之间的关系随着时间而变化。例如,在某个时刻,栈中的第五个 CAM 包含指向高速缓冲存储器的一个特定页的指针,一段时间过后,同样是第五个 CAM 可以指向不同的高速缓冲存储器页。

在一种实施方案中,CAM 阵列包括 32 个 CAM 以及包含 32 个存储器单元(即存储器页)的高速缓冲存储器,每个 CAM 内容指向一个页。假定 CAMs 分别编号为 CAM_0、CAM_1、…、CAM_{31},CAM_0 和 CAM_{31} 是栈中的顶部和底部 CAM。

CAM 阵列由 CAM 控制器控制,CAM 控制器实现成一个状态机,高速缓冲存储器由高速缓冲存储器控制器控制,它也实现成一个状态机。本领域的技术人员从这个操作描述可以了解对这种控制器的需要以及如何把它们实现成状态机或其它形式。为了不至于混淆这些控制器和其它控制器,例如统一存储器控制,我们把这两个控制器分别称为 CAM 状态机和存储器状态机。

作为一个例子来考虑,高速缓存器的状态是它是满的。进一步假定下表中显示了 CAM 栈的内容(地址和指向高速缓冲存储器的指针)和在高速缓冲存储器的每个页号地址上的高速缓冲存储器内容。

CAM	散列	高速缓冲存储器指针		高速缓冲存储器地址	内容
CAM_0	$hash_0$	$page_0$		$page_0$	$entry_0$
CAM_1	$hash_1$	$page_1$		$page_1$	$entry_1$
CAM_2	$hash_2$	$page_2$		$page_2$	$entry_2$
CAM_3	$hash_3$	$page_3$		$page_3$	$entry_3$
CAM_4	$hash_4$	$page_4$		$page_4$	$entry_4$
CAM_5	$hash_5$	$page_5$		$Page_5$	$entry_5$
CAM_6	$hash_6$	$page_6$		$page_6$	$entry_6$
CAM_7	$hash_7$	$page_7$		$page_7$	$entry_7$
……	……	……		……	……
CAM_{29}	$hash_{29}$	$page_{29}$		$page_{29}$	$entry_{29}$
CAM_{30}	$hash_{30}$	$page_{30}$		$page_{30}$	$entry_{30}$
CAM_{31}	$hash_{31}$	$page_{31}$		$page_{31}$	$entry_{31}$

这说明 CAM_4 包含并将匹配散列值 $hash_4$，用 $hash_4$ 进行查找将产生匹配和高速缓冲存储器中的地址 $page_4$。此外，高速缓冲存储器中的 $page_4$ 包含流－入口 $entry_4$，用这个符号表示它和散列值 $hash_4$ 相匹配。这个表还指示对 $hash_0$ 的使用比 $hash_1$ 更近，$hash_5$ 比 $hash_6$ 更近，等等，$hash_{31}$ 是最近最少使用的散列值。进一步假定 LUE 1107 用散列值 $hash_{31}$ 从统一流键缓冲区 1103 获得一个记录。LUE 通过 CAM 阵列查找高速缓存器子系统。CAM_{31} 获得命中并返回命中的页号，即 $page_{31}$。高速缓存器子系统现在向 LUE 1107 指示所提供的散列值产生了命中并向它提供指向高速缓冲存储器的 $page_{31}$ 的指针，$page_{31}$ 包含与 $hash_{31}$ 对应的流－入口，即 $flow_{31}$。LUE 现在从高速缓冲存储器在地址 $page_{31}$ 获得流－入口 $flow_{31}$。在优选实施方案中，对高速缓存器的查找只占用一个时钟周期。

值 $hash_{31}$ 是最近使用过的散列值。因此，根据发明的高速缓冲存储器系统的一个方面，要把最近使用过的记录放在 CAM 栈的顶部。这样 $hash_{31}$ 就被放进 CAM_0 中。此外，$hash_{30}$ 现在是 LRU 散列值，所以把它移到 CAM_{31}。下一个最近最少使用的散列值 $hash_{29}$ 移到 CAM_{30} 中，

等等。这样，在把 MSU 记录放到顶部 CAM 之后所有的 CAM 内容都下移一格。在优选实施方案中下移 CAM 记录只占用一个时钟周期。这样，查找和重排 CAM 阵列以根据最近使用情况维持次序关系。下表显示了 CAM 阵列的新内容以及高速缓冲存储器的内容(无变化)。

CAM	散列	高速缓冲存储器指针		高速缓冲存储器地址	内容
CAM_0	$hash_{31}$	$page_{31}$		$page_0$	$entry_0$
CAM_1	$hash_0$	$page_0$		$page_1$	$entry_1$
CAM_2	$hash_1$	$page_1$		$page_2$	$entry_2$
CAM_3	$hash_2$	$page_2$		$page_3$	$entry_3$
CAM_4	$hash_3$	$page_3$		$page_4$	$entry_4$
CAM_5	$hash_4$	$page_4$		$Page_5$	$entry_5$
CAM_6	$hash_5$	$page_5$		$page_6$	$entry_6$
CAM_7	$hash_6$	$page_6$		$page_7$	$entry_7$
……	……	……		……	……
CAM_{29}	$hash_{28}$	$page_{28}$		$page_{29}$	$entry_{29}$
CAM_{30}	$hash_{29}$	$page_{29}$		$page_{30}$	$entry_{30}$
CAM_{31}	$hash_{30}$	$page_{30}$		$page_{31}$	$entry_{31}$

为了延续这个例子，假定过些时候 LUE 1007 用散列值 $hash_5$ 进行查找。这在指向高速缓冲存储器 $page_5$ 的 CAM_6 中产生命中。这样，在一个时钟周期内，高速缓存器子系统 1115 向 LUE 1007 提供命中指示以及指向指向高速缓冲存储器中的流－入口的指针。最近的记录是 hash5，因此把 $hash_5$ 和高速缓冲存储器地址 $page_5$ 送入 CAM_0 中。剩下的 CAMs 内容都下移一格直到并且包括包含 $hash_5$ 的记录。也就是说，CAM_7、CAM_8、…、CAM_{31} 保持不变。CAM 阵列内容和没有发生变化的高速缓冲存储器内容显示在下表中。

CAM	散列	高速缓冲存储器指针		高速缓冲存储器地址	内容
CAM_0	$hash_5$	$page_5$		$page_0$	$entry_0$
CAM_1	$hash_{31}$	$page_{31}$		$page_1$	$entry_1$
CAM_2	$hash_0$	$page_0$		$page_2$	$entry_2$
CAM_3	$hash_1$	$page_1$		$page_3$	$entry_3$
CAM_4	$hash_2$	$page_2$		$page_4$	$entry_4$
CAM_5	$hash_3$	$page_3$		$Page_5$	$entry_5$
CAM_6	$hash_4$	$page_4$		$page_6$	$entry_6$
CAM_7	$hash_6$	$page_6$		$page_7$	$entry_7$
……	……	……		……	……
CAM_{29}	$hash_{28}$	$page_{28}$		$page_{29}$	$entry_{29}$
CAM_{30}	$hash_{29}$	$page_{29}$		$page_{30}$	$entry_{30}$
CAM_{31}	$hash_{30}$	$page_{30}$		$page_{31}$	$entry_{31}$

这样，在高速缓存器命中的情况下，CAM 阵列总是保持用过的散列值按使用的远近顺序排列，最近使用过的散列值保持在顶部 CAM 中。

将通过继续这个例子描述产生高速缓存器命中时高速缓存器子系统的操作。假定有一个对散列值 $hash_{43}$ 的查找(例如，来自 LUE 1007)。CAM 阵列产生未命中，导致使用该散列在用外部存储器中查找。我们的特定实现的特定操作是 CAM 状态机发送一个 GET 消息给存储器状态机，这导致通过统一存储器控制(UMC)1119 使用该散列的存储器查找。但是，本领域中的那些技术人员也应该了解当 CAM 阵列中产生未命中时实现存储器查找的其它方法。

在流－入口数据库 324(即外部存储器)中的查找产生命中或未命中。假定流－入口数据库 324 没有与散列值 $hash_{43}$ 相匹配的记录。存储器状态机向 CAM 状态机指示未命中，CAM 状态机随后向 LUE 1007 指示未命中。另一方面，假定在数据库 324 中有匹配散列值 $hash_{43}$ 的流－入口—$entry_{43}$。假若这样，就把该流－入口取出装入到高速缓存器中。

根据本发明的另一方面，底部 CAM 记录 CAM_{31} 总是指向高速缓冲存储器中的 LRU 地址。这样，实现真正的 LRU 替换策略包括清洗出 LRU 高速缓冲存储器记录并在底部 CAM 所指的高速缓冲存储器位置中插入一个新记录。并对该 CAM 记录进行修改以反映它所指向的高速缓冲存储器单元中的记录的散列值。这样，散列值 $hash_{43}$ 被放入 CAM_{31} 中并且记录 $entry_{43}$ 也被放入 CAM_{31} 所指的高速缓存器页中。现在 CAM 阵列和改变了的高速缓冲存储器内容是：

CAM	散列	高速缓冲存储器指针		高速缓冲存储器地址	内容
CAM_0	$hash_5$	$page_5$		$page_0$	$entry_0$
CAM_1	$hash_{31}$	$page_{31}$		$page_1$	$entry_1$
CAM_2	$hash_0$	$page_0$		$page_2$	$entry_2$
CAM_3	$hash_1$	$page_1$		$page_3$	$entry_3$
CAM_4	$hash_2$	$page_2$		$page_4$	$entry_4$
CAM_5	$hash_3$	$page_3$		$Page_5$	$entry_5$
CAM_6	$hash_4$	$page_4$		$page_6$	$entry_6$
CAM_7	$hash_6$	$page_6$		$page_7$	$entry_7$
……	……	……		……	……
CAM_{29}	$hash_{28}$	$page_{28}$		$page_{29}$	$entry_{29}$
CAM_{30}	$hash_{29}$	$page_{29}$		$page_{30}$	$entry_{43}$
CAM_{31}	$hash_{43}$	$page_{30}$		$page_{31}$	$entry_{31}$

注意插入的记录是现在的 MRU 流－入口。因此，现在要把 CAM_{31} 的内容移到 CAM_0 并且顶部 31 个 CAMs 中的以前的记录要再下移一次，底部 CAM 指向 LRU 高速缓冲存储器页。

CAM	散列	高速缓冲存储器指针		高速缓冲存储器地址	内容
CAM_0	$hash_{43}$	$page_{30}$		$page_0$	$entry_0$
CAM_1	$hash_5$	$page_5$		$page_1$	$entry_1$
CAM_2	$hash_{31}$	page31		$page_2$	$entry_2$
CAM_3	$hash_0$	$page_0$		$page_3$	$entry_3$
CAM_4	$hash_1$	$page_1$		$page_4$	$entry_4$
CAM_5	$hash_2$	$page_2$		$Page_5$	$entry_5$
CAM_6	$hash_3$	$page_3$		$page_6$	$entry_6$
CAM_7	$hash_4$	$page_4$		$page_7$	$entry_7$
……	$hash_6$ ……	$page_6$ ……		……	……
CAM_{29}				$page_{29}$	$entry_{29}$
CAM_{30}	$hash_{28}$	$page_{28}$		$page_{30}$	$entry_{43}$
CAM_{31}	$hash_{29}$	$page_{29}$		$page_{31}$	$entry_{31}$

除了通过高速缓存器子系统 1115 查找数据库 324 中的记录以获得一个现有的流－入口之外，LUE、SP 或 FIDE 引擎还可以通过高速缓存器更新流－入口。同样，在高速缓存器中也可以有是更新过的流－入口的记录。在这样更新过的记录被写入外部存储器中的流－入口数据库 324 之前，始终称它们为“脏”记录。通常要在高速缓存器系统中提供一种机制来指示高速缓存器中的脏记录。例如，一个脏记录在数据库 324 中的相应记录被更新之前不能被清洗出去。

假定在最后一个例子中，高速缓存器中的记录也由该操作进行了更改。也就是说，$hash_{43}$ 是在 MRU CAM_0 中，CAM_0 正确指向 $page_{30}$，但高速缓存器的 $page_{30}$ 中的信息，也就是 $entry_{43}$，并不对应数据库 324 中的 $entry_{43}$。也就是说，高速缓存器的 $page_{30}$ 的内容是脏的。现在需要更新数据库 324。这被称为备份或清除脏记录。

通常在高速缓存器系统中还用脏标记指示高速缓冲存储器记录是脏的。在优选实施方案中，对高速缓冲存储器中的每个字都有一个脏标记。

所发明的高速缓存器系统的另一个方面是根据最可能被清除出高速缓冲存储器的记录来清除高速缓冲存储器内容。在我们的 LRU 高速缓存器实施方案中，高速缓冲存储器记录的清除按照使用由远至近的顺序进行。这样，LRU 页首先被清除与它们可能被首先清洗的最小可能性一致。

在我们的实施方案中，存储器状态机在它空闲的任意时刻都可以被编程以按照从远到近的顺序，即从 CAM 阵列的底部开始，扫描 CAM 阵列并查找脏标记。只要发现了一个脏标记，就把高速缓冲存储器内容备份到外部存储器中的数据库 324 中。

注意一旦高速缓冲存储器的一个页被清除，为了防止仍然需要它而继续把它保留在高速缓存器中。仅在需要更多高速缓冲存储器页时才把该页冲洗出去。对应的 CAM 也直到需要一个新的高速缓冲存储器页时才被改变。采用这种方式，能够对包括干净记录在内的所有高速缓冲存储器内容进行有效查找。此外，无论何时要冲洗一个高速缓冲存储器记录，都要先进行检查确认该记录是干净记录。如果该记录是脏的，要在冲洗之前先对其进行备份。

高速缓存器子系统 1115 可以一次服务两个读传递。如果一次有两个以上活动的读请求，高速缓存器按照下面的特定顺序为它们服务：

(1) LRU 脏写回。如果最近最少使用高速缓冲存储器记录是脏的，高速缓存器就把它写回存储器中，因此将总有一个空间用于高速缓冲存储器未命中时的获取。

(2) 查找和更新引擎 1107。

(3) 状态处理器 1108。

(4) 流插入和删除引擎 1110。

(5) 分析器主机接口和控制 1118。

(6) 从 LRU－1 到 MRU 的脏写回；当没有其它未决因素时，高速缓存器引擎把脏记录写回到外部存储器。

图 26 显示了高速缓存器子系统 1115 的高速缓冲存储器部件 2600。高速缓冲存储器子系统 2600 包括高速缓冲存储器页的一组双端口存储器 2603。在我们的优选实施方案中有 32 页。每个存储器页都是双端口。也就是说，每个存储器页包括两组输入端口和两组输

出端口，每个输入端口都有地址和数据输入，一组输入和输出端口被连接到统一存储器控制(UMC)1119，用于从高速缓冲存储器和用于流－入口数据库 324 的外部存储器读出或向其写入数据。由多路复用器 2611 使用来自高速冲存器子系统 1115 中的 CAM 存储器子系统部分的高速缓存器页选择信号来选择把输出线 2609 中的哪一个连接到 UMC1119 上。从数据库 324 更新高速缓冲存储器使用来自 UMC 的高速缓存器数据信号 2617 和高速缓存器地址信号 2615。

来自查找/更新引擎(LUE)1107、状态处理器(SP)1108 或流插入/删除引擎(FIDE)1110 的对高速缓冲存储器的查找和数据更新使用高速缓冲存储器存储页 2603 的其它端口。一组输入选择多路复用器 2605 和一组输出选择多路复用器 2607 使用一组选择信号 2619 分别选择输入和输出引擎。

图 27 显示了连接到 CAM 阵列 2705 和存储器状态机 20703 的高速缓存器 CAM 状态机，以及在这些单元之间传递的一些信号。信号名都是自释义的，并且从上面的描述中也能够清楚怎样把这些控制器实现为状态机或其它形式。

尽管对 CAM 阵列操作的上述描述对本领域的技术人员要设计这样一个 CAM 阵列来说已经足够了，并且有多种可能的设计，图 28 显示了一种这样的设计。参考该图，CAM 阵列 2705 对 CAM 存储器的每个页都包括一个 CAM，例如 CAM[7](2807)。查找端口或更新端口取决于 LUE、SP 或 FIDE 中的哪一个正在访问高速缓存器子系统。查找的输入数据通常是散列，图中显示为 REF－DATA 2803。装入、更新或清出高速缓存器用信号 2805 来实现，信号 2805 用选择多路复用器 2809 来选择 CAM 输入数据，这样的数据是命中页或 LRU 页(按照本发明的一个方面就是底部 CAM)。通过 5－>32 解码器 2811 完成任何装入。对阵列中的所有 CAM 进行 CAM 查找的结果被提供给一个 32－>5 低到高的编码器 2813,2813 输出命中 2815 以及产生命中的 CAM 编号 2817。CAM 命中页 2819 是 MUX 2821 的输出，2821 拥有每个 CAM 的 CAM 数据作为输入以及由产生命中的 CAM 的信号 2817 选择的输出。对脏记录的维护从连接到 CAM 状态机 2701 的更新端口执行。MUX 2823 拥有所有 CAM 的数据输入和扫描输入 2827。MUX 2823 产生脏数据 2825。

模式解析和提取数据库格式

能够存在于不同层中的不同协议可以被看作一个或多个由结点链接成的树中的结点。分组类型是树的根(称为基层)。每个协议是下一层上的一些其它协议的父亲结点或终端结点。一个父亲结点把一个协议链接到可以在更高层上的其它协议(子协议)。这样一个协议就可以有零个或多个孩子。

作为树结构的一个实例，考虑以太网分组。它的孩子结点之一可能是 IP 协议，IP 协议的孩子结点之一可能是 TCP 协议。IP 的另一个孩子可能是 UDP 协议。

分组包括所用的每个协议的至少一个报头。分组中所用的特定协议的子协议由该特定协议的报头中一个位置上的内容来标识。分组的内容指定孩子是以子识别模式的形式。

网络分析器最好能够分析多种不同协议。在基层，有很多用在数字通信中的分组类型，包括以太网、HDLC、ISDN、Lap B、ATM、X.25、帧中继、数字数据服务、FDDI(光纤分布数据接口)和 T1 以及其它类型。这些分组类型中有很多使用不同的分组和/或帧格式。例如，数据在 ATM 和帧中继系统中以 53 个八位位组(即字节)的固定长度分组(称为“单元”)进行传输；需要用几个这样的单元来构造可以包括在一些其它类型的单个分组中的信息。

注意术语分组在这里意指包括分组、数据报、帧和单元。通常，分组格式或帧格式指怎样用多个不同的字段和报头封装数据以通过网络进行传输。例如，数据分组通常包括目的地地址字段、长度字段、纠错码(ECC)字段或循环冗余校验(CRC)字段以及标识分组开始和结尾的报头和报尾。术语“分组格式”、“帧格式”和“单元格式”一般是同义的。

分组监控器 300 能够分析不同的协议，并且这样就能够在分组上执行不同的协议特定操作，其中任意协议的协议报头根据父协议和分组中所用的协议被放在不同的位置。这样，分组监控器根据分组的内容能够适应不同的协议。并适应地为特定类型的分组确定从任意分组中要提取的位置和信息。例如，对寻找什么和在哪里寻找它们以形成签名没有固定的定义。在一些现有技术系统中，像 Chiu et al.在美国专利 5，101，402 中所描述的系统中，对特定类型的分组

有固定的位置。随着协议的增殖，指定所有可能的位置以指望确定会话变得越来越困难。同样，添加一个新协议或应用也很困难。在本发明中，对任意协议层数都是一个变量，并且是足够唯一标识到我们想要去的层及所有通往应用层的路径(在 OSI 模型中)的任何数字。

甚至同一协议也可以有不同的变体。例如，以太网分组有几种已知的变体，每一种都拥有基本保持相同的基本格式。以太网分组(根结点)可以是以太类型分组—也称作以太网类型/版本 2 和 DIX(DIGITAL－Intel－Xerox 分组)—或 IEEE 以太网(IEEE 803.x)分组。监控器应该能够处理所有类型的以太网协议。用以太网协议时指示子协议的内容在一个位置，但用 IEEE 类型时子协议被指定在一个不同的位置。子协议由子识别模式指示。

图 16 显示了完整以太网帧(即分组)的报头信息并包括在目的介质访问控制地址(Dst MAC 1602)和源介质访问控制地址(Src MAC 1604)上的信息。图 16 中还显示了在 PDL 文件中为提取签名而定义的一些(不是全部)信息。在解析结构和提取操作数据库 308 中也将定义这样的信息。这以 Dst MAC 信息 1606 的六个字节和 Src MAC 信息 1610 的六个字节形式包括了这一层上的所有报头信息。还分别定义了散列的源和目的地址部分。这些显示为来自 Dst MAC 地址的 2 字节 Dst 散列 1608 和来自 Src MAC 地址的 2 字节 Src 散列 1612。最后包括的信息(1614)是与下一层有关的信息的报头从哪里开始。在这个例子中下一层(第二层)信息在分组位移 12 开始。

图 17A 现在显示了以太类型分组 1700 的下一层(第 2 层)的报头信息。

对以太类型分组 1700 来说，来自分组指示下一层的有关信息是一个两字节类型的字段 1702，其中包含下一层的识别模式。剩下的信息 1704 在图中显示为阴影，因为它与这一层无关。列表 1712 显示了针对以太类型分组的可能的孩子，由在位移 12 发现的子识别模式指示。

还显示了用于解析器记录和定位下一个报头信息的一些提取出的部分。解析器记录的签名部分包括提取出的部分 1702。还包括来自这个信息的 1－字节散列部分。

位移字段 1710 提供转到下一层信息的位移，即定位下一个报头信息的开始。对以太类型分组来说，下一层报头的开始在从帧起始处开始 14 个字节处。

其它分组类型被进行了不同的排列。例如，在 ATM 系统中，每个 ATM 分组包括 5 个 8 位位组的“报头”段，后跟 48 个 8 位位组的“有效负载”段。ATM 单元的报头段关于有效负载段中包含的数据的路由的信息。报头段还包含流量控制信息。报头段的 8 或 12 位包含虚拟路径标识符(VPI)，并且报头段的 16 位包含虚拟通道标识符(VCI)。每个 ATM 交换把由 VPI 和 VCI 位表示的抽象路由信息转化为物理或逻辑网络链路的地址并正确地发送每个 ATM 单元。

图 17B 显示了可能的下一层之一的报头结构，即 IP 协议的报头。表 1752 中显示了 IP 协议可能的孩子。报头根据父协议在不同位置(L3)开始。图 17B 中还包括部分要为签名而提取的一些字段，和指示下一层的报头从分组中的什么地方开始的标记。

注意图 16、17A 和 17B 中显示的信息将以 PDL 文件的形式指定给监控器并编译进模式结构和提取操作数据库 308 中。

解析子系统根据存储在数据库 308 中的信息在分组报头数据上执行操作。因为与协议有关的数据可以被看作组织成树的形式，在解析子系统中需要把最初以树的形式组织的数据仔细查找一遍。既然实时操作是首选，就需要快速地执行这样的查找。

已知有效存储信息的数据结构被组织成树。这样的有效存储方式通常需要用算术计算来确定指向数据结点的指针。对本应用来说使用这样的有效存储数据结构进行查找可能消耗过多的时间。因此希望以某种能够快速查找的格式存储协议数据。

按照本发明的另一个方面，数据库 308 被存储在存储器中并包括用来存储将在分组上执行的协议特定操作的数据结构。尤其压缩表示用来在模式解析和提取数据库 308 中存储信息，数据库 308 由模式识别方法 304 和提取方法 306 在解析器子系统使用。用了通过使用一组一个或多个索引来检索数据结构的内容从而快速定位子协议有关的信息，对该数据结构进行组织。一个数据结构记录包括一个有效性指示。定位并识别子协议包括检索该数据结构直到找到有效记录为止。使用该数据结构存储由模式识别引擎(PRE)1006 所用的协

议信息能够使解析器子系统 301 执行快速查找。

在一种实施方案中，该数据结构是三维结构的形式。注意这个三维结构通常被作为一组二维结构依次存储在存储器中，由此 3－D 结构的三维中的一维被用作特定 2－D 数组的索引。这形成了该数据结构的第一索引。

图 18A 显示了这样一个 3－D 表示 1800(它可被看作 2－D 表示的编了索引的集合)。这个数据结构的三维是:

1. 类型标识符[1:M]。这是标识特定层上的协议类型的标识符。例如，01 指示以太网帧。64 指示 IP，16 指示 IEEE 类型以太网分组，等等。根据分组解析器能够处理多少个协议，M 可能是一个大数；随着向监控器 300 添加处理更多协议的能力 M 可以随着时间而增长。当该 3－D 结构被看作一组 2－D 结构时，类型 ID 就是特定 2－D 结构的索引。
2. 大小[1:64]。分组中感兴趣的字段的大小。
3. 位置[1:512]。这是分组中的偏移位置，表示为多个 8 位位组(字节)。

在这些位置中的任意一个上面可能有或者没有有效数据。通常，在大多数位置上都将没有有效数据。3－D 数据的大小是 M*64*512(译者注：*表示乘)，可以很大；例如 M 是 10，000。这是一个稀疏矩阵，大多数记录都是空的(即无效)。

每个数组记录包括一个“结点代码”，指示内容的种类。这个结点代码可以是三个值中的其中一个：(1)向模式识别方法 304 指示已知协议已经被识别为下一个协议(即子协议)的“协议”结点代码；(2)指向对目前正在查找的协议没有子协议的“终端”结点代码；(3)指示没有有效记录的“空”(也称为“冲洗”)结点代码。

在优选实施方案中，通过初始化把可能的孩子和其它信息装入到该数据结构中，初始化包括基于 PDL 文件 336 和层次选择 338 的编译方法。表示协议的数据结构中的任意记录包含下列信息。

(a) 用于查找下一个的一列孩子(作为类型 ID)。例如，对以太网类型 2 来说，孩子是以太类型(IP、IPX 等等，在图 17 中显示为 1712)。这些孩子被编译进类型代码。IP 的代码是 64，IPX 的是 83，等等。

(b) 对列表中的每个 ID，需要比较一列子识别模式。例如，列表中的 64:0800_{16}指示要查找的值是 0800(十六进制)，对孩子来说是类型 ID 64(它是 IP 协议)。列表中的 83:8137_{16}指示要查找的值是 8137(十六进制)，对孩子来说是类型 ID 83(它是 IPX 协议)，等等。

(c) 为构造该流的标识签名而执行的提取操作。所用的格式是(位移、长度、流_签名_值_标识符)，流_签名_值_标识符指示提取出的记录进入到签名中的什么地方，包括需要执行的操作(与、或等等)。如果还有一个散列键部分，那么也包含它上面的信息。例如，对以太类型的分组，在签名中使用 2-字节类型(图 17 中的 1706)。此外，包括类型的 1-字节的散列。此外注意子协议在位移 14 开始。

附加条目可以是“折叠”。折叠用来减少 3-D 结构的存储需求。既然每个协议 ID 的 2-D 数组可以被稀疏填充，只要单个记录不相互冲突就可以把多个数组结合成单一的 2-D 数组。然后用折叠号把关联每个元素。对给定的查找来说，查找的折叠号必须匹配折叠号记录。下面将更详细地描述折叠。

就以太网来说，下一协议字段可以指示长度，它告诉解析器这是 IEEE 类型的分组，并且下一协议是在别处。通常，下一协议字段包含指示下一个协议(即子协议)的值。

用于解析器子系统的记录点被称为虚拟基层，并且包含可能的第一个孩子，即分组类型。这里包括一个用高级协议描述语言(PDL)书写的一个协议集实例。该集合包括 PDL 语言和描述所有可能的记录点(即虚拟基层)的文件 virtual.pdl。这个文件中只有一个孩子 01 指示以太网。这样，该特定实例只能处理以太网分组。在实践中可以有多个记录点。

在一种实施方案中，分组拦截设备为拦截到的每个分组提供一个报头并向监控器 300 提供指示分组类型的输入。这个报头用来确定虚拟基层记录点给解析器子系统。这样，即使在基层解析器子系统也能够识别分组类型。

最初，查找从在由拦截设备提供的报头中获得的虚拟基层的孩子开始。就这个例子来说，它有 ID 值 01，它是用于以太网分组的整个

3－D结构中的2－D数组。

这样，实现模式分析方法304(例如，图10中的模式识别引擎(PRE)1006)的硬件进行查找以确定有协议ID 01的2－D数组的孩子(如果有的话)。在优选实施方案中采用3－D数据结构，硬件PRE 1006最多可以同时查找四个长度(即大小)。这样，方法304以四个长度为一组进行查找。从协议ID 01开始，查找的前两组3－D位置是:

(1，1，1)　　(1，1，2)

(1，2，1)　　(1，2，2)

(1，3，1)　　(1，3，2)

(1，4，1)　　(1，4，2)

在查找的每一步，分析方法304检查分组和3－D结构以查看是否有匹配(通过查看结点代码)。如果用代码结点没有发现有效数据，就增加大小(最大可以到4)并随后增加位移。

继续这个例子，假定模式分析方法304在1、2、12找到了一些东西。这时，我们意谓方法304已经在分组位移12发现了协议ID值01(以太网)的数据，在该分组中有两字节(8位位组)长的信息与下一(子)协议有关。例如，该信息可以是表示为子识别模式的这个协议的孩子。从该数据结构获得可以获得分组的可能的子识别模式列表部分。

以太网分组结构有两种，以太类型的分组和较新的IEEE类型，并且对它们两个来说指示孩子的分组位置也不相同。以太类型分组中的该位置指示IEEE类型的孩子“长度”，因此对以太网分组要确定“下一协议”位置包含值还是长度(这被称为“LENGTH”操作)。成功的LENGTH操作由小于或等于$05DC_{16}$的内容来指示，那么这是IEEE类型的以太网帧。在这种情况下，要在别处查找子识别模式。否则，该位置包含指示孩子的值。

注意虽然记录的这个能力是值(例如，子协议ID)或长度(指示进一步分析以确定子协议)只用于以太网分组，但将来其它分组可能以正在被修改而结束。因此，这个能力在PDL文件中以宏的形式仍然能够使将来的分组被解码。

继续这个例子，假定LENGTH操作失败。那样我们就有了一个以太类型的分组，并且下一协议字段(包含子识别模式)是从位移12开

始的两个字节，在 17A 中显示为分组字段 1702。这将是图 17A 的表 1712 中所示的以太类型的一个孩子。PRE 使用数据结构中的信息检查用于找到的 2－字节子识别模式的是什么 ID 代码。例如，如果子识别模式是 0800(Hex)，那么该协议是 IP。如果子识别模式是 0BAD(Hex)该协议是 VIP(VINES)。

注意替代实施方案可以保持一个单独的表格，其中包括所有的子识别模式和它们的对应协议 ID。

继续这个例子，假定位于 1、2、12 的子识别模式是 0800_{16}，指示 IP 协议。IP 协议的 ID 代码是 64_{10}。继续以太类型的实例，一旦解析器匹配了该协议—在这个例子中，该协议类型是 IP，ID 是 64_{10}—的可能的孩子中的其中一个，那么解析器就继续查找下一层。ID 是 64，长度未知，位移已知大于等于 14 个字节(12 字节的位移用于类型，加上类型的长度 2 个字节)，因此对 3－D 结构的查找从位于分组位移 14 的位置(64，1)开始。在位于分组位移 14 的位置(64，2)上发现一个填充的结点。报头细节在图 17B 中显示为 1750。可能的孩子显示在表 1752 中。

另外，假定在(1，2，12)上有一个长度 1211_{10}。指示这是一个 IEEE 类型的以太网帧，它在别处存储它的类型。PRE 现在继续在同一层上查找，但是是针对一个新 ID 进行，即 IEEE 类型以太网帧的 ID。IEEE 类型分组的协议 ID 是 16，因此 PRE 从分组位移 14 开始用 ID 16 继续查找 3－D 空间。

在我们的例子中假定在位于分组位移 14 的(16,2)上发现了一个“协议”结点代码，并且下一个协议由子识别模式 0800_{16} 指定。这指示孩子是 IP 协议，它的类型 ID 是 64。这样查找从位于分组位移 16 的(16，1)继续进行。

压缩

像上面所指出的那样，3－D 结构非常大，而且是稀疏填充的。例如，如果在每个位置上存储 32 个字节，那么长度是 M*64*512*32，也就是 M 兆字节。如果 M=10，000，那就是约 10 千兆字节。在解析器子系统中包括 10GB 的存储器用于存储数据库 308 有些不切实际。因此在优选实施方案中使用压缩形式存储数据。压缩最好由编译方法中的优化器单元来执行。

回想一下该数据结构是稀疏的。不同的实施方案可以采用不同的压缩方案，它们均利用该数据结构的稀疏特性。一种实施方案使用多维行程编码的改进形式。

另一种实施方案使用少一些的 2－D 结构来存储信息，而它们又处于一个大的 3－D 结构之中。在优选实施方案中使用第二种方案。

图 18A 描绘了把 3－D 数组怎样被看作是一组 2－D 数组，对每个协议(即协议 ID 的每个值)有一个 2－D 数组。对多达 M 个协议 ID 来说 2－D 结构显示为 1802－1、1802－2、…、1802－M。一个表格记录显示为 1804。注意表格中的空隙是用来描绘每个 2－D 结构表通常也很大。

考虑表示可能的协议的树的集合。每个结点表示一个协议，一个协议可以有孩子或者是一个终端结点。树的基(根)把所有的协议类型都作为孩子。其它结点形成在树中不同层上的结点，从第 1 层到树的终端结点。基结点中的一个元素可以用结点 ID 1 作参考符号，底部结点中的另一个元素可以用结点 ID 2 用参考符号等等。因为树是从根开始遍历的，在树中可能有一些同一结点下一次被引用的点。例如，当像 Telnet 这样的应用协议能够同时运行在几个像 TCP 或 UDP 这样的传输连接上时将会发生这种情况。与其重复 Telnet 结点，不如在模式数据库 308 中只表示一个可以有几个父亲的结点。这就减少了可观的空间占用。

图 18A 中的每个 2－D 数组中表示一个协议。为了能够通过对每个有几个父亲的协议只使用一个数组以节省空间，在一种实施方案中，模式分析子方法保持“当前报头”指针。3－D 结构中的每个协议 2－D 数组的每个位置(位移)索引是从该特定协议的报头起始处开始的相对位置。

2－D 数组中的每一个都是稀疏的。优化的下一步是核对每个 2－D 数组和所有其它 2－D 数组以找出能够节省存储器的 2－D 数组。这些 2－D 数组中有很多通常都是被稀疏填充的，每个当中只有少量有效记录。因此，接下来用一个“折叠”方法把两个或更多 2－D 数组结合成一个物理 2－D 数组，并且不丢失任意一个原始 2－D 数组的同一性(即，所有的 2－D 数组逻辑上连续存在)。折叠可以在任意 2－D 数组之间发生，而不管它们在树中的位置，只要满足一定条件

即可。

假定正在考虑对两个 2－D 数组进行折叠。命名第一个 2－D 数组为 A，第二个为 B。既然这两个 2－D 数组都是被部分填充的，当且仅当这两个 2－D 数组中的相同位置上没有相互冲突的单个元素时可以把 2－D 数组 B 与 2－D 数组 A 结合在一起。如果结果是可折叠，那么 2－D 数组 B 中的有效记录和 2－D 数组 A 中的有效记录结合在一起产生一个物理 2－D 数组。但是，有必要能够区别原始 2－D 数组 A 的记录和 2－D 数组 B 中的记录。例如，如果由 2－D 数组 B 表示的协议的父协议想要引用 2－D 数组 B 的协议 ID，它现在必须改为引用 2－D 数组 A。但是，只有那些原始 2－D 数组 B 中的记录才是要查找的有效记录。为了实现这一点，任意给定 2－D 数组中的每个元素都被标上一个折叠号。当创建原始树时，用零折叠值初始化所有 2－D 数组中的所有元素。随后，如果 2－D 数组 B 被折叠进 2－D 数组 A 中，2－D 数组 B 中的所有有效元素都被复制到 2－D 数组 A 中的相应位置并且给它们与 2－D 数组 A 中的任意元素都不同的折叠号。例如，如果 2－D 数组 A 和 2－D 数组 B 都是树中的原始数组(即以前未折叠过)，那么在折叠后，所有的 2－D 数组 A 的记录将仍然保持折叠号 0，而 2－D 数组 B 的记录现在都有了一个新的折叠值 1。在 2－D 数组 B 被折叠进 2－D 数组 A 之后，需要通知 2－D 数组 B 的父亲在它们的孩子的 2－D 数组物理位置中的更改和在预期的折叠值中的连带更改。

这个折叠方法也可发生在两个已经被折叠过的 2－D 数组之间，只要这两个 2－D 数组中在相同 2－D 数组位置上没有冲突的单个元素。如前所述，2－D 数组 B 中的每个有效元素必须有一个分配给它们的唯一区别于 2－D 数组 A 的那些折叠号的折叠号。在 2－D 数组 B 被合并到 2－D 数组 A 中的时候通过给所有的 2－D 数组 B 折叠号加上一个固定值可以实现这一点。这个固定值是一个比原始 2－D 数组 A 中最大的折叠值还要大的值。需要指出的一点是，任意给定 2－D 数组的折叠号只与那个 2－D 数组有关而且并未跨越 2－D 数组的整个树。

现在可以在两个 2－D 数组的所有组合之间尝试这个折叠方法，直到没有可以折叠的 2－D 数组为止。这样就可以大大减少 2－D 数

组的总数。

无论何时当一个折叠发生时，都要对 3－D 结构(即所有 2－D 数组)进行查找以找到正在被折叠到另一个数组中的 2－D 数组的父亲。以前被映射到标识一个单独 2－D 数组的协议 ID 上的匹配模式现在必须替换成该 2－D 数组 ID 和下一个折叠号(即预期的折叠)。

这样，在压缩后的数据结构中，每个有效记录包括用于该记录的折叠号还有用于孩子的预期的折叠号。

图 18B 中描绘了用在数据库 308 中的数据结构的一种替代实施方案。这样，像上面所描述的 3－D 结构那样，它允许模式识别方法 304 通过在存储器中检索位置而不是执行地址链接计算来执行快速查找。该结构，像图 18A 中的那样，适于用硬件实现，例如对与图 10 中的模式识别引擎一起工作的实现来说就是这样。

表 1850 被称为协议表，对监控器 300 已知的每个协议它都有一个记录，并且包括每个协议的一些特征，包括对在报头中的什么地方可以找到指定下一个协议(子识别模式)的字段的描述、下一协议字段的长度、指示报头长度和类型的标记以及一个或多个限选器命令，限选命令可以为这一层上的这个协议的分组构造键部分和散列部分。

对任意协议还有一个或多个查找表(LUTs)。这样这个实施方案的数据库 308 还包括一组 LUTs 1870。每个 LUT 拥有 256 个记录，由从分组中的下一协议字段提取出的子识别模式中的一个字节进行索引。这样的协议说明可以是几个字节长，因此对任意协议都需要查找 LUTs 1870 中的几个字节。

每个 LUT 记录都包括一个指示内容的种类的 2－位“结点代码”，包括它的有效性。这个结点代码可以是下列四个值之一：(1)“协议”结点代码，向模式识别引擎 1006 指示一个已知协议已经被识别；(2)“中间”结点代码，指示一个多字节协议代码已经被部分识别，这样就允许在此之前把一组 LUTs 链接在一起；(3)“终端”结点代码，指示目前更在被查找的协议没有孩子，即该结点是协议树中的一个最终结点；(4)“空”(也称为“冲洗”或“无效”)结点代码，指示没有有效记录。

除了结点代码之外，每个 LUT 记录还包括下一个 LUT 编号、下一

个协议编号(用于查找协议表 1850)、LUT 记录的折叠和预期的下一个折叠。像实现 3-D 表示的压缩形式的实施方案中那样，折叠用来减少 LUTs 集合的存储需求。既然 LUTs 1870 可以被稀疏填充，只要单个记录之间不相互冲突就可以把多个 LUTs 结合成一个单独的 LUT。随后用折叠号关联每个元素和它的原始 LUT。

对一个给定的查找来说，查找的折叠号必须匹配查找表中的折叠号。预期的折叠从前一个查找表(“预期的下一个折叠”字段)中获得。本实现用 5-位来描述折叠，因此允许把最多 32 个表折叠成一个表。

使用图 18B 中的数据结构时，当分组到达解析器时虚基层已经被预先确定或者是已知的。虚基层记录告诉分组识别引擎在哪里找到分组中的第一个子识别模式。模式识别引擎随后从分组提取出第一个子识别模式字节并用它们作为进入虚基层表(第一个 LUT)的地址。如果用这个方法在指定的下一个 LUT 中查找到的记录与在虚基层记录中所指定的预期的下一个折叠值相匹配，那么就认为查找有效。随后检查结点代码。如果它是中间结点就把从 LUT 查找获得的下一个表字段被用作地址的最高有效位。下一个预期的折叠也从记录中提取。模式识别引擎 1006 随后把来自模式识别引擎的下一个字节用于下一个 LUT 查找。

这样，PRE 的操作继续进行直至找到一个终端结点。在协议表 1850 中查找下一个(最初时是基层)协议以向 PRE1006 提供有关用分组(在解析器子系统 1000 的输入缓冲区存储器 1008 中)中的什么字段来获得下一个协议的子识别模式的信息，包括该字段的大小。子识别模式字节从输入缓冲区存储器 1008 中取出。构成子识别模式的字节数现在也是已知的。

协议代码字节的第一个字节被用作下一个 LUT 中的查找。如果 LUT 查找产生一个指示协议结点或终端结点的结点代码，就设置下一个 LUT 和下一个预期的折叠，并且来自 LUT 查找的“下一协议”也被用作进入协议表 1850 的索引。这就给限选器 1007 提供了指令以及在分组中的什么地方获得用于下一协议的字段。这样，PRE 1006 继续执行直到它完成了对所有字段(即协议)的处理为止，由到达一个终端结点指示完成。

注意当检查一个子识别模式和一个表时总有一个预期的折叠。如果该预期折叠匹配表中的折叠信息，就用它来决定下一步做什么。如果折叠不匹配，优化器操作就结束了。

还要注意替代实施方案可以使用不同大小的 LUTs，然后由不同数量的子识别模式检索 LUT。

这个实施方案的当前实现允许最高达四个字节的子识别模式。超过四个字节的子识别模式被看作特殊情况。

在优选实施方案中，数据库由编译方法产生。编译方法先构造协议之间所有链接的一个单一协议表。链接由父协议和子协议之间的连接组成。每个协议可以有零个或多个孩子。如果一个协议有孩子，就创建一个由父协议、子协议、子识别模式和子识别模式大小组成的链接。编译器首先提取出子识别模式，它超过两个字节长。既然只有这些信息中的少数，就对它们分开处理。为子识别模式大小是两个字节的每个链接创建下一个子链接。

所有的链接随后被编成 256 个记录的 LUTs。

然后执行优化。优化的第一步是核对每个表和所有其它表以找到共享一个表的那些表。这个方法按照和上述 2－D 数组相同的方式进行，但现在是为了稀疏查找表。

初始化方法(例如，编译器方法 310)的一部分用包括指令、源地址、目的地址和长度的数据条目装载限选器指令数据库。当 PRE 1006 发送限选器指令时它把这个指令作为进入限选器指令数据库的位移来发送。指令或操作代码告诉限选器从进入的分组中提取什么并把它放到流签名中的什么位置上。向流签名的特定字段写入信息会自动产生散列。该指令还能够告诉限选器怎样确定特定协议的连接状态。

注意除了编译 PDL 文件之外替代实施方案也可以产生模式、解析和提取数据库。

编译方法

现在更详细地描述编译方法 310。方法 310 包括创建解析模式和提取数据库 308 和状态处理指令数据库 326，308 向解析子系统 301 提供需要用来解析分组并提取标识信息的信息，326 提供需要在状态处理操作 328 中执行的状态方法。

编译器的输入包括描述每个可以出现的协议的一组文件。这些文件是方便的协议描述语言(PDL)，这是一个高级语言。PDL 用于规定新协议和新层，包括新应用。PDL 独立于计算机网络中可用的分组和协议的不同类型。一组 PDL 文件用来描述什么信息与分组有关以及什么分组需要进行解码。PDL 还用来规定状态分析操作。这样，解析器子系统和分析器子系统能够适应并适合多种不同类型的报头、层和成分，并需要被提取或估计，例如为了构造唯一签名。

对每个分组类型和每个协议有一个文件。这样，对以太网分组有一个 PDL 文件，对帧中继分组也有一个 PDL 文件。PDL 文件被编译形成一个或多个数据库，它们能够使监控器 300 在分组上执行不同的协议特定操作，其中任意协议的协议报头根据父协议或分组中所用的协议而被放在不同的位置上。这样，分组监控器就能够根据分组内容适应不同的协议。尤其解析器子系统 301 能够针对不同类型的分组提取不同类型的数据。例如，监控器能够知道怎样解释以太网分组，包括对报头信息进行解码，并且还知道怎样解释帧中继分组，包括对报头信息进行解码。

例如，PDL 文件的集合可以包括普通以太网分组文件。对每个变体以太网分组还包括一个 PDL 文件，例如 IEEE 以太网文件。

协议的 PDL 文件提供编译方法 310 需要用来产生数据库 308 的信息。该数据库依次告诉解析器子系统怎样解析和/提取信息，包括要为流签名提取的一个或多个协议－特定成分，怎样使用这此些成分构造流签名，在分组中的什么地方寻找这些成分，在哪里寻找任意子协议以及要寻找什么子识别模式。对一些协议来说，提取出的成分可能包括源和目的地址，并且 PDL 文件可以包括使用这些地址构造键的顺序。例如，以太网帧有在构造更好的流签名中有用的终点地址。这样以太网分组的 PDL 文件包括关于解析器子系统将怎样提取源和目的地址上的信息，其中包括那些地址在什么位置以及它们的大小。例如，在帧中继基层中没有特定终点地址帮助更好地识别流，因此对那些类型的分组来说 PDL 文件不包括将导致解析器子系统提取终点地址的信息。

一些协议还包括有关连接的信息。TCP 就是这样的协议的一个实例。这样的协议使用在每个分组中都存在的连接标识符。这样一个

协议的 PDL 文件包括关于那些连接标识符是什么、它们在哪里以及它们的长度是多少的信息。在 TCP 实例中，例如运行在 IP 上，这些信息就是端口号。PDL 文件还包括关于是否有应用到连接和断开的状态以及可能的孩子是什么状态的信息。所以，在这些层中的每一层上，分组监控器 300 认识到更多关于该分组的信息。分组监控器 300 使用连接标识符能够识别是特定流的一部分的特定分组。一旦流被识别出来，系统就能够确定当前状态和应用什么状态处理存在于下一层中的连接和断开直到这些特定分组。

对优选实施方案中所用的特定 PDL 来说，一个 PDL 文件可以包括零个或多个 FIELD 语句，每个 FIELD 语句定义分组中一个特定的位或字节串(即一个字段)。一个 PDL 文件还可以包括零个或多个 GROUP 语句，每个 GROUP 语句用来把几个定义过的字段绑在一起。一组这样绑在一起的字段被称为一个组。一个 PDL 文件还可以包括零个或多个 PROTOCOL 语句，每个 PROTOCOL 语句定义在协议的报头中字段和组的顺序。一个 PDL 文件还可以包括零个或多个 FLOW 语句，每个 FLOW 语句通过描述地址、协议类型和端口号在分组中的位置来定义一个流。FLOW 语句包括对怎样用状态操作确定这个协议的子流的描述。相关的状态可以有用于管理和维护随着对流中的更多分组进行分析而认识到的新状态。

图 24 显示了一组 PDL 文件，针对在 IP 之上运行 TCP 的以太网分组的分层结构。Common.pdl(2403)是包含通用协议定义—即不同网络协议中的通用字段的一些字段定义—的文件。Flows.pdl(2405)是包含通用流定义的文件。Virtual.pdl(2407)是包含所用虚基层的定义的 PDL 文件。Ethernet.pdl(2411)是包含以太网分组的定义的文件。这里描述了在以太类型对 IEEE 类型以太网文件上的决定。如果这是以太类型，就从文件 Ethertype.pdl(2413)作出选择。在一种替代实施方案方案中，以太类型选择定义可以在同一个以太网文件 2411 中。在一个典型的实现中，可以包括针对其它以太网类型的 PDL 文件。IP.pdl(2415)是包含 Internet 协议的分组定义的 PDL 文件。TCP.pdl(2417)是包含传输控制协议—在这种情况下就是 IP 协议的传输服务—的分组定义的 PDL 文件。除了提取协议信息之外 TCP 协议定义文件还在连接识别方法中帮助处理状态。在一个典型的文件

集合中，还要有用户数据报协议(UDP)定义的文件 UDP.pdl。RPC.pdl(2419)是包含远程过程调用的分组定义的 PDL 文件。

NFS.pdl(2421)是包含网络文件系统的分组定义的 PDL 文件。对于监控器 300 可能遇到的所有协议通常也要包括其它 PDL 文件。

编译处理 310 的输入是所有感兴趣的协议的 PDL 文件(例如，图 24 中的文件)的集合。方法 310 的输入还可以包括图 3 中显示为数据报层次选择 338 的分层信息。层次选择信息描述了协议的分层情况—在任意特定协议之上运行的是什么协议。例如，IP 可以运行在以太网和很多其它类型的分组上。TCP 可以运行在 IP 之上。UDP 也可以运行在 IP 之上。当没有明确包括分层信息时，分层信息是内在的；PDL 文件包括子协议，这就提供了分层信息。

图 25 中描绘了编译处理 310。编译器把 PDL 源文件装入高速暂存存储器中(2503 步)并检查文件的语法(解析步骤 2505)。一旦完成，编译器就创建一个包含所有解析元素的中间文件(2507 步)。中间文件的格式被称为“编译过的协议语言”(CPL)。CPL 指令有固定的分层格式，并包括每个层及一层的整个树所要求的所有模式、提取和状态。CPL 文件包括协议的数量及协议定义。每个协议的协议定义可以包括一个或多个协议名、协议 ID、报头节、组定义节、任意特定层的节、通告节、有效负载节、子节和状态节。随后由优化器运行该 CPL 文件以创建将由监控器 300 使用的最终数据库。那些本领域的技术人员要清楚编译处理 310 的替代方案可以包括不同形式的中间输出，或者根本没有中间输出直接产生最终数据库。

在创建解析元素之后，编译器创建流签名元素(2509 步)。这在 CPL 中创建了在构造流签名(散列键)和在层间建立链接的每个 PDL 模块的每一层上必需的提取操作。

随着流签名操作完成，PDL 编译器创建(2511 步)从每个 PDL 模块提取有效负载元素所必需的操作。这些有效负载元素由更高层上的其它 PDL 文件中的状态用在处理当中。

最后一步是创建每个 PDL 模块所需要的状态操作。从 PDL 文件编译并以 CPL 形式创建状态操作以为后用(2513 步)。

CPL 文件现在通过优化器运行，优化器产生由监控器 300 使用的最终数据库。

协议定义语言(PDL)参考指南(A0.02版)

这里包含的是协议定义语言(PDL)的参考指南(“指南”)，在本发明的一个方面中 PDL 允许自动生成由解析器和分析器子-系统所使用的数据库，并且还允许监控器有包括新的以及修改过的协议和应用的能力。

版权公告

本文档包含专利在内的部分内容中包含有与版权保护有关的材料。版权所有者(Apptitude 有限公司，圣约瑟，加利福尼亚州，原 Technically Elite 有限公司)不反对由本专利文档或本专利公开内容或这个文档的任何人复制副本，因为它已经出现在专利与商标事务处的文档和记录中，但另外保留所有的版权。

1. 导论

所发明的协议定义语言(PDL)是用来描述网络协议及协议报头中的所有字段的专用语言。

在本指南中，当没有和其它类型的描述发生混淆的危险时，协议描述(PDL 文件)被看作是 PDL 或规则。

PDL 使用和 C 编程语言与 PERL 脚本语言的数据结构定义部分相似的形式和组织方式。因为 PDL 来自一种用来对网络分组接触进行解码的语言，作者已经把该语言形式和分组解码的要求混合在了一起。这产生了一种非常相似并适用于描述分组内容及表示一个流所需的细节的表示语言。

1.1 概述

PDL 是非过程化的第四代语言(4GL)。这意味着描述需要做什么而不描述怎样去做。怎样做的细节隐藏在编译器和编译后的协议规划(CPL)优化程序中。

另外，它还用来通过定义哪个字段是地址字段、哪个是协议类型字段等来描述网络流。

一旦写好了一个 PDL 文件，用 Netscope 编译器(nsc)对其进行编译，产生 MeterFlow 数据库(MeterFlow.db)和 Netscope 数据库(Netscope.db)。MeterFlow 数据库包含流定义，Netscope 数据库包

含协议报头定义。

这些数据库由程序使用，像：mfkeys，产生流键(也称为流签名)；mfcpl，以 CPL 形式产生流定义；mfpkts，产生所有已知协议的实例分组；和 netscope，产生 sniffer™和 tcpdump 文件。

1.2 指南惯例

本指南自始至终都将使用下列惯例：

小写 courier 字体指示函数名的 C 代码实例。书写函数时要在函数名后跟括号[function()]，对变量只写名字[variables]，结构名前面要加“struct”[struct packet]。

斜体指示文件名(例如，mworks/base/h/base.h)。通常相对于分布的根目录书写文件名。

常量用十进制表示，除非写成“0x...”这样十六进制数的 C 语言符号。

注意在 PDL 文件中两个连字符(- -)之后的任意行上的任何内容都被编译器忽略掉。也就是说，它们是注释。

2. 程序结构

一个 MeterFlow 解码与流集合是非空的语句序列。在 MeterFlow PDL 中有四种基本类型的语句或定义可用。

FIELD，

GROUP，

PROTOCOL 和

FLOW。

2.1 FIELD 定义

FIELD 定义用来定义分组中特定的位或字节串。FIELD 定义格式如下：

```
Name FIELD
SYNTAX Type [{Enums}]
DISPLAY-HINT “FormatString”
LENGTH  “Expression”
FLAGS FieldName [, FieldName2]
LOOKUP LookupType [Filename]
ENCODING EncodingType
```

DEFAULT “value”

DESCRIPTION “Description”

只有 FIELD 和 SYNATX 行是必需的。所有其它行都是属性行，定义关于 FIELD 的特殊特征。属性行是可选的并且可以任意顺序出现。下面详细描述了每一个属性行:

2.1.1 SYNTAX Type [{Enums}]

这个属性定义类型，并且如果类型是 INT、BYTESTRING、BITSTRING 或 SNMPSEQUENCE 类型的话定义 FIELD 的枚举类型。当前定义的类型是:

INT(numBits)	numBits 位长的整数
UNSIGNED INT(numBits)	numBits 位长的无符号整数
BYTESTRING(numBytes)	numBytes 字节长的串
BYTESTRING(R1..R2)	大小范围在 R1 到 R2 字节之间的串
BITSTRING(numBits)	numBits 位长的串
LSTRING(lenBytes)	带 lenBytes 报头的字符串
NSTRING	以零结尾的字符串
DBSSTRING	DNS 编码后的字符串
SUMPOID	SNMP 对象标识符
SNMPSEQUENCE	SNMP 序列
SNMPTIMETICKS	SNMP 时间标记
COMBO field1 field2	结合假字段

2.1.2 DISPLAY-HINT “FormatString”

这个属性用于指定怎样打印 FIELD 的值。当前支持的格式是:

Numx	作为 num 字节的十六进制数打印
Numd	作为 num 字节的十进制数打印
Numo	作为 num 字节的八进制数打印
Numb	作为 num 字节的二进制数打印
Numa	以 ASCII 格式打印 num 字节
Text	作为 ASCII 文本打印

HexDump	以十六进制倾印码形式打印

2.1.3 LENGTH "Expression"

这个属性为确定 FIELD 的长度定义一个表达式。表达式是算术表达式，并可以通过在引用的字段的名字前面加$而引用分组中其它 FIELD 的值。例如，如果 tcpHeaderLen 是为当前分组而定义的另一个字段，"($tcpHeaderLen*4) - 20" 就是一个有效表达式。

2.1.4 FLAGS FieldFlags

这个属性定义 FIELD 的几个特殊标记。当前支持的字段标记是:

SAMELAYER	显示同一层上的字段为前一个字段
NOLABEL	不和值一起显示字段名
NOSHOW	对字段解码但不显示它
SWAPPED	整数值是交换的

2.1.5 ENCAP FieldName [, FieldName2]

这个属性定义一个分组怎样被封装在另一个分组中。由 FieldName 字段的值确定要封装到的分组。如果用 FieldName 没有找到分组就尝试用 FieldName2。

2.1.6 LOOKUP LookupType [Filename]

这个属性定义怎样查找特定 FIELD 值的名字。当前支持的 LookupType 是:

SERVICE	用 getservbyport()
HOSTNAME	用 gethostbyaddr()
MACADDRESS	用$METERFLOW/conf/mac2ip.cf
FILE file	用 file 查找值

2.1.7 ENCODING EncodingType

这个属性定义怎样对 FIELD 进行编码。目前仅支持 BER 编码类型(针对由 ASN.1 定义的基本编码规则)。

2.1.8 DEFAULT "value"

这个属性定义当产生这个协议的实例分组时用于这个字段的缺省值。

2.1.9 DESCRIPTION “Description”

这个属性定义对 FILED 的说明。仅用于提供信息目的。

2.2 GROUP 定义

GROUP 定义用来把几个相关 FIELD 绑在一起。GROUP 定义格式如下:

```
Name GROUP
LENGTH “Expression”
OPTIONAL “Condition”
SUMMARIZE “Condition” : “FormatString” [
“Condition” : “FormatString”...]
DESCRIPTION “Description
::={Name=FieldOrGroup [,
Name=FieldOrGroup...}”
```

只有 GROUP 行和::=行是必需的。所有其它行都是属性行，定义 GROUP 的特殊特性。属性行是可选的并且可以任意顺序出现。下面详细描述每个属性行:

2.2.1 LENGTH “Expression”

这个属性为确定 GROUP 的长度定义一个表达式。表达式是算术表达式，并可以通过在引用的字段名字前面加$而引用分组中其它 FIELD 的值。例如，如果 tcpHeaderLen 是为当前分组而定义的另一个分组，“($tcpHeaderLen*4) - 20” 就是一个有效表达式。

2.2.2 OPTIONAL “Condition”

这个属性为确定 GROUP 是否存在定义一个条件。有效条件定义在下面的 Condition 节中。

2.2.3 SUMMARIZE “Condition” : “FormatString” [“Condition” : “FormatString”...]

这个属性定义怎样以详细模式显示一个 GROUP。可以为每个条件(Condition)指定一个不同的格式(FormatString)。有效条件定义在下面的 Condition 节中。通过在 FIELD 的名字前面加上$可以在 FormatString 中引用任意 FIELD 的值。除了 FIELD 名之外还有其它

几种特殊的$关键字。

$LAYER	显示当前协议层
$GROUP	把整个 GROUP 显示成一张表
$LABEL	显示 GROUP 标签
$field	显示 field 值(如果可用的话使用枚举名)
$:field	显示 field 值(以原始格式)

2.2.4 DESCRIPTION “Description”

这个属性定义对 GROUP 的说明。仅用于提供信息目的。

2.2.5 ::={Name=FieldOrGroup[, Name=FieldOrGroup…]}

这个属性定义 GROUP 中字段或子群的顺序。

2.3 PROTOCOL 定义

PROTOCOL 定义用来定义协议报头中 FIELD 和 GROUP 的顺序。PROTOCOL 定义格式如下:

```
Name PROTOCOL
SUMMARIZE “Condition” : “FormatString” [
“Condition” : “FormatString”…]
DESCRIPTION “Description”
REFERENCE “Reference”
::={ Name=FieldOrGroup [,
Name=FieldOrGroup…] }
```

只有 PROTOCOL 行和::=行是必需的。所有其它行都是属性行,为 PROTOCOL 定义特殊的特征。属性行是可选的并且可以任意顺序出现。下面详细描述了每个属性行:

2.3.1 SUMMARIZE “Condition” : “FormatString” [“Condition” : “FormatString”…]

这个属性定义怎样以摘要模式显示一个 PROTOCOL。可以为每个条

件(Condition)指定一个不同的格式(FormatString)。有效条件定义在下面的 Condition 节中。通过在 FIELD 的名字前面加上$可以在 FormatString 中引用任意 FIELD 的值。除了 FIELD 名之外还有其它几种特殊的$关键字。

$LAYER	显示当前协议层
$VARBIND	显示整个 SNMP VarBind 列表
$field	显示 field 值(如果可用的话使用枚举名)
$:field	显示 field 值(以原始格式)
$#field	计算 field 出现的次数
$*field	列出 field 的全部出现

2.3.2 DESCRIPTION "Description"

这个属性定义对 PROTOCOL 的说明。仅用于提供信息的目的。

2.3.3 REFERENCE "Reference"

这个属性定义用来确定协议格式的参考材料。仅用于提供信息的目的。

2.3.4 ::={Name=FieldOrGroup[, Name=FieldOrGroup…]}

这个属性定义 PROTOCOL 中 FIELD 和 GROUP 的顺序。

2.4 FLOW 定义

FLOW 定义用来通过描述地址、协议类型和端口号在分组中的什么地方来定义网络流。FLOW 定义格式如下:

```
Name FLOW
HEADER {Option [, Option…] }
DLC-LAYER {Option [, Option…] }
NET-LAYER {Option [, Option…] }
CONNECTION {Option [, Option…] }
PAYLOAD {Option [, Option…] }
CHILDREN {Option [, Option…] }
STATE-BASED
STATES "Definitions"
```

只有 FLOW 行是必需的。所有其它行都是属性行,定义 FLOW 的一些特殊特征。属性行是可选的并可以任意顺序出现。但是,必须至少有一个属性行。下面详细描述了每个属性行:

2.4.1 HEADER {Option [, Option…] }

这个属性用来描述协议报头的长度。当前支持的选项是:

LENGTH=number	报头是 number 大小的固定长度
LENGTH=field	报头是由 field 的值确定的可变长度
IN-WORDS	报头长度单位是 32-位字而不是字节

2.4.2 DLC-LAYER {Option [, Option…] }

如果协议是数据链路层协议，就由这个属性描述它。当前支持的选项是:

DESTINAION=field	指示 field 是 DLC 目的地址
SOURCE=field	指示 field 是 DLC 源地址
PROTOCOL	指示这是一个数据链路层协议
TUNNELING	指示这是一个隧道协议

2.4.3 NET-LAYER {Option [, Option…] }

如果该协议是网络层协议，就由这个属性描述。当前支持的选项是:

DESTINATION=field	指示 field 是网络目的地址
SOURCE=field	指示 field 是网络源地址
TUNNELING	指示这是一个隧道协议
FRAMENTATION=type	指示这个协议支持分段。目前有两种分段类型: IPV4 和 IPV6

2.4.4 CONNECTION {Option [, Option…] }

如果该协议是面向连接的协议,那么这个属性描述怎样建立及断开连接。当前支持的选项是:

IDENTIFIER=field	指示连接标识符 field
CONNECT-START="flag"	指示正在初始化连接
CONNECT-COMPLETE="flag"	指示已经建立了连接

DISCONNECT - START="flag"	指示正在断开连接
DISCONNECT - COMPLETE="flag"	指示已经断开连接
INHERITED	指示这是面向连接的协议，但父协议在建立连接的地方

2.4.5 PAYLOAD {Option [, Option...] }

这个属性描述应该存储多少来自这种类型的分组的有效负载以在后来的分析中使用。当前支持的选项是:

INCLUDE - HEADER	指示应该包括该协议报头
LENGTH=number	指示应该存储有效负载的多少字节
DATA=field	指示 field 包含有效负载

2.4.6 CHILDREN {Option [, Option...] }

这个属性描述怎样确定子协议。当前支持的选项是:

DESTINATION=field	指示 field 是目的端口
SOURCE=field	指示 field 是源端口
LLCCHECK=flow	指示如果 DESTINAION 字段小于 0x50DC，那么就用 flow 代替当前流定义

2.4.6 STATE - BASED

这个属性指示该流是基于状态的流。

2.4.7 STATES "Definitions"

这个属性描述怎样用状态确定这个协议的子流。从下面的状态定义可以知道怎样定义这些状态。

2.5 条件

条件和 OPTIONAL 及 SUMMARIZE 属性一起使用，由下列组成:

Value1= =Value2	Value1 等于 Value2。适用于字符串值
Value1!=Value2	Value1 不等于 Value2。适用于字符串值
Value1<=Value2	Value1 小于或等于 Value2
Value1>=Value2	Value1 大于或等于 Value2
Value1<Value2	Value1 小于 Value2
Value2>Value2	Value1 大于 Value2
Field m/regex/	Field 匹配正则表达式 regex

Value1 和 Value2 可以是 FIELD 引用(字段名前面加$)或常量值。注意目前不支持复合条件语句(使用 AND 和 OR)。

2.6 STATE 定义

运行在数据网络上的很多应用利用复杂的方法通过使用多种状态来给流量分类。状态定义用于管理和维护从来自网络的流量中认识到的状态。

状态定义的基本格式是:

StateName: Operand Parameters [Operand Parameters]

用下面的操作数描述特定流的不同状态:

2.6.1 CHECKCONNECT, operand

检查连接。一旦连接上就执行 operand。

2.6.2 GOTO state

转到 state, 使用当前分组。

2.6.3 NEXT state

转到 state, 使用下一个分组。

2.6.4 DEFAULT operand

当所有其它操作都失败时执行 operand。

2.6.5 CHILD protocol

跳转到子 protocol 并在这个孩子中执行基于状态的处理(如果有的话)。

2.6.6 WAIT numPackets, operand1, operand2

等待指定数量的分组。当已经接收到指定数量的分组时执行 operand1。当接收到一个分组但还小于指定的分组数量时执行 operand2。

2.6.7 MATCH 'string' weight offset LF-offset range LF-range, operand

在分组中查找 'string', 如果找到了就执行 operand

2.6.8 CONSTANT number offset range, operand

在分组中检查常数, 如果找到就执行 operand

2.6.9 EXTRACTIP offset destination, operand

从分组中提取 IP 地址然后执行 operand。

2.6.10 EXTRACTPORT offset destination, operand

从分组中提取端口号然后执行 operand。

2.6.11CREATEREDIRECTEDFLOW, operand

创建一个改变了方向的流然后执行 operand。

3. 实例 PDL 规则

下面一节包含 PDL 规则文件的几个实例。

3.1 以太网

下面是针对以太网的 PDL 实例:

```
MacAddress FIELD
           SYNTAX            BYTESTRING(6)
           DISPLAY-HINT      "1x:"
           LOOKUP            MACADDRESS
           DESCRIPTION
                "MAC layer physical address"

etherType  FIELD
           SYNTAX             INT(16)
           DISPLAY-HINT       "1x:"
           LOOKUP             FILE    "EtherType.cf"
           DESCRIPTION
                "Ethernet type field"

etherData  FIELD
           SYNTAX               BYTESTRING(46..1500)
           ENCAP                etherType
```

```
            DISPLAY-HIND        "HexDump"
            DESCRIPTION
                  "Ethernet data"

ethernet    PROTOCOL
            DESCRIPTION
                  "Protocol format for an Ethernet frame"
            REFERENCE      "RFC   894"
    ::={      MacDest=macAddress  ,   MacSrc=macAddress  ,
EtherType=etherType,
        Data=etherData }

ethernet    FLOW
            HEADER                {  LENGTH=14  }
            DLC-LAYER             {
                  SOURCE=MacSrc,
                  DESTINATION=MacDest,
                  TUNNELING,
                  PROTOCOL
            }
            CHILDREN          { DESTINATION=EtherType , LLC-
CHECK=llc }
```

3.2 IPV4

这里是针对 IP 协议的 PDL 实例:

```
ipAddress   FIELD
            SYNTAX              BYTESTRING(4)
            DISPLAY-HINT        "1d."
            LOOKUP              HOSTNAME
            DESCRIPTION
                  "IP   address"
```

```
ipVersion  FIELD
           SYNATX        INT(4)
           DEFAULT       "4"

ipHeaderLength   FIELD
                 SYNTAX          INT(4)

ipTypeofService FIELD
                 SYNTAX     BITSTRING(8){  minCost(1),
    maxReliability(2), maxThruput(3), minDelay(4)}

ipLength   FIELD
           SYNTAX     UNSIGNEDINT(16)

ipFlags    FIELD
           SYNTAX     BITSTRING(3)  {     moreFrags(0)     ,
dontFrag(1)    }

ipProtocol FIELD
           SYNTAX     INT(8)
           LOOKUP  FILE "IpProtocol.cf"

ipData     FIELD
           SYNTAX        BYTESTRING(0..1500)
           ENCAP      ipProtocol
           DISPLAY-HINT "HexDump"

ip         PROTOCOL
           SUMMARIZE
           "$FragmentOffset != 0" :
                "IPFragment   ID=$Identification
  Offset=$FragmentOffset"
```

```
            "Default" :
                  "IP   Protocol=$Protocol"
            DESCRIPTION
                  "Protocol   format for the Internet Protocol"
            REFERENCE     "RFC 791"
::={ Version=ipVersion, HeaderLength=ipHeadLength,
    TypeOfService=ip TypeOfService, Length=ipLength,
    Identification=UInt16, IpFlags=ipFlags,
    FragmentOffset=ipFragmentOffset, TimeToLive=Int8,
    Protocol=ipProtocol, Checksum=ByteStr2,
    IpSrc=ipAddress, IpDest=ipAddress, Options=ipOptions,
    Fragment=ipFragment, Data=ipData }

ip          FLOW
            HEADER   { LENGTH=HeaderLength, IN-WORD }
            NET-LAYER {
                  SOURCE=IpSrc,
                  DESTINATION=IpDest,
                  FRAGMENTATION=IPV4,
                  TUNNELING
                  }
            CHILDREN { DESTINATION=Protocol }

ipFragData FIELD
            SYNTAX          BYTESTRING(1..1500)
            LENGTH          "ipLength - ipHeaderLength*4"
            DISPLAY-HINT    "HexDump"

ipFragment GROUP
            OPTIONAL   "$FragmentOffset !=0"
::= { Data=ipFragData }
```

```
ipOptionLength   FIELD
                 SYNTAX      UNSIGNEDINT(8)
                 DESCRIPTION
                    "Length of IP option "

ipOptionData     FIELD
                 SYNTAX      BYTESTRING(0..1500)
                 ENCAP       ipOptionCode
    DISPLAY-HINT  "HexDump"

ipOptions        GROUP
                 LENGTH        "(ipHeaderLength*4) - 20"
::={    Code=ipOptionCode    ,    Length=ipOptionLength    ,
Pointer=UInt8,
    Data=ipOptionData }
```

3.3 TCP

这里是针对 TCP 协议的 PDL 实例:

```
tcpPort     FIELD
            SYNTAX      UNSIGNEDINT(16)
            LOOKUP      FILE "TcpPort.cf"

tcpHeaderLength     FIELD
                    SYNTAX      INT(4)

tcpFlags    FIELD
            SYNTAX      BITSTRING(12){fin(0), syn(1), rst(2),
psh(3),
    ack(4), urg(5)}

tcpData     FIELD
            SYNTAX      BYTESTRING(0..1564)
```

```
            LENGTH      "($ipLength  -  ($ipHeaderLength*4))-
($tcpHeaderLen*4)"
            ENACP       tcpPort
            DISPLAY     "HexDump"

tcp         PROTOCOL
            SUMMARIZE
            "Default"  :
            "TCP  ACK=$Ack  WIN=$WindowSize"
            DESCRIPTION
            "Protocol  format  the  Transmission  Control
Protocol"
            REFERENCE     "RFC  973"
::={ SrcPort=tcpPort, DestPort=tcpPort, SequenceNum=UInt32,
    Ack=UInt32      ,       HeaderLength=tcpHeaderLen      ,
TcpFlags=tcpFlags,
    WindowSize=UInt16, Checksum=ByteStr2,
    UrgentPointer=UInt16, Options=tcpOptions, Data=tcpData}

tcp         FLOW
            HEADER      { LENGTH=HeaderLength, IN-WORDS }
            CONNECTION {
            IDENTIFIER=SequenceNum,
            CONNECT-START= "TcpFlags:1",
            CONNECT-COMPLETE= "TcpFlags:4",
            DISCONNECT-START= "TcpFlags:0",
            DISCONNECT-COMPLETE= "TcpFlags:4"
            }
      PAYLOAD  { INCLUDE-HEADER }
      CHILDREN   { DESTINATION=DestPort, SOURCE=SrcPort }

tcpOptionKind   FIELD
```

```
                SYNTAX      UNSIGNEDINT(8){  tcpOptEnd(0),
tcpNop(1)},
tcpMSS(2), tcpWscale(3), tcpTimestamp(4)}
                DESCRIPTION
                "Type of TCP option"

    tcpOptionData     FIELD
                      SYNTAX          BYTESTRING(0..1500)
                      ENCAP           tcoOptionKind
                      FLAGS           SAMELAYER
                      DISPLAY-HIN     "HexDump"

tcpOptions    GROUP
              LENGTH          "($tcpHeaderLen*4) - 20"
::={ Option=tcpOptionKind, OptionLength=UInt8,
        OptionData=tcpOptionData }

tcpMSS        PROTOCOL
::={ MaxSegmentSize=UInt16 }
```

3.4 HTTP(带状态)

这里是针对 HTTP 协议的一个 PDL 实例:

```
httpData   FIELD
           SYNTAX          BYTESTRING(1..1500)
           LENGTH          "($ipLength - ($ipHeaderLength*4))
- ($tcpHeaderLen*4)"
           DISPLAY-HINT "Text"
           FLAGS       NOLABEL

http       PROTOCOL
           SUMMARIZE
             "$httpData m/^GET|^HTTP|^HEAD|^POST/" :
```

```
            "HTTP  $httpData"
            "$httpData m/^[Dd]ate|^[Ss]erver|^[Ll]ast  -
[Mm]odified/":
            "HTTP $httpData"
          "$httpData m/^[Cc]ontent-/":
            "HTTP $httpData"
          "$httpData m/^<HTML>/":
            "HTTP [HTML  document]"
          "$httpData m/^GIF/":
            "HTTP [GIF    image]"
          "Default":
            "HTTP [Data]"
          DESCRIPTION
            "Protocol  format  for  HTTP"
::=  {  Data=httpData  }

http      FLOW
HEADER  {LENGTH=0}
CONNECTION    {INHERITED}
PAYLOAD {INCLUDE-HEADER, DATA=Data, LENGTH=256}
STATES
          "S0:CHECKCONNECT, GOTO  S1
              DEFAULT  NEXT  S0

          S1:WAIT 2, GOTO S2, NEXTS1
            DEFAULT NEXT  S0

          S2:MATCH
            '\n\r\n'      900  0  0  255  0, NEXT S3
            '\n\n'     900  0  0  255  0, NEXT S3
            'POST  /tds?'     50  0  0  127  1, CHILD
  sybaseWebsql
```

```
                '.hts  HTTP/1.0'  50  4  0  127  1, CHILD
    sybaseJdbc
                'jdbc:sybase:Tds'   50  4  0  127  1, CHILD
    sybaseTds
                'PCN-The  Poin' 500  4  1  255  0, CHILD
    pointcast
                't:  BW-C-'    100  4  1  255  0, CHILD
    backweb
                DEFAULT NEXT S3

            S3:MATCH
                '\n\r\n'        50  0  0    0  0, NEXT S3
                '\n\n'       50   0  0    0  0, NEXT  S3
                'Content-Type:' 800  0  0  255  0, CHILEmime
                'PCN-The  Poin' 500  4  1  255  0, CHILD
    pointcast
                't:  BW-C-'    100  4  1  255  0, CHILD
    backweb
                DEFAULT NEXT S0

sybaseWebsql    FLOW
                    STATE-BASED

sybaseJdbc      FLOW
                STATE-BASED

sybaseTds       FLOW
                STATE-BASED

pointcast       FLOW
                STATE-BASED
```

```
backweb        FLOW
               STATE - BASED

mime           FLOW
               STATE - BASED
               STATES

'application'     900   0   0     1   0, CHILDmimeApplication
'audio'         900   0   0     1   0, CHILDmimeAudio
'image'          50   0   0     1   0, CHILD   mimeImage
'text'        50   0   0     1   0, CHILD   mimeText
'video'          50   0   0     1   0, CHILDmimeVideo
'x - world'        500   4   1   255   0, CHILEmimeXworld
DEFAULT GOTO S0"

mimeApplication FLOW
                STATE - BASED

mimeAudio  FLOW
           STATE - BASED
           STATES
           "S0:  MATCH
                 'basic'          100   0   0   1   0, CHILD
   pdBasicAudio
                 'midi'       100   0   0   1   0, CHILD   pdMidi
                 'mpeg'       100   0   0   1   0, CHILD
   pdMpeg2Audio
                 'vnd.rn - realaudio'    100    0    0    1    0,
CHILD pdRealAudio
                 'wav'        100   0   0   1   0, CHILD   pdWav
                 'x - aiff'          100   0   0   1   0, CHILD
   pdAiff
```

```
                'x-midi'          100  0  0  1  0, CHILD
    pdMidi
                'x-mpeg'          100  0  0  1  0, CHILD
    pdMpeg2Audio
                'x-mpgurl'    100  0  0  1  0, CHILD
    pdMpeg3Audio
                'x-pn-realaudio' 100  0  0  1  0, CHILD
    pdRealAudio
                'x-wav'       100  0  0  1  0, CHILD
    pdWav
          DEFAULT GOTO S0"

mineImage  FLOW
           STATE-BASED

mineText   FLOW
           STATE-BASED

mineVideo  FLOW
           STATE-BASED

mineXworld FLOW
           STATE-BASED

pdBasicAideo  FLOW
           STATE-BASED

pdMidi     FLOW
           STATE-BASED

pdMpeg2Audio  FLOW
              STATE-BASED
```

```
pdMpeg3Audio  FLOW
              STATE - BASED

pdRealAudio   FLOW
              STATE - BASED

pdWav         FLOW
              STATE - BASED

pdAiff        FLOW
              STATE - BASED
```

重复利用来自流的信息以维持度量

每个流的流－入口存储了该流的一组统计测量信息，包括流中分组的总数、到达时间和从上次到达的微分时间。

再来看图 3，状态处理方法 328 执行为流的状态而定义的操作，例如迄今为止针对该流而识别出的特定协议。本发明的一个方面是有时可以用存储在流－入口中的一个或多个统计测量来确定一组一个或多个与该流有关的度量。例如，可以由状态处理器运行状态处理器指令与模式数据库 326 中的指令来执行这样的度量确定。随后可以由分析器子系统把这样的度量发往连接到监控器的主机上。或者也可以由连接到流－入口数据库 324 的处理器执行这样的度量确定。在图 10 中所示的我们的优选硬件实现中，可以对分析器主机接口和控制 1118 进行配置以通过高速缓冲存储器系统 1115 访问流－入口记录进而通过主机总线接口向处理器进行输出。随后处理器可以报告基本度量。

图 15 描述了监控器系统可以和主机 1504 装配在一起。监控器 300 不时地向主机 1504 发送度量信息，并且主机 1504 执行部分分析。

下面一节描述所发明的监控器怎样被用来通过提供 QOS 度量来监控服务质量(QOS)。

服务质量流量统计(度量)

下面这一节按照本发明的一个方面定义可能应用于服务质量(QOS)度量的通用结构。还定义了可以在本发明的实施方案中确定的“原始”(或“基本”)度量集以支持QOS。基本度量被作为状态处理的一部分或由连接到监控器300的处理器确定，由主机1504从基本度量确定QOS度量。这么分工的主要原因是完整的QOS度量需要复杂的计算，涉及到平方根和其它一些可能无法实时获得所需计算资源的函数。因此选择一些基本函数简化实时计算并从中确定完整的QOS度量。在本发明范围内也可以有其它函数分工。

可以由状态处理器运行状态处理器指令和模式数据库326中的指令来执行这样的度量确定。随后由分析器子系统通过连接到监控器的处理器或逻辑电路把这些基本度量发送出去。或者可以由连接到流-入口数据库324的微处理器(或其它一些逻辑)来执行这样的度量确定。在图10和图11中所示我们的优选硬件实现中，这样一个微处理器通过分析器主机接口和控制1118以及主机总线接口连接到高速缓冲存储器系统1115。可以对这些单元进行配置以通过高速缓冲存储器系统1115访问流-入口记录进而使微处理器能够确定并报告基本度量。

QOS度量可以分成下列度量组。名字是描述性的。列表并不详尽，还有其它度量可以使用。下面的QOS度量包括客户机到服务器(CS)和服务器到客户机(SC)度量。

像CSTraffic和SCTraffic这样的流量度量。

像CSJitter和SCJitter这样的振动度量。

像CSExchangeReponseTimeStartToStart、CSExchangeResponseTimeEndToStart、CSExchangeResponseTimeStartToEnd、SCExchangeReponseTimeStartToStart、SCExchangeResponseTimeEndToStart、SCExchangeResponseTimeStartToEnd这样的交换响应度量。

像CSTransactionReponseTimeStartToStart、CSApplicationReponseTimeEndToStart、CSApplicationReponseTimeStartToEnd、

SCTransactionReponseTimeStartToStart、SCApplicationReponseTimeEndToStart、SCApplicationReponseTimeStartToEnd这样的事务响应度量。

像ConnectionEstablishment、ConnectionGracefulTermination和ConnectionTimeoutTermination这样的连接度量。

像CSConnectionRetransmissions、SCConnectionRetransmissions和CSConnectionOutOfOrders、SCConnectionOutOfOrders这样的连接顺序度量。

连接窗口度量有CSCoonnectionWindow、SCConnectionWindow、CSConnectionFrozenWindows、SCConnectionFrozenWindows、CSConnectionClosedWindows以及SCConnectionClosedWindows。

QOS基本度量

表示一组数据最简单的方法是通过子范围内的频率分布。在优选实施方案中有一些创建子范围的规则。首先范围需要是已知的。其次需要确定子范围大小。固定子范围大小是首选的，替代实施方面可以使用可变的子范围大小。

确定完整的频率分布可能要进行大量的计算。这样，优选实施方案使用由对象总体中的单个数据元素上的求和函数确定的度量。

度量报告方法提供可以用来计算有用的统计测量的数据。在一种实施方案中，度量报告方法是根据状态而不时执行的状态处理的一部分，在另一种实施方案中，度量报告方法由访问流记录的微处理器不时地执行。优选地，度量报告方法提供基本度量，由主机1504执行最终的QOS度量计算。除了保持实时状态处理简单之外，用这种方式分割任务还提供可升级的度量。例如，来自两个间隔的度量可以被结合成更大间隔的度量。

例如，考虑把算术平均值定义为数据的和除以数据元素的个数。

$$\overline{X}=\frac{\sum x}{N}$$

由度量报告方法提供的两个基本度量是x的和以及元素个数N。主机1504执行除法操作以得到平均值。此外，通过加上x'的和以及它的元素个数把来自两个间隔的两组基本度量结合在一起可以得到

合并后的和以及元素个数。

已经对基本度量进行了选择以使可用数据的数量最大，用来存储度量所需的存储量最小并使产生度量所需的处理开销最小。在一个包含五个无符号整数值的数据结构中提供基本度量。

- N　　度量的数据点个数
- $\sum X$　　度量的所有数据点值的和
- $\sum(X^2)$　　度量的所有数据点值的平方和
- X_{max}　　度量的最大数据点值
- X_{min}　　度量的最小数据点值

度量用来描述时间间隔上的事件。从保存在流－入口中的统计测量确定基本度量。没有必要把所有事件都高速缓存起来然后在间隔的末尾对它们计数。基本度量还被设计成在合并相邻间隔时易于升级。

当合并连续时间间隔的基本度量时应用下列规则：

- N　　$\sum N$
- $\sum X$　　$\sum(\sum X)$
- $\sum(X^2)$　　$\sum(\sum(X^2))$
- X_{max}　　$MAX(X_{max})$
- X_{min}　　$MIN(X_{min})$

除了上面五个值之外，优选实施方案数据结构中还包括一个“趋势”指示符。由枚举类型提供。这么做的原因是产生趋势信息的优选方法是通过从间隔的最终值减去该间隔的最初值来产生趋势信息。只有结果的符号才对确定趋势指示有价值。

可以在基本度量上执行的典型操作包括：

- 数量　　N
- 频率　　$\frac{N}{TimeInterval}$
- 最大值　　Xmax
- 最小值　　Xmin
- 范围　　R=Xmax－Xmin
- 算术平均值　$\overline{X}=\frac{\sum X}{N}$

●均方根值 $RMS=\sqrt{\frac{\sum(X^2)}{N}}$

●方差 $\sigma^2=\frac{\sum(X-\overline{X})^2}{N}=\frac{(\sum X^2)-2\overline{X}(\sum X)+N(\overline{X}^2)}{N}$

●标准偏差 $\sigma=\sqrt{\frac{\sum(X-\overline{X})^2}{N}}=\sqrt{\frac{(\sum(X^2))-2\overline{X}(\sum X)+N(\overline{X}^2)}{N}}$

●趋势信息可以是选中的间隔之间的趋势或一个间隔内的趋势。选中的间隔之间的趋势是管理应用函数。通常管理站倾向于报告的间隔的平均值。一个间隔内的趋势被表示成枚举类型，并且可以通过从该间隔中的最后一个值减去第一个值并根据结果的符号值给趋势赋值很容易地生成。

替代实施方案

在度量的不同实现中可以包括下列不同数据元素中的一个或多个:

- 增量和(即微分值)。趋势枚举可以基于这个简单的计算。
- 增量值的绝对值的和。提供对间隔内的全部活动的测量。
- 正增量值的和以及负增量值的和。把这些值中的每一个和相关的计数及最大值扩展到一起就可以给出精确的信息
- 斜交的统计测量，可以通过给现有的度量增加 $\Sigma(X^3)$ 而获得。
- 峰度的统计测量，可以通过给现有的度量增加 $\Sigma(X^3)$ 和 $\Sigma(X^4)$ 而获得。
- 用来通过其计算最小二乘方线的斜率的数据。

现在详细描述不同的度量。

流量度量

CSTraffic

定义

这个度量包含有关为给定应用以及特定客户机－服务器对或特定服务器和它的所有客户机而测量的流量大小的信息。

在标准、RMON II、AL/NL 矩阵表中可以找到这个信息副本。在这里包括它是为了给应用以及提高性能后的相关收益提供方便，提

高性能是通过在执行 QOS 分析时避免访问不同的功能 RMON 区域的需要。

度量规范

度量	适用性	单元	说明
N	适用	分组	从客户机到服务器的分组数量
Σ	适用	8 位位组	从客户机到服务器的这些分组中的 8 位位组的总数量
Maximum	不适用		
Minimum	不适用		

SCTRAFFIC

定义

这个度量包含有关为给定应用以及特定客户机－服务器对或特定服务器和它的全部客户机而测量的流量大小的信息。

在标准、RMON II、AL/NL 矩阵表中可以找到这个信息副本。在这里包括它是为了给应用以及提高性能后的相关收益提供方便，提高性能是通过在执行 QOS 分析时避免访问不同的功能 RMON 区域的需要。

度量规范

度量	适用性	单元	说明
N	适用	分组	从服务器到客户机的分组数量
Σ	适用	8 位位组	从服务器到客户机的这些分组中的 8 位位组的总数
Maximum	不适用		
Minimum	不适用		

振动度量

CSJitter

定义

这个度量包含有关为数据分组测量的振动(例如，分组间间隔)

的信息，该数据分组是给定的应用以及特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 CSJitter 测量从客户机到服务器的数据消息的振动。

数据分组以从客户机到服务器的 1st 传输协议数据分组/单元(TPDU)开始，并以另一方向上的 1st 后继数据分组作为界线(或终止)。在该消息内的数据分组之间测量客户机到服务器分组间间隔。注意在我们的实现中，ACKnowledgement 在这个度量的测量中并不考虑确认。

另外，在测量中也不考虑重传无序数据分组。在从客户机到服务器的数据消息中的最后一个分组和同一方向上下一个数据消息的 1st 分组之间的间隔并不被表示为分组间间隔。

度量规范

度量	适用性	单元	说明
N	适用	分组间间隔	为从客户机到服务器的数据测量的分组间间隔的数量
Σ	适用	useconds	这些分组间间隔中的时间差的总和
Maximum	适用	useconds	测量的分组间最大时间差
Minimum	适用	useconds	测量的分组间最小时间差

SCJitter

定义

这个度量包含有关为数据分组测量的振动(例如，分组间间隔)的信息，该数据分组是给定的应用以及特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 SCJitter 测量从服务器到客户机的数据消息的振动。

数据消息以从服务器到客户机的 1st 传输协议数据分组/单元(TPDU)开始，并以另一方向上的 1st 后继数据分组作为界线(或终止)。在该消息内的数据分组间测量服务器到客户机分组间间隔。注意在我们的实现中，在这个度量的测量中并未考虑确认。

度量规范

度量	适用性	单元	说明
N	适用	分组间间隔	为从服务器到客户机的数据测量方法分组间间隔数量
Σ	适用	useconds	这些分组间间隔中的时间差的总和
Maximum	适用	useconds	测量的分组间最大时间差
Minimum	适用	useconds	测量的分组间最小时间差

交换响应度量

CSExchangeResponseTimeStartToStart

定义

这个度量包含有关为数据分组测量的传输级响应时间的信息，该数据分组是给定的应用以及特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 CSExchangeReponseTimeStartToStart 测量在客户机到服务器数据消息的开始和从服务器到客户机的它们的后继响应数据消息的开始之间的响应时间。

客户机到服务器数据消息以从客户机到服务器的 1^{st} 传输协议数据分组/单元(TPDU)开始，并以另一方向上的 1^{st} 后继数据分组作为界线(或终止)。用这个度量测量在客户机到服务器数据消息的开始和服务器到客户机数据消息的开始之间的总时间。注意该度量的测量中没有考虑确认。

另外，在该测量中也没有考虑重传或无序数据分组。

度量定义

度量	适用性	单元	说明
N	适用	客户机到服务器消息	为从客户机到服务的数据交换测量的客户机到服务器消息数量
Σ	适用	useconds	这些交换响应时间中开始到开始时间差的总和
Maximum	适用	useconds	这些交换响应时间中最大的开始到开始时间差
Minimum	适用	useconds	这些交换响应时间中最小的开始到开始时间差

CSExchangeResponseTimeEndToStart

定义

这个度量包含有关为数据分组测量的传输级响应时间的信息，该数据分组是给定的应用以及特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 CSExchangeResponseTimeEndToStart 测量在从客户机到服务器的数据消息的结尾和从服务器到客户机的它们的后继响应数据消息的开始之间的响应时间。

客户机到服务器数据消息以从客户机到服务器的 1^{st} 传输协议数据分组/单元(TPDU)开始，并以另一方向上的 1^{st} 后继数据分组作为界线(或终止)。用这个度量测量在客户机到服务器数据消息的结尾和服务器到客户机的数据消息的开始之间的总时间。注意该度量的测量中没有考虑确认。

另外，在该测量中也没有考虑对无序数据分组的重传。

度量规范

度量	适用性	单元	说明
N	适用	客户机到服务器消息	为从客户机到服务的数据交换测量的客户机到服务器消息数量
Σ	适用	useconds	这些交换响应时间中结尾到开始时间差的总和
Maximum	适用	useconds	这些交换响应时间中最大的结尾到开始时间差
Minimum	适用	useconds	这些交换响应时间中最小的结尾到开始时间差

CSExchangeReponseTimeStartToEnd

定义

这个度量包含有关为数据分组测量的传输级响应时间的信息，该数据分组是给定的应用以及特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 CSExchangeResponseTimeStartToEnd 测量在从客户机到服务器的数据消息的开始和从服务器到客户机的它们的后继响应数据消息的结尾之间的响应时间。

客户机到服务器数据消息以从客户机到服务器的 1st 传输协议数据分组/单元(TPDU)开始，并以另一方向上的 1st 后继数据分组作为界线(或终止)。另一方向(例如，从服务器到客户机)上的响应消息的结尾以下一个客户机到服务器消息的 1st 数据分组之前的消息的最后数据作为界线。用这个度量测量在客户机到服务器数据消息的开始和服务器到客户机数据消息的结尾之间的总时间。注意该度量的测量中没有考虑确认。

另外，在该测量中也没有考虑重传或无序数据分组。

度量规范

度量	适用性	单元	说明
N	适用	客户机到服务器消息	为从客户机到服务的数据交换测量的客户机到服务器和服务器到客户机交换消息对的数量
Σ	适用	useconds	这些交换响应时间中开始到结尾时间差的总和
Maximum	适用	useconds	这些交换响应时间中最大的开始到结尾时间差
Minimum	适用	useconds	这些交换响应时间中最小的开始到结尾时间差

SCExchangeResponseTimeStartToStart

定义

这个度量包含有关为数据分组测量的传输级响应时间的信息，该数据分组是给定的应用以及特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 SCExchangeResponseTimeStartToStart 测量在从服务器到客户机数据消息的开始和从客户机到服务器的它们的后继响应数据消息的开始之间的响应时间。

服务器到客户机数据消息以从服务器到客户机的 1^{st} 传输协议数据分组/单元(TPDU)开始，并以另一方向上的 1^{st} 后继数据分组作为界线(或终止)。用这个度量测量在服务器到客户机数据消息的开始和客户机到服务器数据消息的开始之间的总时间。注意该度量的测量中没有考虑确认。

另外，在该测量中也没有考虑重传或无序数据分组。

度量规范

度量	适用性	单元	说明
N	适用	服务器到客户机消息	为从客户机到服务器的数据交换测量的服务器到客户机消息的数量
Σ	适用	useconds	这些交换响应时间中开始到开始时间差的总和
Maximum	适用	useconds	这些交换响应时间中最大的开始到开始时间差
Minimum	适用	useconds	这些交换响应时间中最小的开始到开始时间差

SCExchangeResponseTimeEndtToStart

定义

这个度量包含有关为数据分组测量的传输级响应时间的信息，该数据分组是给定的应用以及特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 SCExchangeResponseTimeEndToStart 测量在服务器到客户机数据消息的结尾和客户机到服务器的它们的后继响应数据消息的开始之间的响应时间。

服务器到客户机数据消息以从服务器到客户机的 1^{st} 传输协议数据分组/单元(TPDU)开始，并以另一方向上的 1^{st} 后继数据分组作为界线(或终止)。用这个度量测量在服务器到客户机数据消息的结尾和客户机到服务器数据消息的开始之间的总时间。注意该度量的测量中没有考虑确认。

另外，在该测量中也没有考虑重传或无序数据分组。

度量规范

度量	适用性	单元	说明
N	适用	服务器到客户机消息	为从客户机到服务器的数据交换测量的服务器到客户机消息的数量
Σ	适用	useconds	这些交换响应时间中结尾到开始时间差的总和
Maximum	适用	useconds	这些交换响应时间中最大的结尾到开始时间差
Minimum	适用	useconds	这些交换响应时间中最小的结尾到开始时间差

SCExchangeResponseTimeStartToEnd

定义

这个度量包含有关为数据分组测量的传输级响应时间的信息，该数据分组是给定的应用以及特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 SCExchangeResponseTimeStartToEnd 测量在从服务器到客户机数据消息的开始和从客户机到服务器的它们的后继响应数据消息的结尾之间的响应时间。

服务器到客户机数据消息以从服务器到客户机的 1^{st} 传输协议数据分组/单元(TPDU)开始，并以另一方向上的 1^{st} 后继数据分组作为界线(或终止)。另一方向(例如，从客户机到服务器)上的响应消息的结尾以下一个服务器到客户机消息的 1^{st} 数据分组之前的消息的最后数据作为界线。用这个度量测量在服务器到客户机数据消息的开始和客户机到服务器数据消息的结尾之间的总时间。注意该度量的测量中没有考虑确认。

另外，在该测量中也没有考虑对无序数据分组的重传。

度量规范

度量	适用性	单元	说明
N	适用	服务器到客户机消息	为从服务器到客户机的数据交换测量的服务器到客户机和客户机到服务器交换消息对的数量
Σ	适用	useconds	这些交换响应时间中开始到结尾时间差的总和
Maximum	适用	useconds	这些交换响应时间中最大的开始到结尾时间差
Minimum	适用	useconds	这些交换响应时间中最小的开始到结尾时间差

事务响应度量

CSTransactionResponseTimeStartToStart

定义

这个度量包含有关为应用事务测量的应用级响应时间的信息，该应用事务是给定的应用以及特定客户机－服务器对或特定服务器和它的所有客户的。尤其 CSTransactionResponseTimeStartToStart 测量在客户机到服务器应用事务的开始和服务器到客户机的它们的后继事务响应的开始之间的响应时间。

客户机到服务器事务以从客户机到服务器的事务请求的 1^{st} 传输协议数据分组/单元(TPDU)开始，并以对该事务请求的响应的 1^{st} 后继数据分组作为界线(或终止)。用这个度量测量在客户机到服务器事务请求的开始和服务器到客户机的实际事务响应的开始之间的总时间。

这个度量被看作是“最好－成果”测量。实现这个度量的系统产生“最好－成果”以便用特定应用的逻辑事务定义划定请求和响应的开始和结尾。对这个度量最低级的支持也使这个度量等效于 CSExchangeResponseTimeStartToStart。

度量规范

度量	适用性	单元	说明
N	适用	客户机到服务器事务请求	为从客户机到服务的应用请求测量的客户机到服务器事务请求的数量
Σ	适用	useconds	这些应用响应时间中开始到开始时间差的总和
Maximum	适用	useconds	这些应用响应时间中最大的开始到开始时间差
Minimum	适用	useconds	这些应用响应时间中最小的开始到开始时间差

CSTransactionResponseTimeEndToStart

定义

这个度量包含有关为应用事务测量的应用级响应时间的信息，该应用事务是给定的应用以及特定客户机－服务器对或特定服务器和它的所有客户的。尤其 CSTransactionResponseTimeEndToStart 测量在客户机到服务器的应用事务的结尾和服务器到客户机的它们的后继事务响应的开始之间的响应时间。

客户机到服务器事务以从客户机到服务器的事务请求的 1st 传输协议数据分组/单元(TPDU)开始，并以对该事务请求的响应的 1st 后继数据分组作为界线(或终止)。用这个度量测量在客户机到服务器事务请求的结尾和服务器到客户机的实际事务响应的开始之间的总时间。

这个度量被看作是“最好－成果”测量。实现这个度量的系统产生“最好－成果”以便用特定应用的逻辑事务定义划定请求和响应的开始和结尾。对这个度量最低级的支持也使这个度量等效于 CSExchangeResponseTimeEndToStart。

度量规范

度量	适用性	单元	说明
N	适用	客户机到服务器事务请求	为从客户机到服务的应用请求测量的客户机到服务器事务请求的数量
Σ	适用	useconds	这些应用响应时间中结尾到开始时间差的总和
Maximum	适用	useconds	这些应用响应时间中最大的结尾到开始时间差
Minimum	适用	useconds	这些应用响应时间中最小的结尾到开始时间差

CSTransactionResponseTimeStartToEnd

定义

这个度量包含有关为应用事务测量的应用级响应时间的信息，该应用事务是给定的应用以及特定客户机－服务器对或特定服务器和它的所有客户的。尤其 CSTransactionResponseTimeStartToEndt 测量在客户机到服务器应用事务的开始和服务器到客户机的它们的后继事务响应的结尾之间的响应时间。

客户机到服务器事务以从客户机到服务器的事务请求的 1^{st} 传输协议数据分组/单元(TPDU)开始，并以对该事务请求的响应的 1^{st} 后继数据分组作为界线(或终止)。另一方向(例如，从服务器到客户机)上的事务响应的结尾由下一个客户机到服务器事务请求的 1^{st} 数据之前的事务响应的最后数据划定。用这个度量测量在客户机到服务器事务请求的开始和服务器到客户机的实际事务响应的结尾之间的总时间。

这个度量被看作是“最好－成果”测量。实现这个度量的系统产生“最好－成果”以便用特定应用的逻辑事务定义划定请求和响应的开始和结尾。对这个度量最低级的支持也使这个度量等效于 CSExchangeResponseTimeStartToEnd。

度量规范

度量	适用性	单元	说明
N	适用	客户机到服务器事务	为从客户机到服务的事务测量的客户机到服务器和服务器到客户机请求/响应对的数量
Σ	适用	useconds	这些应用响应时间中开始到结尾时间差的总和
Maximum	适用	useconds	这些应用响应时间中最大的开始到结尾时间差
Minimum	适用	useconds	这些应用响应时间中最小的开始到结尾时间差

SCTransactionResponseTimeStartToStart

定义

这个度量包含有关为应用事务测量的应用级响应时间的信息,该应用事务是给定的应用以及特定客户机-服务器对或特定服务器和它的所有客户的。尤其 SCTransactionResponseTimeStartToStart 测量在服务器到客户机应用事务的开始和客户机到服务器的它们的后继事务响应的开始之间的响应时间。

服务器到客户机事务以从服务器到客户机的事务请求的 1st 传输协议数据分组/单元(TPDU)开始,并以对该事务请求的响应的 1st 后继数据分组作为界线(或终止)。用这个度量测量在服务器到客户机事务请求的开始和客户机到服务器的实际事务响应的开始之间的总时间。

这个度量被看作是“最好-成果”测量。实现这个度量的系统产生“最好-成果”以便用特定应用的逻辑事务定义划定请求和响应的开始和结尾。对这个度量最低级的支持也使该度量等效于 SCExchangeResponseTimeStartToStart。

度量规范

度量	适用性	单元	说明
N	适用	服务器到客户机事务请求	为从服务器到客户机的应用请求测量的 服务器到客户机事务请求的数量
Σ	适用	useconds	这些应用响应时间中开始到开始时间差的总和
Maximum	适用	useconds	这些应用响应时间中最大的开始到开始时间差
Minimum	适用	useconds	这些应用响应时间中最小的开始到开始时间差

SCTransactionResponseTimeEndToStart

定义

这个度量包含有关为应用事务测量的应用级响应时间的信息，该应用事务是给定的应用以及特定客户机－服务器对或特定服务器和它的所有客户的。尤其 SCTransactionResponseTimeEndToStart 测量在服务器到客户机应用事务的结尾和客户机到服务器的它们的后继事务响应的开始之间的响应时间。

服务器到客户机事务以从服务器到客户机的事务请求的 1^{st} 传输协议数据分组/单元(TPDU)开始，并以对该事务请求的响应的 1^{st} 后继数据分组作为界线(或终止)。用这个度量测量在服务器到客户机事务请求的结尾和客户机到服务器的实际事务响应的开始之间的总时间。

这个度量被看作是“最好－成果”测量。实现这个度量的系统产生“最好－成果”以便用特定应用的逻辑事务定义划定请求和响应的开始和结尾。对这个度量最低级的支持也使该度量等效于 SCExchangeResponseTimeEndtToStart。

度量规范

度量	适用性	单元	说明
N	适用	服务器到客户机事务请求	为从服务器到客户机的应用请求测量的 服务器到客户机事务请求的数量
Σ	适用	useconds	这些应用响应时间中结尾到开始时间差的总和
Maximum	适用	useconds	这些应用响应时间中最大的结尾到开始时间差
Minimum	适用	useconds	这些应用响应时间中最小的结尾到开始时间差

SCTransactionResponseTimeStartToEnd

定义

这个度量包含有关为应用事务测量的应用级响应时间的信息，该应用事务是给定的应用以及特定客户机－服务器对或特定服务器和它的所有客户的。尤其 SCTransactionResponseTimeStartToEnd 测量在服务器到客户机应用事务的开始和客户机到服务器的它们的后继事务响应的结尾之间的响应时间。

服务器到客户机事务以从服务器到客户机的事务请求的 1st 传输协议数据分组/单元(TPDU)开始，并以对该事务请求的响应的 1st 后继数据分组作为界线(或终止)。另一方向(例如，从客户机到服务器)上的事务响应的结尾由下一个服务器到客户机事务请求的 1st 数据之前的事务响应的最后数据划定。用这个度量测量在服务器到客户机事务请求的开始和客户机到服务器的实际事务响应的结尾之间的总时间。

这个度量被看作是“最好－成果”测量。实现这个度量的系统产生“最好－成果”以便用特定应用的逻辑事务定义划定请求和响应的开始和结尾。对这个度量最低级的支持也使该度量等效于 SCExchangeResponseTimeStartToEnd。

度量规范

度量	适用性	单元	说明
N	适用	服务器到客户机事务请求	为从服务器到客户机的事务测量的服务器到客户机和客户机到服务器请求/响应对的数量
Σ	适用	useconds	这些应用响应时间中开始到结尾时间差的总和
Maximum	适用	useconds	这些应用响应时间中最大的开始到结尾时间差
Minimum	适用	useconds	这些应用响应时间中最小的开始到结尾时间差

连接度量

ConnectionEstablishment

定义

这个度量包含有关给定应用和特定客户机－服务器对或特定服务器和它的所有客户机的传输级连接建立的信息。尤其 ConnectionsEstablishment 测量客户机到服务器建立的连接数。该信息包含，实际包括:

- 成功建立的传输连接数
- 建立的连接的准备时间
- 同时建立的最大连接数
- 失败的连接建立尝试(由于超时或拒绝)数

注意从这个度量和 ConnectionGracefulTermination 和 ConnectionTimeoutTermination 度量可以获得“当前建立的传输连接数”，方法如下:

当前连接数: ==“成功建立数”

－“正常终止数”

－“超时终止数”

连接的准备时间被定义为在第一个传输级连接建立请求(即，SYN、CR－TPDU 等)和在该连接上交换的第一个数据分组之间的时间差。

度量规范

度量	适用性	单元	说明
N	适用	连接	从客户机到服务建立的连接数
Σ	适用	useconds	这些已建立的连接中连接准备时间的总和
Maximum	适用	连接	从客户机到服务器同时建立的连接的最大数
Minimum	不适用	连接	从客户机到服务器建立连接同时失败的数量

ConnectionGracefulTermination

定义

这个度量包含有关正常终止的传输级连接的信息，该传输级连接是给定应用和特定客户机－服务器对或特定服务器和所有它的客户机的。尤其 ConnectionsGracefulTermination 以数量和总计连接持续时间测量正常终止的连接。该信息包含，实际上包括：

- 正常终止的传输连接数
- 正常终止的连接的持续时间(生命期)

度量规范

度量	适用性	单元	说明
N	适用	连接	在客户机和服务器之间正常终止的连接数
Σ	适用	毫秒(ms)	这些终止的连接的连接持续时间(生命期)的总和
Maximum	不适用		
Minimum	不适用		

ConnectionTimeoutTermination

定义

这个度量包含有关非正常(例如，超时)终止的传输级连接的信

息，该传输级连接是给定的应用和特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 ConnectionsTimeoutTermination 以数量和总计连接持续时间测量以前建立并超时的连接。该信息包含，实际上包括：

●超时传输连接数

●超时终止的连接的持续时间(生命期)

这个度量的持续因子被看作是“最好－成果”测量。当网络实体实际探测到连接超时状态时独立的网络监控器不能真正的认识到这一点，因此当连接超时真正发生时需要进行推断和估计。

度量规范

度量	适用性	单元	说明
N	适用	连接	在客户机和服务器之间超时连接的数量
Σ	适用	毫秒(ms)	这些终止的连接的连接持续时间(生命期)的总和
Maximum	不适用		
Minimum	不适用		

连接顺序度量

CSConnectionRetransmissions

定义

这个度量包含有关传输级连接健康状态的信息，该传输级连接健康状态是针对给定应用和特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 CSConnectionRetransmissions 测量在从客户机到服务器的数据意义的 PDU(分组)被进行了重传的传输中建立的连接的生命期中真实事件的数量。

注意由网络监控设备看到的重传事件指示从存在网络上获得的 TPDU 的“副本”。

度量规范

度量	适用性	单元	说明
N	适用	事件	从客户机到服务数据 TPDU 重传的数量
Σ	不适用		
Maximum	不适用		
Minimum	不适用		

SCConnectionRetransmisions

定义

这个度量包含有关传输级连接健康状态的信息，该传输级连接健康状态是针对给定应用和特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 SCConnectionRetransmissions 测量在从服务器到客户机的数据意义的 PDU(分组)被进行了重传的传输中建立的连接的生命期中真实事件的数量。

注意由网络监控设备看到的重传事件指示从存在网络上获得的 TPDU 的“副本”。

度量规范

度量	适用性	单元	说明
N	适用	事件	从服务器到客户机数据 TPDU 重传的数量
Σ	不适用		
Maximum	不适用		
Minimum	不适用		

CSConnectionOutOfOrders

定义

这个度量包含有关传输级连接健康状态的信息，该传输级连接健康状态是针对给定应用和特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 CSConnectionOutOfOrders 测量在从客户机到服务器的数据意义的 PDU(分组)被检测到是不连续的传输中建立的连接的生命期中真实事件的数量。

注意重传（或复制）被看作是与无序不同的事件，并在

CSConnectionRetransmissions 中对其进行单独跟踪。

度量规范

度量	适用性	单元	说明
N	适用	事件	从客户机到服务器的无序 TPDU 事件的数量
Σ	不适用		
Maximum	不适用		
Minimum	不适用		

SCConnectionOutOfOrder

定义

这个度量包含有关传输级连接健康状态的信息，该传输级连接健康状态是针对给定应用和特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 SCConnectionOutOfOrders 测量在从服务器到客户机的数据意义的 PDU(分组)被检测到是不连续的传输中建立的连接的生命期中真实事件的数量。

注意重传(或复制)被看作是与无序不同的事件，并在 SCConnectionRetransmissions 中对其进行单独跟踪。

度量规范

度量	适用性	单元	说明
N	适用	事件	从服务器到客户机无序 TPDU 事件的数量
Σ	不适用		
Maximum	不适用		
Minimum	不适用		

连接窗口度量

CSConnectionWindow

定义

这个度量包含有关传输级连接窗口的信息，该窗口是针对给定的应用和特定客户机－服务器对或特定服务器和它的所有客户机的。

尤其 CSConnectionWindow 测量从客户机到服务器建立的连接的生命期中传输级确认的数量和它们的相对大小。

注意通过区别这个度量的确认数和从客户机到服务器的全部流量(见上面的 CSTraffic)可以估计数据 TPDU(分组)的数量。因为连接建立和终止 TPDUS 的缘故在这个计算中会有轻微的误差，但应该是无足轻重的。

度量规范

度量	适用性	单元	说明
N	适用	事件	从客户机到服务器 ACK TPDU 重传的数量
Σ	适用	增量	确认的窗口大小的总和
Maximum	适用	增量	这些确认中最大的窗口大小
Minimum	适用	增量	这些确认中最小的窗口大小

SCConnectionWindow

定义

这个度量包含有关传输级连接窗口的信息，该窗口是针对给定的应用和特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 SCConnectionWindow 测量从服务器到客户机建立的连接的生命期中传输级确认的数量和它们的相对大小。

注意通过区别这个度量的确认数和从服务器到客户机的全部流量(见上面的 SCTraffic)可以估计数据 TPDU(分组)的数量。因为连接建立和终止 TPDUS 的缘故在这个计算中会有轻微的误差，但应该是无足轻重的。

度量规范

度量	适用性	单元	说明
N	适用	事件	从服务器到客户机的 ACK TPDU 重传的数量
Σ	适用	增量	确认的窗口大小的总和
Maximum	适用	增量	这些确认中最大的窗口大小
Minimum	适用	增量	这些确认中最小的窗口大小

CSConnectionFrozenWindows

定义

这个度量包含有关传输级协议窗口的信息，该窗口是针对给定应用和特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 CSConnectionFrozenWinddow 测量在建立的连接生命期中从客户机到服务器有效确认数据的传输级确认的数量，但这些确认：

- 不能增加窗口上部边缘
- 减小窗口上部边缘

度量规范

度量	适用性	单元	说明
N	适用	事件	从客户机到服务器带有冻结/减小窗口的 ACK TPDU 的数量
Σ	不适用		
Maximum	不适用		
Minimum	不适用		

SCConnectionFrozenWindows

定义

这个度量包含有关传输级协议窗口的信息，该窗口是针对给定应用和特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 SCConnectionFrozenWinddow 测量在建立的连接生命期中从服务器到客户机有效确认数据传输级确认的数量，但这些确认：

- 不能增加窗口上部边缘
- 减小窗口上部边缘

度量规范

度量	适用性	单元	说明
N	适用	事件	从客户机到服务器的带有冻结/减小窗口的 ACK TPDU 的数量
Σ	不适用		
Maximum	不适用		
Minimum	不适用		

CSConnectionClosedWindows

定义

这个度量包含有关传输级协议窗口的信息，该窗口是针对给定应用和特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 CSConnectionClosedWinddow 测量在建立的连接生命期中从客户机到服务器完全关闭确认/顺序窗口的传输级确认的数量。

度量规范

度量	适用性	单元	说明
N	适用	事件	从客户机到服务器带有关闭的窗口的 ACK TPDU 的数量
Σ	不适用		
Maximum	不适用		
Minimum	不适用		

SCConnectionClosedWindows

定义

这个度量包含有关传输级协议窗口的信息，该窗口是针对给定应用和特定客户机－服务器对或特定服务器和它的所有客户机的。尤其 SCConnectionClosedWinddow 测量在建立的连接生命期中从服务器到客户机完全关闭确认/顺序窗口的传输级确认的数量。

度量规范

度量	适用性	单元	说明
N	适用	事件	从客户机到服务器带有关闭的窗口的 ACK TPDU 的数量
Σ	不适用		
Maximum	不适用		
Minimum	不适用		

本发明的实施方案用必要的识别模式和状态转换程序自动地产生流签名。这来自根据解析规则分析分组并产生状态转换进行查找。任意层上的应用和协议都可以通过分组序列的状态分析进行识别。

注意本领域的技术人员应该理解用来连接多种不同类型设备的计算机网络，这些设备包括像电话、“Internet”广播、寻呼机等。这里所用的术语“计算机”包括所有这样的设备，“计算机网络”包括这样的计算机的网络。

虽然已经根据目前的优选实施方案对本发明进行了描述，但可以理解这些内容不应该被解释为对本发明的限制。不怀疑那些本领域的技术人员在阅读了上面的内容之后能够明白多种不同的变更和修改。因此，意图把权利要求表示为覆盖所有的变更和修改并符合本发明的真正精神和范围。

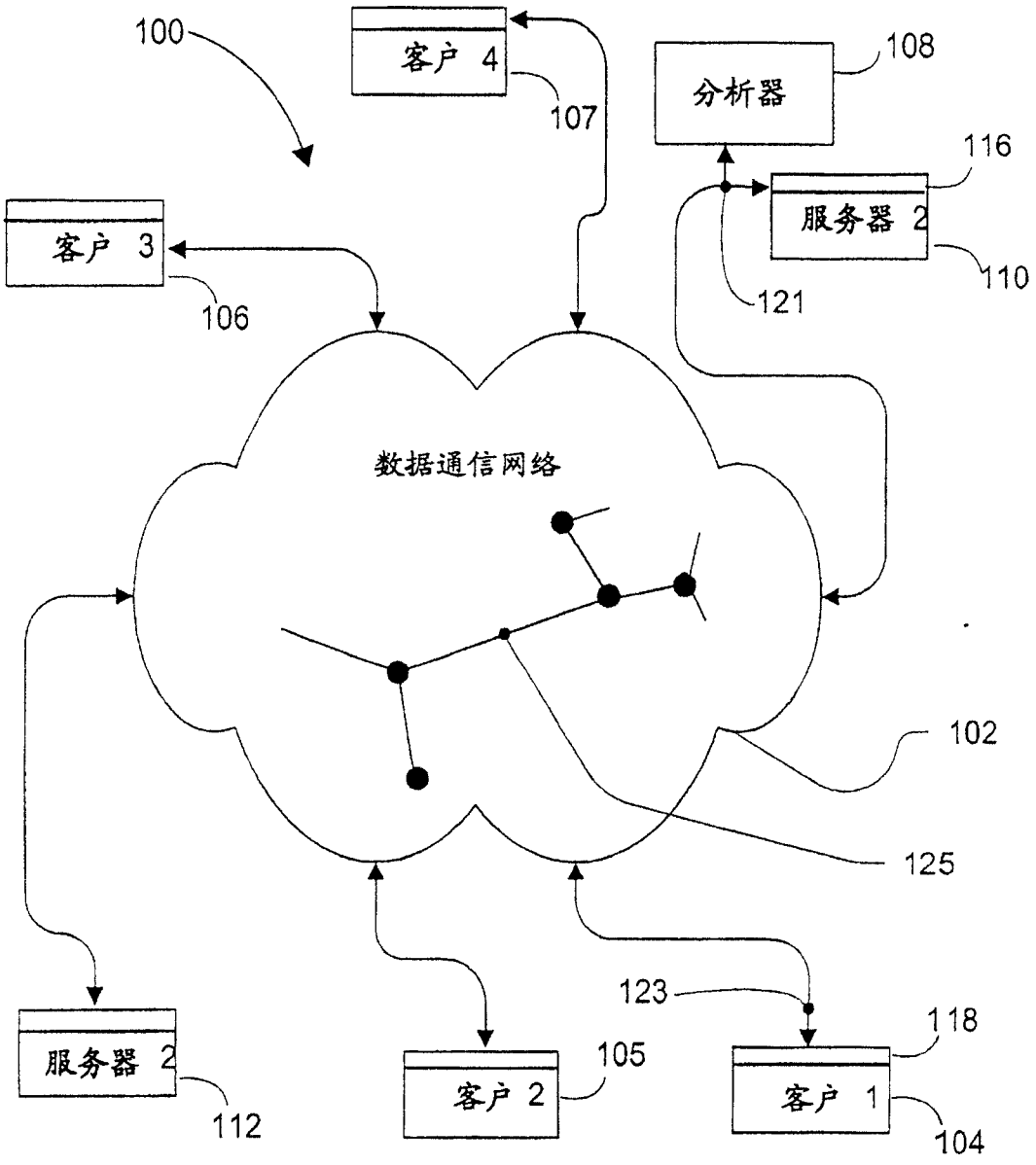

100
客户 4
107
分析器
108
116
服务器 2
110
121
客户 3
106
数据通信网络
102
125
123
118
服务器 2
112
客户 2
105
客户 1
104

图 1

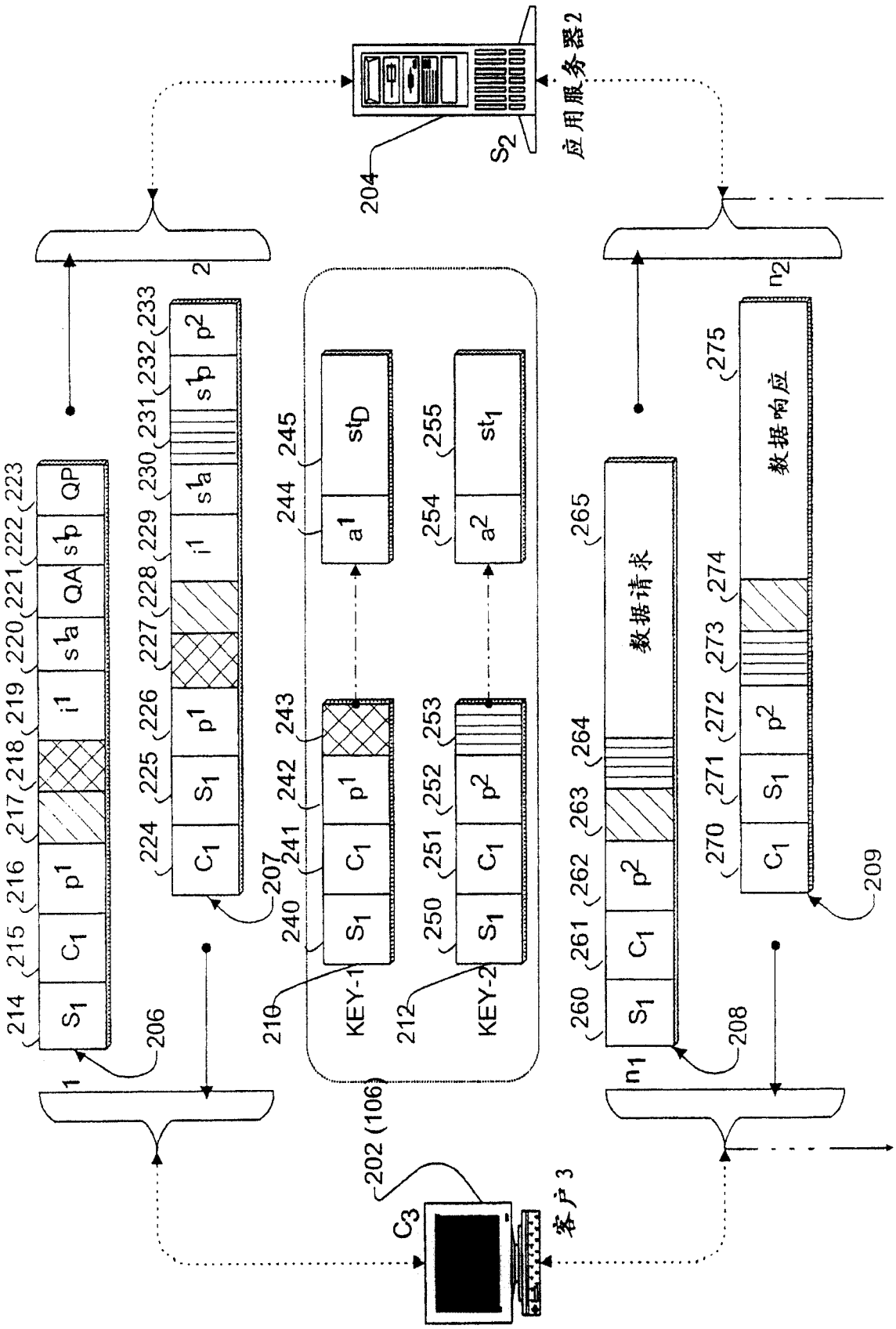

应用服务器2
S_2
204
2
1
n_2
n_1
客户3
C_3
202 (106)
KEY-1
KEY-2
210
212
206
207
208
209
数据请求
数据响应
214 215 216 217 218 219 220 221 222 223
224 225 226 227 228 229 230 231 232 233
240 241 242 243 244 245
250 251 252 253 254 255
260 261 262 263 264 265
270 271 272 273 274 275
S_1 C_1 p^1 i^1 s^1a QA s^1b QP
C_1 S_1 p^1 s^1a s^1b p^2
S_1 C_1 p^1 a^1 st_D
S_1 C_1 p^2 a^2 st_1
S_1 C_1 p^2 数据请求
C_1 S_1 p^2 数据响应

图 2

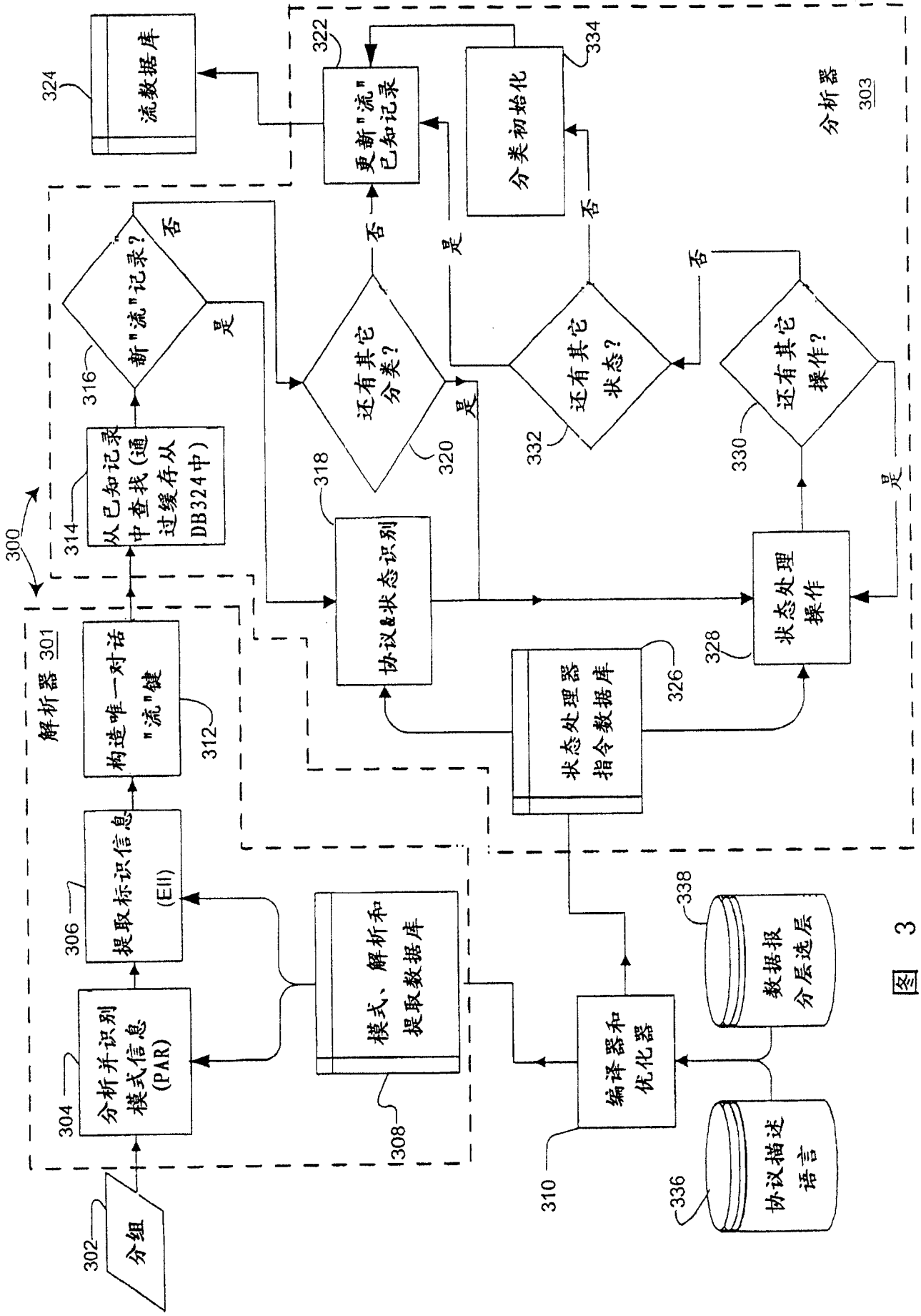

分组
302
分析并识别模式信息(PAR)
304
提取标识信息(EII)
306
模式、解析和提取数据库
308
编译器和优化器
310
协议描述语言
336
数据报分层选层
338
解析器 301
构造唯一对话"流"键
312
从已知记录中查找(通过缓存从DB324中)
314
新"流"记录?
316
协议&状态识别
318
还有其它分类?
320
更新"流"已知记录
322
流数据库
324
状态处理器指令数据库
326
状态处理操作
328
还有其它状态?
332
分类初始化
334
还有其它操作?
330
分析器
303
300
是
否

图 3

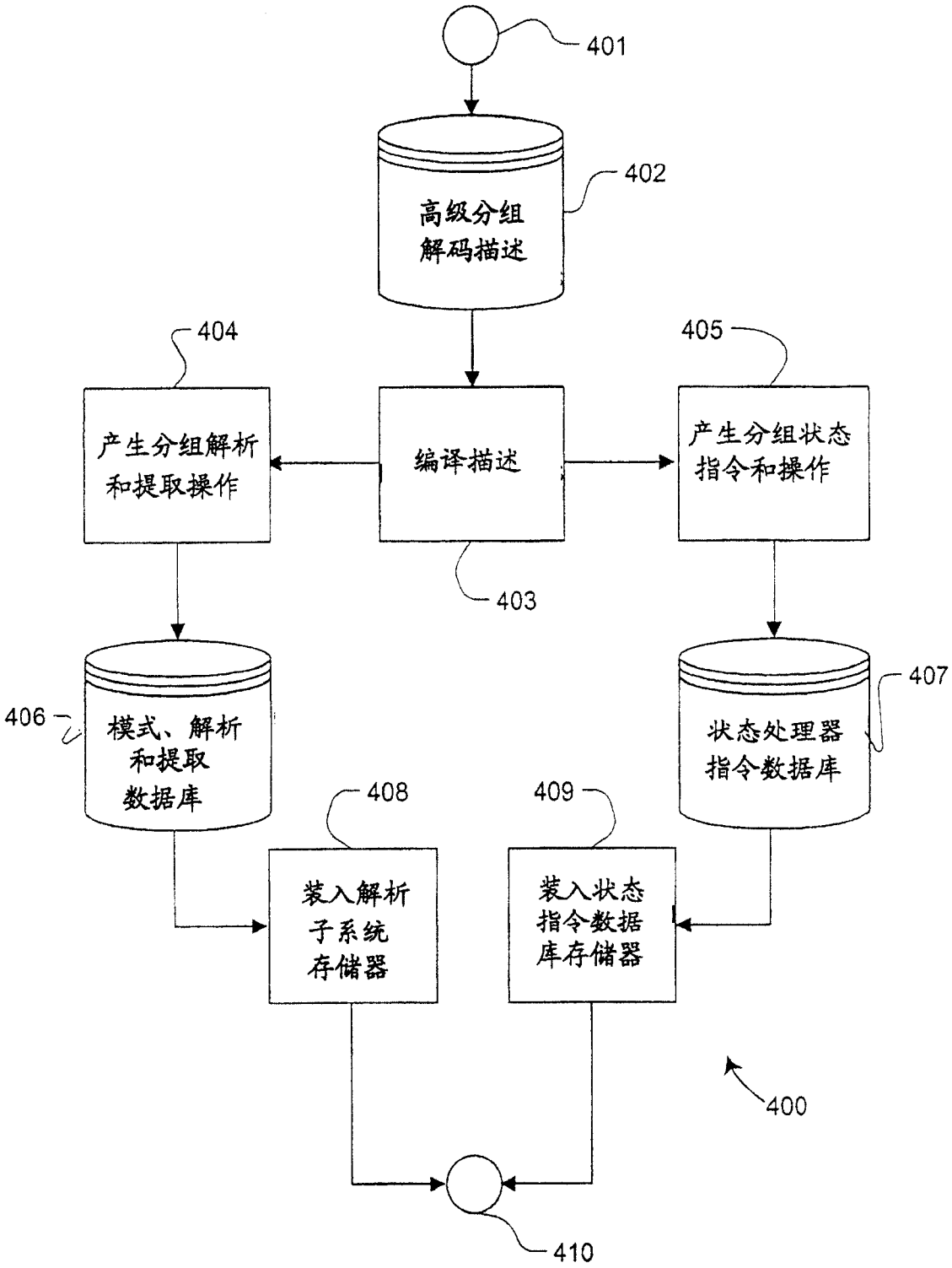

401
高级分组
解码描述
402
404
405
产生分组解析
和提取操作
编译描述
产生分组状态
指令和操作
403
406
模式、解析
和提取
数据库
407
状态处理器
指令数据库
408
409
装入解析
子系统
存储器
装入状态
指令数据
库存储器
400
410

图 4

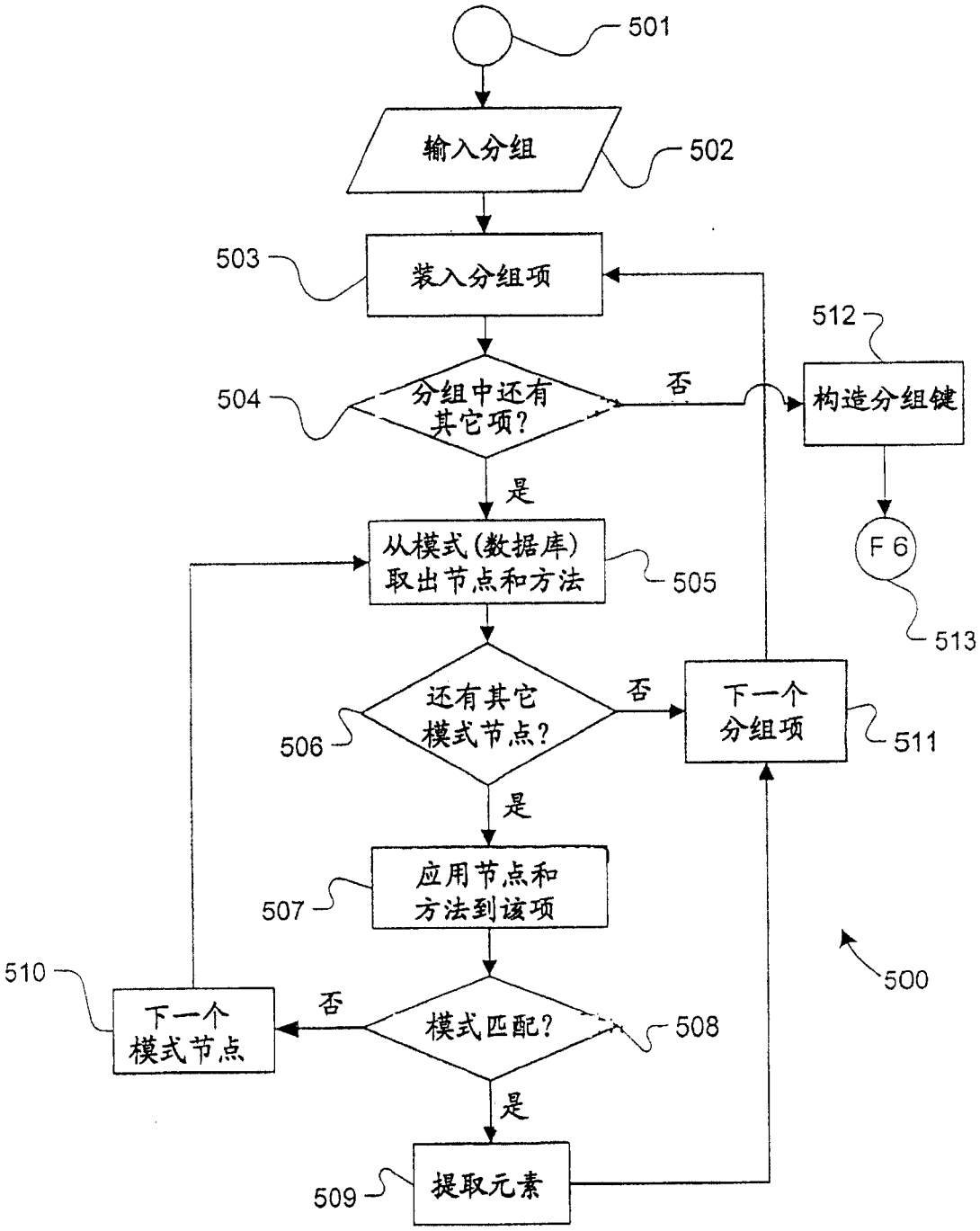

501
输入分组
502
503
装入分组项
512
504
分组中还有其它项?
否
构造分组键
是
F 6
513
从模式(数据库)取出节点和方法
505
还有其它模式节点?
506
否
下一个分组项
511
是
应用节点和方法到该项
507
500
510
下一个模式节点
否
模式匹配?
508
是
509
提取元素

图 5

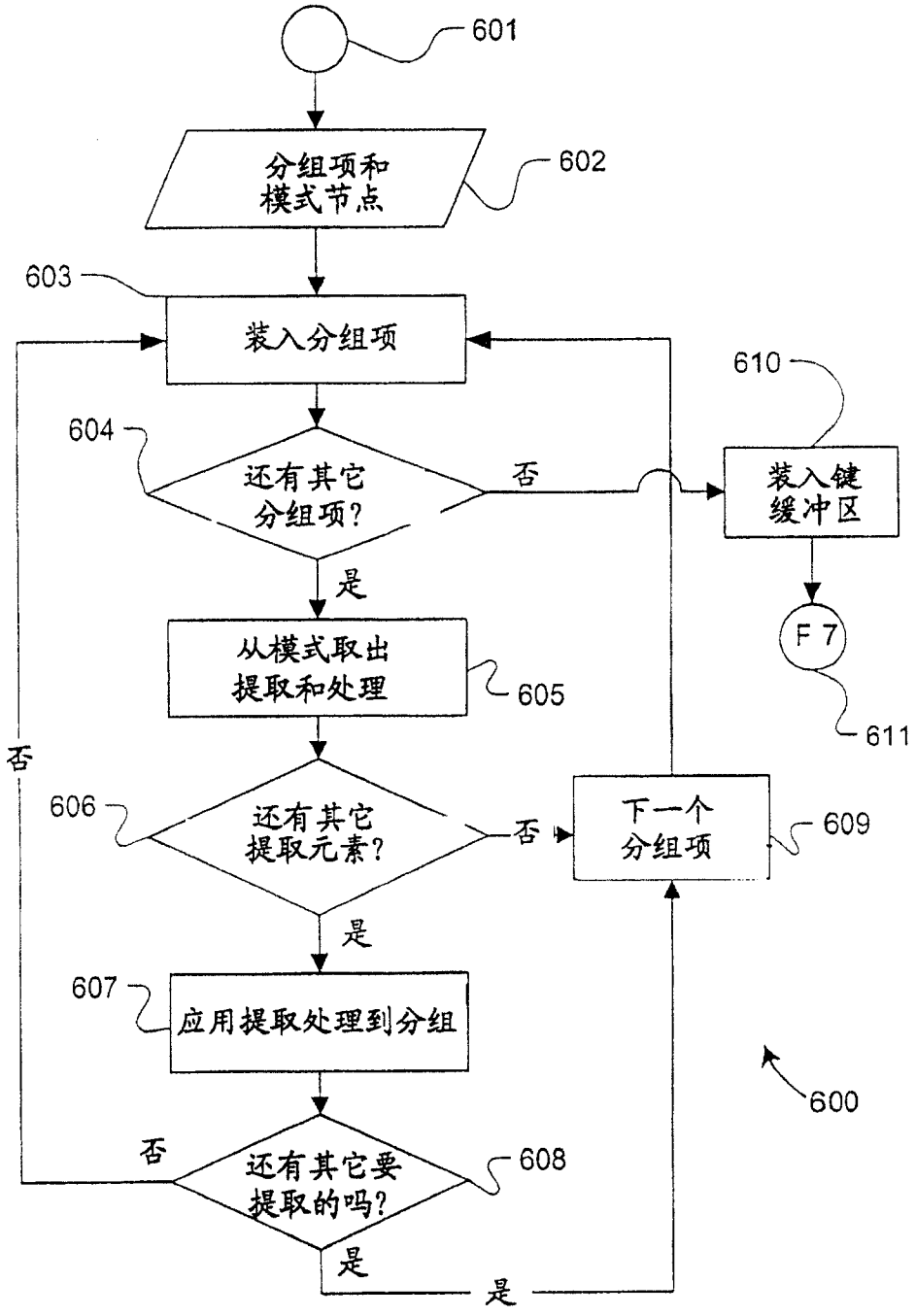

601
分组项和
模式节点
602
603
装入分组项
610
604
还有其它
分组项?
否
装入键
缓冲区
是
F 7
从模式取出
提取和处理
605
611
否
606
还有其它
提取元素?
否
下一个
分组项
609
是
607
应用提取处理到分组
600
否
还有其它要
提取的吗?
608
是
是

图 6

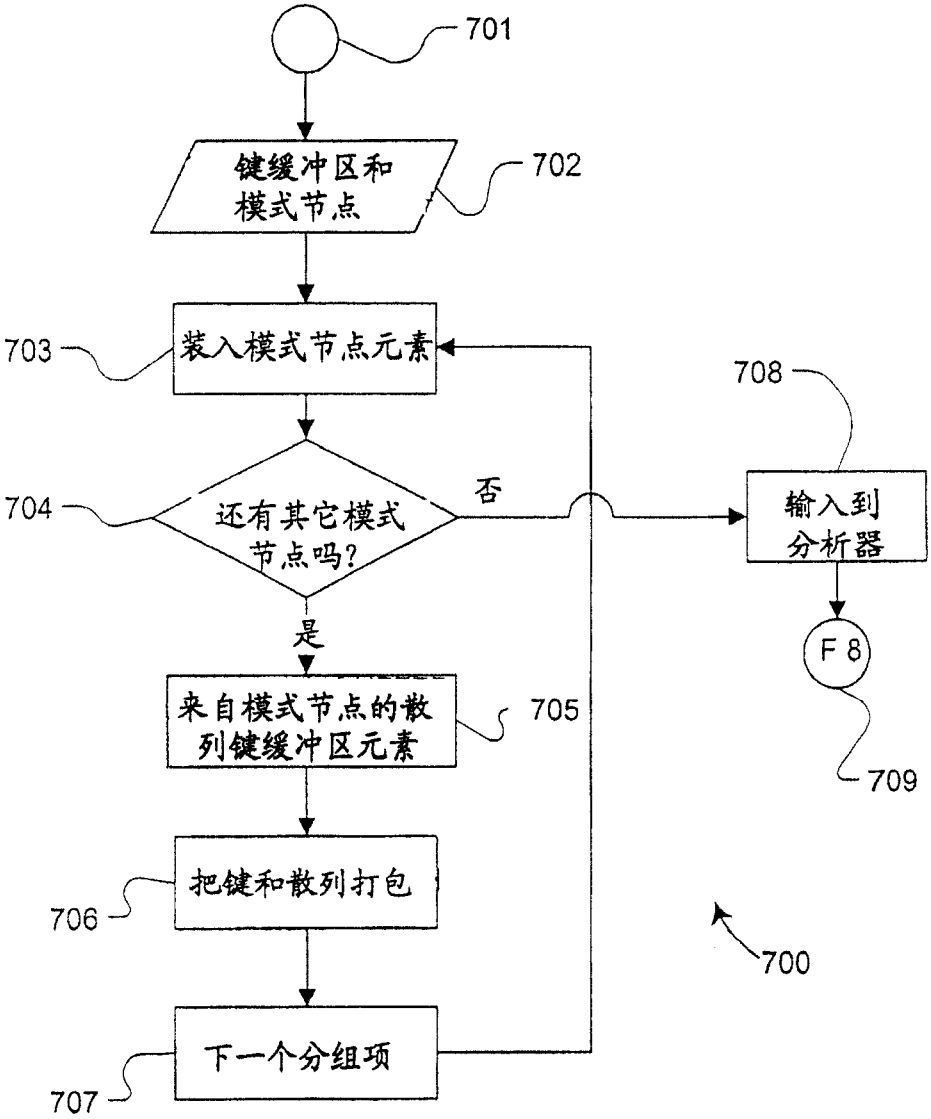

701
键缓冲区和
模式节点
702
703
装入模式节点元素
704
还有其它模式
节点吗?
否
是
708
输入到
分析器
F 8
709
来自模式节点的散
列键缓冲区元素
705
706
把键和散列打包
707
下一个分组项
700

图 7

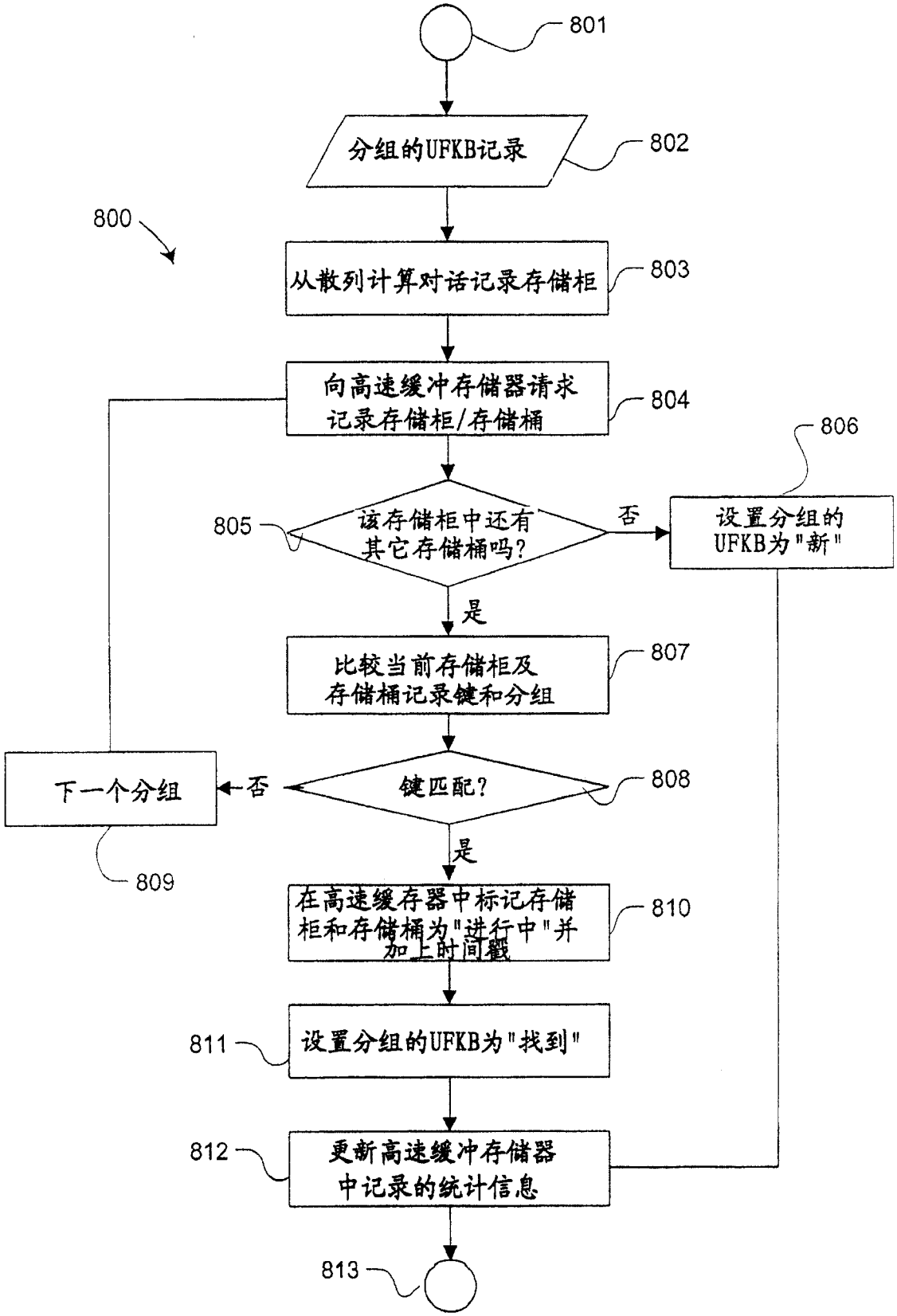

801
分组的UFKB记录
802
800
从散列计算对话记录存储柜
803
向高速缓冲存储器请求
记录存储柜/存储桶
804
806
805
该存储柜中还有
其它存储桶吗?
否
设置分组的
UFKB为"新"
是
比较当前存储柜及
存储桶记录键和分组
807
下一个分组
否
键匹配?
808
是
809
在高速缓存器中标记存储
柜和存储桶为"进行中"并
加上时间戳
810
811
设置分组的UFKB为"找到"
812
更新高速缓冲存储器
中记录的统计信息
813

图 8

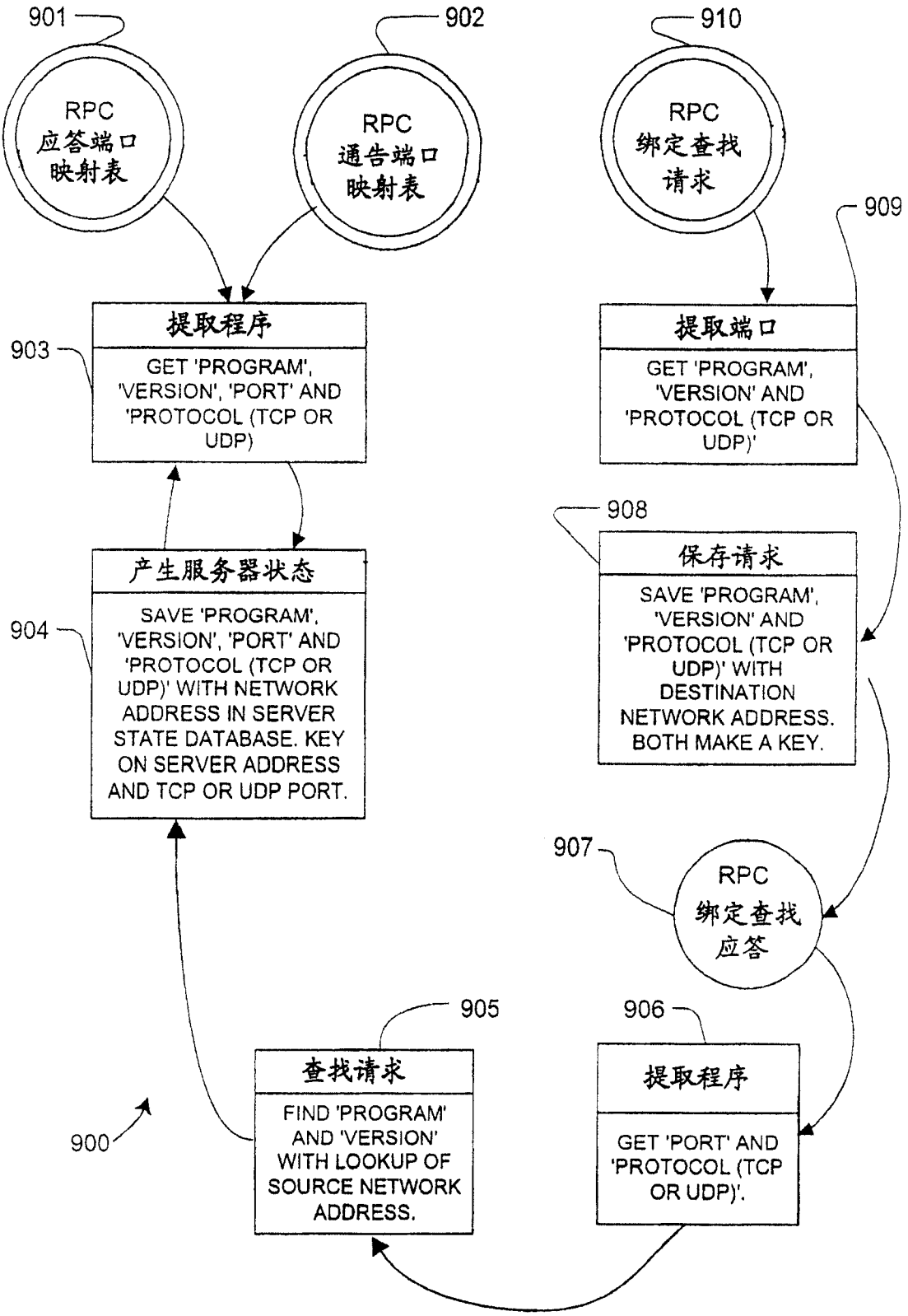

901
RPC
应答端口
映射表
902
RPC
通告端口
映射表
910
RPC
绑定查找
请求
909
提取程序
903
GET 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)
提取端口
GET 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)'
908
保存请求
SAVE 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)' WITH DESTINATION NETWORK ADDRESS. BOTH MAKE A KEY.
产生服务器状态
904
SAVE 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)' WITH NETWORK ADDRESS IN SERVER STATE DATABASE. KEY ON SERVER ADDRESS AND TCP OR UDP PORT.
907
RPC
绑定查找
应答
905
906
查找请求
FIND 'PROGRAM' AND 'VERSION' WITH LOOKUP OF SOURCE NETWORK ADDRESS.
提取程序
GET 'PORT' AND 'PROTOCOL (TCP OR UDP)'.
900

图 9

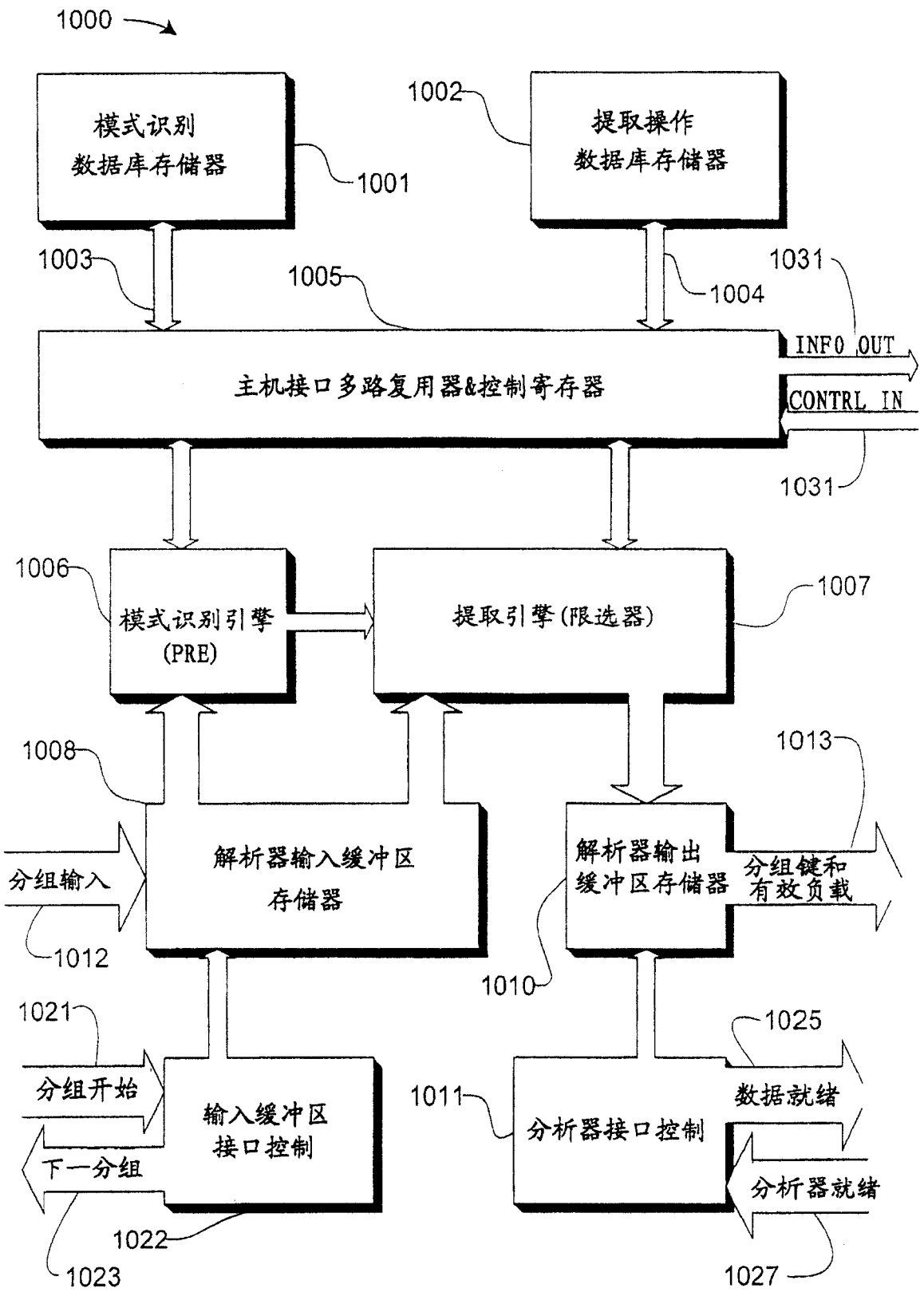

1000
模式识别
数据库存储器
1001
1002
提取操作
数据库存储器
1003
1005
1004
1031
INFO OUT
主机接口多路复用器&控制寄存器
CONTRL IN
1031
1006
模式识别引擎
(PRE)
提取引擎(限选器)
1007
1008
1013
分组输入
解析器输入缓冲区
存储器
解析器输出
缓冲区存储器
分组键和
有效负载
1012
1010
1021
1025
分组开始
输入缓冲区
接口控制
1011
分析器接口控制
数据就绪
下一分组
分析器就绪
1022
1023
1027

图 10

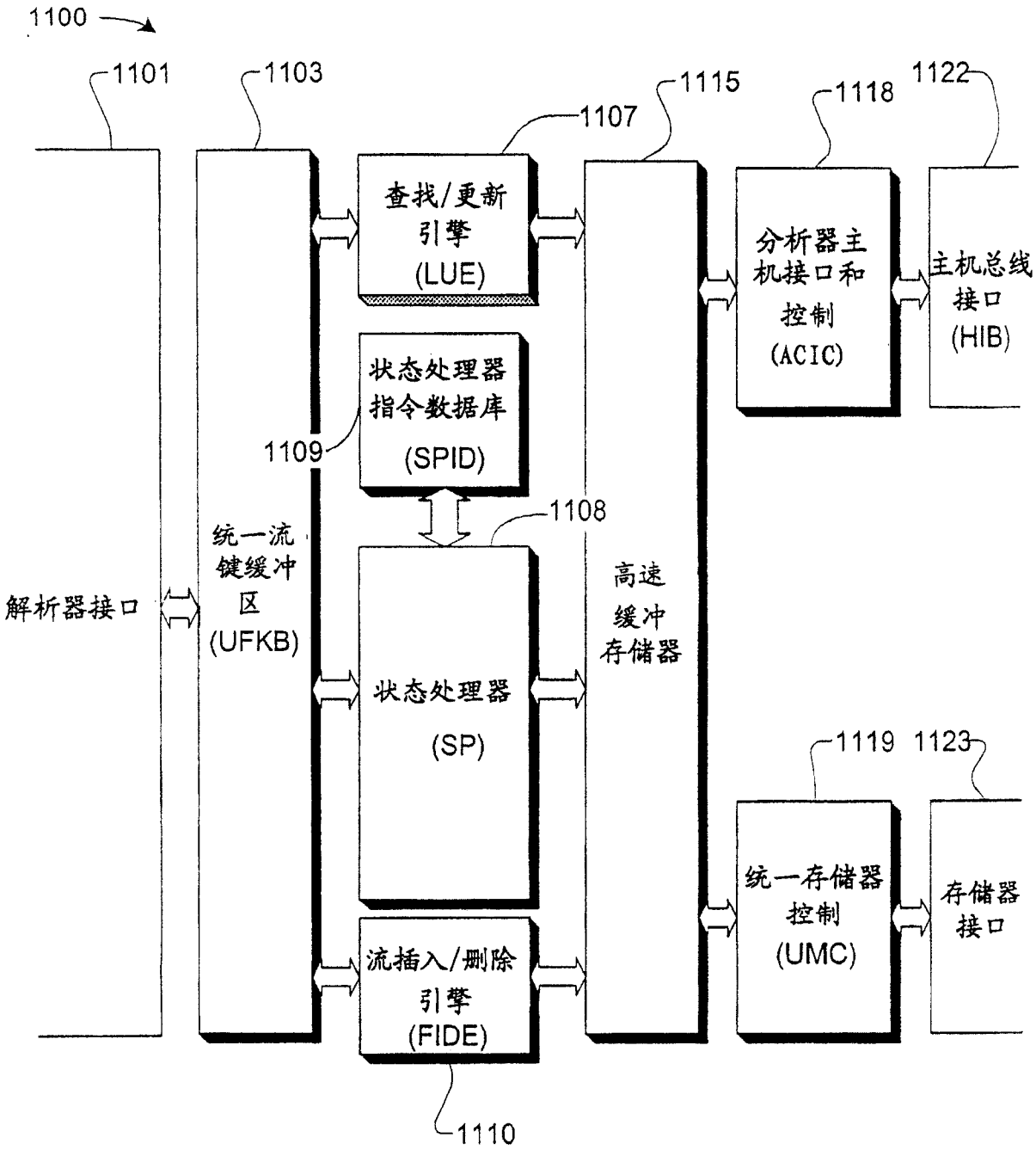

1100
1101
1103
1107
1115
1118
1122
查找/更新
引擎
(LUE)
分析器主
机接口和
控制
(ACIC)
主机总线
接口
(HIB)
状态处理器
指令数据库
(SPID)
1109
1108
解析器接口
统一流
键缓冲
区
(UFKB)
高速
缓冲
存储器
状态处理器
(SP)
1119
1123
统一存储器
控制
(UMC)
存储器
接口
流插入/删除
引擎
(FIDE)
1110

图 11

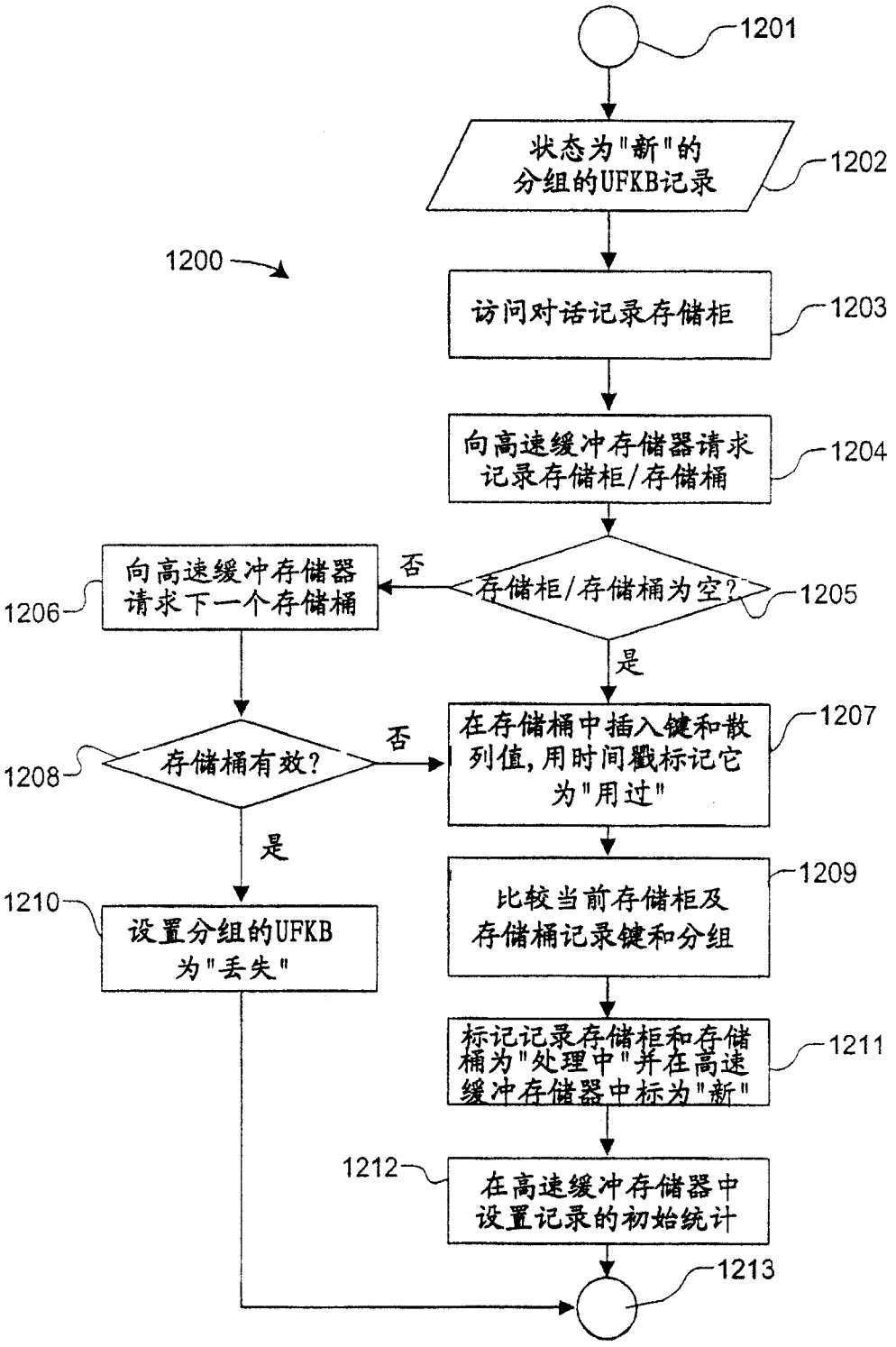

1200
1201
状态为"新"的分组的UFKB记录
1202
访问对话记录存储柜
1203
向高速缓冲存储器请求记录存储柜/存储桶
1204
存储柜/存储桶为空?
1205
否
向高速缓冲存储器请求下一个存储桶
1206
是
存储桶有效?
1208
否
在存储桶中插入键和散列值,用时间戳标记它为"用过"
1207
是
设置分组的UFKB为"丢失"
1210
比较当前存储柜及存储桶记录键和分组
1209
标记记录存储柜和存储桶为"处理中"并在高速缓冲存储器中标为"新"
1211
在高速缓冲存储器中设置记录的初始统计
1212
1213

图 12

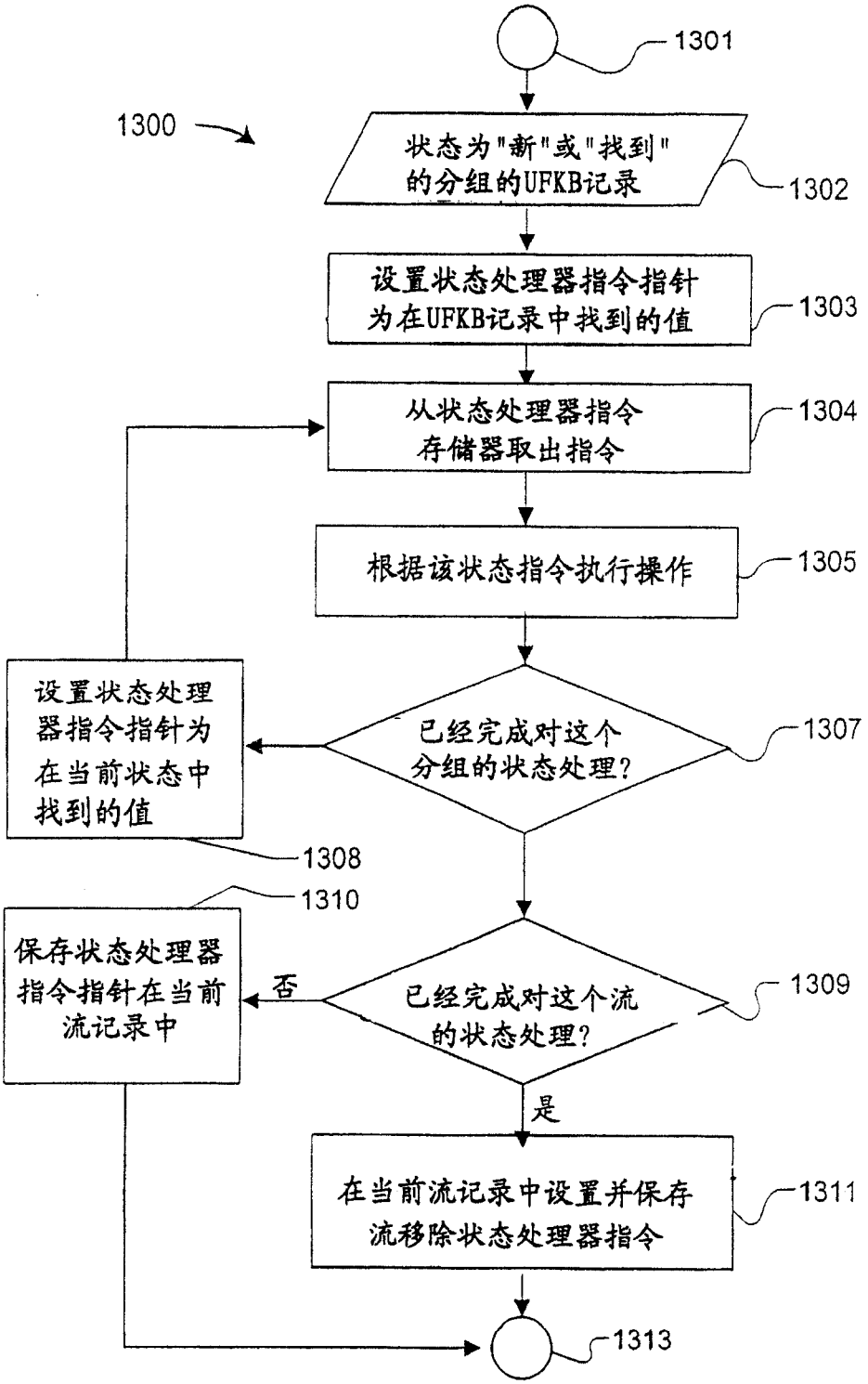

1300
1301
状态为"新"或"找到"的分组的UFKB记录
1302
设置状态处理器指令指针为在UFKB记录中找到的值
1303
从状态处理器指令存储器取出指令
1304
根据该状态指令执行操作
1305
已经完成对这个分组的状态处理?
1307
设置状态处理器指令指针为在当前状态中找到的值
1308
1310
保存状态处理器指令指针在当前流记录中
否
已经完成对这个流的状态处理?
1309
是
在当前流记录中设置并保存流移除状态处理器指令
1311
1313

图 13

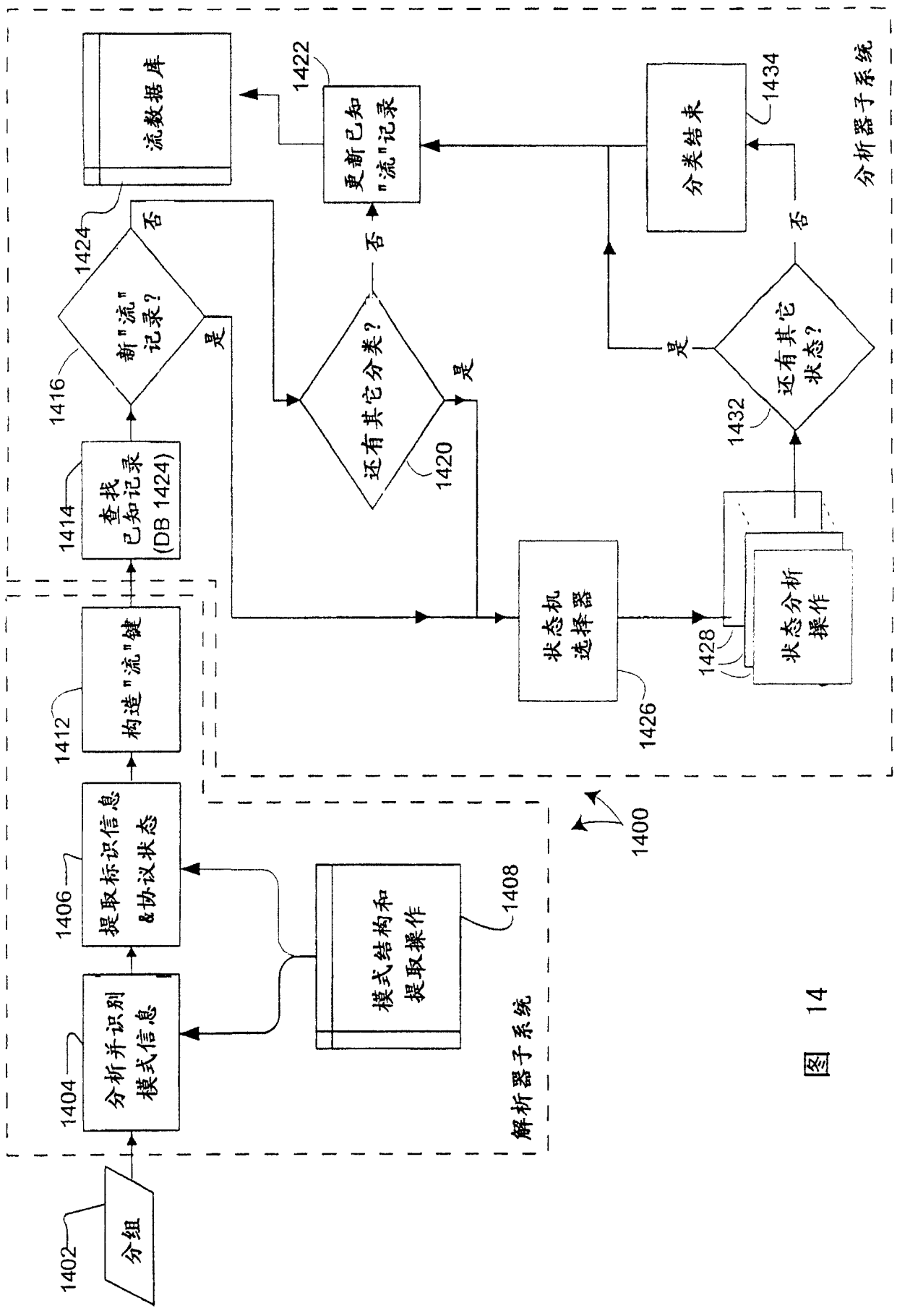

1402
分组
1404
分析并识别模式信息
1406
提取标识信息&协议状态
1408
模式结构和提取操作
解析器子系统
1400
1412
构造"流"键
1414
查找已知记录(DB 1424)
1416
新"流"记录?
是
否
1424
流数据库
1420
还有其它分类?
是
否
1422
更新已知"流"记录
状态机选择器
1426
1428
状态分析操作
1432
还有其它状态?
是
否
1434
分类结束
分析器子系统

图 14

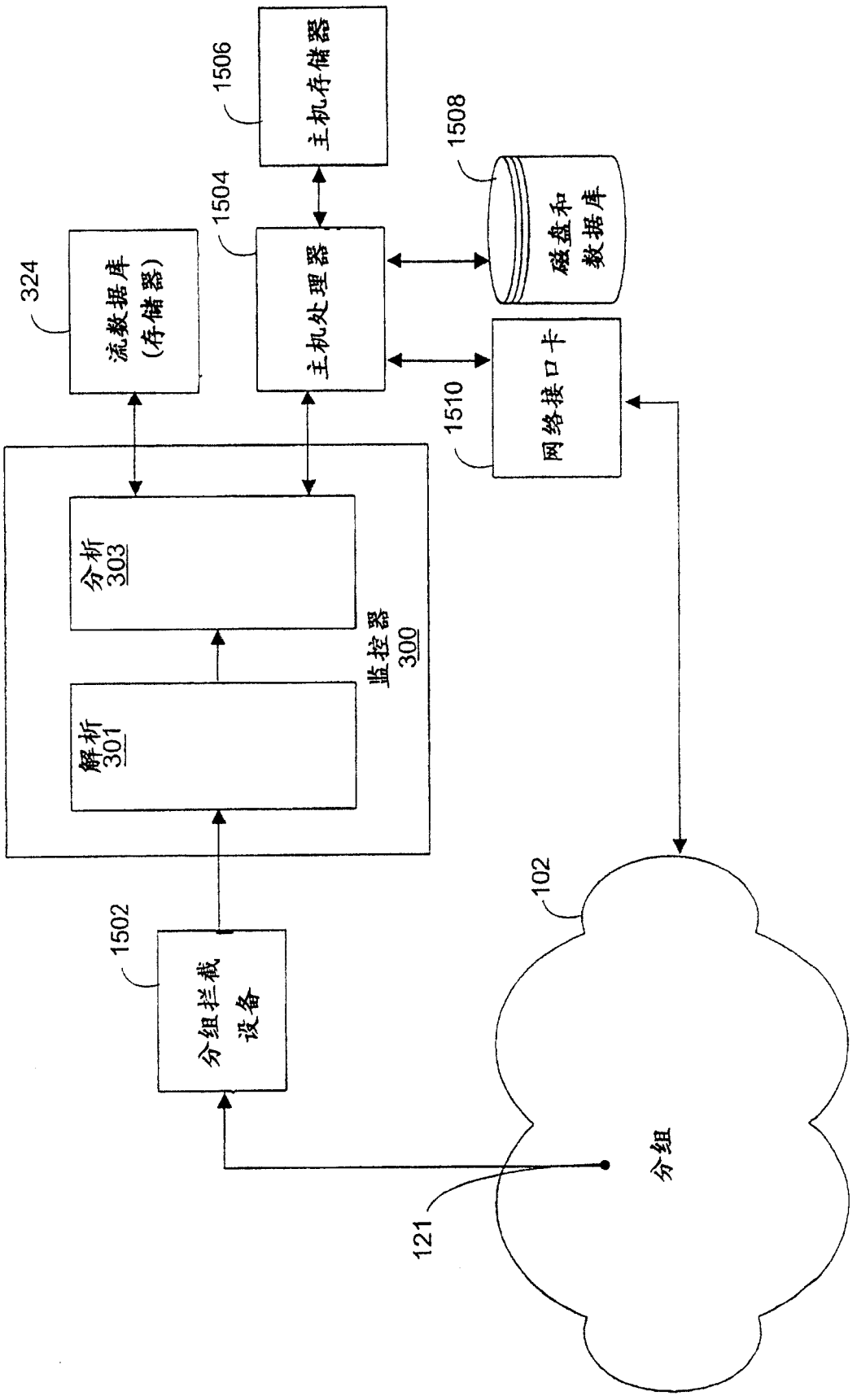

主机存储器
1506
1508
磁盘和数据库
1504
主机处理器
流数据库(存储器)
324
网络接口卡
1510
分析303
解析301
监控器300
分组拦截设备
1502
102
分组
121

图 15

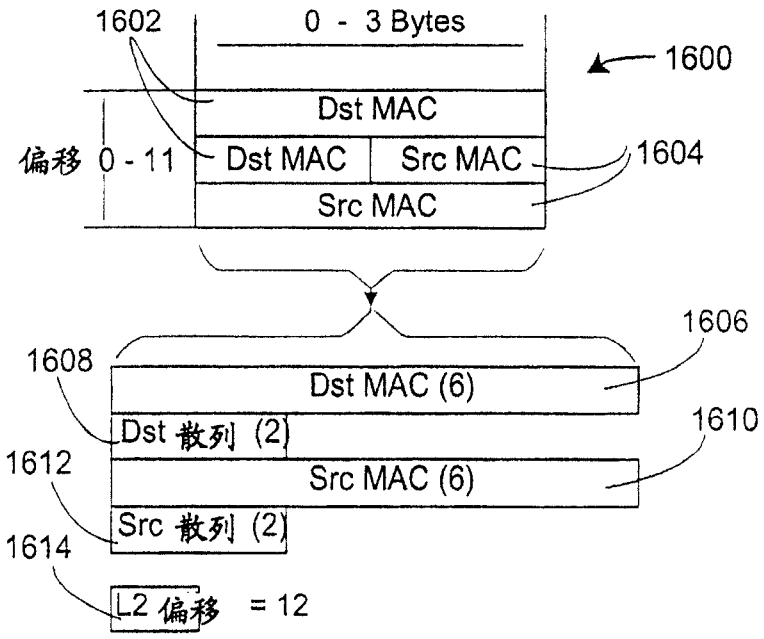

1602
0 - 3 Bytes
1600
Dst MAC
偏移 0 - 11
Dst MAC
Src MAC
1604
Src MAC
1606
1608
Dst MAC (6)
Dst 散列 (2)
1610
1612
Src MAC (6)
Src 散列 (2)
1614
L2 偏移 = 12

图 16

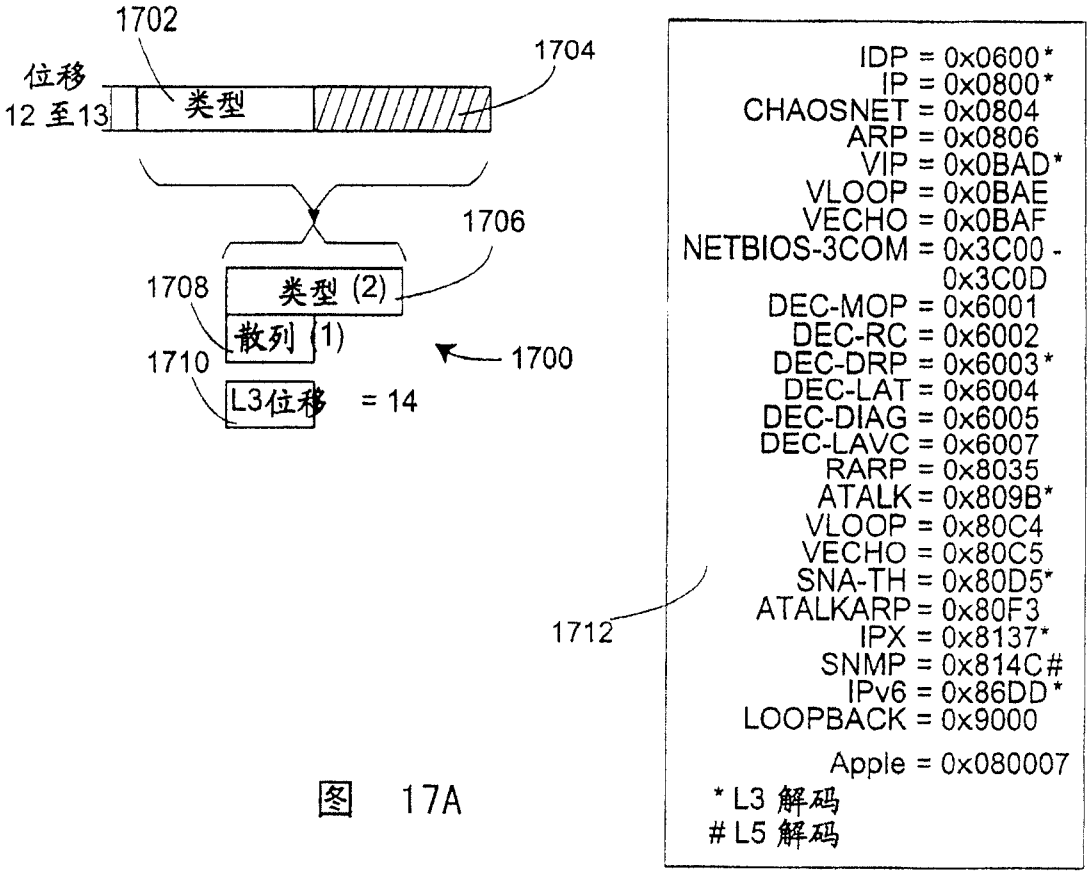

1702
位移
12 至13
类型
1704
1706
1708
类型 (2)
散列 (1)
1710
L3位移 = 14
1700
1712
IDP = 0x0600*
IP = 0x0800*
CHAOSNET = 0x0804
ARP = 0x0806
VIP = 0x0BAD*
VLOOP = 0x0BAE
VECHO = 0x0BAF
NETBIOS-3COM = 0x3C00 - 0x3C0D
DEC-MOP = 0x6001
DEC-RC = 0x6002
DEC-DRP = 0x6003*
DEC-LAT = 0x6004
DEC-DIAG = 0x6005
DEC-LAVC = 0x6007
RARP = 0x8035
ATALK = 0x809B*
VLOOP = 0x80C4
VECHO = 0x80C5
SNA-TH = 0x80D5*
ATALKARP = 0x80F3
IPX = 0x8137*
SNMP = 0x814C#
IPv6 = 0x86DD*
LOOPBACK = 0x9000
Apple = 0x080007
* L3 解码
# L5 解码

图 17A

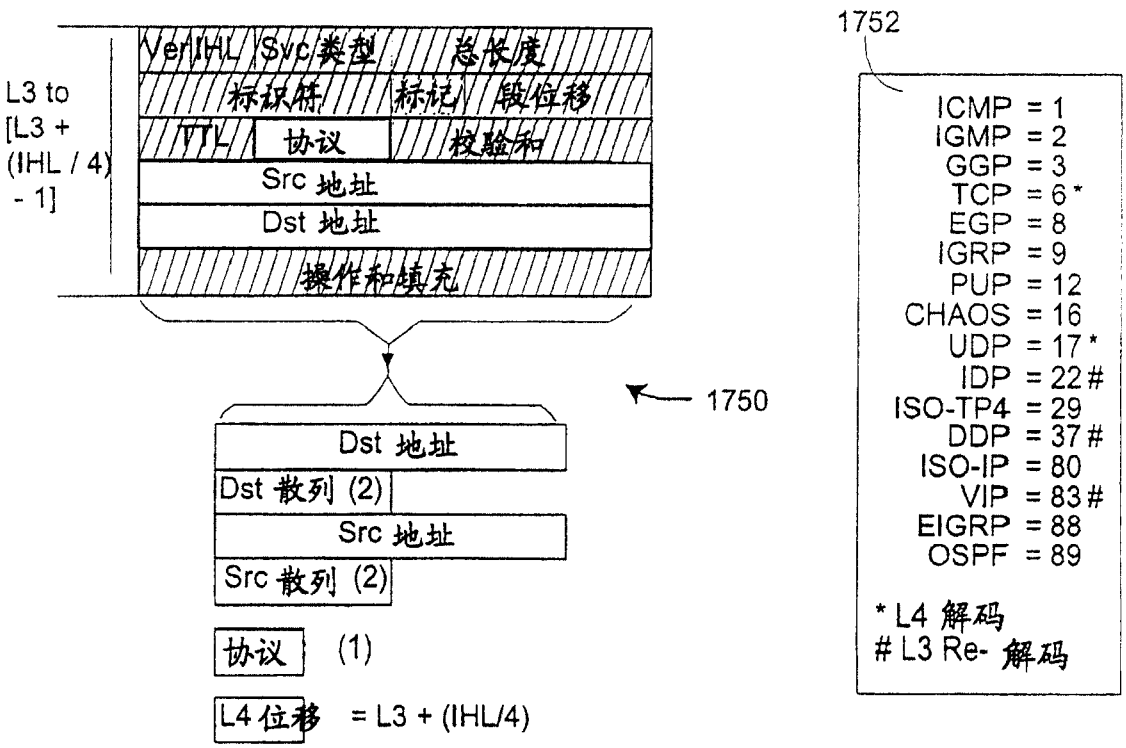

L3 to [L3 + (IHL / 4) - 1]
Ver/IHL
Svc 类型
总长度
标识符
标记
段位移
TTL
协议
校验和
Src 地址
Dst 地址
操作和填充
1750
Dst 地址
Dst 散列 (2)
Src 地址
Src 散列 (2)
协议 (1)
L4 位移 = L3 + (IHL/4)
1752
ICMP = 1
IGMP = 2
GGP = 3
TCP = 6*
EGP = 8
IGRP = 9
PUP = 12
CHAOS = 16
UDP = 17*
IDP = 22#
ISO-TP4 = 29
DDP = 37#
ISO-IP = 80
VIP = 83#
EIGRP = 88
OSPF = 89
* L4 解码
# L3 Re- 解码

图 17B

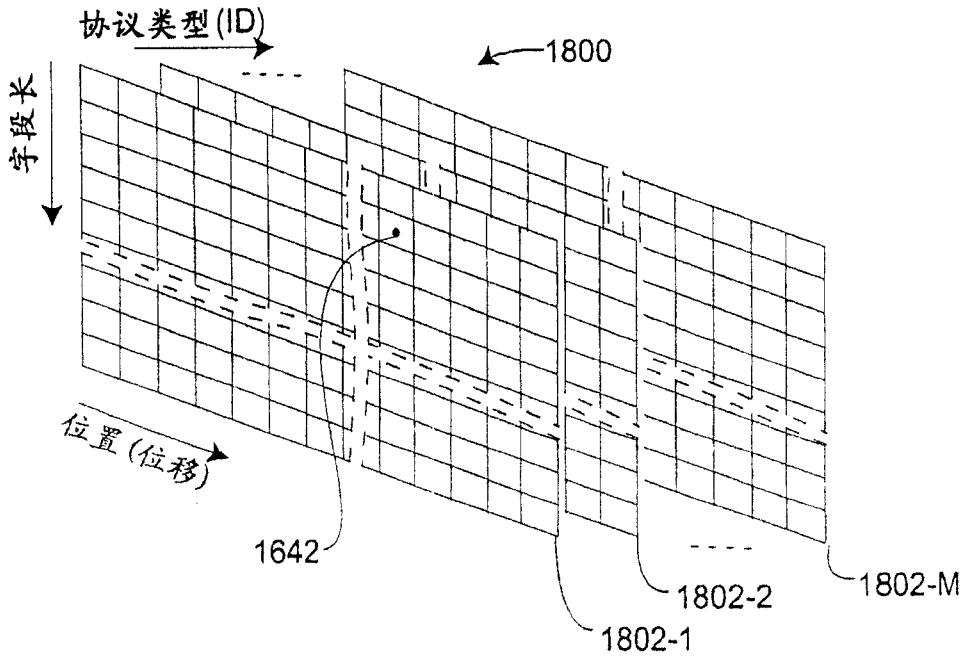

协议类型(ID)
1800
字段长
位置(位移)
1642
1802-1
1802-2
1802-M

图 18A

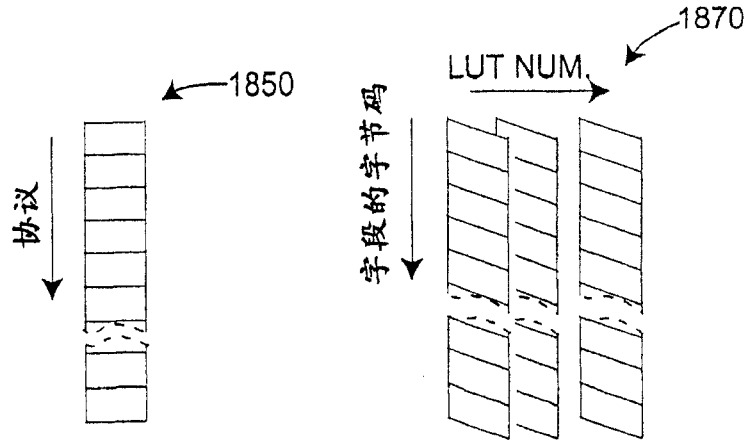

1870
LUT NUM.
1850
协议
字段的字节码

图 18B

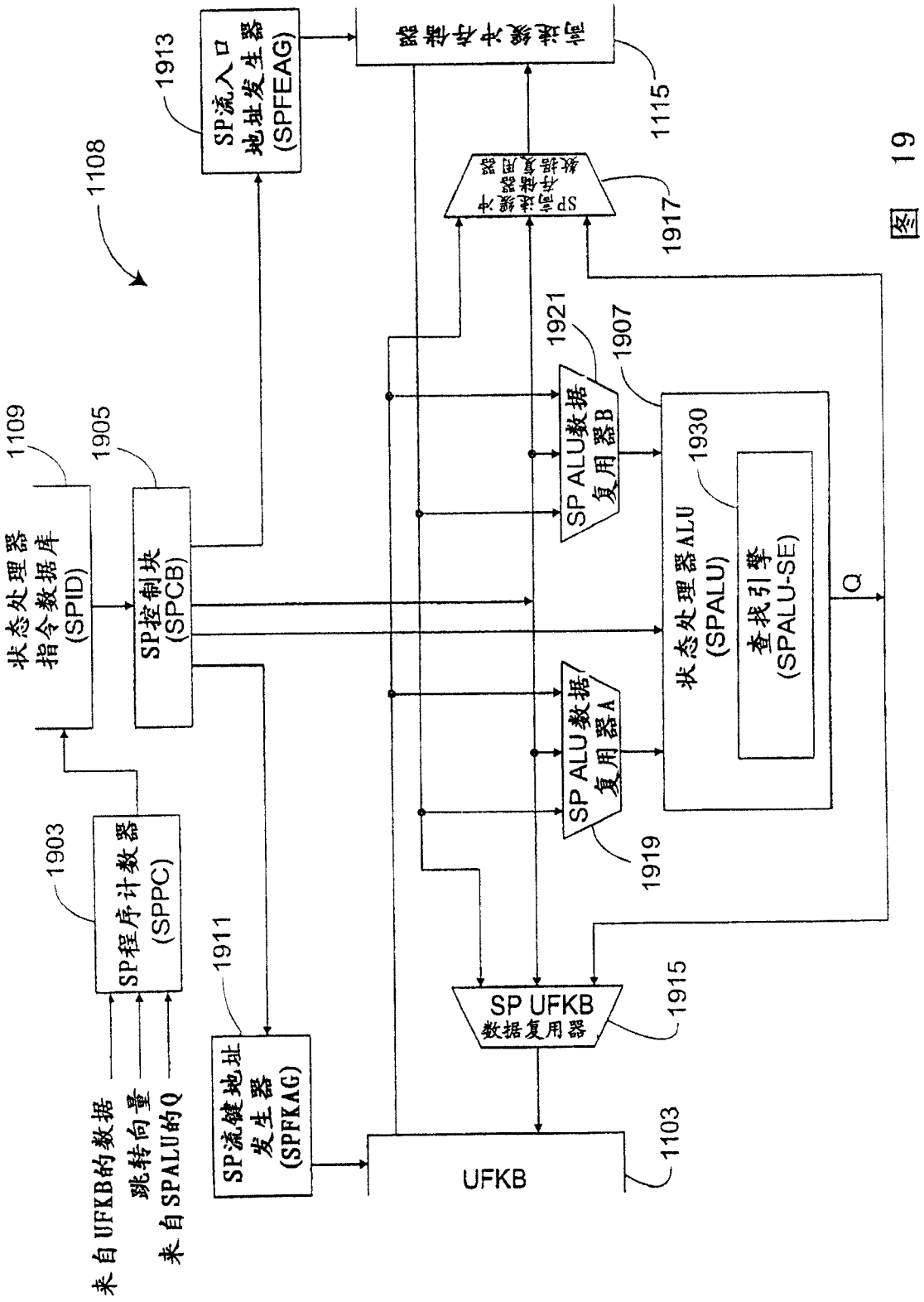

1108
1913
SP流入口地址发生器(SPFEAG)
1115
1917
1921
1907
1109
1905
状态处理器指令数据库(SPID)
SP控制块(SPCB)
SP ALU数据复用器B
1930
状态处理器ALU(SPALU)
查找引擎(SPALU-SE)
Q
1903
SP程序计数器(SPPC)
1911
SP流键地址发生器(SPFKAG)
SP ALU数据复用器A
1919
SP UFKB数据复用器
1915
来自UFKB的数据
跳转向量
来自SPALU的Q
UFKB
1103

图 19

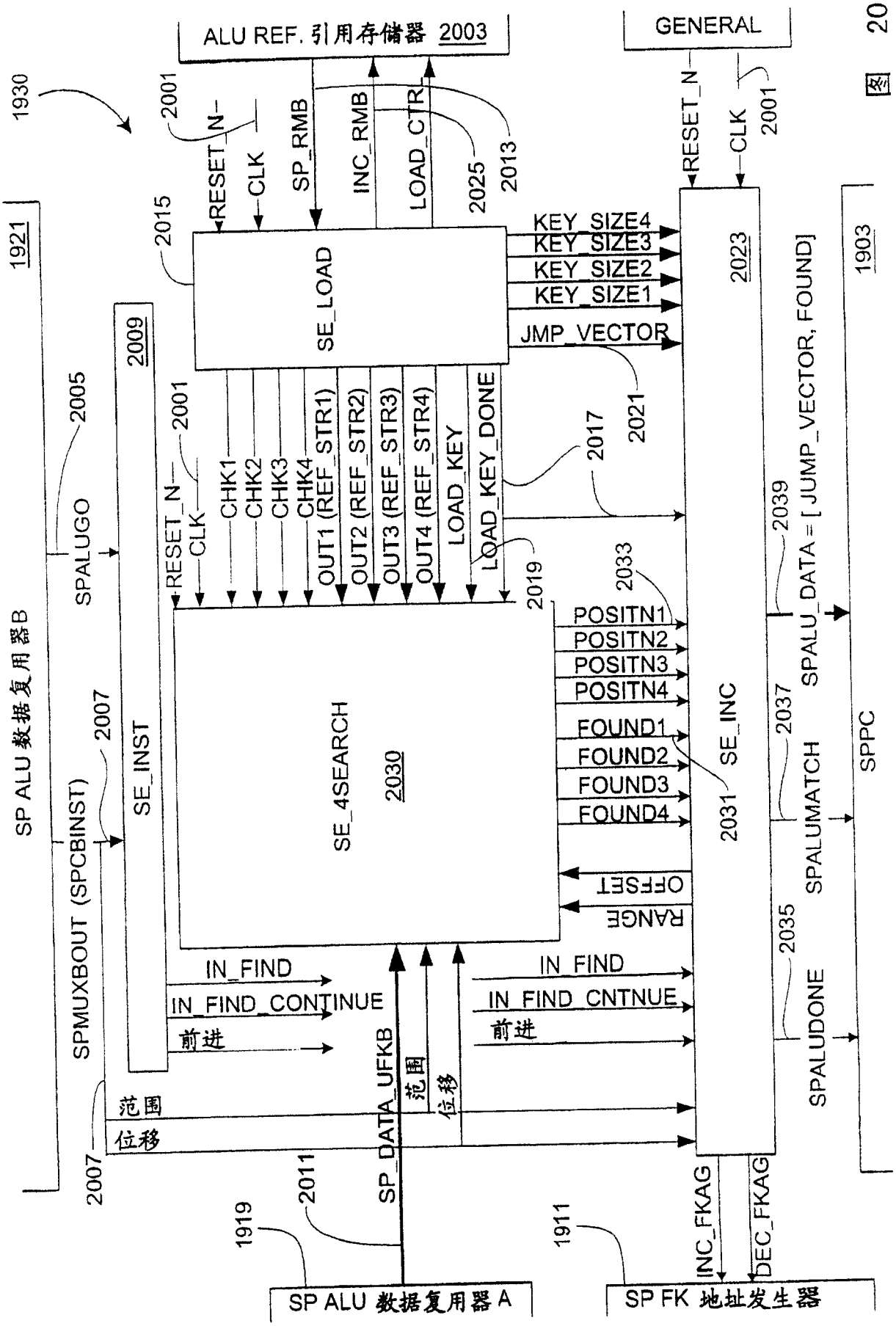

ALU REF. 引用存储器 2003
GENERAL
1930
2001
RESET_N
CLK
SP_RMB
INC_RMB
LOAD_CTR
2025
2013
2015
SE_LOAD
KEY_SIZE4
KEY_SIZE3
KEY_SIZE2
KEY_SIZE1
JMP_VECTOR
2021
2023
1921
1903
2009
2005
SPALUGO
CHK1
CHK2
CHK3
CHK4
OUT1 (REF_STR1)
OUT2 (REF_STR2)
OUT3 (REF_STR3)
OUT4 (REF_STR4)
LOAD_KEY
LOAD_KEY_DONE
2017
2019
2033
2039
SPALU_DATA = [JUMP_VECTOR, FOUND]
POSITN1
POSITN2
POSITN3
POSITN4
FOUND1
FOUND2
FOUND3
FOUND4
SE_4SEARCH
2030
SE_INC
2031
2037
SPALUMATCH
SPPC
OFFSET
RANGE
2035
SPALUDONE
SP ALU 数据复用器B
SPMUXBOUT (SPCBINST)
2007
SE_INST
IN_FIND
IN_FIND_CONTINUE
前进
范围
位移
SP_DATA_UFKB
2011
1919
SP ALU 数据复用器A
1911
INC_FKAG
DEC_FKAG
SP FK 地址发生器

图 20

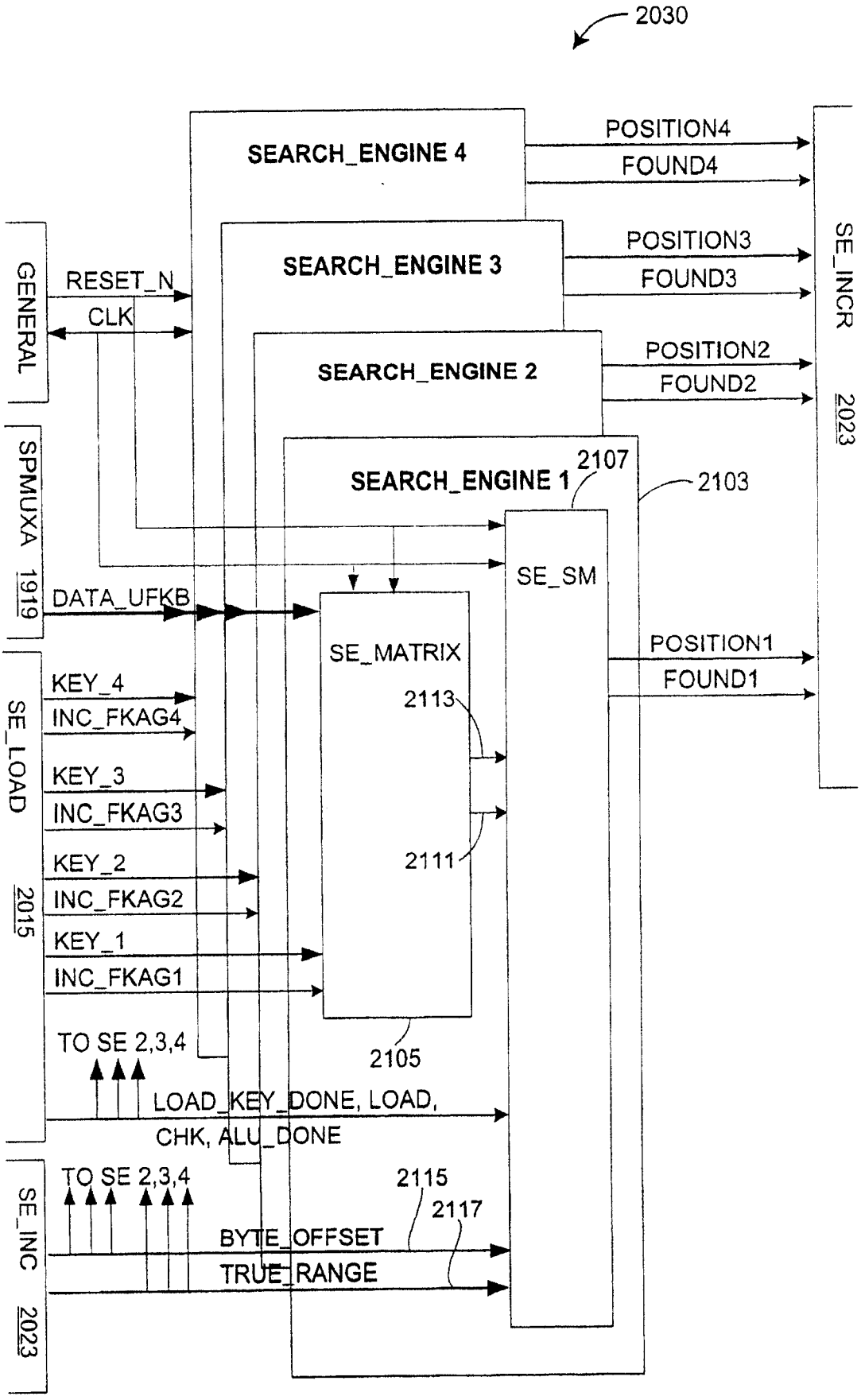

2030
SEARCH_ENGINE 4
SEARCH_ENGINE 3
SEARCH_ENGINE 2
SEARCH_ENGINE 1
2107
2103
SE_SM
SE_MATRIX
2113
2111
2105
POSITION4
FOUND4
POSITION3
FOUND3
POSITION2
FOUND2
POSITION1
FOUND1
SE_INCR 2023
GENERAL
RESET_N
CLK
SPMUXA 1919
DATA_UFKB
SE_LOAD 2015
KEY_4
INC_FKAG4
KEY_3
INC_FKAG3
KEY_2
INC_FKAG2
KEY_1
INC_FKAG1
TO SE 2,3,4
LOAD_KEY_DONE, LOAD,
CHK, ALU_DONE
SE_INC 2023
TO SE 2,3,4
2115
2117
BYTE_OFFSET
TRUE_RANGE

图 21

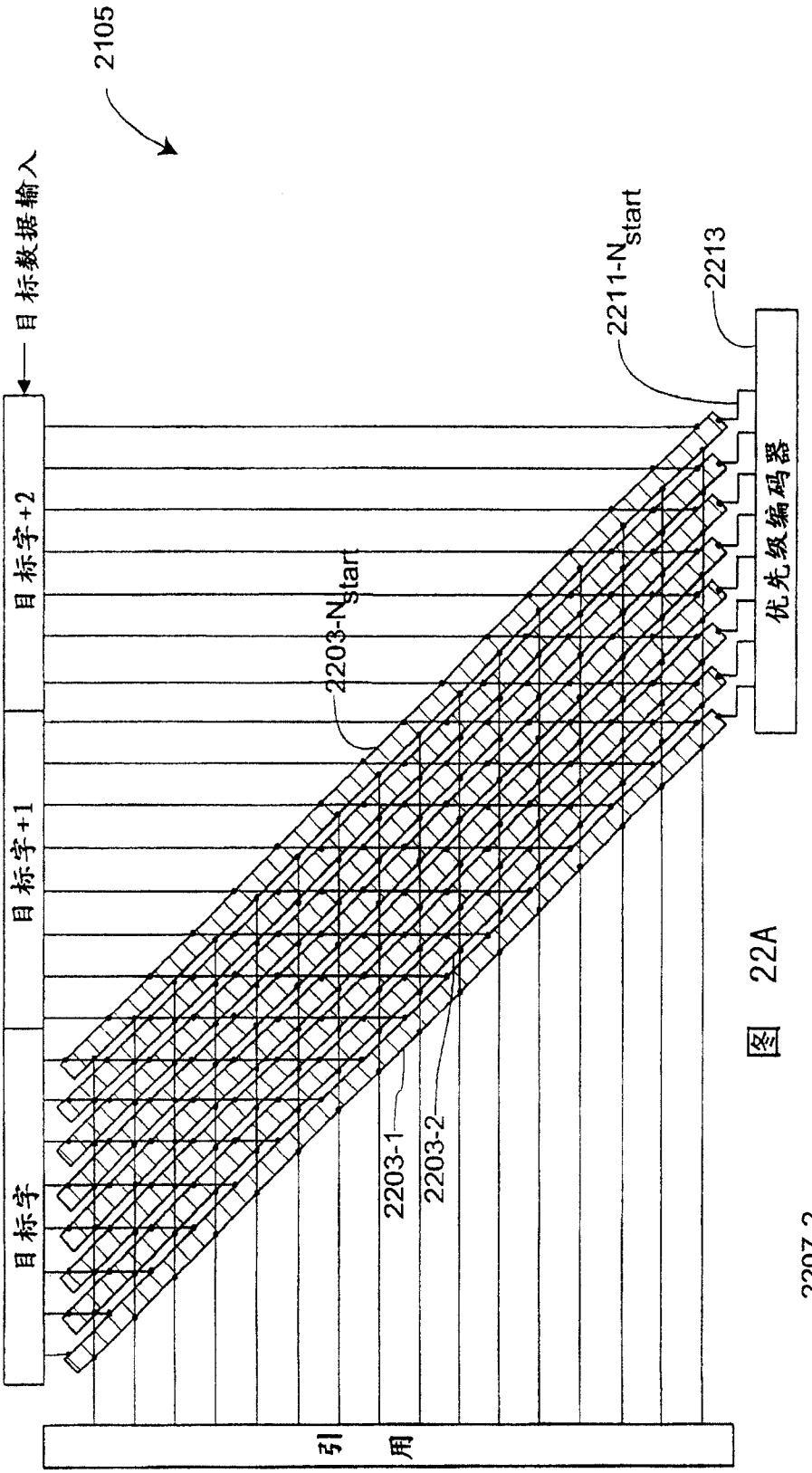

2105
目标数据输入
目标字
目标字+1
目标字+2
2203-Nstart
2203-1
2203-2
2211-Nstart
2213
优先级编码器
引用

图 22A

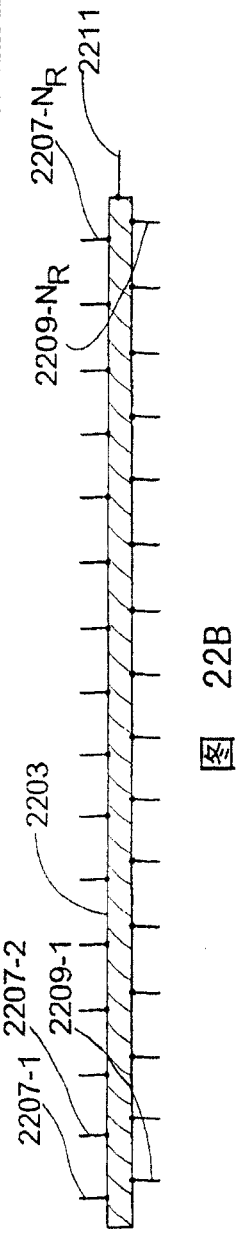

2207-NR
2211
2209-NR
2203
2207-2
2209-1
2207-1

图 22B

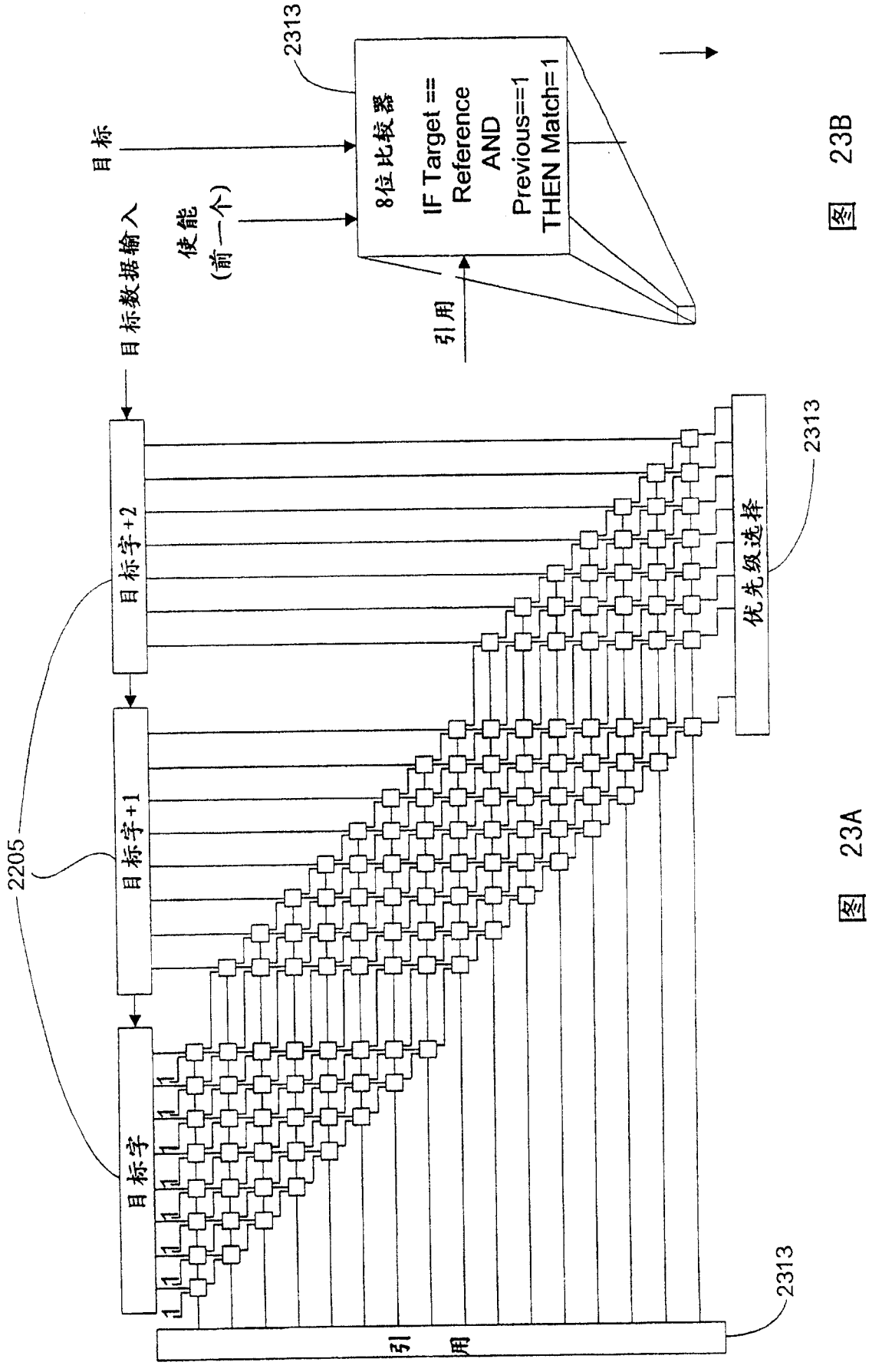

2313
8位比较器
IF Target ==
Reference
AND
Previous==1
THEN Match=1
目标
使能
(前一个)
引用
目标数据输入
目标字+2
目标字+1
目标字
2205
优先级选择
2313
引用
2313

图 23B

图 23A

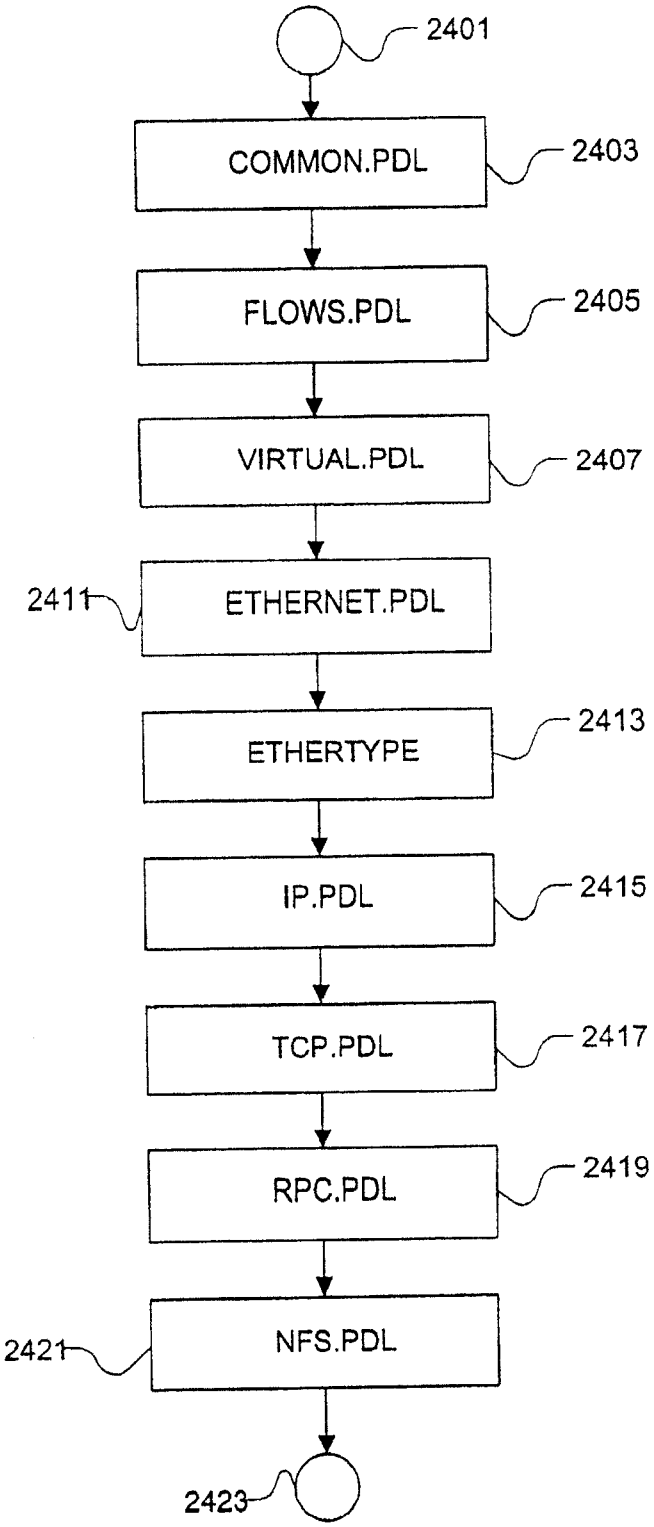

2401
COMMON.PDL
2403
FLOWS.PDL
2405
VIRTUAL.PDL
2407
2411
ETHERNET.PDL
ETHERTYPE
2413
IP.PDL
2415
TCP.PDL
2417
RPC.PDL
2419
2421
NFS.PDL
2423

图 24

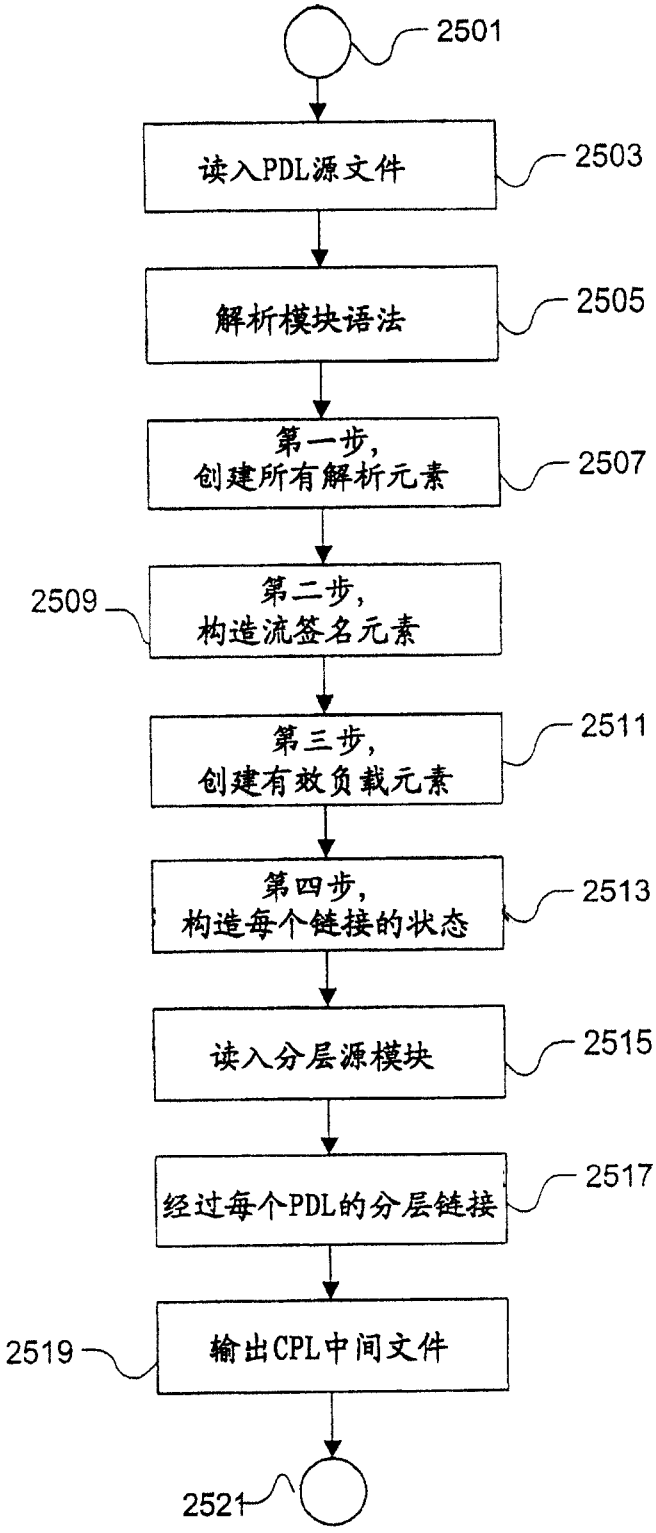

2501
读入PDL源文件
2503
解析模块语法
2505
第一步，
创建所有解析元素
2507
2509
第二步，
构造流签名元素
第三步，
创建有效负载元素
2511
第四步，
构造每个链接的状态
2513
读入分层源模块
2515
经过每个PDL的分层链接
2517
2519
输出CPL中间文件
2521

图 25

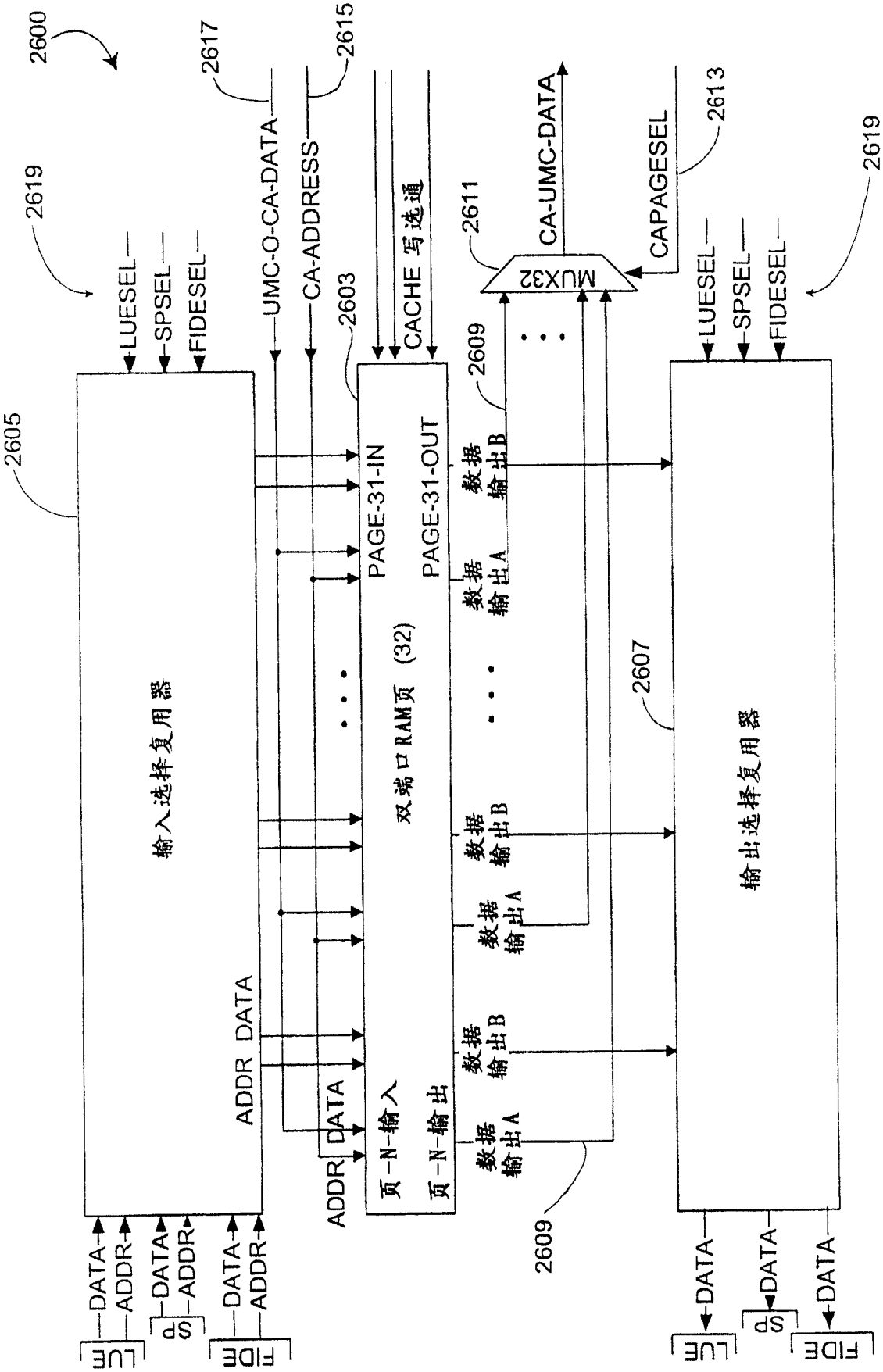

2600
2619
2617
2615
2605
2603
2609
2611
2613
2607
LUESEL
SPSEL
FIDESEL
UMC-O-CA-DATA
CA-ADDRESS
CACHE 写选通
CA-UMC-DATA
CAPAGESEL
输入选择复用器
双端口RAM页 (32)
PAGE-31-IN
PAGE-31-OUT
页-N-输入
页-N-输出
ADDR DATA
ADDR DATA
数据输出A
数据输出B
2:1 X32
输出选择复用器
DATA
ADDR

图 26

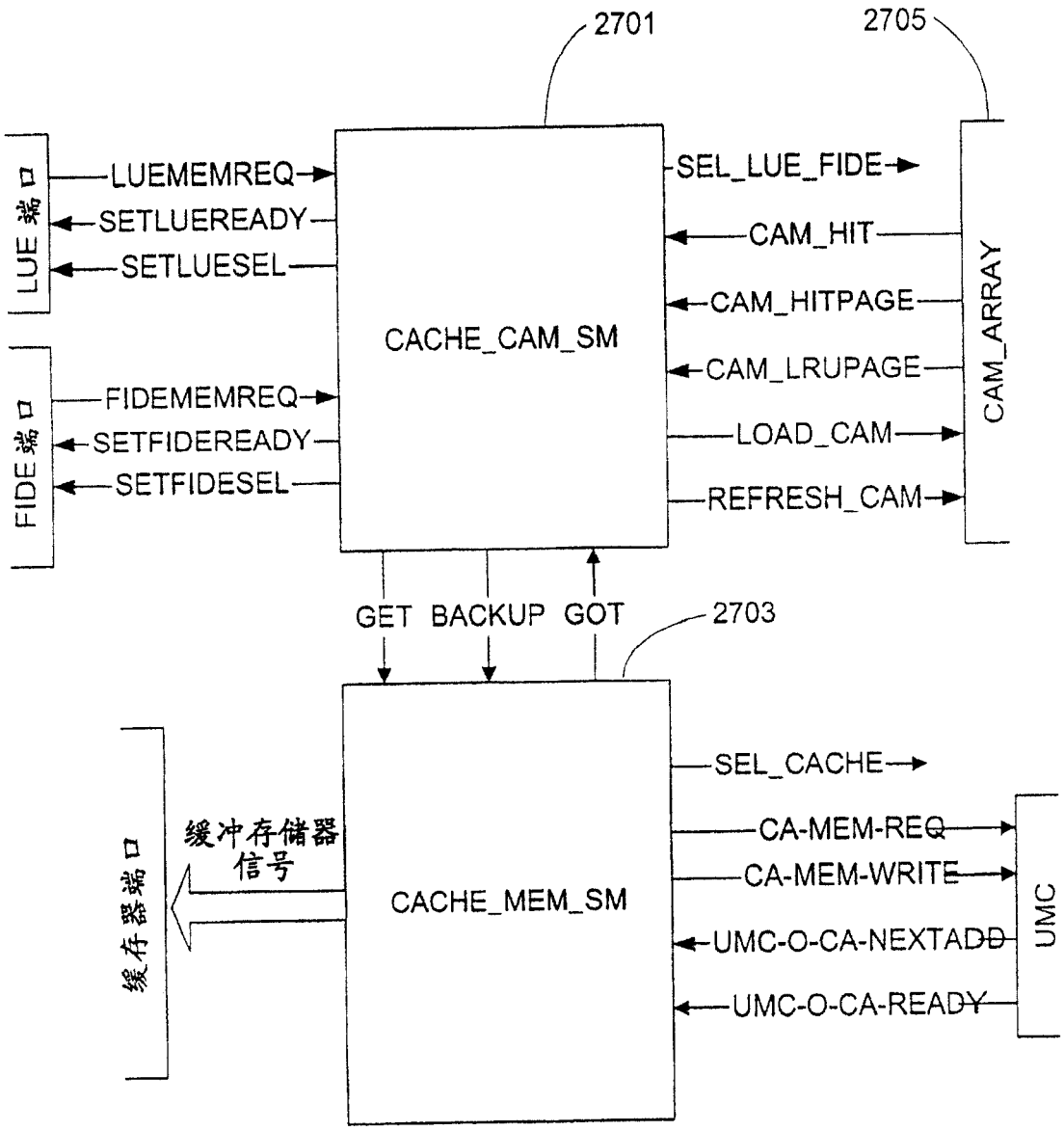

2701
2705
LUE 端口
LUEMEMREQ
SETLUEREADY
SETLUESEL
FIDE 端口
FIDEMEMREQ
SETFIDEREADY
SETFIDESEL
CACHE_CAM_SM
SEL_LUE_FIDE
CAM_HIT
CAM_HITPAGE
CAM_LRUPAGE
LOAD_CAM
REFRESH_CAM
CAM_ARRAY
GET
BACKUP
GOT
2703
缓存器端口
缓冲存储器信号
CACHE_MEM_SM
SEL_CACHE
CA-MEM-REQ
CA-MEM-WRITE
UMC-O-CA-NEXTADD
UMC-O-CA-READY
UMC

图 27

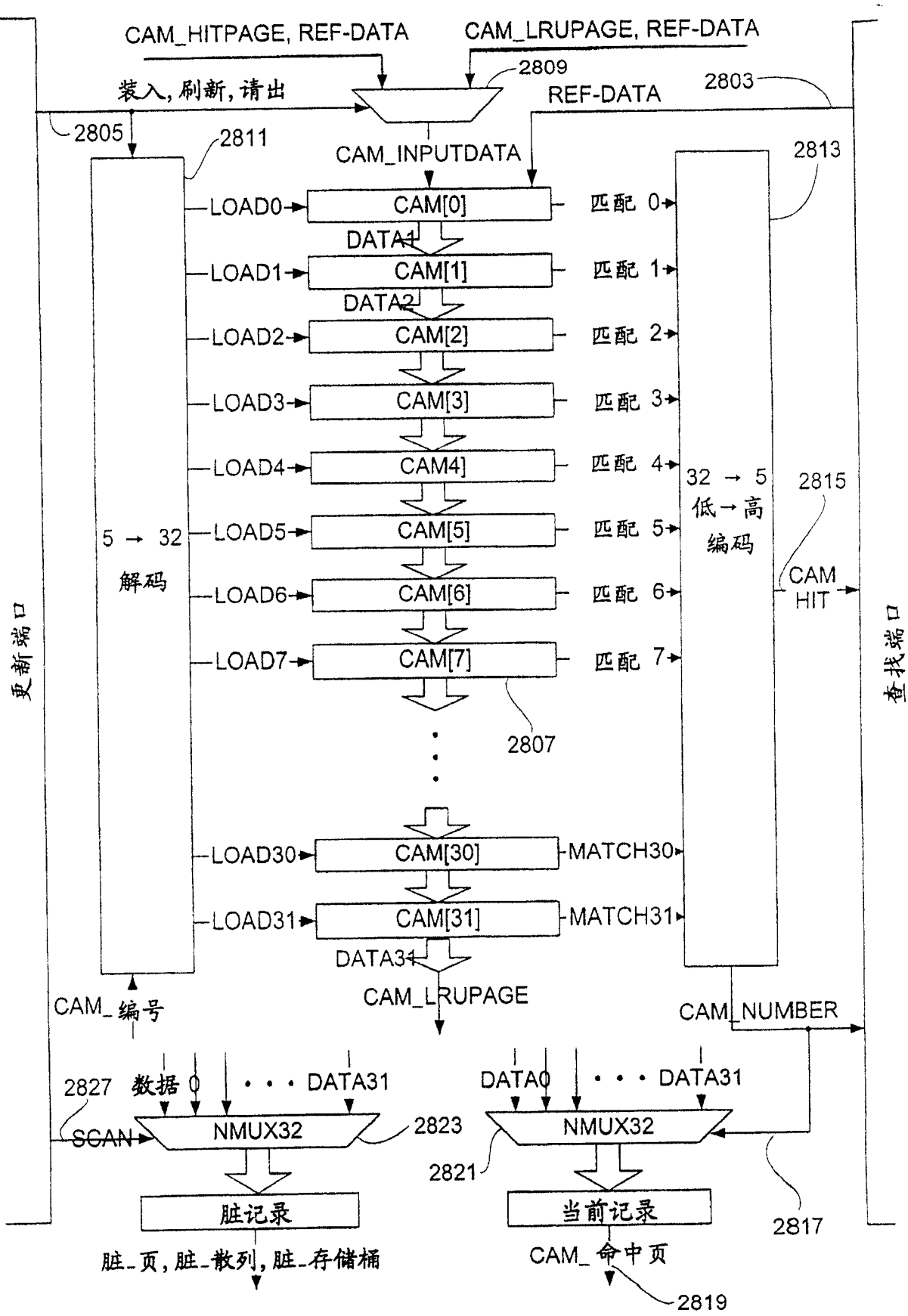

CAM_HITPAGE, REF-DATA
CAM_LRUPAGE, REF-DATA
装入, 刷新, 请出
2809
2803
REF-DATA
2805
2811
CAM_INPUTDATA
2813
LOAD0
CAM[0]
匹配 0
DATA1
LOAD1
CAM[1]
匹配 1
DATA2
LOAD2
CAM[2]
匹配 2
LOAD3
CAM[3]
匹配 3
LOAD4
CAM4]
匹配 4
32 → 5
低→高
编码
2815
5 → 32
解码
LOAD5
CAM[5]
匹配 5
LOAD6
CAM[6]
匹配 6
CAM
HIT
LOAD7
CAM[7]
匹配 7
更新端口
查找端口
2807
LOAD30
CAM[30]
MATCH30
LOAD31
CAM[31]
MATCH31
DATA31
CAM_编号
CAM_LRUPAGE
CAM_NUMBER
2827
数据 0
DATA31
DATA0
DATA31
SCAN
NMUX32
2823
NMUX32
2821
脏记录
当前记录
2817
脏_页, 脏_散列, 脏_存储桶
CAM_命中页
2819

图 28