[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 13/14 (2006.01)
G06F 13/42 (2006.01)

[12] 发明专利说明书

专利号 ZL 99803953.5

[45] 授权公告日 2008 年 6 月 25 日

[11] 授权公告号 CN 100397372C

[22] 申请日 1999.1.21 [21] 申请号 99803953.5

[30] 优先权

[32] 1998.1.22 [33] US [31] 60/072,128

[86] 国际申请 PCT/US1999/001234 1999.1.21

[87] 国际公布 WO1999/038084 英 1999.7.29

[85] 进入国家阶段日期 2000.9.13

[73] 专利权人 英纳瑞公司

地址 美国犹他州

[72] 发明人 A·沃拜克 T·N·李

[56] 参考文献

US5630204A 1997.5.13

US5608720A 1997.3.4

审查员 张一良

[74] 专利代理机构 北京纪凯知识产权代理有限公司

代理人 赵蓉民 彭益群

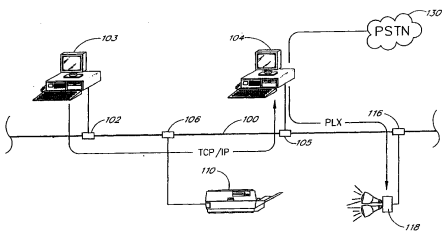
权利要求书 4 页 说明书 76 页 附图 20 页

[54] 发明名称

用于通用数据交换网关的方法和装置

[57] 摘要

本发明涉及允许在一种或多种网络协议以及一种或多种控制协议之间传送数据的一种通用网关(104)。不同的协议可以在相同的物理网络介质(100)上同时使用，或在各自的网络(100，130)上同时使用。通过使用所述网关(104)，终端用户可以把传统上相互独立、互不兼容的网络联网成为一个普遍可访问的、集中管理的“超级网络”。所述网关(104)提供集中式节点数据库，对传统协议的支持，规则引擎，以及面向目标级别的程序库接口。通过备用服务器节点提供的系统容错性能，增强对该集中式节点数据库的高可靠性访问。此外，所述网关(104)提供在电源线上分配不同类型的数据流的能力。所述网关(104)提供的路由处理器，允许在电源线上路由实际上任何传统的数据联网服务，如 TCP/IP。



1、 一种配置为允许在一个计算机网络上的多个节点使用一种或多种数据协议进行通信的网关，其中使用一种介质协议在一种网络介质上传输所述的一种或多种数据协议，所述网关进一步提供一个应用编程接口，用于与所述多个节点进行通信，所述网关包括：

包含在一个网络上节点的信息的一个内部节点数据库；

配置为用来提供一种活动模式和一种备用模式的一个软件模块，所述活动模式被配置用于维护所述内部节点数据库，并提供到所述节点数据库的访问，所述备用模式被配置用于维护所述内部节点数据库作为一个外部节点数据库的一个镜像复制，其中所述软件模块被配置为当检测到一个未被确认的客户请求后，转变为所述活动模式。

2、 如权利要求 1 中所述的网关，其中所述内部节点数据库进一步包括根据一个客户节点的一种状态变化确定所采取的行动的规则。

3、 如权利要求 2 中所述的网关，进一步包括配置为用于翻译所述规则的一个规则引擎。

4、 如权利要求 2 中所述的网关，进一步被配置为把规则翻译为一种规则定义语言。

5、 如权利要求 2 中所述的网关，其中所述状态变化包括在所述客户节点的一个实例变量中的一个变化。

6、 如权利要求 5 中所述的网关，进一步包括配置为当在所述客户节点的一个实例变量中的一个变化发生时通知一个用户应用的一事件处理器。

7、 如权利要求 1 中所述的网关，其中所述内部节点数据库通过发布连接测试程序请求进行更新。

8、 如权利要求 1 中所述的网关，进一步配置为通过隧道方式使一个第一种协议通过一个第二种协议。

9、 如权利要求 8 中所述的网关，其中所述介质是一电源线，并且所述介质协议是一电源线协议。

10、 如权利要求 1 中所述的网关，其中所述介质是一电源线，并且所述介质协议是一 PLX 协议。

11、 如权利要求 1 中所述的网关，进一步包括一个面向目标的应用编程接口。

12、 如权利要求 11 中所述的网关，进一步包括一互联网浏览器，所述浏览器配置为用于提供到所述内部节点数据库中信息的一个用户接口。

13、 如权利要求 12 中所述的网关，其中所述用户接口配置为允许一个用户控制在一个电源线网络上的节点。

14、 使用一所需的协议在一个网络上的节点之间进行通信的一种方法，所述方法包括：

建立包含关于所述节点信息的一节点数据库；

指定一活动的网关节点来维护所述节点数据库，所述活动的网关节点提供一种或多种访问所述节点数据库的访问方法；

在一个或多个备份服务器节点中，镜像所述节点数据库；并且

当检测到一个未被确认的客户请求后，从一种备用模式转变为一种活动模式。

15、 如权利要求 14 中所述的方法，进一步包括翻译和执行规则，当一状态变化在一个客户节点中发生时，所述规则确定将采取的行动。

16、 如权利要求 15 中所述的方法，其中所述规则由一个规则引擎翻译。

17、 如权利要求 15 中所述的方法，进一步包括步骤：当所述状态变化发生时，产生事件通知。

18、 如权利要求 17 中所述的方法，其中所述通知提供给一调度。

19、 如权利要求 15 中所述的方法，进一步包括步骤：把接收的数据翻译为一规则定义语言。

20、 如权利要求 15 中所述的方法，其中所述状态变化包括所述客户节点的一个实例变量中的一变化。

21、 如权利要求 14 中所述的方法，进一步包括发布连接测试程序请求并侦听对所述连接测试程序请求的响应的步骤，所述响应用于更新所述节点数据库。

22、 如权利要求 14 中所述的方法，进一步包括步骤：把在第一种协议中的原始包装入所述的所需协议中的封装包，并以隧道方式使所述原始包通过所述的所需协议。

23、 如权利要求 14 中所述的方法，其中所述介质是一电源线，并且所述介质协议是一电源线协议。

24、 如权利要求 14 中所述的方法，其中所述介质是一电源线，并且所述介质协议是一 PLX 协议。

25、 如权利要求 14 中所述的方法，进一步包括步骤：在所述客户节点的一个实例变量中，一个变化发生时，通知一用户应用。

26、 如权利要求 14 中所述的方法，进一步包括步骤：使用一互

联网浏览器来浏览所述节点数据库中的信息。

27、 如权利要求 14 中所述的方法，进一步包括步骤：使用一互联网浏览器来控制在一个电源线网络上的节点。

用于通用数据交换网关的方法和装置

发明背景

发明领域

本发明涉及计算机网络协议网关，特别是涉及适于电源线联网系统的网关。

相关技术的描述

计算机特别是个人计算机的广泛应用，造成了计算机网络数量的迅速增长。将两台或多台计算机联网到一起允许计算机共享信息、文件资源、打印机等。把两台或多台个人计算机和打印机连接到一起形成一个网络，原则上，是一个简单的任务。计算机和打印机可以使用一根电缆简单地连接到一起，并且在计算机上安装必要的软件。在网络术语中，电缆是网络介质，计算机和打印机是网络节点。网络节点使用一个或多个协议，如传输控制协议、网间协议（TCP/IP），进行相互“对话”。

网关用于执行从一个协议到另一个协议的协议转换，这样使用不同协议的网络能够相互连接。例如，Prodigy 网络服务具有用于在它的内部专用电子邮件格式和国际互联网电子邮件格式之间进行转换的一个网关。

标准网络协议，一般在设计时假设每个网络节点都是一个具有基本的数据处理和存储能力的“智能”设备。例如，一台典型的个人计算机（PC）有足够的处理和存储能力，来处理几乎任何网络协议。但是，一台典型的打印机是一个“哑的”设备，并不具有必须的处理和存储能力。一些制造商提供允许打印机被连接到网络的网络打印机适配器。该打印机适配器，是提供类似于全配置个人计算机的数据处理和存储能力的单板计算机。该网络打印机适配器，因此把“哑的”打印机转换为一个“智能”设备。尽管网络打印机适配器确实起作用，

但是它们的价格相对昂贵，因此不适于在一些家庭和小型办公环境中使用。此外，该打印机适配器也不适于用来将其它非 PC 设备连接到网络。例如，用户经常希望将哑的设备，如户外的灯、报警系统、电话系统等诸如此类的设备，连接到他们的计算机网络。购买网络适配器插件，来把这些哑设备的每一个变为一个智能设备，将不可避免地费用昂贵。

用于智能设备的协议一般称为“网络协议”。用于哑设备的协议经常称为“控制协议”。在性能和复杂性两方面，网络协议与控制协议都大不相同。由于这些不同，设计用于在网络协议之间进行数据转换的网关，一般不适于在控制协议之间进行数据转换的任务。而更为困难的任务，是在网络协议和控制协议之间进行数据转换。

已有的家庭用控制/自动产品，倾向于使用基于对等模型的控制协议而不是集中式的客户/服务器模型。这严重限制了这些产品的应用，因为每个节点需要在本地存储状态信息和规则。由于缺少容易使用的、集中式的用户接口部件，配置网络很困难。而且，在竞争产品间（如 X-10 和 CEBus）的互操作性基本上是不可能的。

发明概述

通过提供一个低成本、容易使用、灵活、可靠并且可伸缩的网关结构，解决了这些以及其它的问题，该结构允许在一个或多个网络协议和一个或多个控制协议之间进行数据转换。各种不同的协议能够在相同的物理网络介质上同时使用。该网关也提供使网络协议以隧道方式通过一个选定的协议和集中控制。

因此，为数据和控制网络的终端用户，尤其是在家庭和小型办公环境中，该网关提供了极大的益处。通过使用所述网关，终端用户可以容易而方便地把传统上相互独立、互不兼容的网络联网成为一个普遍可访问的、集中管理的“超级网络”，该网络连接计算机、打印机、报警系统、家庭设备、照明系统和电话系统等。

该网关提供一个集中式节点数据库，支持传统的协议如 TCP/IP，

规则引擎以及面向目标类库接口。使用普通的、容易使用的图形用户接口，如国际互联网浏览器，该网关提供广泛的家庭/小型办公室自动化和控制能力。通过自动的设备发现，简化了配置。通过备用服务器提供的系统容错性能，增强了对集中式节点数据库的访问。

当与电源线网络一起使用时，该网关提供在电源线上分配不同类型的数据流的能力。例如，拥有电缆调制解调器或其它类型的高速国际互联网连接的网络用户，可以把国际互联网业务分配到家庭/办公室中的任何地点，而不需要除了已有的电力线以外的附加附加线路。音频数据也很容易通过电源线在整个家庭/办公室中分配。该网关提供的路由处理器，允许实际上任何传统的数据联网服务和现在通用的协议，在电源线上被路由。

附图的简要描述

当与以下列出的附图一起阅读时，在本技术领域内熟练的人员，从以下的详细描述中，将很容易地意识到公开的本发明的优点和特征。

图 1 是具有如个人计算机的智能节点和如外部安全照明的哑节点的网络的一个框图。

图 2 是七层 OSI 网络模型的框图。

图 3 是通用的网关结构的框图。

图 4 是展示服务器激活算法的框图。

图 5 是展示一个规则的各部分的数据图表。

图 6 是用于智能设备的一个 PLX 网络模型的框图。

图 7 是用于哑设备的一个 PLX 网络模型框图。

图 8 是展示介质访问算法的一个流程图。

图 9A 是展示活动的网络服务器发信号（spitting）算法的一个流程图。

图 9B 是展示客户发信号（spitting）算法的一个流程图。

图 10 是展示活动的网络服务器轮询算法的一个流程图。

图 11 是展示一个 PLX 逻辑组隔离（LoGI）包的字段框图。

图 12 是展示一个 PLX 原始数据包的字段的框图。

图 13 是展示一个 PLX 令牌包的字段的框图。

图 14 是展示一个 PLX 直接确认（DACK）包的字段的框图。

图 15 是展示一个 PLX 掩码的排队插入包（LIPG）的字段的框图。

图 16 是展示一个 PLX 直接排队插入（LIPD）包的字段的框图。

图 17 是展示一个 PLX 内部主机包的字段的框图。

图 18 是展示一个 PLX 公共应用语言（Common Application Language）(CAL)请求包的框图。

图 19 是展示一个 PLX CAL 响应包的框图。

图 20 是展示一个 PLX 单个信道传输状态包的框图。

图 21 是展示一个 PLX 多信道传输状态包的框图。

图 22 是展示一个 PLX 包定时的一个定时图。

在图中，任何三位数字的第一位数字，一般标识组件第一次出现的图的号码。使用四位参考号码的地方，前两位数字标识图的号码。

最佳实施例的详细描述

图 1 展示了适于使用通用的网关的一个计算机网络系统。图 1 中的系统展示了两种不同的物理网络。其中第一种物理网络使用网络介质 100（示为电缆）。一个智能节点（展示为一台个人计算机 103）通过一个连接器 102 被连接到该网络介质 100。一台打印机 110，一台计算机 104 和一个安全照明系统 118 也被连接到该网络介质 100。该安全照明系统 118，是具有相对少的计算功能或存储的一个“哑”节点的一个举例。计算机 104 也连接到示为公共电话交换网（PSTN）的第二个网络 130 上。

在一个典型的网络环境中，网络介质 100 配置为承载不同数据协议的数据业务。因此，如图 1 中例子所示，计算机 103 可以使用 TCP/IP 协议与计算机 104 通信，而计算机 104 使用一控制协议，如在公开的本申请中标为附录 A 的部分中所描述的电源线交换（PLX）协议，与照明系统 118 进行通信。附录 A 包括申请号为 09/211950，标题为“电源线交换协议的方法和装置”的美国申请的部分，在此引证作为参考。

一个通用的网关作为一个软件程序，在一台计算机上运行，如计算机 104 上。该通用的网关在不同的网络协议之间，以及不同的物理网络之间，提供连接。例如，如图 1 中所示，该通用的网关作为一个软件程序，在计算机 104 上运行，连接 TCP/IP 协议到 PLX 协议，因此允许计算机 103 与照明系统 118 通信。

在计算机 104 上运行的通用网关软件，也提供在各自的物理网络之间的数据传输，因此允许在各自的物理网络上的设备相互通信。在图 1 中，该通用的网关提供在网络 130 和照明系统 118 之间的数据传输。

该通用的网关与分层的网络协议兼容。为智能节点（如计算机 103 和 104）配置的大部分网络，是基于由开放系统接口（OSI）委员会开发的一个网络结构模型。该 OSI 结构定义了一个网络模型，其中列出了在一个通信系统中的每一个单独的硬件和软件层，层之间的内部相关性，以及每一层所执行的单一功能。

图 2 展示了该 OSI 结构被分为七个层，从最低到最高为：物理层 201；数据链路层 202；网络层 203；传送层 204；对话层 205；表示层 206；和应用层 207。每层使用直接在其下面的层，并且为直接在其上面的层提供服务。在一些实施中，一层可能本身由多个子层组成。一层是两个或多个通信设备或计算机的软件和/或硬件环境，一个特定的网络协议在该通信设备或计算机中运行。一个网络连接可以被看成为一组或多或少独立的协议，每一个在不同的层或级中。最低层管理在不同节点的硬件之间直接的节点到节点的通信；最高层由用户应用程序组成。每层使用在其下面的层，并且为在其上面的层提供服务。在一个主机上的每个联网组成部分的硬件或软件，使用适用于它的层的协议，与在另一个节点上的相应的组成部分（它的“对等”）进行通信。这样的分层协议有时被称为对等协议。

分层协议的优点，在于由一层到另一层传递信息的方法，被清晰地确定为协议的组成部分，并且防止了在一个协议层内的改变影响到另一个协议层。这简化了设计和维护通信系统的任务。

物理层 201 是 OSI 分层的模型中的最低层。它涉及网络中电的和机械的连接，包括介质访问控制（MAC）部分。介质访问控制是指对于数据传输介质 100（例如网络电缆）的控制和访问。物理层 201 被数据链路层 202 使用。

数据链路层 202 是 OSI 模型中的由下数的第二层。数据链路层 202 把数据分成帧（为了在物理层 201 上发送），并接收确认帧。数据链路层 202 执行错误检测和重传未被正确接收的帧。数据链路层 202 为网络层 203 提供一个无差错的虚拟信道。数据链路层 202 被典型地分为一个高的子层，逻辑链路控制（LLC），和包括介质访问控制（MAC）

的部分的一个低的子层。

网络层 203 是 OSI 七层模型中的由下数的第三层。网络层 203 决定通过数据链路层 202 的由发送者到接受者的数据包的路由，并且被传送层 204 使用。最常用的网络层协议是 IP。

传送层 204（或“主机—主机层”）是 OSI 模型中处于中间的层。传送层 204 决定怎样使用网络层 203，来提供一个虚拟的无差错的、点对点的连接，因此第一个节点可以传送信息到第二个节点，并且信息将未被破坏地和以正确的次序到达。传送层 204 建立和释放节点间的连接。

对话层 205 是 OSI 模型中的由上数的第三层。对话层 205 使用传送层 204 来建立在不同节点上的过程间的连接。对话层 205 处理安全性和对话的建立。

表示层 206 是 OSI 模型中的由上数的第二层。表示层 206 执行如文本压缩、编码或格式转换等功能，来消除节点间的差异。表示层 206 允许在应用层中不兼容的过程，通过对话层进行通信。

应用层 207 是 OSI 模型中的最高层。应用层 207 与用户对于网络的视图（如格式化的电子邮件信息）相关。表示层 206 为应用层 207 提供独立于在网络上使用格式的熟悉的本地数据表示。应用层协议的例子包括：远程登录、文件传送协议（FTP）、简单网络管理协议（SNMP）、简单邮件传送协议（SMTP）、网间控制报文协议（ICMP）、NetWare 核心协议（NCP）、路由信息协议（RIP）、服务广告协议（SAP）、一般的文件传送协议（TFTP）、系统错误容限协议（SFTP）。

图 3 展示了通用网关 300 的结构，以及该通用网关 300 如何与其它软件组成部分，如应用程序、硬件驱动等，相互作用。应用程序 302，使用一组网关基础级 304，与网关 300 进行通信。网关基础级 304 与数据库访问模块 306、事件处理器 310、调度 320、运行时间操作系统服务（RTOS）311 进行通信。数据库访问模块 306 包括一被调用的信

息储存库（一节点数据库）308。节点数据库 308 可以组织为数据库、链接列表、表、图、容器等。数据库访问模块 306 还与事件处理器 310 以及有效负荷/协议处理装置 312 通信。该有效负荷/协议处理装置包括一个或多个有效负荷/协议处理器，如原始数据处理器 316、流式数据处理器 317 和 CAL 控制处理器 318。事件处理器 310 还与规则引擎 314 以及调度 320 进行通信。调度 320 还与 RTOS 311、规则引擎 314、PLX 设备驱动 322 以及传统设备驱动 326 进行通信。该 PLX 设备驱动包括 PLX 控制设备驱动 323(用于哑设备)和 PLX 数据设备驱动 324（用于智能设备)。

网关 300 提供由一个协议到另一个协议的协议转换，这样两个使用不同协议的网络可以互联。该网络可以是真正的、物理网络，也可以是虚拟网络（如，只存在于软件中)。网关 300 在传统协议和主要的（如所需的）协议（如电源线协议）之间提供接口。传统协议这一术语并不限于以前使用的协议。传统协议这一术语，而是指除了主要协议以外的其它任何协议。传统设备驱动包括，如 X-10 驱动 329CEBus 驱动 332。而且每个传统设备驱动，可选择地，包括用于使传统设备驱动与网关适配的一个调整（shim)。在一个实施例中，电源线协议作为主要协议，但是网关 300 并非如此限定。因此，主要协议可以是任何协议，包括，如 TCP/IP、IPX、ADSL 以及在本公开文本其它地方所列或本领域内所知的任何协议。

对于流式传统设备驱动 362，如以太网驱动 362 和 ADSL 驱动 364，网关提供了传统堆栈 352。传统堆栈 352 支持 OSI 协议堆栈，如 TCP/IP 堆栈 353 和 SPX/IPX 堆栈 354。传统堆栈 352 与流式传统设备驱动 362 以及路由处理器 355 通信。路由处理器 355 与流式传统设备驱动 362、传统堆栈 352 以及有效负荷/协议处理装置 312 通信。传统堆栈 352 还与传统应用 350 通信。

网关 300 作为不同类型网络之间的一个链接点，为控制和数据网络两者提供支持，其中控制和数据网络包括但不限于以太网、数字用户线（DSL 和 ADSL)、电源线交换（PLX)、X-10 以及 CEBus。

网关 300 具有两个主要功能，即节点管理和数据路由。作为节点管理者，网关 300 负责发现、列举、检索、存储以及处理网络节点状态信息。状态信息存储在节点数据库 308 中。节点数据库 308 基于 Generic 公共应用语言（CAL）技术规格。EIA-600 中定义的 CEBus，是用于控制总线设备的一种工业标准控制语言。EIA-600 为用于家庭的 LAN 中的公共应用语言提供了一个框架。Generic CAL 在 EIA-721 系列标准（包括 EIA-721.1、EIA-721.2、EIA-721.3 和 EIA-721.4）中定义。CEBus 工业委员会（CIC）定义了一个家庭即插即用（HPP）技术规格，通过为使用 Generic CAL 定义“语法”规则，该 HPP 技术规格使框架细节化。

HPP 技术规格为家庭中的产品和系统，细节化了一组行为特征，这些行为特征允许产品或系统根据家庭中的状态采取行动。例如，该技术规格定义了家庭内的不同状况，如“居住者不在”或“居住者在家睡觉”，来允许系统采取适当的行动，如启动安全系统、关闭室内的灯或设置温度。HPP 技术规格也包括用于开发基于 Windows ’95 PC 的家庭控制应用的信息。

EIA-600 中定义的公共应用语言，提供了用于在多个产业部门（如娱乐、计算机、加热/制冷、厨房设备等）生产的家庭 LAN 产品中通信的一个框架。每个产业部门定义产品使用语言的“应用环境”（如语法规则）。作为支持组织建立了 CIC，该组织协助多个产业部门开发“协调的”应用环境。对于那些为基于 CAL 的可互操作产品，寻求家用 LAN 市场的产业部门，CIC 的 HPP 是兼容的应用环境的一个纲要。

CEBus/Generic CAL 技术规格全部在此引证，作为参考。

网关 300 也提供规则引擎 314，规则引擎监视节点状态变化，并根据状态变化，根据使用通用的规则定义语言（RDL）的语法定义的规则，采取行动。

通过使用调度控制块（DCB），网关 300 也处理数据流和路由任务。

路由处理器 355 提供这样的功能，如 TCP/IP 隧道通过 PLX、代理服务器以及网址翻译等。提供调整（shim）（328、330），用于允许基于非 PLX 电源线的节点无缝隙地插入 PLX 网络。网关基础级 304 提供被用户接口和系统管理应用 302 使用的面向对象的 API 库。网关 300 进一步提供支持，允许使用传统的、非通用的网络 API（MFC、TLI、套接字、家庭 API 等）与网络节点进行通信。这允许传统网络数据（以太网、ADSL 等）的透明的数据流在网络上通过，包括控制网关 300 使用 Java 和 Web Browser 管理的节点的能力。

节点数据库 308 是由网关 300 存储的节点数据储存库。节点数据包括由每个客户节点得到的节点配置文件夹，事件序列，客户节点联编和规则。网关 300 的其它组成部分，通过一组访问方法 306 访问节点数据库 308。

事件处理器 310 和规则引擎 314，负责当客户节点中状态发生变化，作为结果而发生的行为。规则引擎 314 翻译与状态变化相关的规则，并负责规划通知或作为规则求值结果可能触发的 CAL 发送请求。规则引擎 314 与节点数据库 308 和事件处理器 310 一起工作。事件处理器 310 处理事件序列和执行事件通知。这些事件通知包括到网关 300 其它组成的内部通知，和到已经注册以接收事件通知的应用 302 的外部通知。在网关 300 和应用 302 之间的互操作通过网关基础级 304 完成。

调度 320 负责管理网关 300 运行时的操作，包括发送/接收序列处理、线路初始化以及管理等。需要管理的网关活动包括应用 CAL 发送请求、原始流式数据请求、接收事件和后台管理任务。调度 320 提供普通的接口到网络设备驱动 322、326，允许支持不同类型的网络节点，而不需要对于驱动 322、326 服务的潜在的网络控制器硬件的详细信息。

设备驱动 322、326 直接与网络通信硬件（USB 或并行端口 PLX 节点、X-10 调整（shim）、CEBus 调整（shim）等）通信，并且处理数据链路层 202 功能（如产生/翻译 MAC 头、低级定时等）。直接与

潜在的硬件对话的驱动 322、326 部分，是典型的平台式技术规格，用于支持不同类型的可能存在于驱动和硬件之间的连接（如通用串行总线（UBS）、并行端口、防火端口、PCI 插槽、ISA 插槽、串行端口等）。网关 300 提供调整（shim）组成 328、330，用于处理由传统的 X-10 和 CEBus 节点接收的控制数据到网关 300 使用格式的转换。

在有效负荷/协议处理装置 312 中的有效负荷/协议处理器 316-318，负责解释和执行接收的包数据。数据处理器 316-317 与路由处理器 355 共同工作。路由处理器 355 解释原始数据包的头。路由处理器 355 也查询地址和路由表，该表协助导向在不同的传统协议堆栈 352 和传统的流式驱动 362 之间的数据流。

为了维护在不同的平台和操作系统上编码的可移植性，平台特定编码在 RTOS 311 提供的平台抽象层中被隔离。典型的运行时服务包括线路计划、储存器管理、中断处理、时钟、同步、优先权计划和初始化等。

网关 300 中的许多组成，直接或间接地与节点数据库 308 相互作用。最初建立和维护数据库的任务，主要由调度 320、驱动 322、326 以及有效负荷/包处理装置 312 处理。用户接口和管理应用，通过网关基础级 304 访问和/或修改节点数据库 308。

节点数据库 308 是结构化的，这样实时访问快且高效。在速度/存储使用之间进行了折衷，通常其中存储器使用效率比访问速度被赋予更高的优先权。在一些实施例中，节点数据库 308 的部分，存储在非挥发性储存器中。

节点数据库 308 包含客户节点配置和节点状态信息。这包括与每个客户节点相关的使用环境、对象和实例变量（IV）信息。在使用网络命令和网络请求的节点发现过程中，获得这个信息。这个信息中，动态的 IV 存储在永久性储存器中。对于大多数对象（不包括节点控制和环境控制），这包括定义节点状态的当前值 IV。

节点数据库 308 也包含一些节点确定的信息。在每个节点存在的节点环境之外，节点数据库 308 包含包括用于节点管理的对象的数据库服务器环境。该数据库服务器环境包括网络节点列表对象，该对象中包含网关 300 了解的客户节点地址矩阵。

通过一组普通的访问方法 306 访问节点数据库 308。由于大多数访问直接或间接来自网络信息，数据库访问方法 306，使用典型的网络取得（Get）和设置（Set）方法，负责取得和设置变量值。这种方法包括 GetValue、GetArray、SetValue、SetArray、SetON 和 SetOFF。哪种方法适用，决定于所作用的实例变量的类型。例如，CAL 网络定义了四种 IV 数据类型：布尔型（GetValue、SetValue、SetON、SetOFF），数值型（GetValue、SetValue），字符串型（GetValue、SetValue），和数据型（GetArray、SetArray）。

大部分时候，这些方法使用一组相似的键，它们确定起作用的数据和输入变量的任何数值。每种方法随后返回所请求的信息（如果有）和状态。例如，作用于 CAL IV 的用于以上方法的普通的原型如下：

ENUM_STATUS <method> (NodeID, ContextID, ObjectID, IVID, args, ..., *returnVal)

在此例中，NodeID, ContextID, ObjectID 和 IVID 为键。对于非 CAL 目标，使用不同的一组键。

当在一个实例变量上进行设置时，通过事件处理器 310 产生事件通知到规则引擎 314，指示一个状态变化已经发生。然后，规则引擎 314 翻译关于发生变化的实例变量的规则/报告状态，如果已经规定了操作，则通过事件处理器 310 规划事件。这样做的结果可能是通过调度 320 发送一个网络请求，或者通过基础级 304 通知到较高层应用 302。

如上文所述，当产生一个访问请求来 GET 或 SET 一个节点 IV 时一般要提供四条信息（或键）。这些信息包括一 NodeID, 一 ContextID, 一 ObjectID 和一 IVID。

NodeID 是设置过程中分配的节点地址（这可以在制造时或在运行时动态设置）。对于 PLX 网络，这是一个 4 字节字段，取值在 0x00000001 和 0xffffffef 之间；高于此范围的地址专供广播和其它 PLX 特定应用。对于非 PLX 网络，一般需要进行一些地址映像/翻译。路由处理器 355 负责地址翻译。

ContextID 长度为 2 字节，其高字节为一个可选择的应用环境编号，取值范围为 0xA0-0xDE 或在未被使用时为 0。ContextID 的低字节是一个应用等级值，取值范围为 0x00-0x9E（这些在通用 CAL 技术规格中定义）。

ObjectID 是一个 1 字节的目标编号值,取值范围为 0x01-0x3E。

IVID 为一个 ASCII 码字符（或字符串），其在一个给定的应用环境和目标类型之中确定一实例变量。

通过一组应用环境描述一个节点；每种应用环境包括一组目标；每个目标包括一组 IV。数据库是结构化的，这样访问给定所述四条信息的一个 IV，包括以下查询算法。

1） NodeID 映像到一个节点列表入口（一般使用一散列算法），其中每个节点指向一应用环境记录矩阵。ContextID 未定义，这样它可以被用作到所需的应用环境记录的直接索引，因此，在此出现线性查询。一个节点一般仅有几个（2 或 3 个）应用环境。每个应用环境记录包含到一个目标记录矩阵的一个指针（或一个链接的列表）。

2） ObjectID 是格式型的，这样它可以用作到所需目标记录的一个直接索引。每个目标记录包含到 IV 记录矩阵的一个指针。

3） 线性检索用于定位所需的 IV，基于 IVID 将作用于此 IV。这样操作是有效的，因为大部分客户节点只在节点数据库 308 存储几个 IV。大部分 GET/SET 请求与所述 IV 的当前值有关。如果当前值总是存储在 IV 列表的开头，则几乎不需要扩展的线性检索。

网关 300 用于填充网络节点列表的发现过程，根据节点类型的不同而不同。对于使用 CAL 的典型的 PLX 节点，使用以下算法：

1） 在网络上广播一个 CAL “连接测试程序（Ping）”请求（一般在初始化过程中完成此操作，但也可以定期进行）。这种请求的格式与其它任何 CAL 命令/请求的格式相似，并使用以下参数：

Context Class = 0 (通用的应用环境),

Object Number = 1 (节点控制目标),

Method = 50h = PING_REQUEST (未在普通 CAL 技术规格中定义),

IV = < 任何 IV 均可 >.

这种请求与标准 CAL 请求不同，标准 CAL 请求可以直接发送到一个节点或者广播到所有节点。连接测试程序（Ping）请求不需要来自接收者的导典型的 CAL 格式响应包。

2） 接收 CAL 连接测试程序（Ping）请求的节点，发回一个 CAL 连接测试程序响应到发送的节点。尽管这个包被称为一个“响应”，该包的语法与传统的 CAL 响应包不同。该响应在次序上不同，以允许连接测试程序请求被作为广播发送。除了方法编码为 51h 而不是 50h 之外，响应包的格式（它实际上类似 CAL 命令/请求，而不是 CAL 响应）与连接测试程序请求相似。

3） 当网关 300 由未列在网络节点列表中的一个节点接收到一个连接测试程序响应时，网关 300 增加该节点到列表中，并把该节点的 IV 信息加入到节点数据库 308。

4） 可选择地，网关 300 发送一个 CAL 请求，到节点列表中的在一定长度的时间内未收到信息的任何节点。如果该节点无法响应，它被由列表中移去。

5） 当一个客户节点加电或复位时，也发送连接测试程序响应包。

对于传统的控制节点（如 X-10 或 CEBus 节点），根据该节点的传统技术规格，节点发现过程不同。网关 300 在调整（shim） 328，330 协助下，处理这些节点。调整（shim） 328，330 对于上述的 CAL 节点发现方法和非 PLX 驱动使用的传统方法都了解。作为对一个连接测试程序请求的响应，该调整（shim）把连接测试程序请求转换为在

传统控制网络上执行节点发现所需的相同请求（或请求组）。当节点被发现后，调整（shim） 328，330 产生连接测试程序响应包，并把它们上传到网关 300 的较高层。对于网关 300 的其余部分来说，这些传统的控制节点是 PLX 节点。以下将进一步论述调整（shim） 328，330。

网关 300 允许用户接口和管理应用 302 通过基础级 304 访问节点数据库 308。在这方面，网关 300 作为应用服务器。为了增强应用 302 对节点数据库 308 的可访问性，网关 300 支持多个应用服务器的存在，其中一个服务器作为活动的应用服务器，其它作为备用应用服务器。

每个服务器存储相同的节点数据库 308，并通过侦听网络业务、监视网络节点状态变化，相应地更新它的节点数据库 308，保持它的节点数据库 308 的信息与其它节点的信息一致（同步）。但是，活动的应用服务器是真正评估与状态变化相关的规则并执行所需的事件通知的仅有节点。如果活动的应用服务器由于某些原因变得无法工作，则一台备用服务器将检测到这种情况并成为“活动的”。在图 4 中，展示了一个应用服务器节点 402 激活的过程，由同步框 404 开始，在此，节点 402 更新其节点数据库 308。更新节点数据库 308 之后，过程前进到一个判定框 406。在判定框 406 中，如果该节点处于活动的服务器状态，则过程前进到活动的服务器框 408；否则，过程前进到备用服务器框 420。在完成活动的服务器框 408 后，过程前进到判定框 412。在判定框 412 中，如果检测到服务器静止请求，则过程前进到设定备用框 410；否则，过程前进到判定框 414。如果在判定框 414 中，检测到一客户请求，过程前进到响应框 416；否则，过程返回同步框 404。在完成设定备用框 410 或响应框 416 后，过程返回同步框 404。

在备用服务器框 420 完成后，过程前进到判定框 422。在判定框 422 中，如果检测到一个未被确认的客户请求，过程前进到设置活动的框 424；否则，过程返回到同步框 404。在完成设置活动的框 424 后，过程前进到通知框 426，通知其它应用服务器高性能节点当前服务器节点将成为活动的。在框 426 完成后，过程返回到同步框 404。

当客户节点中发生状态变化时，将会涉及到规则引擎 314 和事件处理器 310。因为当执行设置（SET）方法时发生状态变化，所以每个 SET 方法通过事件处理器 310 将变化通知规则引擎 314。然后，规则引擎 314 检查是否存在与所改变的 IV 相关的任何规则，并决定是否需要事件通知。规则可以由用户通过用户接口使用基础级 304，把不同的节点相互结合来创建。对于一些不需要明确的用户定义的节点和常规操作，也存在默认规则。

规则可以简单或复杂。简单规则是那些特征为用户节点之间是一对一或一对多结合关系的规则。简单一对一规则的举例，如“如果照明开关 A 切换到开状态，则接通照明灯泡 A。”一对多规则的举例，如“如果照明开关 A 切换到开状态，则接通照明灯泡 A、B、C 和 D。”复杂规则处理多对一或多对多结合（如“如果照明开关 A 切换到开状态并且用户具有安全间距，则接通照明灯泡 A、B、C 和 D。”）。

网关 300 为规则定义提供一个通用的语法，称为通用 Rules Definition Language (RDL)。图 5 是展示规则的各部分以及应用 502 与规则 504 之间相互作用的结构图。如图 5 中所示，应用 502 创建包括规则 504 的一个或多个规则。规则 504 可以包含涉及条件表示 506 的“if”描述。条件表示 506 可以包括一个或多个常量 506、一个或多个运算符 512 以及一个或多个 IV。在规则 504 中的一个“then”子句确定返回到应用 502 的通知 514。每个常量包含一个值。每个运算符包含一个运算标识符（ID）。每个 IV 包括一个 ID、一个值和一个触发状态（state）。触发状态（state）包括边缘触发和非边缘触发。每个通知包括一个通知 ID 和一个信息 ID。

一个规则包括一个事件和一个操作。该操作是一个应用明确的字符串矩阵。事件是一个描述是否网络上存在给定状态的逻辑表达。使用以下语法确定事件字符串。该语法具有未定义的终端符号*整型_常量（integer_constant）*、*字符型_常量（character_constant）*、*浮点_常量（floating_constant）和字符串（string）*。注意*标识符*终端具有确定的格式。插入记号‘^’或磅符号‘#’后的第一个 *integer_constant* 是

一个十六进制的节点地址。第二个（可选择的）*integer_constant* 是一个十六进制的 CAL 应用环境编号。第三个 *integer_constant* 是一个十六进制的 CAL 应用环境类型。第四个 *integer_constant* 是一个十六进制的 CAL 目标编号。*character_constant* 是一个 CAL 实例变量（IV）。在标识符之前的插入符号确定一个“边缘触发的”或“just changed” iv。磅符号表示该 iv 是“level”或“已有值”。一个规则只能有一个“边缘触发的”标识符。

规则语法

IF OR_表达 THEN ACTION_表达

OR_表达

AND_表达

OR_表达||AND_表达

AND_表达

等式_表达

AND_表达&&等式_表达

等式_表达

关系_表达

等式_表达==关系_表达

等式_表达！=关系_表达

关系_表达

相加_表达

关系_表达<相加_表达

关系_表达>相加_表达

关系_表达<=相加_表达

关系_表达>=相加_表达

相加_表达

相乘_表达

相加_表达+相乘_表达

相加_表达 -相乘_表达

相乘_表达

初始_表达

*相乘_表达*初始_表达*

相乘_表达/初始_表达

相乘_表达%初始_表达

初始_表达

标识符

常量

字符串

（OR_表达）

标识符:

^*integer_constant.integer_constant*$_{opt\ opt}$*integer_constant.integer_constant.character_constant*

#*integer_constant.integer_constant*$_{opt\ opt}$*integer_constant.integer_constant.character_constant*

常量:

整型_常量（integer_constant）

字符型_常量（character_constant）

浮点_常量（floating_constant）

ACTION_表达

通知_应用_为_此_规则_注册

发送_PLX_包_到_节点

对于包含嵌入的使用传统的语法（如 RDL 之外的一种语法）规则的非 PLX 节点，规则引擎 314（在调整（shim）328，330 的帮助下）把传统语法转换为 RDL 语法。

调度 320 负责导向出现在网关 300 中的命令和操作流。在初始化过程中，调度 320 调用 RTOS 311 来创建用于管理运行时操作的不同的处理器线程。按照优先级顺序列表，所述处理器线程包括：1）接收有效负荷线程，2）原始数据（流式）传输线程，3）CAL 传输线程，以及 4）低优先级的空闲线程（可选用的）。

以上所列的大部分处理器线程，发送包到客户节点。在调用一个设备驱动发送的例程之前，这些线程首先在 TxFreeCount 信号装置上等待（静止），该信号装置表明所述驱动能够处理发送请求。而且，CAL 发送线程确保 CAL 包不被发送到处于“延迟”模式的客户。延迟模式将在下文论述。

调度 320 还管理处理器线程所作用的队列。处理器线程和其它网关 300 组成部分调用调度 320，来增加并移去包到/由队列。调度 320 使用调度控制块来描述 CAL 请求/响应包和原始数据流片段。

DCB 结构定义（用 C/C++语法）如下：

```
Typedef struct sDCB
{
void            *link;
UNITE8 *buffer ;
UNIT32          destDevAddress;
UNITE32         srcDevAddress;
UNIT16          TimeStamp;
UNIT16          reserved1;
UNIT8           destSocket;
UNIT8           srcSocket;
UNIT8           sequenceNo;
UNIT8           bufferSize;
UNIT8           controlInfo;
UNIT8 reserved2[5];
}
tDCB,   *PDCB;
```

其中 *link* 域可以由 DCB 拥有者用于任何用途。它典型地用于 DCB 队列管理。*buffer* 和 *bufferSize* 域指向并定义 DCB 所描述的包/片段数据的长度。*destDevAddress* 和 *srcDevAddress* 确定与 DCB 相关的目标和源节点。*destSocket* 和 *srcSocket* 可以用于进一步确定在节点中的目标/源应用。timeStamp 域被网关 300 用来协助处理不同的时效条件。sequenceNo 用于排序包，主要用于原始数据包流。controlInfo 域用来储存用于指明需要 DCB 和相关数据包的特定特征/处理的不同位域和标志。这些位控制诸如数据加密、授权和数据流等特性。reserved1 和 reserved2 域由网关 300 在内部使用，并被外部应用使用。

所述接收有效负荷线程一般是由调度 320 发起的最高优先级线程，并且无论何时当设备驱动 322、325 把接收的包排队后，就执行。在设备驱动 322、325 中的中断/轮询服务例程，可以优先于此线程。此线程负责从接收队列移去包，并把包传递给有效负荷/协议处理装置 312 来进一步处理。此线程一直执行到接收队列变空为止（或者可以直到达到一些门限为止）然后处于静止直至排队更多的接收包之后。

当以下两个条件成立时，原始数据传输线程执行：1）一个原始数据/流式传输请求已被一个应用排队；和 2）任何具有更高优先级的线程/中断处理器已执行结束。原始传输线程在发送队列中得到另一个入口，并把它传给适当的设备驱动发送例程。这个过程一直重复到在发送队列中没有任何报告，或已经达到一些门限。原始数据传输线程然后处于静止，直到安排了新的传输之后。

CAL 传输线程与原始数据线程相似。两者之间的主要差别在于 CAL 传输线程处理不同的发送队列。CAL 传输线程根据 CAL 请求事件工作。 CAL 请求事件一般作为应用 304 的请求结果而被安排。这些应用请求被调度 320 放到 CAL 发送队列上。该 CAL 传输可以用于读（GET）和写（SET）储存在远端节点中或节点数据库 308 中的状态信息。CAL 传输也可以由内部的和外部的状态变化来产生，所述状态变化触发由规则引擎 314 翻译的规则中定义的事件。

空闲线程在系统中工作在低优先级并且是可选择的。如果被选用，该空闲线程处理在较高优先级线程执行时被不确定地延迟而可以不受影响的低优先级任务。低优先级任务一般包括以下操作：1）储存器管理和垃圾收集任务；2）发送看门狗/连接测试程序（Ping）包来检测和老化无响应节点；和3）使高速缓存数据库信息与非高速缓存（持久的）备份储存同步。如果支持空闲线程，则调度 320 提供到其它模块的接口，这样可以计划空闲线程调用低优先级任务。

设备驱动 322、326 接收来自调度 320 的一般请求（由调度控制块或 DCB 描述），并把这些请求转换为适合于潜在的网络节点的 I/O 请求。

设备驱动 322、326 也处理由网络硬件接收的包，并创建 DCB，该 DCB 然后被传递到调度 320 用于进一步处理。网关 300 与设备驱动 322、326 仅通过调度线程部分接口。设备驱动 322、326 提供的到调度 320 的接口包括传递一个 DCB 的一个驱动发送（DriverSend）接口。介质抽象组成（MAC）模块建立一个 MAC 头，并起始实际的到网络控制器硬件的包发送。一驱动中断/轮询在有效负荷接收队列上的例程队列接收事件，该事件然后由接收有效负荷线程在后边的阶段处理。

以下的大部分论述，假设设备驱动驱动一个基于 PLX 的节点。但是，提供到较高层的接口是通用的，这样网关也支持其它的设备驱动。提供调整（shim）328、330 以允许网关 300 无隙地支持基于非 PLX 的节点如 X-10 和当地 CEBus 节点。

当一个驱动，如 PLX 控制驱动 323 安装后，一个驱动初始化模块，把驱动所服务的网络接口硬件置于一个功能状态。初始化完成之后，驱动 323 和硬件就准备好发送和接收包。驱动初始化部分一般包括设置一接收处理 ISR/轮询例程，确定/保留硬件资源，以及开始任何管理线程。一般一个时效线程用于检查传输超时和设备驱动/硬件死机问题。

发送时，调度 320 使用一 DCB，把包/片段数据地址以及控制信息

提供给驱动 323。驱动 323 然后创建适当的 MAC 头，并起始传输。能够同时处理的传输数量是一个依赖与硬件/固件的值（在此称为 TxFreeCount），该值可以在初始化时读出。

在把传输数据复制到网络控制器后，DriverSend 例程启动一个发送时效计时器。此时，除非同步地收到表明传输完成的响应（在发送之后，DriverSend 例程返回之前），该传输将一直处于一种“悬挂”状态，直到网络硬件指示传输完成状态。当传输处于悬挂状态时，只要网络控制器还有可用的缓存空间，这由 TxFreeCount 定义，就可以继续进行更多的发送。一旦悬挂的发送数量等于 TxFreeCount，DriverSend 例程就被锁住（意味着不可以出现更多的发送），并且保持锁住状态，直到接收到传输成功状态或传输超时状态之后，ISR/轮询例程为它解锁。

驱动中断/轮询服务例程（或 ISR 处理器）以最高优先级在系统中运行（尤其是中断应用环境中），并负责解释/去除接收的包的 MAC 头，并且使用调度 320 排队包，以被适当的有效负荷处理器用于后期处理。接收包为以下类型之一：1）内部（或本地）控制/状态包；2）CAL 命令/请求包；3）CAL 立即响应包；4）CAL 延迟响应；5）原始流数据。

内部控制/状态包主要用于表示传输完成或错误状态。在 ISR 处理器中传输状态包的处理，一般根据状态所用的传输包类型（如 CAL/原始、本地/远端）的不同而不同。

对于原始片段和 CAL 包，如果状态指示传输已经成功完成，则 ISR 例程调整 TxFreeCount 以允许下一个发送进行。万一发生超时错误，CAL 传输首先将重传适当的次数直到发送成功，或达到最大重传次数而放弃。

对原始数据片段传输超时的处理与对 CAL 传输的处理相似，例外的是，在一些情况下（取决于原始数据流的特性和接收方的设置），可能需要一些附加的重传处理。原始流片段一般使用包排队来协助管

理数据流。根据接收原始流应用的需要，对于超时重传码，不仅需要重传超时的传输，而且需要重传额外的悬起的原始传输（那些具有较高序列号的），而不管这些悬起的传输是否成功结束。这将协助确保接收的应用或路由处理器将以升序接收包。接收的应用/路由器将丢掉无序的包。

CAL 请求/响应和原始流包由 ISR 处理器排队，然后由接收有效负荷线程和适当的有效负荷/协议处理器处理。

特殊的处理还被用于 CAL 包，来核实没有新的 CAL 请求被发送到 CAL 响应正在悬起的远端节点。另外，如果从远端节点接收到一个 CAL 请求，一般，在适当的 CAL 响应发送之前，不允许发送任何新的 CAL 包到该节点。

对于传统的控制网络设备驱动（如 X-10 驱动 328 和 CEBus 驱动 332），调整（shim） 323，330 提供允许这些非 PLX 驱动使用网关 300 的接口。传输时，调整（shim） 328，330 检查由调度 320 发送的 CAL 包，并把这些包转换为潜在的传统驱动/设备能识别的等价格式。接收时，调整（shim） 328，330 把接收的传统数据转换为 CAL 格式，建立一个 DCB，并把该 DCB 传给调度 320。

在发送和传输过程中，调整（shim） 328，330 一般执行一些地址翻译，把 4 字节的 DCB 地址转换为潜在的传统驱动使用的地址格式。

接收到的包可能包含 CAL 控制信息或原始流数据（可能包括打印机数据、隧道的以太网/ADSL 数据等）。这些有效负荷由有效负荷/协议处理装置 312 提供的适当的有效负荷/协议处理器处理。CAL 处理器 318 包括 CAL 翻译器，该翻译器负责解释和处理由控制网络节点接收的 CAL 请求/响应控制包。有效负荷处理器线程把 CAL 包由客户传到 CAL 处理器，用于进一步处理。CAL 翻译器也提供一些基本的 CAL 解释例程到其它处理 CAL 信息的网关组成部分。

当调度 320 调用 CAL 处理器 318 时，调度把指向 CAL 信息的一

个指针放到其它数据中，传到发出包的客户节点地址，以及所要作用的客户节点地址。除了包是由应用 302 发出的那些情况之外，这两个地址一般相同。

CAL 信息被分类为命令（一般也称为请求）或响应信息。来自客户节点的 CAL 命令，一般是状态改变通知，告诉网关 300“设置我的状态变量为一些新的值”的一些结果。特殊功能客户，如那些由应用 304 控制的客户，根据客户的请求，也发送 CAL 命令来取得或设置其它客户状态（实例）变量。这些包一般需要在下一部分所描述的特殊处理。CAL 命令包可以包含一个或多个命令。

单个命令 CAL 包的格式：

<contextID> <object#> <method/macro> [<IV> [<arguments>]]

CAL 翻译器把信息分成它的组成部分，映象方法标识符到适当的数据库访问方法，然后使用该 ID 和变量参数调用该数据库访问方法。数据库访问方法 306 执行被请求的操作，并且如果 IV 被改变，就通知规则引擎 314。规则引擎 314 评估施加于改变的 IV 的任何规则，并且如果被许可，就通过上述的事件处理器 310 计划事件通知/行动。

客户节点使用 CAL 响应包响应 CAL 命令。CAL 响应的形式为 <status token>[<returned data>], 其中<status token>是响应类型的一个一字节指示（完成、失败 false 或错误），<returned>数据是作为命令信息结果而返回的任何数据。

当接收到这些包之一时，把它与原始 CAL 命令/请求相联系。当客户立即响应请求时，这种联系相对容易。当响应被延迟时，响应复杂一些。在发送对前一个请求的响应之前，客户不可以发送 CAL 请求包到网络上，这使得这种联系成为可能。网关 300 不发送请求到处于“延迟”模式的客户。这允许网关 300，按客户储存发送到客户的最后请求（或命令）。当接收到客户响应后，这个信息允许 CAL 处理器 318 决定怎样处理返回的信息。

原始数据处理器 316 处理非 CAL 接收数据。网关 300 支持可以在不同的应用中发起的多个同时的数据流。原始数据处理器 316 区分原始数据流的方法，涉及套接字段，单独的套接字编号与每个原始数据流类型相联系。原始数据流的例子包括以太网/ADSL 网络包、打印机数据、音频流及其它类似数据。除了检查套接字编号之外，原始数据处理器 316 极少分析原始数据。包数据以及相联系的 DCB 然后被传递到负责那个套接字编号的路由处理器 355。

路由处理器 355 提供允许网关 300 由传统数据网络（以太网、ADSL、令牌环等）重定向网络数据业务到所需网络的功能。例如，使用网关 300 来重定向网络数据业务由一个传统网络到诸如 PLX 电源线网络的一个网络，提供了许多需要的特征，包括但不限于：1）允许扩展传统的以太网，来容纳基于电源线的节点，而不需要附加的以太网电缆连接；2）允许宽带数据在电源线上传送；3）对于基于电源线的网络客户，允许代理服务器性能；4）允许使用隧道技术使传统协议堆栈（TCP/IP、PX/IPX、NetBEUI 等）通过电源线。

路由处理器 355 执行的一个重要任务是地址翻译。在初始化时，路由处理器 355 获得在由网关 300 发起的 DCB 传输中用作 srcDevAddress 的 4 字节地址。路由处理器 355，把这个地址翻译成适于处理器与其通信的传统堆栈和/或驱动的形式。

以下部分将描述路由处理器 355 使用隧道技术，使网络数据包通过使用 DCB 的不同协议（如 PLX 协议）。使用隧道技术是把一个由第一个协议的包封装或包装到用于第二个协议的包内。封装的包然后通过第二个协议传过网络。到达它的目的地之后，封装的包被拆封，并现出来自第一个协议的原始包。

在初始化过程中，路由处理器 355 设置前述的地址映象/翻译表。路由处理器 355 也了解每个 DCB 支持的包/片段的最大长度。路由处理器 355 从调度 320 获得这些信息。不同的路由处理器 355 通过使用套接字编号来确定。公知的套接字地址可以为专门使用保留或在初始化过程中动态地获得。

当路由处理器 355 由 传统堆栈或驱动获得一个目标为电源线设备的发送请求时，路由处理器 355 把要发送的数据分成不大于所支持的最大 DCB 数据长度的片段。然后创建 DCB，并把序列号赋给每个片段。第一个片段总是序列号为 0，最后的片段具有序列设置的高位。可选择地，如果在其它层中校验和功能不能满足要求，路由处理器 355 可以把校验和加到包中。这些 DCB 包然后传给调度 320，用于在原始发送序列上排队。这将唤醒原始发送线程，为了在电源线上传输，该线程把 DCB 传递给适当的设备驱动。

设备驱动完成每次传输时，Tx 状态标记到 DCB 中，DCB 被返回到路由处理器 355。在所有的 DCB 被返回后，路由处理器 355 执行传统协议所需的发送完成操作。由于超时/重传问题由调度 320 和设备驱动处理，并且在许多情况下也由传统堆栈处理，路由处理器 355 可以选择不执行任何附加的重传操作。

由于路由处理器 355 可能同时由多个发送者接收数据片段，接收处理稍微复杂一些。另外，在一些情况下，片段可能顺序不对或被丢掉。路由处理器 355 有一个能够处理所有这些可能性的接收模块。一般，当有效负荷/协议处理器 316 传递一个开始 DCB/片段到路由处理器之一时，该 DCB/片段被放到为发送地址保留的接收队列上。当收到每个 DCB 后，接收者检查最后片段序列号或接收超时。

当所有片段被收到并通过完整性校验后，路由处理器 355 执行传统协议/驱动所需的步骤，来指示接收事件。这一般将涉及重组包和可能复制相关的接收数据到传统模块 352 提供的缓冲器之中。在每个接收 DCB 所描述的被处理后，DCB 被返回到自由 DCB 列表。

网关基础级 304 是基于面向对象的概念（如 Java、C++、smalltalk 等），它为不同类型的应用提供一种访问和管理节点信息的方法，所述节点信息储存于节点数据库 306、规则引擎 314、事件处理器 310 以及网关 300 提供的其它服务之中。这允许终端用户应用 302 为终端用户提供很宽范围的有用性能。例如，网关基础级 304 使独立的应用

和 Jave 浏览器程序能够列举、监视并控制节点数据库 308 中描述的节点。通过使用规则定义语言（Rules Definition Language）定义简单或复杂的规则，这些应用/程序可以定义节点之间的不同组合。通过把应用 302 自己注册为事件通知目标，也可以使应用 302 在变化发生时了解数据库变化。

尽管以上对于本发明的特定实施例进行了描述和说明，但是本领域内的熟练人员仍然可以对其进行不同的改变和改进，而不背离附录 A 之后的权利要求中所定义的本发明的范围和精神。

附录 A

PLX 协议一个廉价的、易于使用的、灵活的、可靠的、并且可伸缩的网络结构/协议，该网络结构/协议允许多个智能的和哑的节点通过一个共用的数据/控制信道进行通信。该联网协议允许网络上的任何节点把自己指定为活动的网络服务器。该活动的网络服务器轮询基于一个排队卡的客户节点。未激活的节点被自动从排队卡移去，因而减少不必要的轮询业务量。这种结构减少了冲突，而为真正的数据传输保留了带宽。该协议提供了为控制和数据联网两者所需的支持。通过在网络上分配时隙，并允许两个智能节点互相直接对话且由活动的网络服务器仲裁，为流式数据和同步数据提供支持。该活动的网络服务器也能够分配单独的数据信道，因此大量的数据业务可以独立于该主网络的运行地流动。作为活动的网络服务器的该网络节点，可以在动态的基础上更换，并且典型地由在一个静止的网络上起始一个传输请求的第一个节点决定。客户节点由使用寻址隔离模式的动态轮询进行寻址。

该 PLX 结构，包括 PLX 协议，非常适于使用在建筑内已有的电力电源导线（电源线）作为网络介质的网络。使用已有的电源线来传输数据意味着用户不需要安装网络电缆。

该 PLX 结构为网络节点提供强大的、确定性的媒体接入能力。节

点通过使用寻址隔离模式的动态轮询进行寻址。提供了一个可行的数据信道，用于诊断、变元传送和一般的数据传送的应用系统。

在一个实施例中，该 PLX 协议提供了全球单一的识别码、节点文件夹和 32 位的虚拟寻址能力。这使得该 PLX 协议与即插即用型的网络相兼容。

在一个实施例中，该 PLX 结构提供了诸如同级、多个服务器、简单的配置、安全、数据报检测、多种数据格式以及优先权计划的特征。错误检测，如 CRC 和校验和以及数据完整性能力是一些 PLX 实施例的一部分。该 PLX 结构用于智能节点、哑的节点，并且该结构用于从简单控制到复杂数据流的数据处理。

在一个实施例中，PLX 可以由状态（state）机器逻辑或一个微控制器来实施。一个流线型的低端的节点（哑的节点）可以被实施来使用整个 PLX 能力的一个子集。中端的节点，如器具，适合在此公开的该协议。高端的节点（智能节点），如 PC、PBX、内部通信/监视系统、打印机、鼠标以及其它数据密集型的节点也可以在 PLX 结构中找到适用性。

该 PLX 为数据链路层、网络层和传送层定义了操作规则。在一个实施例中，PLX 包括数据链路层的介质访问控制（MAC）部分。该 MAC 协议是管理怎样以及何时物理介质能够被每个节点访问的一组规则。在一个实施例中，该 MAC 协议使用一个减少在电源线上发生冲突的动态中心分布式令牌传递结构。

该 PLX 结构允许网络上的任何节点指定自己作为负责仲裁令牌请求的活动的网络服务器。当节点为非激活状态时，它们进入一种“静止”状态，因而减少了任何不必要的“轮询”业务量。这种结构减少了冲突，而为真正的数据传输保留了宝贵的带宽。

该 PLX 结构，在一些方面，是一个为控制和数据两方面联网所需的支持包的客户/服务器联网结构。通过在网络上分配时隙，并允许两

个智能节点互相直接对话且由活动的网络服务器仲裁，能够为流式数据和同步数据提供支持。该活动的网络服务器也能够分配单独的数据信道，因此大量的数据业务可以独立于该主网络的运行地流动。作为活动的网络服务器的该网络节点，可以在动态的基础上更换，并且典型地由在一个静止的网络上起始一个传输请求的第一个节点决定。另外，该活动的网络服务器是独立于应用服务器地被选定的。应用服务器典型地在一固定节点位置。该网络服务器可以是任何能胜任的服务器节点。

在一个实施例中，PLX 提供组合的介质访问能力，包括一个为在未激活的（静止的）网络介质上的最初访问的数据报检测算法，被为插入到一个激活的网络上的中心控制的令牌传递跟随。这有效地把多个访问与一个无冲突的、令牌传递型的环境，以及确定性的附加的益处联系在一起。在一个实施例中，PLX 用一个数据报的出现，来决定最初的介质可访问性。特别是通过匹配一个特定的前同步码/长度序列组合，来检测该数据报。

在一个实施例中，通过使用一个仅仅传递令牌到在系统上的激活的节点的、集中式动态轮询算法，PLX 减少在网络上的业务量。一旦一个节点变为非激活的，该节点就被从轮询的列表中移去。这种选择性的轮询过程，是基于节点通过一个被称为“在总线上发信号（spitting）”，把它们自己插入到轮询的列表中的能力。

这种发信号（spitting）过程，提供对于轮询列表的实时的、飞击式（on-the-fly）插入。这种发信号（spitting）过程，允许多个节点响应被视为一个单个的系统响应。这个系统响应允许活动的服务器节点（正在进行轮询的节点）来进一步把要求插入到轮询列表的特定的节点区分出来。

从轮询列表中实时的、飞击式（on-the-fly）的退出（de-insertion）插入由一个老化（aging）过程提供。经过一个预先定义的时间段后，如果他们没有使用令牌，未激活的节点最终被从轮询列表中移去（分离插入）。在一个实施例中，如果一个节点未能响应一个令牌请求，

该时效过程进一步被加速。

在一个实施例中，基于介质的带宽能力，轮询列表被设置为一个固定的容量（节点的数量）。传输具有较低的优先权的数据（如用于照明系统的控制数据）的节点，被从轮询列表中移去，目的是为传输具有较高的优先权的数据（如音频/视频流式数据）的节点让出空间。

在一个实施例中，在该 PLX 中的介质访问控制（MAC）层，通过使用一个备用的接收缓冲器和忙响应信号交换，提供一种自我调节机制。在一个实施例中，自我调节是通过提供一个 MAC 报头，在每个节点中的用于保持该 MAC 报头的一个拷贝的足够大的接收区来完成。即使一个节点被先前的包请求完全淹没，通过一个忙响应，这个被淹没的节点也能够对一个请求作出响应。该忙响应通知正在传输的节点，正在传输的节点必须截止它的包突发或序列，因此根据每个接收节点的能力协调该系统。

一个节点的加电时的自动通告特征，提供了远端数据库服务器的再同步。在一个新节点加电时，该新节点将通告它新出现在介质上。

在一个实施例中，PLX 提供优选的服务器选择和 kick-start 算法。因为 PLX 是一个客户/服务器型的结构，一个单个的节点被典型地选定来仲裁介质访问。在一个典型的电源线网络上，并非所有的节点要建立为平等的。因此，PLX 的一个实施例允许一个用户选择一个处于最中心位置（如邻近一个开关板）的节点作为优选的“活动的网络服务器”。如果该优选的服务器未激活，远端的节点可以激活该优选的服务器。一个简单的唤醒算法允许一个未激活的优选的服务器再次变为激活的。

在一个客户/服务器模型中，开始，一个节点请求访问介质的令牌。一旦一个客户节点被发给令牌，在一个特定的时间内，它可以接管该介质。在此期间，它可以与系统上的任何节点直接通信，而独立于服务器的涉及之外。在此期间结束后，介质访问控制被释放回服务器节点。因此，介质的仲裁首先以客户/服务器的方式进行，接下来是一个

对等关系的时隙。

在一个实施例中，PLX 包括一个动态的仲裁服务器。该仲裁到介质访问的服务器，基于活动性被动态地指定。当第一个要传输包的节点，发现系统是“未激活的”，并且经过唤醒优选的服务器（如果有一个存在）的几次尝试后，出现这种动态的指定，承担活动的网络服务器的职责。在 PLX 网络上，任何具有服务器能力的节点都能成为活动的网络服务器。

在一个实施例中，本网络协议提供了通过电源线介质发送和接收流式数据。在一个实施例中，流式数据包括数字语音数据。在一个实施例中，流式数据包括数字视频数据。

在一个实施例中，通过电源线介质，该网络协议被用来提供数字 PBX 型功能和/或数字内部通信功能。在家里已有的电源线上，该网络协议可以被用来扩展宽带数字联网服务（如 DSL、电缆、ISDN 等）遍及家里。

该网络协议可以同时处理和管理三种或多种联网业务：控制业务；数据业务；以及流式数据业务（流多媒体数据）。该网络协议提供区分优先权方案，以根据一个给定的节点的联网需求（如一个为语音设备决定的需求），允许有保证的访问时间。

PLX OSI 模型

图 2 中示出的 OSI 上面的五层 203—207 中的每一层，都给网络应用增加了有效的开销。如图 3 所示，PLX 使用被称为公共应用语言（CAL）的一个相对小的应用层 607，和一个相对小的传送/网络层 603，来补充下边的数据链路层 602 和物理层 601。601—603 和 607 中的每一层，都被典型地表示在 PLX 顺从性节点中。如图 3 所示，PLX 数据联网节点（智能节点），也可以包括在应用层 207、网络层 203 和传送层 204 中的传统的 OSI 网络性能（如 TCP/IP，IPX，Windows，NetWare，等）。PLX 顺从性节点典型地包含数量减少的控制信息，该控制信息仅使用 PLX 堆栈的在 PLX 节点之间传递，如包含在 601—603

和 607 层中的。

PLX 物理层

PLX 物理层 601 处理与网络硬件、网络电缆，并且典型地，包括实际的硬件本身相接口的硬件细节。PLX 物理层包括诸如调制技术、使用的频率、输出功率等属性。在一个实施例中，PLX 使用如下所述的数字电源线（DPL）技术。

PLX 数据链路层

PLX 数据链路层 602 处理与介质 100 接口的细节，如寻址能力、介质仲裁计划、间隙之间的间隔、退出算法等。数据链路层 602 典型地包括一个报头，其中包括源/目的地址、长度和错误检测/校正数据，如循环冗余校验（CRC）或校验和数据。

PLX 网络层

网络/传送层 603，有时被称为网间网层，负责为数据包在网络上由一个地点到另一个地点进行路由。在 PLX 内，网络层 603 典型地处理使用在一个 MAC 报头字段中的系统、单个节点、套接字、和网络地址字段。

PLX 传送层

PLX 网络/传送层 603，为驻留在其上面的应用层 607，提供在两个主机之间的一个数据流。传送层 603 也包括序列号和/或请求/应答型的确认信息。与 OSI 传送层 203 相比，在 PLX 中，传送层 603 被减小规模并连成一个整体，来允许控制应用。传送层 603 提供请求/应答信号交换算法、重传算法、超时算法等。PLX 几乎完全在一个 MAC 报头的控制字段中，实施网络/传送层 603。

PLX 应用层

PLX 应用层 607 处理应用的细节，并且根据正在使用的是何种传输，PLX 应用层 607 可以使用信号交换协议和/或请求/应答协议来确保包的发送。在 OSI 各层的协议中存在大量的重复字段。这种重复转换为更多的开销，使用更多的空间，并且需要附加的处理能力。在 PLX

协议中，许多 OSI 字段并不需要而被典型地省略掉。

包括在不同的 OSI 协议中的不同的元件的检测，显示出数据链路层 602 在没有上面三层时，可以作许多过滤。这种过滤是有益的，因为数据链路层 602 经常典型地被限制在也负责硬件问题的硬件逻辑中，如多个节点为相同的通信信道竞争（如多个网卡为相同的网络连线竞争）。在一个实施例中，一个特定的网络节点的网络硬件，过滤除了目的地为该特定的网络节点的数据包以外的所有的东西。在这样的一个系统下，节点仅需要对一个数据包的数据部分进行语法分析。

用于 DPL 的两个协议

PLX 优先地定义了两个协议用于数字电源线（DPL）；一个低级协议和一个高级协议。

低级协议定义。该低级协议提供了数据链路层 602 的一个限定，并限定了如何从相同的介质 100，以相对较少的联网和传送功能进行包的过滤、发送和接收。

高级协议定义。PLX 节点包含数量减少的控制信息。每一个 PLX 节点使用一个通用的应用层 607 来控制特殊的节点属性。这允 PLX 系统可以具有不管节点类型的特征。在硬件报头被分离出来后，应用层 607 译码或分析控制信息。

物理层：数字电源线（DPL）技术规格

PLX 协议是一个多用途的协议，该协议可以与许多种类型的网络介质（如数据传输系统）一起使用，网络介质包括光传输、光纤传输、射频传输系统、双绞线传输系统、同轴电缆传输系统、卫星传输系统、数字电源线（DPL）系统等。

DPL 系统，也称为电源线载波系统，使用电力供给连线（如建筑物中的 110 伏特交流（VAC）电路）来承载数字数据。在一个实施例中，PLX 协议被用于具有 DPL 的连接中，该 DPL 具有一个单个的低速率信道（350—1000kbps）、约为 5.6MHz 的低速载频、约为 80dB 或

更好的动态范围、低的带宽使用率（依赖于速率，但约为 1MHz）。

在一个实施例中，PLX 协议被用于具有 DPL 的连接中，该 DPL 具有多个速率信道（共 4-8mbps）、直至 30MHz 或更高的高速载频、和约为 80dB 或更好的动态范围。

在一个典型的 DPL 系统上，在数据之前的传输载波被典型地允许为至少 20 微秒，并且直到接收器检测不到载波为止，发送器停止发送之间的时间可以为 15 微秒或更长。

低级协议层：PLX 技术规格

从简单的控制到复杂的数据流网络，该 PLX 协议大小可调。在一个实施例中，该 PLX 协议被适配为可以权衡 Generic CAL 技术规格的大部分特征。在 EIA-600 中定义的 CEBus，是用来控制总线设备的一种工业标准控制语言。EIA-600 为用在家庭中的 LAN 中的通用应用语言提供了一个框架。Generic CAL 在 EIA-721 系列标准（包括 EIA-721.1、EIA-721.2、EIA-721.3 和 EIA-721.4）中定义。CEBus 工业委员会（CIC）定义了一个家用的即插即用（HPP）技术规格，通过为使用该语言定义“语法”规则，该技术规格具体化了该框架。

该 HPP 技术规格细节化了家庭中的产品和系统的一组行为特征，这将允许它们基于家庭的状态采取行动。例如，该技术规格区分了家庭中的不同条件，如“居住者不在”或“居住者在家并在睡觉”，来允许家庭系统采取适当的行动，如启动安全系统、关掉内部电灯，或设置温度。HPP 技术规格也包括为开发基于 Windows '95 PC 的家庭控制应用的信息。

在 EIA-600 中定义的通用应用语言，为不同工业部门（如娱乐、计算机、加热/制冷、厨房用具等）生产的家庭中 LAN 产品之间的通信提供了一个框架。

每个工业部门定义它的产品所使用语言的“应用环境”（如语法规则）。CIC 作为支持组织被创立，该组织帮助不同的工业部门开发“协

调的” 应用环境。对于那些追求具有基于可互操作的产品的 CAL 的家庭 LAN 市场的工业部门，CIC 的 HPP 是协调的应用环境的纲要。

CEBus/Generic CAL 技术规格在此全部插入作为参考。

介质访问概述

PLX 可以是具有中心控制的令牌传递计划或 DSMA/CTP 的数据报检测多访问协议的特征。因为多个对等体被允许访问相同的物理介质 100，PLX 提出了为每个节点把数据放到介质 100 上时，使用的一组通用的规则。

PLX 从不同数量的协议集成了几个特征，来创建一个单独的、高效的、确定性的环境。PLX 提供数据报检测。每个 PLX 节点可以检测介质 100 的业务量，如果介质 100 目前静止时，指定它自己。通过一个有组织的令牌传递型机制避免冲突。PLX 包括为处理到介质的访问而选择一个单独的、中心的仲裁节点的方法。该中心节点（活动的服务器）负责确保在一个活动的系统上有一个令牌。PLX 使用选择性的动态轮询来提供设计的简化、实施的简易性、无冲突的访问、系统的接收和随后释放令牌、和用于数据的可靠传递（请求/应答）的一个确认序列。

当节点是“未激活”时，PLX 提供维持“静止的”介质 100 的能力。典型地，在 PLX 中，只有“激活的”节点在介质 100 上通信。为即插即用功能，PLX 也提供一个全球寻址计划，和隔离多节点争用介质 100 的一种算法。

PLX 也为流应用提供时间确定机制、或确保的时隙，和为快速回转时隙（times）而提供减少的单元长度（包长度）。

PLX 提供多速率支持、热交换、确认（athentication ）和安全、控制和管理包。

此外，在高层协议中，PLX 提供许多控制联网的功能。结果，通

过应用许多不同拓扑的先进功能，介质访问的方法被高度地优化。

介质访问方法

介质访问方法列出了用于获取对介质 100 的访问的规则。PLX 获取对介质 100 的访问的方法典型地包括三步：

1. 数据报检测或“侦听”；
2. 在总线上发信号（spitting）；和
3. 中心控制的令牌传递。

根据在系统上出现的令牌，节点具有作为活动的网络服务器节点或作为一个客户节点的特征。在一个 PLX 系统上，对介质 100 的起始访问是通过侦听活动来完成，然后自我指定为活动的网络服务器，最后是活动的网络服务器的系统的中心控制的令牌传递。

图 5 是展示介质访问算法的一个流程图，PLX 用该算法来仲裁哪个节点被允许在介质 100 上“通话”。图 5 中的流程图由一个加电和通告过程框 801 开始，其中每个节点在加电后，通告它在介质 100 上的出现。通告完成后，过程前进到一个判定框 802。该节点（空闲的）在判定框 802 处循环，直到接收到一个传输（Tx）准备好的命令，据此，过程前进到一个判定框 803。如果在判定框 803 中，节点不在排队卡上或者节点是活动的服务器，过程前进到一个数据报判定框 804；否则，过程前进到一个判定框 816。在判定框 816 中，如果该节点接收到令牌，那么过程前进到一个传输包框 814；否则，过程前进到一个超时（timeout）判定框 810。在判定框 810 中，如果未发生超时，则过程返回到判定框 816；否则，过程前进到数据报检测框 804。在传输包框 814 中，过程传送一个传输包并前进到一个轮询框 815。在轮询框 815 中，活动的网络服务器轮询激活的节点，如与图 7 相关的描述的那样，或者返回，如果节点是客户。完成轮询框 815 后，过程前进到一个判定框 802。

在数据报检测框 804 中，节点侦听介质一段特定的时间，然后前进到一个判定框 805。在过程框 804 的侦听阶段中，如果介质醒来，则过程前进到一个 LIP 请求判定框 806；否则，过程前进到过程框 812。

在过程框 812 中，节点发送一个“唤醒”包，并前进到一个判定框 814。在判定框 814 中，如果已经发送了三个唤醒包而未得到响应，则过程前进到一个自我指定框 813；否则，过程返回到数据报检测框 804。在自我指定框 813 中，节点指定自己作为活动的服务器节点，并且过程前进到传输包框 814。

在 LIP 请求判定框 806 中，过程检测 LIP 请求的出现。如果没有 LIP 请求出现，过程前进到一个超时判定框 809，否则，过程前进到一个过程框 807。在超时判定框 809 中，过程检测来看一个特定包的时效期间是否已经过去。如果该期间已经过去，则过程返回到判定框 802；否则，过程返回到 LIP 请求判定框 806。

在过程框 807 中，节点在总线上发信号（spit）然后前进到一个判定框 808。在判定框 808 中，过程检测来看该节点是否已经被选中（draft）。如果节点被选中（draft），则过程返回到接收令牌判定框 816；否则，过程返回到 LIP 请求判定框 806。

框 802、803、810 和 814-816，是中心控制的令牌传递算法的部分。框 804、805 和 811-813，是数据报检测（侦听）算法的部分。框 806-809 是在总线上发信号（spitting） 算法的部分。

如图 5 所示，开始的访问介质 100 是根据介质是“睡眠的”或“醒来的”，通过两种不同的方法之一来完成。如果介质是睡眠的，一个期望访问的节点将自我指定为活动的网络服务器。如果介质 100 是激活的（也就是说，正在被一个活动的网络服务器使用），则一个期望访问的客户节点将向该活动的网络服务器请求访问。该活动的网络服务器维持一个发出访问请求的客户节点排队卡。通过被称为“在总线上发信号（spitting）”的一个过程，一个客户节点请求被放到排队卡上。

典型地任何具有服务器性能的节点，都可以指定自己作为活动的网络服务器，但是，在一个给定的节点中，并不要求包括服务器性能的特性。

一旦一个活动的网络服务器被选定，它必须能够建立并维持包括一个要被轮询的激活的节点列表的一个“排队卡”。当所有激活的节点变为静止后（通过一个老化过程），活动的网络服务器释放作为活动的服务器的当前状态，并且介质 100 再次变为静止的（睡着的）。典型地，活动的网络服务器是被一个要在介质 100 上传输的节点自我指定的。

当一个节点沉默一段时间后，该激活的节点被从排队卡上移去。当一个具有较高优先权数据的节点需要访问排队卡时，激活的节点也将被从排队卡上移去。排队卡典型地有一个最大的时隙（slots）数量。换句话说，排队卡有一个最大的可以进入排队卡的节点数量。时隙（slots）数量一般由在介质 100 上可用的带宽和不同的节点所需的带宽决定。如果 N 是排队卡中时隙的最大数量，t 是一个特定的节点可以保持令牌的最长时间（以毫秒计），那么每个激活的节点至少约每隔 N*t 毫秒获得一次令牌。因此，排队卡提供决定机制，在此机制中一个激活的节点将在一个规则的、可预测的基础上得到轮询。

例如，流式视频数据具有比流式音频数据更高的优先权。因此，如果 N 个流式视频节点已经进入到排队卡上，一个流式音频节点请求进入到排队卡上将被拒绝。但是，当该流式音频节点每次请求进入到排队卡上时，将被发给令牌。这说明了排队卡的一个属性。列在排队卡上的节点将被自动轮询，因此将在一个规则的基础上获得令牌而不必请求令牌。未列在排队卡上的节点，只有在请求令牌后或一个请求被放到排队卡上后，才收到令牌。

一个特定的节点提供的数据的优先权，决定于与在如下所述的节点文件夹目标中共同描述的网络级别字段。对于一个特定的节点的网络级别，也可以在该节点地址（设备类型字段）的最高四位中找到。

节点信号标志

每个 PLX 节点管理两个本地信号标志，这些信号标志反应系统当前的状态和在系统中节点的涉及。这些信号标志帮助节点决定是否需

要开始侦听过程。典型地，节点管理这两个信号标志，因为它们被用来获得到介质 100 的访问（当节点要传输时）。

第一个信号标志反映“系统状态”。系统的状态是“正在使用（awake）”或“静止(asleep)”，这决定于介质 100 是否处于激活的状态（如，在介质 100 上检测到包）。

第二个信号标志被称为“本地节点状态”。该本地节点状态反映一个节点的三个可能的状态之一，如下：（1）该节点是一个活动的网络服务器节点；（2）该节点是一个激活的客户节点，或（3）该节点是一个未激活的客户节点。本地节点状态说明一个节点是否应开始侦听算法，该节点是否目前在排队卡上（正在被轮询），或该节点目前是活动的服务器。

“系统状态”信号标志

每个节点对系统是否正在使用或处于静止得出各自的结论。这个结论是基于排队插入请求包（LIP）在介质 100 上的出现。当一个节点检测到一个 LIP 包时，系统状态信号标志变为正在使用。如果经过一段时间后，LIP 包未被检测到，该节点把系统状态切换为静止。这含义是，如果一个活动的网络服务器存在，它应该不时地发送 LIP 包来使客户节点保持于正在使用状态。

一个节点使用这个信号标志来决定是否它必须侦听介质 100。只有当系统状态为静止时，节点才需要通过一个侦听过程来争用介质 100。

“本地节点状态”信号标志

在一个客户节点的最后传输后，活动的网络服务器，将持续一至十秒，分配令牌（轮询）到目前在它的排队卡上的这个客户节点。在此时，活动的网络服务器确定该节点完成了传输，并使该节点“老化”，从排队卡上去掉。客户节点必须能够检测这个。当客户节点正在接收到令牌时，它被认为是激活的。当客户节点目前没有接收到令牌，它被认为是未激活的。只有当通过一个称为“在总线上发信号(spitting)”的处理过程，被活动的网络服务器插入到排队卡中后，一个未激活的

客户才能在介质 100 上传输。下表 A1 列出了可能的节点信号标志状态，和对于在介质上传输来说每个状态的含义。

系统状态	节点状态	下一步传输行动
正在工作	激活的	在排队卡上：等待令牌
正在工作	未激活的	不在排队卡上：在总线上发信号（Spit）
静止	激活的	不良状态：侦听，然后指定作为服务器
静止	未激活的	侦听，然后指定作为服务器

表 A1. 有一个准备好的新的传输的节点的下一步行动

数据报检测或“侦听”

上文所讨论的系统状态信号标志，是决定一个节点是否开始侦听的主要因素。它也是决定节点是否应把自己指定为活动的网络服务器，或是否接受作为一个客户的顺从角色的主要因素。典型地，侦听只在开始在一个静止的系统上传输前执行。如果有任何节点在介质 100 上传输，则一个活动的网络服务器早已被选定来发送 LIP 包和仲裁令牌的分配，并且系统正在使用中。如果系统正在使用，节点应作为一个客户。

当一个节点确定它有一个包准备好发送到介质 100 上，并且系统状态信号标志是静止，该节点执行一个侦听过程来决定它的下一步，并在这个起始过程中把冲突减到最小。这应是在 PLX 网络上，两个节点可能为介质 100 竞争的仅有的期间，并且会发生可能的未知冲突。因而，提供了一个加强的补偿算法。

有两个在侦听中寻址的可能的情况：（1）节点刚加电并需要传输它的“通告”或“CAL-ping”包，来宣布它加入到当前系统；或（2）节点是未激活的并正在尝试唤醒系统。在其中任一情况，如果一个服务器在侦听时被检测到，该节点应立即开始查找一个 LIP 包。一个 LIP 包将允许该节点插入到活动的网络服务器的排队卡上，并进行随后的

令牌传递和节点传输。

开始“侦听/连接测试程序（Ping）”通告

一旦一个节点加电，则通过发送一个广播 CAL-ping 包，该节点通告其在系统中的出现。通过“推进”信息而不是总是尝试去“取出”信息，这样使得自动发现机制更加强大。因为刚刚加电的节点，没有关于系统的历史记录，与一个普通的唤醒过程相比，它的侦听算法稍有不同。

在广播一个 CAL-ping 包之前，初始的侦听可能持续 800ms。这是通过对业务实际侦听一段限定的时间，然后在那段时间内随机地传送一个广播唤醒包三次，如果该节点存在，允许优选的服务器轮询这个节点。这个序列重复三次，在每次结束时，广播一个 CAL-ping 包到所有节点，表明已成功地进入到系统上。侦听/连接测试程序（Ping）过程的这个序列由伪代码给出如下：

1）

a）侦听介质 100，持续一个小于 125ms 的随机数量的时间长度（查找一个 LIP 包）。

b）发送一个广播唤醒包三次，其中间隔 600μs 间隙空间。

c）继续侦听，以完成一整个 125ms 的时间段。

2）

a) 侦听介质 100，持续一个小于 125ms 的随机数量的时间长度（查找一个 LIP 包）。

b) 发送一个广播唤醒包三次，其中间隔 600μs 间隙空间。

c) 继续侦听，以完成一整个 125ms 的时间段。

3）

a) “侦听”介质 100，持续一个小于 125ms 的随机数量的时间长度（查找一个 LIP 包）。

b) 发送一个广播唤醒包三次，其中间隔 600μs 间隙空间。

c) 继续“侦听”，以完成一整个 125ms 的时间段。

4）指定作为活动的网络服务器，并且发送一个“CAL-ping”包来表明出现。

5）取消指定作为活动的网络服务器。

以上侦听/连接测试程序(ping)过程在节点加电后立即发生，因此这个过程所用的执行时间一般并非很长。如下所述，运行时的唤醒过程，经常被执行，因此理想地具有一个短的执行时间。

运行时“侦听/唤醒”序列

一旦一个节点加电后并在系统上通告它的出现，该节点开始工作在一种运行时模式中。在该节点运行时模式工作中，如果一个节点需要传输一个包到一个静止的系统上，它经过一个相似的事件序列，来尝试并唤醒一个优选的服务器。如果优选的服务器不存在，并且没有活动的网络服务器存在，那么该节点指定它自己作为活动的网络服务器，并开始轮询客户节点。为侦听/唤醒算法的一个伪代码列表给出如下。在如下给出的算法之外，为了更快的响应时间，节点也可以选择性地监视介质 100 并使用本地节点信号标志来反映系统的状态。该本地节点信号标志与唤醒包一起使用来进一步减少与这个过程相关的等待时间。

1）

a）侦听介质 100，持续一个小于 125ms 的随机数量的时间长度（查找一个 LIP 包）。

b）发送一个广播唤醒包三次，其中间隔 600μs 间隙空间。

c）继续“侦听”，以完成一整个 125ms 的时间段。

2）

a) 侦听介质 100，持续一个小于 125ms 的随机数量的时间长度（查找一个 LIP 包）。

b) 发送一个广播唤醒包三次，其中间隔 600μs 间隙空间。

c) 继续侦听，以完成一整个 125ms 的时间段。

3）

a) 侦听介质 100，持续一个小于 125ms 的随机数量的时间长度（查找一个 LIP 包）。

b) 发送一个广播唤醒包三次，其中间隔 600μs 间隙空间。

c) 继续侦听，以完成一整个 125ms 的时间段。

4）指定作为活动的网络服务器，并且发送一个“CAL-ping”包来表明出现。

5）取消指定作为活动的网络服务器。

在总线上发信号（Spitting）

当系统上的一个节点有一个包准备好要发送，并且系统处于工作状态（一个活动的网络服务器存在并正在分配令牌）时，“发信号（spitting)”过程发生。该活动的网络服务器是被授权来允许在介质 100 上访问的唯一节点。该活动的网络服务器的排队卡是一个机制，通过这个机制，未激活的客户节点可以获得到介质 100 的访问。各节点发信号（spit）来进入活动的网络服务器的排队卡。

在典型的运行时工作期间，网络将呈现两种状态之一：静止状态或工作状态。根据网络当前的状态，发信号（spitting） 过程稍有不同。

静止和工作状态

当活动的网络服务器确定没有节点目前需要服务（发送包），网络进入静止状态，并且作为结果，停止传送令牌。使网络停止活动之前，在一个特定的时间内，活动的网络服务器发送一系列掩码组 LIP(LIPG) 请求包。如果 LIPG 请求包序列没有得到来自任何客户节点的响应，活动的网络服务器变为未激活状态，并且网络变为静止状态。请求传输的节点到网络上的后续进入，随后通过上文所述的通常的争用处理、侦听算法来完成。

工作状态象征特定网络上的节点，这些节点正在与一个或多个远端节点交换信息。在工作状态，介质访问由活动的网络服务器和它的排队卡控制。通过使用为目前在排队卡上的节点的一个令牌传递计划，和通过为尝试进入到排队卡上的节点的发信号（spitting），减少了冲突。

“在总线上发信号（spitting）”序列

在总线上发信号（spitting）的序列允许活动的网络服务器周期性地发送一个 LIPG 包。静止的客户节点被允许响应 LIPG 包。一旦发现一个响应，该活动的网络服务器发送一个未掩码的 LIPD 请求到所有的节点，希望得到带有需要令牌的节点的地址的一个单独响应。如果多于一个节点竞争令牌，将见不到响应，并且活动的网络服务器进入一个节点隔离序列。

图 6A 和 6B 分别说明了一个活动的网络服务器和客户节点在总线上发信号（spitting）的过程。在图 6A 中，一个活动的网络服务器在总线上发信号（spitting）的过程，始于起始框 901，此时开始一个节点变为活动的网络服务器。该过程由起始框 901 前进到一个轮询框 902。在轮询框 902 中，活动的网络服务器轮询当前在排队卡上的所有节点。一旦轮询结束，该过程前进到发送框 903。在发送框 903 中，活动的服务器发送一个未掩码的 LIP 请求，然后前进到一个判定框 904。在判定框 904 中活动的服务器检测 LoGI 响应。如果收到一个 LoGI 响应，则过程前进到一个过程框 905；否则，该过程返回到轮询框 902。

在过程框 905 中，该活动的网络服务器发送一个未掩码的 LIPD 请求，然后前进到一个判定框 906。 在判定框 906 中，活动的服务器检测一个直接 ACK（DACK）响应。如果一个单独的 DACK 响应被接收到，则该过程前进到一个过程框 907。如果多个 DACK 响应被接收到，或者如果没有接收到 DACK 响应，则该过程前进到一个节点隔离框 910。在过程框 907 中，发送 DACK 响应的客户节点被加到排队卡上，然后过程返回到轮询框 902。

在过程框 910（开始节点隔离算法）中，过程初始化一个 LIPG 掩

码，并前进到一个过程框 911。在过程框 911 中，该掩码被更新（如在掩码中的一个下一位被切换），并且过程前进到一个发送框 912。在发送框 912 中，一个掩码的 LIPG 请求被发送，并且过程前进到一个判定框 913。在判定框 913 中，过程检测一个 LoGI 响应。如果一个 LoGI 响应被接收到，过程前进到一个判定框 915，否则，过程前进到一个过程框 914。在过程框 914 中，在过程框 911 中最近被切换的掩码位，被恢复到切换前的值，并且过程前进到一个判定框 915。

在判定框 915 中，如果掩码中所有的位已经被切换，过程前进到一个过程框 916，否则，过程返回到过程框 911。在过程框 916 中，活动的网络服务器发送一个掩码的 LIPG 请求，并前进到一个判定框 917。在判定框 917 中，如果一个 DACK 响应被接收到，则过程前进到一个过程框 907；否则，过程返回到轮询框 902。

过程框 903-907 是一个服务器发信号（spitting）起始的序列的部分。过程框 910-917 是一个服务器发信号（spitting） 节点隔离序列的部分。

图 6B 是展示客户发信号（spitting）算法的一个流程图，在一个正在工作的网络上的一个客户，由一个开始框 931 开始。由开始框 981，过程前进到一个判定框 982，在此检测发送状态。如果该发送状态是“已准备好”，则过程前进到一个判定框 983；否则，过程前进到一个空闲框 988（该空闲框返回到判定框 982）。

在判定框 983 中，如果节点已经接收到系统令牌，则过程前进到一个发送框 989；否则，过程前进到一个判定框 984。在发送框 989 中，该节点发送一个数据包，然后过程返回到判定框 982。在判定框 984 中，如果该节点接收到一个 LIPD 请求，则过程前进到一个过程框 990；否则，过程前进到一个判定框 986。在判定框 986 中，过程检查超时或系统静止状态。如果过程检测到超时和静止，则过程前进到一个过程框 987，在此当前的节点指定它自己作为活动的服务器。

在过程框 990 中，由 LIPD 的掩码与当前节点的节点地址相比较，并且过程前进到一个判定框 991。在判定框 991 中，如果掩码与该节

点匹配，则过程前进到一个响应框 992；否则，过程返回到判定框 982。在响应框 992 中，该节点响应网络服务器（恰当地带有一个 LoGI 或 DACK），并且过程返回到判定框 982。

组 LIP（LIPG）查询

当网络正在工作时，活动的网络服务器定时地广播组 LIP 查询。一个组 LIP（LIPG）查询，要求从任何数量的节点的一个逻辑的组隔离（LoGI）响应。在一个无冲突的机制中的一个正在工作的网络期间，这种机制给客户节点一个机会来被插入到排队卡上。LoGI 包的优点在于，多个节点可以同时发送这种包（假设它们在同一个时间段上），结果将是一个单独的 LoGI 包。因此，由接收的节点看来，多个 LoGI 响应结果是一个单独的 LoGI 包。

起始的 LIP 序列包是一个未掩码的组 LIP（LIPG）查询，该未掩码的组 LIP（LIPG）查询被发送来确定是否网络上的任一节点要开始 LIP 序列以插入到排队卡。如果发现一个 LoGI 响应，可能仅仅是一个单个的节点要插入，因此下一步发送一个未掩码的直接的 LIP（LIPG）包。如果未发现一个直接的响应，则发送带有一个掩码地址的作为组包的后续的 LIPG 包。这是一个艰巨而低效的隔离机制，用来隔离一个特定的节点以插入到排队卡。这是通过系统地传送一个位掩码实现的，该位掩码一次隔离远端节点 32 位地址的一个单个的位。如果两个或多个冲突的节点同时请求令牌，必须执行这种隔离机制。

直接的 LIP（LIPD）查询

直接的 LIP（LIPD）查询被作为由一个 LIPG 查询的一个 LoGI 响应的结果发送。该 LIPD 查询的目的是加速 LIP 过程，这通过发送一个未掩码的 LIPD 请求到所有节点，希望只有一个单个的节点响应（这应是大部分时候的情况）。这个 LIPD 包由包括响应节点的地址的一个普通的 DACK 响应来响应。如果是一个单个的节点响应，则响应被发现，并且该节点的地址被恰当地加入到排队卡上。但是，如果 LIPD 请求未被发现，（由于多个节点同时响应）则活动的网络服务器继续隔离，通过普通的隔离算法，使用 LIPG 包来只选择竞争的节点中的

一个，插入到“排队卡”。

因此，该 LIPD 包仅仅用来加速隔离过程，希望仅仅有一个单个的节点响应请求。

节点隔离序列

如果一个节点响应起始的 LIPG，但是由于某种原因由 LIPD 查询未发现一个单个的响应，则活动的网络服务器自动地进入节点隔离。隔离序列使用要求一个 LoGI 响应的 LIPG 包。这允许多个同时的响应被活动的网络服务器发现。

通过发送带有第一个地址（最低有效的）位组的一个包，“活动的网络服务器”开始这个序列。当且仅当这个特定的地址位与它们自己的相匹配时，需要发送的节点响应这个包。这一算法是一种简单的“与”，随后和原始的掩码相比较。如果两个值相匹配，则用一个 LoGI 响应这个包。

该活动的网络服务器，然后发送具有未用的预先匹配的掩码的下一个包，这个包带有下一个位组。再次，当整个位序列匹配时，节点将响应。如果没有节点响应，活动的网络服务器清除当前位，并重传这个包。这个过程将继续直到所有 32 位被识别并发现一个匹配。这时，单独地被识别的节点被加到活动的网络服务器的排队卡。

中心控制的令牌传递（轮询）

当系统正在工作时，希望给包括在排队卡上的每个节点（通过发信号（spitting） 过程）一个确定性的时隙，在此时隙中节点可以访问介质 100。进一步还希望给每个节点相同的机会在繁忙的介质 100 上发送。以太网缺乏上述的两个优点中的任何一个，而令牌环具有以上两个优点。

令牌环具有一个缺点，它需要每个节点了解它的上行和下行邻居的地址，并且需要一个令牌持续存在/循环。传统的令牌环网络的开销需求，与 PLX 所关注的的哑节点不兼容。而且，一个电源线网络的

特别的联网需求，也并非有益于这样严格的令牌循环。因此 PLX 引入具有一个动态的排队卡的中心控制的令牌传递（CTP）机制。

在 CTP 中，活动的网络服务器节点负责确保一个令牌存在、每个需要令牌的节点得到它、静止的节点可以醒来并接收令牌、并且令牌被以一个确定性的方式公平地分配。在 CTP 下，活动的服务器以外的节点被作为客户来引用。活动的网络服务器的职责，是通过前述的数据报检测或侦听过程，自我指定的。经过一段预先确定的在介质 100 上无活动的时间段后，活动的网络服务器的职责被释放。在一个实施例中，活动的服务器的职责，经过无活动的约 5 秒钟后，被释放。在系统活动中，该活动的网络服务器负责轮询在排队卡中的每个客户节点，也允许新节点有机会通过发信号（spitting） 过程把它们自己插入到排队卡中。

图 7 是展示网络服务器轮询算法的一个流程图，由开始框 1001 开始，在此一个节点变为活动的服务器。过程由开始框 1001 前进到一个判定框 1002，在此过程确定发送一个周期性的 LIP 包的需求。如果需要一个 LIP 包，则过程前进到过程框 1010，否则，过程前进到一个过程框 1003。在过程框 1010 中，节点执行与图 6A 一起描述的活动的服务器发信号（spitting） 过程。在完成过程框 1010 后，过程前进到过程框 1003。

在过程框 1003 中，过程获得在排队卡中的下一入口，并前进到一个判定框 1004。在过程框 1004 中，如果排队卡上的所有入口都已处理过（也就是，如果所有的客户节点都已经有一次机会讲话），则过程前进到一个过程框 1011；否则，过程前进到一个过程框 1005。在过程框 1011 中，令牌被分给活动的服务器（因此允许活动的服务器讲话），并且过程前进到过程框 1005。

在过程框 1005 中，令牌被发给从排队卡上得到的下一个节点，并且过程前进到一个判定框 1007。在判定框 1007 中，如果一个响应超时发生，则过程前进到过程框 1012；否则，过程前进到一个判定框 1007。在判定框 1007 中，如果客户节点没有使用令牌，则过程前进

到过程框 1012。在过程框 1012 中，一个激活的节点的计数被减少，并且过程前进到判定框 1008。

在判定框 1008 中，如果所有的节点都是未激活的，则过程前进到一个过程框 1009；否则，过程返回到判定框 1002。在过程框 1009 中，活动的服务器回复为一个未激活的客户节点。

包的类型和格式

一个 PLX 网络上的包，可以根据包的目的使用不同的格式。不同的包的格式可以方便地分组为三个不同的类别。

一种格式允许多个节点同时发送/接收相同的响应包而没有干扰或解调问题。这些包被称为逻辑组隔离（LoGI）包，并且主要被用来广播/重广播和确认。

其它两种类型的包，被称为原始数据有效负荷包和命令有效负荷包，当一个单个的节点在任何给定点在线上通信时使用这两种包。一个原始数据有效负荷包，被一个需要发送/接收与它的应用有关的信息的应用使用。来自主节点的包是原始数据有效负荷包，任何 CAL 包也是原始数据有效负荷包。

一个 PLX 命令有效负荷包被用来管理介质访问和流。PLX 命令包在适配器的固件和硬件中产生和终结，并且不被传递到主节点上。PLX 命令包便于令牌、确认、排队插入等的平滑流动，是所有的 PLX 网络所固有的。

逻辑组隔离（LoGI）响应包

当一个节点发出一个组请求（一个可能得到多个同时响应的请求）到网络上时，使用第一种形式。因为 PLX 希望是一个较少冲突的、或在一些情况下无冲突的环境，所以很难检测到冲突。因此，同时响应是可能的。如图 8 所示的 LoGI 包 1100，包括一个两字节的 NULL 字段，跟着多个两字节的全“1”字段，并由一个两字节的 NULL 字段结束。在这种包中的数据是非常秘密的，但是它确实达到它的帮助

分离组响应到一个单个的节点的目的。

一个掩码的 LIPG 包一个在 LoGI 包之前。掩码意味着多于一个节点可以与掩码的地址匹配，因此，多个同时响应可能发生。LIPG 包在后文描述。

通过加长一个特定包中的 1 序列，LoGI 包也可以包含一些非常简单化的数据。加长的包必须与一个时间位移一起使用，来标识不同类型的响应。广播包使用这种特性，来允许由一个或多个节点以同时的方式标识的一个忙的响应。

有效负荷包

第二种形式被用来在网络上承载一个有效负荷。这是用在网络上的最普通的形式，并且是用于发送和接收有用的数据信息的有效的形式。

有效负荷包使用另外的两种形式，这两种形式指明接收者的范围以及它们期望收到何种形式的响应。它们具有组寻址（典型的广播包）和直接寻址的包类型。因为只期望一个单个的响应，所以组寻址的包只能接收LoGI响应包，而直接寻址的包接收直接的ACK确认或DACK包。

有效负荷包类型进一步细分为两个不同的类型，这两种不同的类型决定在包中有效负荷的使用。它们是：原始数据包和 PLX 命令包。

原始数据包

图 9 示出了一个原始数据包 1200 的格式，格式中包括一个前同步码字段 1201、一个长度字段 1202、一个长度字段 1203、一个控制字段 1204、一个目标地址字段 1205、一个源地址字段 1206、一个序列字段 1207、一个确认字段 1208、一个 DSK 字段 1209、一个 SSK 字段 1210、一个有效负荷字段 1211 和一个 CRC 字段 1212。原始数据包 1200 由一个活动的服务器节点或客户节点发送。长度字段 1202、长度字段 1203、控制字段 1204、目标地址字段 1205、源地址字段 1206、

序列字段 1207、然后为确认字段 1208、DSK 字段 1209 和 SSK 字段 1210，是一个 MAC 报头 1215 的组成部分。有效负荷字段 1211 包括要被一个适当的有效负荷处理器分析的应用层信息。主机 PC 和 CAL 译码器是有效负荷处理器的例子。在一个实施例中，原始数据包 1200 有一个三字节的前同步码字段 1201，一个 13-15 字节的 MAC 报头 1215、一个直至 255 字节的有效负荷部分 1211 和一个 2 字节的 CRC 1212。

PLX（外部的）命令包

通过为两个节点通过简短的包序列通信提供一个方法，PLX 命令包被使用以便于数据流上下介质 100。关于 PLX 命令包的变化的描述如下：

令牌包：一个 PLX 令牌包 1300 的格式如图 10 所示，包括前同步码字段 1201、长度字段 1202、长度字段 1203、控制字段 1204、目标地址字段 1205 和 CRC 字段 1212。长度字段 1202、长度字段 1203 和控制字段 1204，分别具有 0x05、0x05 和 0x17 的（十六位进制）值。

令牌包 1300 被发送到直接的地址节点，并请求有效负荷包的两种中的一种。不需要留意的节点应简单地直接确认（DACK）（状态字段设置为 0x03），意思是它们没有任何东西要表示并且将不使用该令牌。

客户节点在正在工作的网络上传送之前，需要调用一个令牌（通过 LIP 过程）。只要一个客户节点继续使用令牌，活动的网络服务器将继续交给它令牌。但是，如果客户节点重复地以一个“令牌未被使用”的响应来回应，则活动的网络服务器将使该节点老化并且该节点被从排队中移去。

一个令牌包包括通常的 MAC 报头（减去一个源地址）和 CRC，但是，数据字段未被使用（数据字段的长度为 0）。令牌只能来自地址固定为 0xfffffffe 的“活动的网络服务器”，因此不需要源地址字段。

直接确认（DACK）包：一个 PLX 令牌包 1400 的格式如图 11 所示，包括前同步码字段 1201、长度字段 1202、长度字段 1203、控制

字段 1204、目标地址字段 1205、一个状态字段 1401 和 CRC 字段 1212。长度字段 1202、长度字段 1203 和控制字段 1204，分别具有 0x06、0x06 和 0x07 的（十六位进制）值。

一个 DACK 包被一个接收节点发送，来确认包或包序列的接收有效。DACK 包只由直接地址信息包返回（LIPD 包除外）。

一个 DACK 包被用来终结在网络上两个节点间的典型的信号交换序列，并且结果涉及三个节点...1）活动的网络服务器、2）请求的节点和 3）响应的节点。（如果活动的网络服务器是当前请求的目的地，请求/响应节点也可以是“活动的网络服务器”。）该 DACK 的状态字段根据接收包的节点类型(活动的网络服务器或客户节点)改变。DACK 包送回请求的节点（由响应的节点），释放控制给请求的节点来继续一个包流，DACK 包送回“活动的网络服务器”（由请求的节点），释放控制给“活动的网络服务器”，表示一个包流的结束。如果一个响应或 DACK 包没有接收到，请求的节点负责重复请求一个包。

该 DACK 包包括一个典型的 MAC 报头和 CRC，以及一个 1 字节有效负荷。状态字段包括被接收包的信息，并在这个字段中被返回。状态字段 1401 的值列在表 A2 中。

表 A2. DACK 状态字段 1101 的值

DACK	节点	描述
0x0	全部	接收缓冲区满（失败）
0x1	全部	失败（多信道响应）
0x2	服务器	令牌由节点使用
0x3	服务器	令牌未被节点使用
0x4	服务器	令牌响应唤醒请求
0x9	全部	打印机序列编号错误
0xa	全部	打印机未插入错误
0xb	全部	打印机脱机错误
0xc	全部	打印机一般错误

0xd	全部	打印机无纸错误
0xe	全部	打印机不能识别错误
0xf	全部	成功

应当注意，这些信息是在实际的介质 100 本身上被传递，并且可能不是传递到主机节点的状态。请见关于内部 PLX 包的部分，关于被传递到主机的状态信息的更多信息的 Tx 状态。

排队插入包（LIPD 和 LIPG）：图 12 展示了一个 PLX LIPG 包 1500 的格式，PLX LIPG 包包括前同步码字段 1201、长度字段 1202、长度字段 1203、控制字段 1204、目标地址字段 1205、一个掩码字段 1501 和 CRC 字段 1212。长度字段 1202、长度字段 1203 和控制字段 1204，分别具有 0x09、0x09 和 0x23 的（十六位进制）值。

图 13 展示了一个 PLX LIPD 包 1600 的格式，PLX LIPD 包包括前同步码字段 1201、长度字段 1202、长度字段 1203、控制字段 1204、目标地址字段 1205、一个 NULL 字段 1601 和 CRC 字段 1212。长度字段 1202、长度字段 1203 和控制字段 1204，分别具有 0x09、0x09 和 0x23 的（十六位进制）值。

排队插入包（LIP）被定时地由“活动的网络服务器”发送，以允许新接收的成员进入已有的排队卡。这是由两个单独的包来完成的，它们都被广播到正在侦听的节点。第一个包，LIPG 包 1500，包含 LIP 掩码字段 1501。在用一个 LoGI 响应回应之前，该掩码 1501 必须与远端的地址相匹配。第二个包，LIPD 包 1600，通过使响应的节点用包含它的源地址（以便插入到排队卡中）的一个 DACK 包来响应，被用来加速插入过程。

因此，LIPG 包被掩码，并且具有在 LIP 掩码字段中的一个相应的位序列。一个节点应使用一个 LoGI 包来响应 LIPG 包。相似地，LIPD 包未被掩码，意思是，任何希望进入排队卡的节点（这意味着该节点并未在排队卡上），应使用一个 DACK 来响应。

有效负荷包帧格式

以下是可能出现在一个有效负载类型的包中的每个字段的描述。这对原始数据和 PLX 命令包类型两者都成立。

尽管前同步码/开始序列并不是包格式的部分，但是它是预先确定的位模式，这种位模式被用来进行检测载波、把硬件与输入的包同步、决定位计数或在当前包内的后续字节的线速率。前同步码的长度由建立一个有效载波的出现和在线路上同步所需的位计数时间的最小值决定。前同步码 1201 的位模式为：

值	序列
0xaa	第一个同步字节
0x31	第一个同步字节
0xnn	速率/第三个同步字节

速率（或第三个同步）字节确定输入数据（由长度字节 1202 开始）的速率，概述如下：

值	速率
0x55	低速率 - 650k
0xdd	中速率 - 1000k
0x99	高速率 - 1.19m
0x11	保留

最后，该前同步码之后是两个双重的长度字节 1202-1203，1202-1203 描述包的长度。这些字节将以新的速率进入。

长度字段

长度字段 1202-1203 被用来指示输入包的长度。长度字段 1202-1203 被硬件使用（在缺少载波检测信号时）来确定有效包的接收。一旦包的长度达到后，就为有效性检测 CRC 字段 1212。一个 PLX 包的长度因此最好限制在总共 256 字节之内（不包括前同步码字段 1201 和 CRC 字段 1212）。长度包括 MAC 报头 1215（控制、地址等）、可选字段和有效负荷字段 1211。

长度字段被重复两次（1202、1203）来确保输入数据流的有效性（它作为前同步码的一个延伸）。在包接收开始前，长度字段 1202-1203 必须相互匹配（前同步码也匹配）。

控制字段

如上文所示，有效负荷包可以是以下两种类型之一：PLX 命令包或原始数据包。

PLX 命令包类型可以进一步分类为两个子类型：外部和内部 PLX 命令。内部 PLX 命令包，是指通过本地连接（USB、1584、串行等）在硬件和主机节点的驱动之间的信号交换。外部 PLX 命令包，是指在电源线介质 100 本身上的信号交换包，它管理对介质 100 的访问。

控制字段 1204 根据包的类型而改变，如表 A3 所示，每一个位表示一个特定的定义。

表 A3 控制字段 1204 中的位

位	PLX （外部）	PLX （内部）	原始 （非 PLX）
0	包_类型（1）	包_类型（1）	包_类型（0）
1	PLX_子类型（1）	PLX_子类型（0）	原始_ACK_类型 0
2	PLX_ACK_类型	保留（0）	原始_ACK_类型 1
3	保留（0）	保留（0）	密码
4	外部_子类型	内部_子类型	套接字
5	外部_子类型	内部_子类型	保留（0）
6	外部_子类型	内部_子类型	PID
7	外部_子类型	内部_子类型	保留（0）

包类型

包类型位被用来指示一个给定的包是否是 PLX 类型、原始数据类型或非-PLX 类型。因为 PLX 协议请求得到不同的处理，并在大多情况下由微控制器固件处理，并且原始数据包典型地由一个单独的应用或主机软件处理，所以在控制字段中进行区分是有益的。原始数据包，典型地包含要被传递到适当的应用软件的原始有效负荷信息。这种情

况的一个例外，是包含微控制器中的翻译器的部分和在主机中的部分的 CAL 包。

位 0	包类型
1	PLX 命令包 =1
0	原始数据包 =0

PLX 子包类型

PLX 命令典型地为两种形式之一。第一种形式是来自电线的由另一个节点的一个请求，第二种形式是来自主机的一个请求，来自主机的请求并不送到电线上。因为微控制器固件对这两种类型的响应进行区分，并且这两种类型彼此是完全独立的，所以建立这个位。

位 1	PLX 子包类型
1	外部的 PLX 命令包 =1
0	内部的 PLX 命令包 =0

PLX ACK 类型

令牌和 DACK 命令包被用来传送到介质 100 访问的权利，并终结一个序列，在此“活动的网络服务器”暂时释放对介质 100 的控制到另一个节点。另外两个 PLX 命令包，LIPG 和 LIPD，要求一个响应包。该响应类型是 LoGI 类型或 DACK 类型两者之一。这个位确定节点应使用什么类型的响应。

位 2	PLX ACK 类型
1	用一个 DACK 响应 =1
0	用一个 DACK 响应 =0

PLX 子包外部的类型

PLX 技术规格，提供在一个中心控制的（服务器仲裁令牌）令牌传递系统中，两个节点之间的无连接的、有确认的和无确认的数据传送服务。这些位允许这种通信进行。

在一个客户开始发送前，活动的网络服务器把一个直接的令牌放到介质 100 上。一个客户节点，使用直接返回活动的网络服务器节点的一个 DACK 响应包，终结到介质 100 的访问权利。当轮询客户节点时，活动的网络服务器维持激活的节点的一个排队卡。为进入到排队卡上，一个客户节点恰当地响应一个直接的 LIP 请求（LIPD）或一个组 LIP 请求（LIPG）。

节点一旦在排队卡上，它们将被轮询，并且它们可以以确认的或非确认的形式，发送和接收具有有效负荷信息的包。以下是一个在介质 100 上被允许的有效的 PLX 子包外部的类型的一个表。

位（7、6、5、4）	字节的值	包的子类型
0 0 0 0	0x07	DACK
0 0 0 1	0x17	令牌
0 0 1 0	0x27	LIPD
	0x23	LIPG
其它...		保留

注：如果一个 DACK/GACK 未被请求的节点，在预先确定的间隙间隔要求内接收到，则发送的节点（请求的或响应的）负责重传该请求（响应）。

PLX 子包内部的类型

PLX 技术规格允许在一个主机节点上，如一台 PC 上，存在协议端口。周期性地，该主机节点需要访问在所连接的节点上的信息，主机节点物理上与该节点连接。这被认为是一个内部的 PLX 请求，因为该请求是为所连接的节点而发，并且一般不应被放到电线上来被发送到一个远端的节点。以下是可能的内部 PLX 子类型的描述。

位（7、6、5、4）	字节的值	包的子类型
1 1 1 1	0xf1	信号交换错误
0 0 0 1	0x11	CAL 请求
0 0 1 0	0x21	CAL 响应

0 0 1 1	0x31	Tx 状态
1 1 x x		保留

内部的子类型被从主机发送并被硬件吸收，并且一个适当的响应被返回到主机节点。内部的包从来不被发送到介质 100 上。因而，这种包类型不在有效负荷包部分下定义，但是在 PLX（内部的）主机包定义的部分中。

原始 ACK 类型

原始 ACK 类型确定了什么类型的响应应跟随在当前的原始数据包之后。响应类型具有以下四种形式之一：突发（无响应）、双 LoGI、LoGI 和 DACK。

突发的类型是自我说明性的，包被一个接一个地发送。一个突发序列的最后一个包，应当具有一个指定的不同的 ACK 确认类型（为完成该突发序列，使用一个响应）。

一个双 LoGI 序列允许发送组或广播请求。如果一个节点不能缓存这个包，则该节点在第一个间隙间隔内响应，如果该节点正确地接收和分析这个包，则它在一个延迟的间隙间隔内响应。

LoGI 响应被导向一个单个的节点，并且是最有效的响应机制。一个 LoGI 包的长度具有最高的带宽利用率，但是不能包含关于响应的很多信息。

DACK 响应被导向一个特定的节点，但是与 LoGI 类型相比，能够包含响应内的很多信息。

位 （2、1）	包的子类型
0 0	突发
0 1	双 LoGI
1 0	LoGI
1 1	DACK

密码

密码位允许由验证字节开始的包内容被加密。一个加密方案使用一个 256 位 Diffie-Hellman 信号交换来进行密钥交换，此后，一个秘密的 32 字节矩阵被通过介质 100 安全地发送。随后的事务可以为安全通信使用加密矩阵。

位 3：密码

当前的包被加密 =1

当前的包未被加密 =0

套接字

典型地，一个 PLX 原始数据有效负荷包将包含以下字段长度。

字段	长度
前同步码 1201	3 字节
长度 1202、903	重复的 2 字节
控制 1204	1 字节
目的地址 1205	4 字节
源地址 1206	4 字节
有效负荷 1211	0-255 字节
CRC 1212	2 字节

当同一个节点上有多个应用时，通过使用一种机制来使包能够被路由到在一个特定的节点地址中的适当的应用。这些类型的应用使用一个套接字字段。第一个字节是目标套接字地址，而第二个字节是源套接字地址。因此，由于设定这个位，MAC 报头的长度增加 2 个字节。当使用时，这个字段将紧跟着确认字节字段，并且如果以下的位被置 1，则它就被包括进来。

位 4	套接字
1	包括套接字字段
0	不包括套接字字段

协议 ID（PID）

每个包都包含能够被较高级协议，如 IPX、TCP/IP 或 CAL，解析的信息。PLX 仅仅被用作传送装置，以封装要穿过网络被发送/接收的这些类型的包。典型地，较高级的解析程序存在于一个主机系统上；但是，需要硬件来包含 CAL 解析功能的一个最小设置。这样，硬件解析 CAL 请求，并把所有的其它请求交给适当的有效负荷处理程序。一些协议信息可以驻留在硬件中（如在 ROM、FLASH 存储器等之中），其它协议信息由主机节点解析。这个位决定是否需要硬件协议处理器来开始解析这个包。

位 6	协议 ID（PID）
1	协议 ID 存在（微解析）
0	协议 ID 不存在 （原始 – 主机解析）

原始包的含义是数据的第一个字节不是这种类型的协议的一个字节编码，而是协议报头本身的第一个字节。具有 PID 解析性能的包译码第一个字节编码来决定由哪个协议解析这个包。

以下是当 PID 位被设定时可用的选择。第一个数据字节代表解析当前的包所需的协议类型：

字节的值	定义	类型
0xff	保留	n/a
0xfe	完成的包	cebusResp
0xfd	失败的包	cebusResp
0xfc	错误的包	cebusResp
0xdf – 0xfb	保留	n/a
0xa0 – 0xde	前后关系编号（CAL）	cebusCmd
0x9f	保留（CAL）	cebusCmd
0x00 – 0x9e	前后关系级别（CAL）	cebusCmd

目的地址字段

目的地址字段 1205 包含当前包的目的节点。

当一个节点有一个请求或响应另一个节点的请求时，它把响应包的目的地址，放到目的地址字段 1205 中。如果该节点只能与活动的网络服务器或数据库服务器通信，它将把那个地址放到目的地址字段 1205 中。否则，目的地址一般由请求包的源地址字段 1206 中得到。

一些 PLX 地址是众所周知的。这些众所周知的 PLX 地址列出如下：

地址	描述
0x00000000-0xffffffef	有效的单个节点地址
0xfffffff0-0xfffffffc	保留
0xfffffffd	应用服务器节点地址
0xfffffffe	活动的网络服务器节点地址
0xffffffff	广播节点地址

源地址字段

源地址 1206 包含用于当前包的节点地址。当一个节点有一个请求或响应另一个节点的请求时，它把它自己的节点地址，放到源地址字段 1206 中。该节点地址使用 8 字节 GUID 的一部分，与节点的类型结合，形成一个四字节的节点地址。使用由 GUID 的最低有效的 7 个半字节，并将节点的类型盖写节点地址的最高有效半字节（第 8 半字节）。

例如：

```
If...
    GUID = 0x123456789ABCDEF
    AND Note Type = 0x03
Then...
    Source Address = 0x39ABCDEF
End If
```

序列编号字段

序列字段 1207 为一个主机应用，提供重建或重组一个数据包或序

列的能力，为了在介质 100 上传输，该数据包或序列被分为较小的包。重复的序列编号可以被抛弃，而未收到的序列编号可以被重传。编号为较长的数据流提供了数据完整性。在序列字段 1207 内的值决定于应用，并且如果需要，可以为另一个应用使用。

确认字段

确认字段 1208 允许每个包在接收完成前被确认。确认字段 1208 典型地通过对于加密矩阵的第一个两字节求异或而被植入（seeded）。因此，在一个安全系统内的所有节点将植入(seeded)相同的确认值，并且所有这些节点都应通过这个确认程序。该确认字段为增强完整性被进一步加密。

有效负荷字段

数据有效负荷字段 1211 被用来为接收节点提供信息。有效负荷数据的第一个字节可以包含决定怎样解析内容的一个字节编码。这个第一个数据字节与上文所述的原始（RAW）位共同使用。

循环冗余检验（CRC）字段

在已发送的包中，循环冗余检验（CRC）字段 1212 被用来提供一个可靠的错误检测技术。为了确认，根据完成和比较，它被重新评估。不能通过这项检测的包被抛弃。

CRC 算法被选得足够高效而且简单，以提供理想水平的可靠性而没有不适当的开销（在软件和硬件中）。提供一个这样的 CRC 算法是理想的，该算法足够快可以为发送和接收的包进行实时的（on-the-fly）CRC 计算。

实时计算（当一个位或字节被收到，CRC 就被更新，而不是等待整个包被收到，对于发送也是同样）不是强制性的，但是有助于系统的总的处理能力和性能。

在一个实施例中，G(X)由 $G(x)=x^{16}+x^{15}+x^{11}+x^{8}+x^{6}+x^{5}+x^{4}+x^{3}+x+1$ 给出。

PLX（内部的）主机包

PLX 内部的主机包永远不到介质 100 上，这样，包的描述看起来较简单。不需要前同步码 1201，不需要重复的长度字段 1202、903，不需要地址字段 1205、906，也不需要 CRC 字段 1212。图 14 展示了一个 PLX 内部的主机包的格式，包括一个长度字段 1701、一个控制字段 1702 和一个数据字段 1703。数据字段 1703 包含控制字段指定的任何东西。如前面的控制字段的定义（它也应用于 PLX 内部的主机包）所示，存在在硬件和主机节点间通过的许多包，它们促进业务流。以下是每种类型的包的定义。

CAL 请求包

图 15 展示了一个 CAL 请求包 1800 的格式，包括一个长度字段 1701、一个控制字段 1702 和一个 CAL 数据字段 1803。控制字段 1702 的值为 0x11。

一个 CAL 请求包 1800 被主机节点发送到硬件节点，以接收表示在硬件上的 CAL 信息。因为 PLX 节点可以具有应用编码或独立于硬件/ASIC 的一个主机处理器，CAL 信息也可以在这两个独立的处理器之间传播。这样，主机处理器从所连接的节点定时地收集 CAL 信息。

CAL 响应包

图 16 展示了一个 CAL 响应包 1900 的格式，包括一个长度字段 1701、一个控制字段 1702 和一个 CAL 响应字段 1903。控制字段 1702 的值为 0x21。

由于与上文所述同样的原因，一个 CAL 响应包被由硬件节点发往所连接的主机节点。这个响应包 1900 作为对一个在前的 CAL 请求包 1800 的响应而被发送。

Tx 状态包（单信道、速率）

图 17 展示了一个单信道 CAL 响应包 2000 的格式，包括一个长度字段 1701、一个控制字段 1702 和一个数据字段 1903。控制字段 1702

的值为 0x21。图 18 展示了一个多信道 CAL 响应包 2100 的格式，包括一个长度字段 1701、一个控制字段 1702 和一个数据字段 2103。控制字段 1702 的值为 0x31。

有两种格式的 Tx 状态包。一种格式是为单信道、单速率的应用，使用时控制字段的值为 0x21。第二种格式是为多信道、多速率的解决方案，使用时控制字段的值为 0x31。

单信道、单速率的解决方案只有两个可用的 Tx 缓冲区，这两个 Tx 缓冲区的状态被定时地通过一个内部的 PLX 信号交换传回主机节点。这些 Tx 状态包的目的是闭合关于被从主机节点交给硬件的显著的传送事件的环。经常，在一个 DACK 包内被返回的相同的值，为了与这个发送事件有关的信息，将被传递给主机，但是，很多时候 DACK 是到一个外部的 PLX 事件，在此情况下，该 DACK 值不应被交给主机节点。当主机节点发起传送请求时，该 DACK 值被交回给主机节点。

相应地，PLX 使用如下所示的重复的 DACK 状态值。

在介质上得到的 DACK 状态值

0x0=接收缓冲区满（失败）
0x2=令牌被节点使用（未被传给主机）
0x3=令牌未被节点使用（未被传给主机）
0x4=令牌响应“唤醒”请求（未被传给主机）
0x9=打印机序列编号错误
0xa=打印机未插入错误
0xb=打印机脱机错误
0xc=打印机一般错误
0xd=打印机无纸错误
0xe=打印机未知错误
0xf=成功

值 0x9 到 0xe 来自打印机节点的 DACK 响应。打印机响应值被未

改变地传回主机节点。

值 0xf 是一个成功的 DACK 响应，如果是主机发起请求，则这个值被未加改变地传回主机节点。

值 0x2 到 0x4 是到外部的 PLX 命令包的 DACK 响应值，不应被传回主机节点。

唯一奇怪的状态值是 0x0，0x0 在电线上的意思是接收节点忙，因此无法接收包。硬件识别这种状态并将重传这个包（经常是节点并非忙）一个特定数量的次数。如果接收节点在一段非常长的时间一直处于忙状态，则该包被最终放弃，并且一个“失败-0xf”响应状态被传回主机节点。一个 0x0 的值被传回主机节点不表示任何含义。这是传输事件未被完成的缺省值，主机节点将等待直到一个非零状态被放入这个字段中。值 0x1 从来不在电线上被返回。如果一个节点接收到一个有错误数据的包，它仅简单地不响应这个包，发送的节点将被要求重传这个包。只有当一个发送包超时并达到它的最大重传次数，值 0x1 才被传回主机。

以下是一个展示一般被返回主机节点的 Tx 状态值的表（注意这些值并非在所有的情况下与 DACK 响应值相同）：

Tx 状态数据字段值

0x0=对此 Tx 缓冲区无 Tx 状态
0x1=失败（Tx 超时或接收缓冲区满）
0x9=打印机序列编号错误
0xa=打印机未插入错误
0xb=打印机脱机错误
0xc=打印机一般错误
0xd=打印机无纸错误
0xe=打印机未知错误
0xf=成功

这个意思是以下的 DACK 信息未被通过一个内部的 Tx 状态包传给主机节点。

未传给主机节点的另外的 Tx 状态信息

0x0=接收缓冲区满（失败）

0x2=令牌被节点使用（未被传给主机）

0x3=令牌未被节点使用（未被传给主机）

0x4=令牌响应“唤醒”请求（未被传给主机）

Tx 状态字节被进一步分为两部分，每部分半字节，代表两个 Tx 缓冲区状态。在 Tx 状态字段中的值以及它们各自的含义列出如下。

Tx 状态值举例

0x0f= 第一个 Tx 缓冲区成功发送

0xf0= 第二个 Tx 缓冲区成功发送

0xff= 两个 Tx 缓冲区成功发送

0x1f= 第二个 Tx 缓冲区失败，第一个 Tx 缓冲区成功

等...

Tx 状态包（多信道、速率）

Tx 状态包的第二种格式用于多信道、多速率的解决方案。全部前面论述的关于单信道 Tx 状态包，以及它如何与 DACK 值相关，依然适用。不同点在于，包含在多信道/速率 Tx 状态包中的数据信息的量。该包将基本包含一个代表每个信道的单个的前文定义的状态字节。结果是多个数据字节，每个字节代表一个具有两个独立的 Tx 缓冲区的单个的信道。

包定时、间隔和重发

为在介质 100 上传输的所有的包，必须符合严格的定时要求。这些定时要求是使该系统平稳而无冲突地运行的规则。为了恰当地运行，必须严格地、强制性地坚持这些规则。

在平时运行中，一个“活动的网络服务器”出现在系统上，并且

仲裁所有激活的节点到介质 100 的访问。以下假设适用于这样一个出现在介质 100 上的激活的状态。在介质 100 上未激活，意味着每个节点都处于静止状态，并且在指定节点作为“活动的网络服务器”前，必须经过平常的“侦听”过程。

进一步，该 PLX 系统的特征在于确认的信号交换序列。确认包要在特定的时间间隔内被返回。除了确认包（DACK、LoGI 或双 LoGI）外，传送任何信息前都需要令牌包。活动的网络服务器是具有传送令牌和 LIP 包权利的唯一节点。客户节点仅仅传送有效负荷和确认包。

典型的包定时

图 19 是展示包定时和间隔的一个定时图。包定时的定义涉及第一个参考时间 2202 和第二个参考时间 2204。第二个参考时间跟着第一个参考时间 2202，之间有一个平均的包间间隙（I/Gap）50μs（微秒）。

以上所示的图假定一个运行在 650kbps 的系统的定时。除了间隙定时外，所有的值都应被调整为如表 A4 所给出，表 A4 中上标 1 表示一个参考第一个参考 2202 的时间，上标 2 表示一个参考第二个参考 2204 的时间。

表 A4 包定时

	350kbps	700kbps	1.2mbps	1.4mbps
最小 I/Gap[1]	15μs	15μs	15μs	15μs
平均 I/Gap[1]	50μs	50μs	50μs	50μs
前同步码	130μs	65μs	38μs	33μs
LoGI 包[2]	140μs	70μs	40μs	35μs
DloGI 包[2]	185μs	92μs	54μs	46μs
DACK 包[2]	335μs	168μs	98μs	84μs
TxRetry LoGI[1]	205μs	103μs	61μs	52μs
TxRetry DACK[1]	400μs	200μs	117μs	100μs
TxRetry DloGI[1]	320μs	160μs	94μs	80μs
内部令牌[1]	3+ms	3+ms	3+ms	3+ms

在平时状态下，典型的包定时需要接收包的节点在一个预先确定

的时间内响应。除 LoGI/双 LoGI 确认包外，这个响应时间对于所有的包是一致的。因此，包定时的两种情况是 1）LoGI/双 LoGI 响应和 2）所有的其它响应。

其它包定时

在一个特定的时间内，节点把一个包传回有效负荷包发出的节点，突发包和确认包除外，它们不需要响应包。响应包的类型可以是：DACK 包、LoGI 包或有效负荷包。

响应包符合上文中图 19 所示的间隙间隔需要。最小的响应时间典型地长于 15 微秒，最大的响应时间典型地不应超过 50 微秒。

如果一个发送节点未接收到前面的发送的确认，为了增加传递的可靠性，它必须开始一个重传过程。这个重传过程典型地在最长的可能的确认序列或一个 DACK 包加上最长的可能间隙间隔或在 650kbps 时约 700 微秒后开始。

节点特定信息

每个节点用一个特定数量的信息配置，形成了该特定节点的特性。PLX 节点需要这个最小量的信息，以在系统上实现完全的功能。

唯一标识、可寻址性和全球唯一标识（GUID）

当一 PLX 节点插入一个电力系统中时，它可以马上准备好行动。每个节点具有一个烧入的序列号，其中最低有效的 28 位被用作该节点运行时的地址。这并不能确保全球的单一性，但是她确实限定了可能性，因为你找到具有冲突地址的两个节点的机会是 1 比 268 百万。这个运行时的长地址只轻微减少吞吐量，但是在简单化系统时（因为节点从工厂出来时已预先设置），它增强了即插即用的能力并使用容易。

普通的设备环境和节点文件夹目标

CEBus/Generic CAL 兼容节点，至少有一个普通的设备环境和一个具有相关的实例变量（IVs）的节点控制目标。PLX 背离 CEBus/Generic

CAL 定义的报告条件和节点寻址，（两者都与该 PLX 客户/服务器结构有关，而与 CEBus/Generic CAL 的对等结构相反。因此，PLX 重新定义了普通的设备环境/节点控制目标，作为具有稍微不同的 IV 描述的节点文件夹目标。而且，每个 PLX 顺从性节点包含与节点文件夹目标相关的实例变量。

每个节点都负责包含一个预先定义的属性组，该属性组标识节点并把节点放到具有一般已知属性的一组节点类型中。每个节点的节点文件夹目标信息，最好被硬性编码进在该节点中的非挥发性存储器中。该信息根据请求被发送到服务器。一个节点文件夹目标由一个实例变量的列表组成。每个 PLX 节点至少包括，一个普通的设备环境（0x00）、一个节点文件夹目标（0x01）和特定的实例变量（IV），如以下表 A5 所示（其中 R/W 表示读/写）。

表 A5

IV	R/W	类型	名称	描述
o	R/W	d	context_list	包含这个特定的节点所支持的所有的设备环境的列表
w	R/W	b	Power	控制到这个特定节点的整体（global）电源
s	R	d	serial_number	包含一个设备商分配的产品序列号，其最低有效的 8 个字节也是该设备的 GUID（全球唯一标识）（18 字节）
c	R	n	manufacture_name	制造商的特定名称（最长 18 字节）
m	R	c	manufacture_model	制造商的特定型号（最长 18 字节）
c	R	n	product_class	根据 Generic CAL 技术规格(2 ASII 字节)
v	R	c	conformance_level	CAL/PLX 支持这个特定的设备的标识其目前级别的字符串（4 ASII

				字节）
h	R/W	d	area_address	用于路由和网络标识目的（1 字节）。这个 IV 总是全球可读（与 network_name 一起）。
a	R/W	d	unit_address	用于直接寻址包的节点 ID（4 字节）
t	R	d	network_class	定义设备的网络种类，并将被用于重写该设备 MAC 地址的最高四位，以区分令牌分配的优先次序。以下是优先次序和与网络类型相关的值： 0x01 视频系统 I 0x02 视频系统 II 0x03 音频系统 I 0x04 音频系统 II 0x05 保留 0x06 安全系统 0x07 应用监视系统 0x08 HVAC 系统 0x09 照明系统 0x0a 应用系统 0x0b 数据联网系统 0x0c 保留 0x0d 保留 0x0e 保留 0x0f 通用系统
f	R	d	buffering	以字节为单位的接收缓冲区大小
x	R	c	product_rev	产品修订等级（ASCII 字节）
b	R/W	d	dynamic_ mask	包括可以由一个单个位表示特征的一些动态节点功能。 位 0：混合模式 1=允许

				0=不允许 位 1：MAC 服务器 1=MAC 服务器 0=非 MAC 服务器 位 2：规则/数据库服务器 1=数据库服务器 0=非数据库服务器 位 3：未激活的/激活的设备（被轮询/轮询） 1=目前为激活的 0=目前为未激活的
u	R	d	static_mask	包括可以由一个单个位表示特征的一些静态节点功能。 位 0：远端闪烁能力 1=远端能够 0=远端不能够 位 1：确认能力 1=能确认（需要 NV Mem） 0=不能确认 位 2：复杂方法支持 1=复杂方法支持 0=复杂方法不支持 位 3：Diffie/Hellman 最大密钥大小 1=812 位 0=256 位
y	R	d	statistics	保持在此节点中的所有相关的计数器的一个统计表，具有以下格式：

				位 0：表版本 位 1：位掩码计数器
r	R/W	d	reset	允许这个节点被复位。写一个值 0x52 “R” 到这个实例变量，开始复位功能。
I	R/W	b	sleep	为了服务或人工控制，允许节点继续或脱机。
G	R/W	d	group_ address_list	这个节点支持的所有组地址的一个长度优先列表。因此，第一个 16 位值是跟着的组地址号。
j i k g d	R/W	d	authentication	根据配置和初始化，传到节点的确认 ID 值。 在 Diffie-hellman 中使用的 XOR 的密码矩阵 在 Diffie-hellman 中使用的非 XOR 的密码矩阵 公共密钥 公共的产生器 随机号
q	R/W	c	network_ name	允许一个节点，被用户使用一个可理解的名称，放入一个特定的安全网络中。
e	R/W	c	product_name	允许一个节点被一个逻辑名称引用。
p	R/W	c	product_ location	允许一个节点，被用户使用一个可理解的名称，放入一个特定的位置。
	R/W	d	*system_id_ list	在这个环境领域中，所有被分配的系统 ID 的一个列表。
	R/W	c	*last_log	最后被记录的事件

下表 A6 列出了被“应用服务器”存储、管理和维护，并存在于应用服务器的数据库中的客户 IV。因此，客户不需要涉及有关存储或提供关于这些 IV 的信息。

仅为主要情况的通用设备环境的部分，也是一个规则目标（0x03），该规则目标使用由 CAL 定义的数据存储目标，以及为我们的目的而定义的一些单独的 IV。

表 A6

IV	R/W	类型	名称	实例变量描述
r	R/W	d	current_rule	包含被当前的目录变量指向的有效的规则。
C	R/W	n	current_index	包含当前规则变量所示的规则的目录（名号）。
s	R	n	Rule_length	包含当前规则变量所示的规则的长度。
m	R	n	Maximum_ index	包含可以被放入当前的目录变量中的最大的目录值。
p	R/W	d	Previous_ value	包含在相应的规则中提到的每个 IV 的以前值的一个字符串。每个 IV 都是长度在先，NULL 在结束/填充。
l	R	n	Previous_ Value_length	Previous_value 字符串的最大长度。
n	R/W	c	Rule_name	分配给这个特定规则的逻辑名称。用来使得客户接口更具可读性。
z	R/W	n	Status	包含当前规则的状态。如果它的值为零，该规则有效，如果它的值为非零，则被这个规则指向的 IV 之一无效（脱机）。

规则目标允许远端的节点使用一个方法，来在规则列表中增加（继承）、删除（不继承）以及查看（获取矩阵）规则。

通过提供一个通用的设备环境，该网络能够包含一个节点列表。节点能够包含一个设备环境列表。节点能够有每个设备环境的一个目标列表。给出目标列表，节点也能够包含特定的实例变量。许多这些列表在 GenericCAL 技术规格中（而不是网络和节点列表）被说明。

当被要求时，节点使用上文所示的为它的特定配置的节点文件夹的特定的部分进行响应。该节点文件夹允许一个自动配置特定节点的方法，该节点在所考虑的网络中是唯一的。为了被唯一地标识，重复的节点能够提供另一水平的的配置。

安全

通过一个两步过程实现安全。开始，在网络上加电的每个节点立即被放入公共网络中。公共网络是为所有节点分配的默认的网络，它们为所有其它的公共节点所见，并且它们的确认 ID 被赋值为 NULL。一旦一个节点，通过下述的密钥交换过程变为安全时，它的确认 ID 变为由加密矩阵规定的一个值。每个节点被指派到这个专用/安全网络，它们被给与 32 字节的矩阵，由此它们加密或解密随后的包。这是通过一个被称为 Diffie-Hellman 的密钥交换技术，使用一个 256 位密钥实现的。使用一个有效的取幂算法，来减少被用在密钥交换中计算数值所需的时间。一旦加密矩阵被存储在网络上每个节点的存储器中，就执行加密和解密。在一个实施例中，加密和解密使用了基于一个带反馈异或的一个流密码技术。也可以使用其它算法，包括，例如，DES、RC4、MD5 等等。

附加的特征

报告状态详细说明

因为与它们在 CAL 下相比，在 PLX 下报告状态被不同地处理，该 PLX 处理规则的方法在此示出。这些变化用来说明在一个严格的 CAL 报告状态方法中的许多限制。不同点在下表 A7 中示出。

表 A7 与 Generic CAL 相比 PLX 的优点

Cebus CAL	PLX
每个目标一个规则	每个目标多个规则
每个目标一个有效的 IV	每个目标多个有效的 IV
只有简单规则	简单的和复杂的规则
固定的规则	灵活的规则

因为 PLX 规则存在于服务器上，与在 Generic CAL 下的分布式规则相对，通过它的单个的、功能强大的引擎，PLX 在它如何处理规则上，功能更强大。PLX 客户节点将它们的 IV 中的变化，汇报到服务器。任何 IV 变化都如此。当服务器检测到一个 IV 变化时，该服务器查看变化的特定目标/IV 组合，该服务器查看它的规则列表，并且该服务器测试每个规则的有效性。因此，每个目标都被配置为包含以下两个 IV，这两个 IV 处理为特定目标建立的每个规则，相关的 IV 列出如下。

IV	R/W	类型	名称	设备环境功能
R	R/W	d	rules_arry	包含一个目录的指针矩阵到主用的规则列表（规则目标）。当在这个目标中的 IV 被改变时，每个入口表示一个要被测试的完全的规则。
P	R/W	n	number_of_rules	包含在 rules_arry 中的规则数量。

实际的 report_header 、 report_address 、 report_condition 和 previous_value 变量，每个被保存在矩阵所指向的规则中。调用程序简单地传递这个指针（或目录）到规则引擎，而规则引擎将从主用的规则列表，解析它需要的适当的信息。

非挥发性存储器使用

每个节点在一个静态的存储位置，如 ROM 中，包含节点文件夹信息。另外，节点也可以在非挥发性存储器中，存储其它信息，如确认

密钥，但是，这是一个选择，并非任何 PLX 顺从性节点都需要。其它的选择性的存储器需求，包括路由信息和其它的动态表。

客户变化通知

客户节点典型地报告状态变化到应用服务器节点。意思是，即使应用服务器告诉一个客户改变它的状态，该客户回报应用服务器它的状态已经改变。这减少了应用服务器数据库未与真正的客户节点变量同步的问题。

这是所希望的，因为应用服务器包含，与客户变量的变化相关的报告的状态和规则。该客户在此方面智能较少，因此它们应当把适当的变化通知应用服务器。

该应用服务器典型地不更新属于一个特定的客户节点的它的数据库变量，直到从那个客户节点接收到确认，通知该“应用服务器”该客户节点已经改变状态。

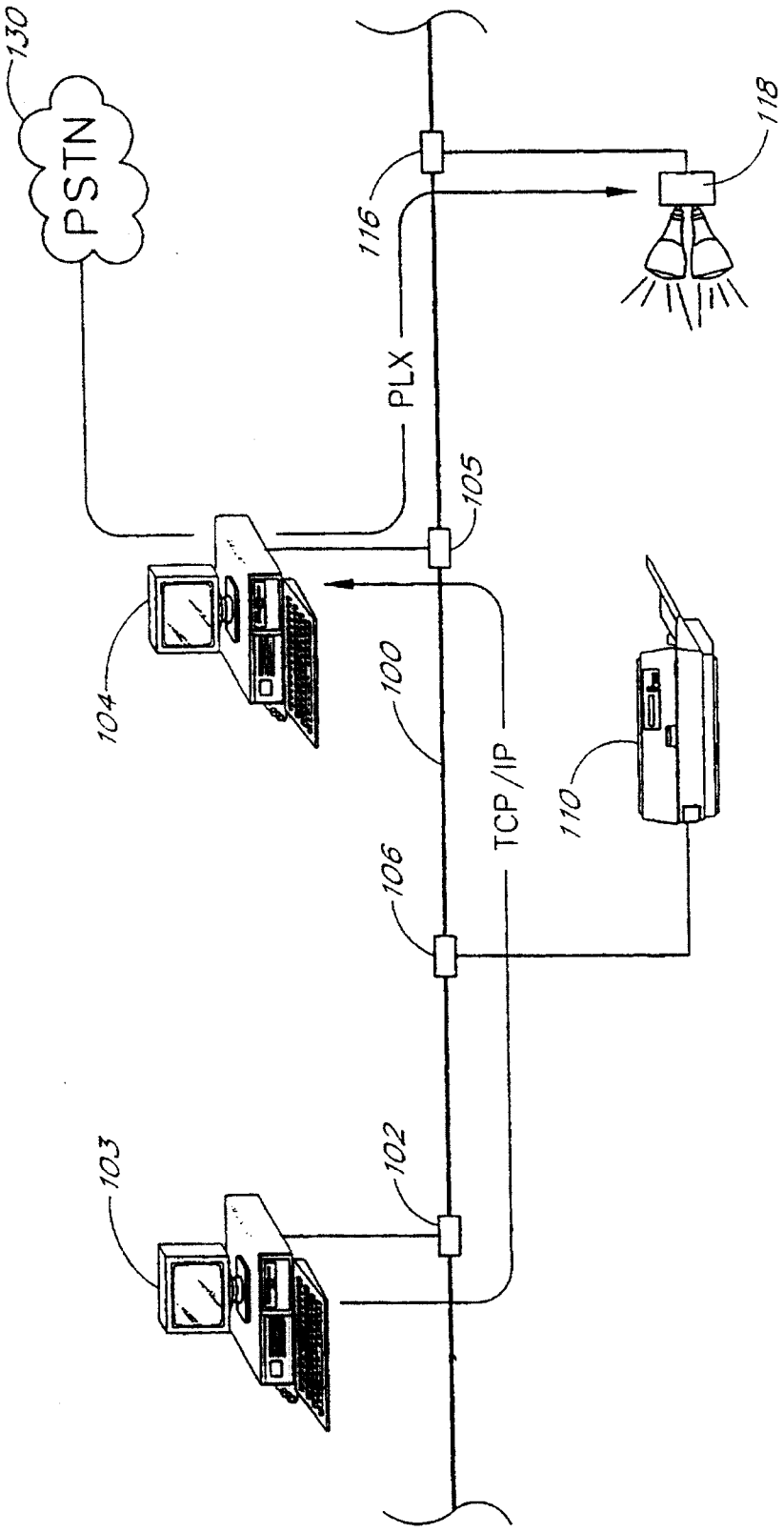


图 1

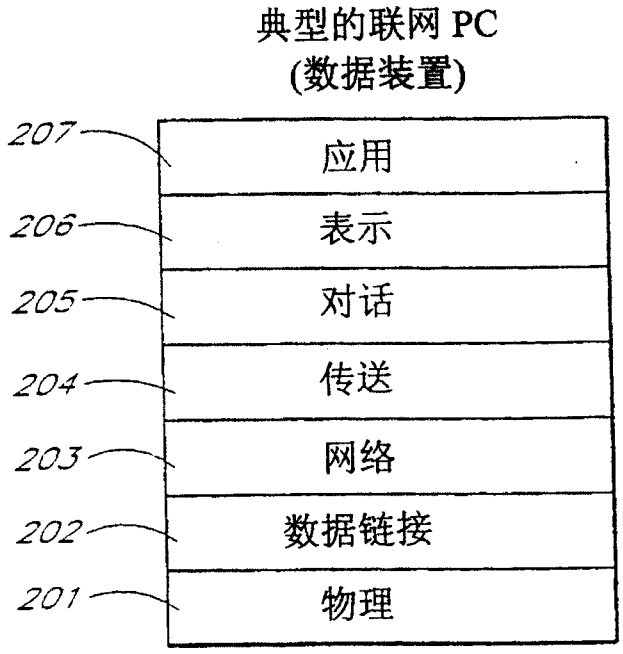


图 2

图 3A	图 3B
图 3C	图 3D

图 3

图 3A

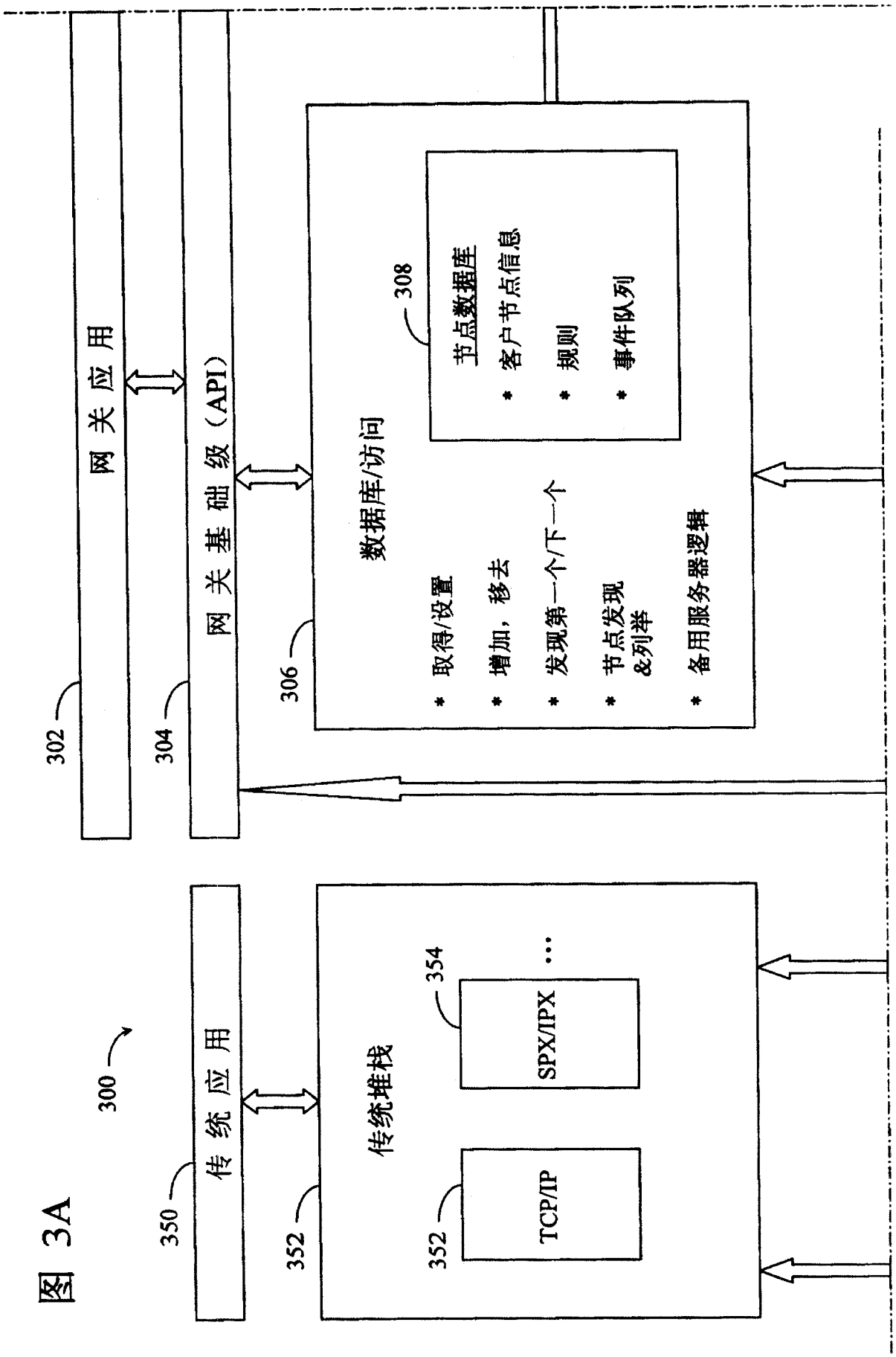
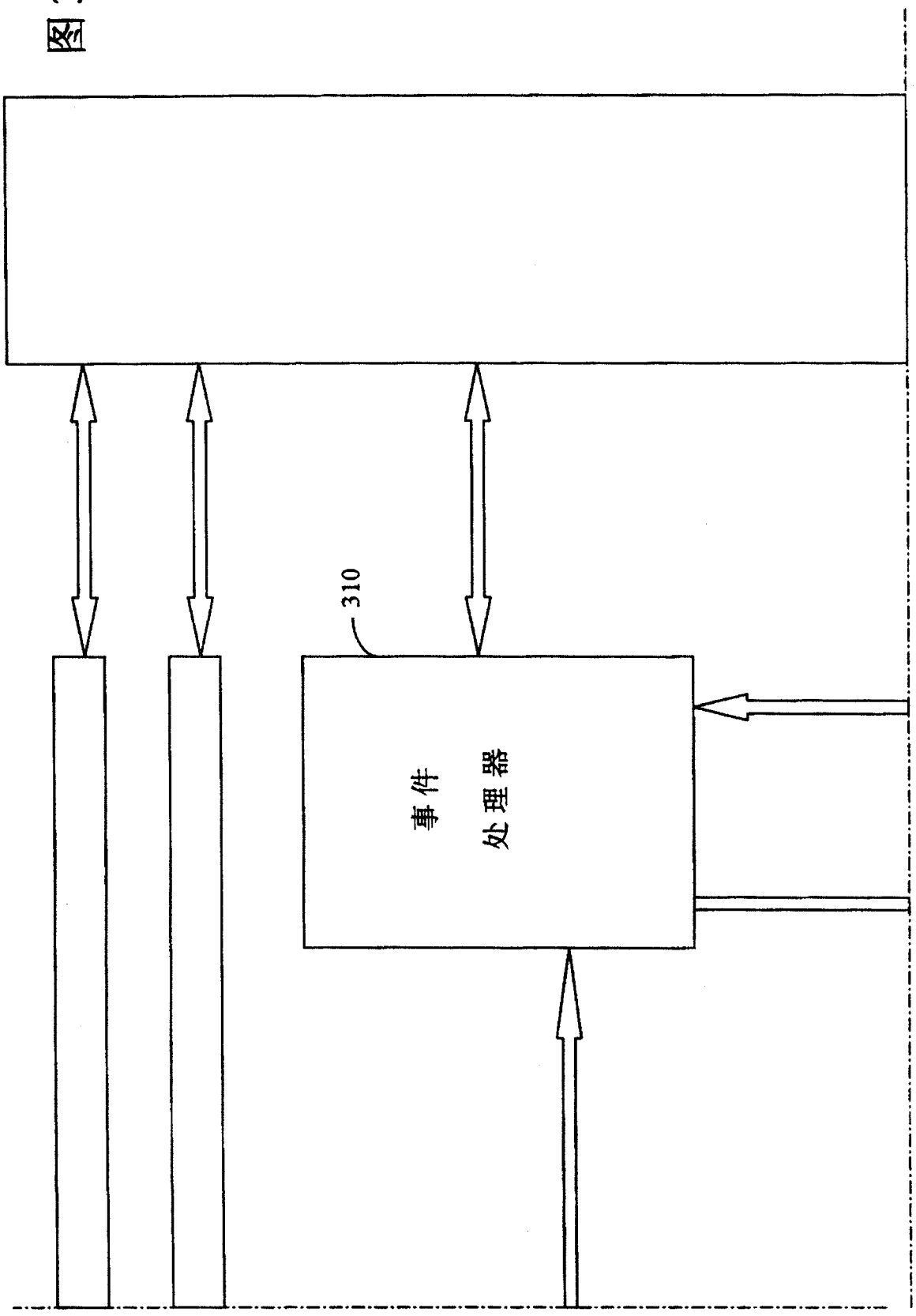

300
网关应用
302
网关基础级（API）
304
数据库/访问
306
* 取得/设置
* 增加，移去
* 发现第一个/下一个
* 节点发现 &列举
* 备用服务器逻辑
节点数据库
308
* 客户节点信息
* 规则
* 事件队列
传统应用
350
传统堆栈
352
TCP/IP
352
SPX/IPX
354
...

图3B



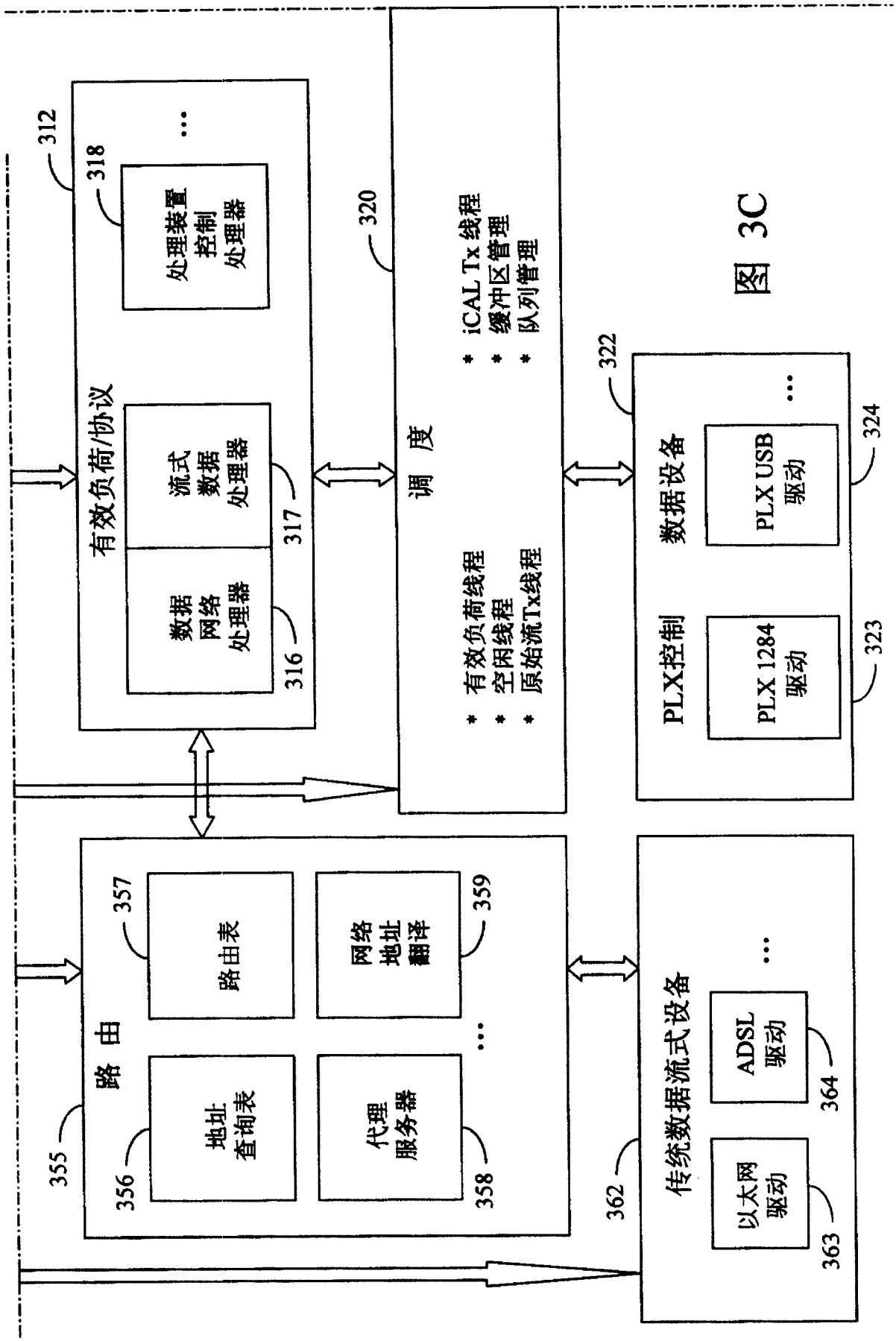

312
318
处理装置
控制
处理器
...
有效负荷/协议
流式
数据
处理器
317
数据
网络
处理器
316
320
调 度
* iCAL Tx 线程
* 缓冲区管理
* 队列管理
* 有效负荷线程
* 空闲线程
* 原始流Tx线程
322
PLX控制
数据设备
PLX USB
驱动
324
PLX 1284
驱动
323
路 由
355
357
路由表
网络
地址
翻译
359
地址
查询表
356
代理
服务器
358
...
传统数据流式设备
362
以太网
驱动
363
ADSL
驱动
364
...

图 3C

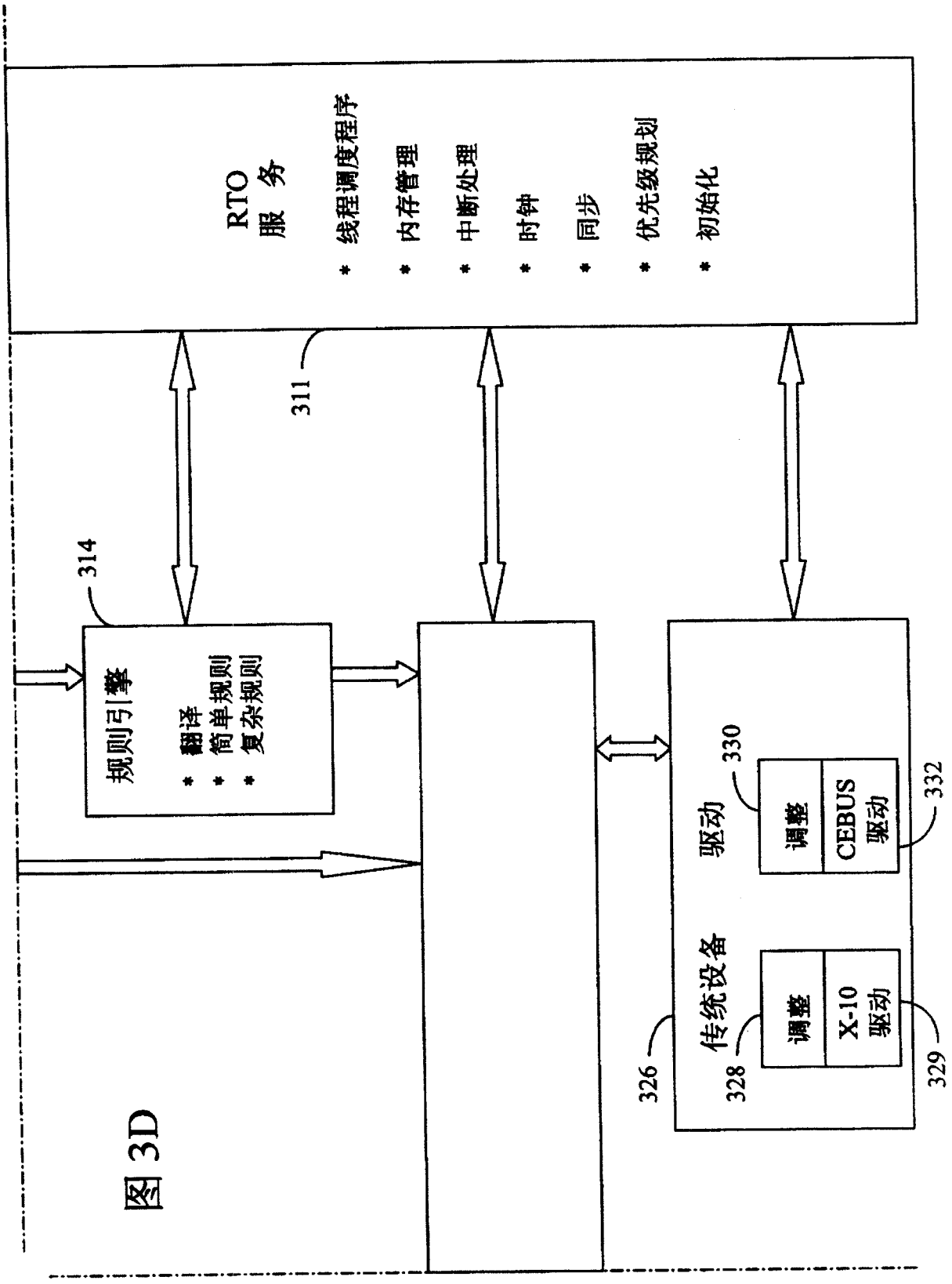

RTO
服 务
* 线程调度程序
* 内存管理
* 中断处理
* 时钟
* 同步
* 优先级规划
* 初始化
311
314
规则引擎
* 翻译
* 简单规则
* 复杂规则
驱动
330
调整
CEBUS
驱动
332
传统设备
326
328
调整
X-10
驱动
329

图 3D

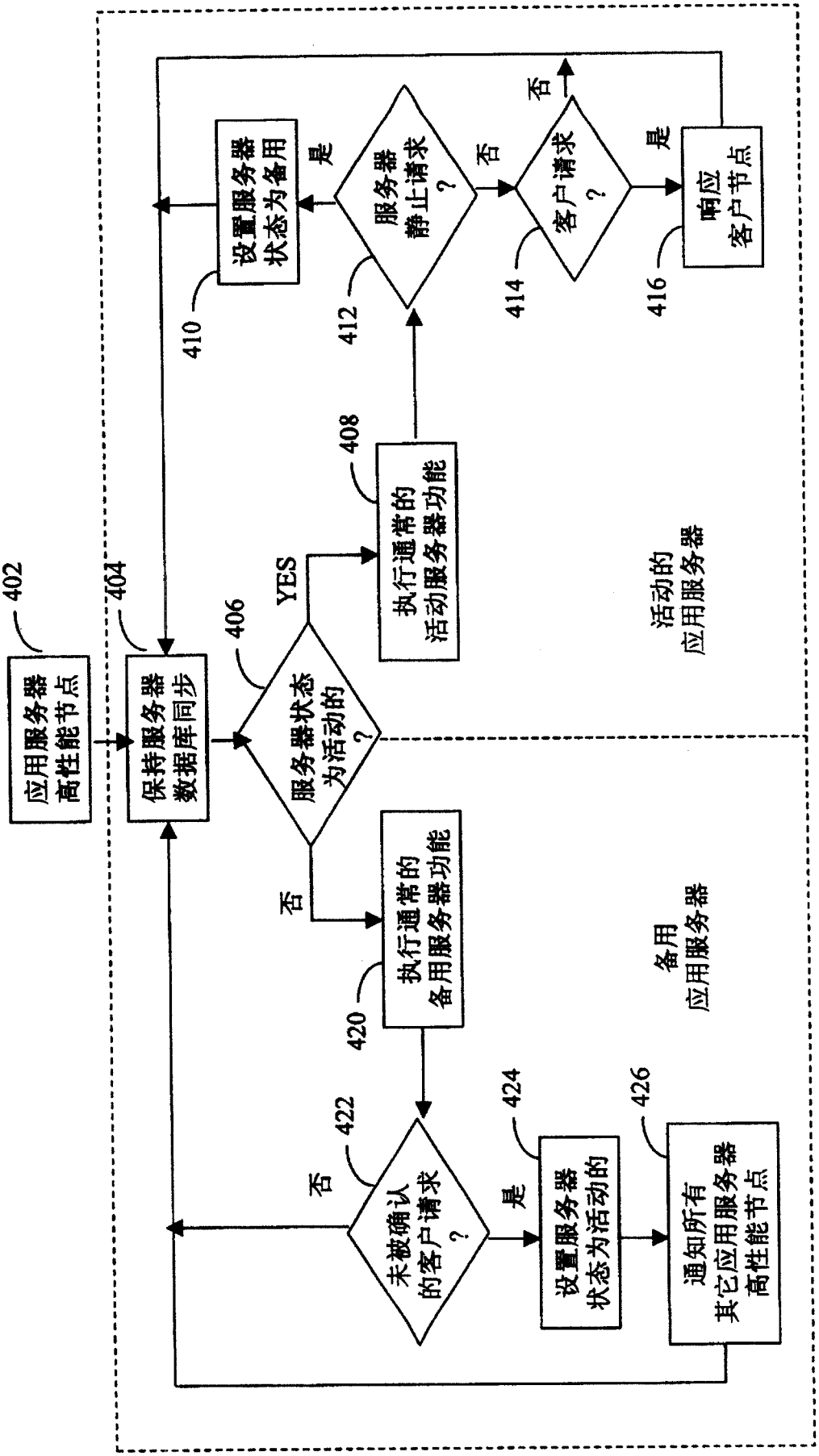

应用服务器高性能节点
402
保持服务器数据库同步
404
服务器状态为活动的？
406
YES
否
执行通常的活动服务器功能
408
执行通常的备用服务器功能
420
活动的应用服务器
备用应用服务器
设置服务器状态为备用
410
服务器静止请求？
412
是
否
客户请求？
414
是
否
响应客户节点
416
未被确认的客户请求？
422
否
是
设置服务器状态为活动的
424
通知所有其它应用服务器高性能节点
426

图 4

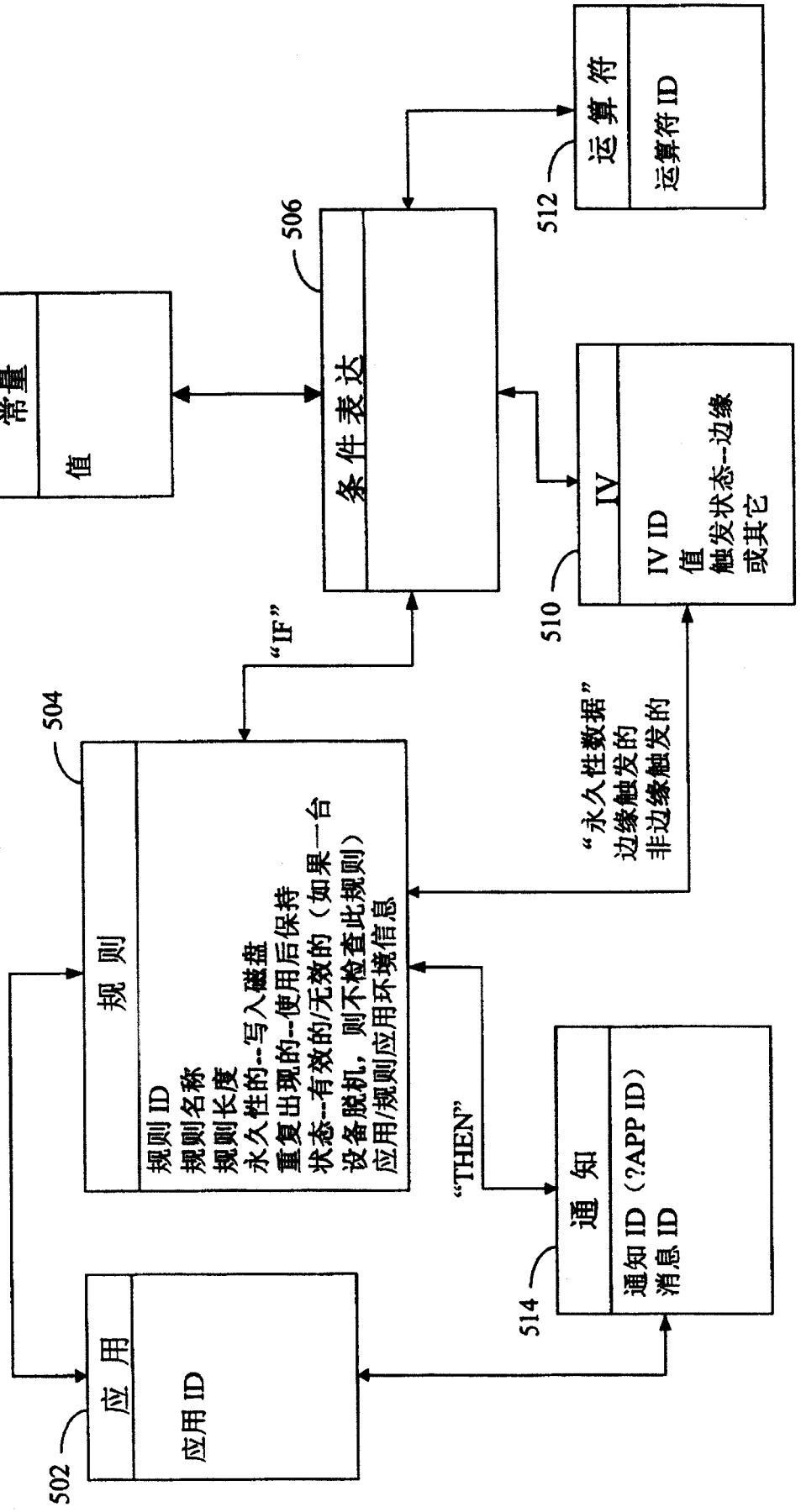

应 用
应用 ID
502
规 则
规则 ID
规则名称
规则长度
永久性的--写入磁盘
重复出现的--使用后保持
状态--有效的/无效的（如果一台
设备脱机，则不检查此规则）
应用/规则应用环境信息
504
常量
值
508
条件表达
506
运算符
运算符 ID
512
IV
IV ID
值
触发状态--边缘
或其它
510
通 知
通知 ID（?APP ID）
消息 ID
514
“IF”
“THEN”
“永久性数据”
边缘触发的
非边缘触发的

图 5

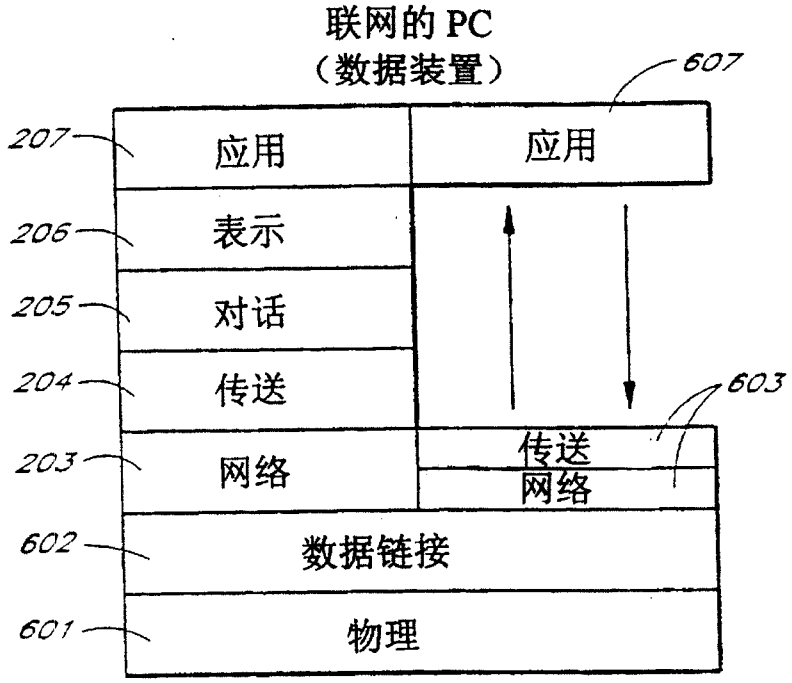

联网的PC
（数据装置）
607
207
应用
应用
206
表示
205
对话
204
传送
603
203
网络
传送
网络
602
数据链接
601
物理

图 6

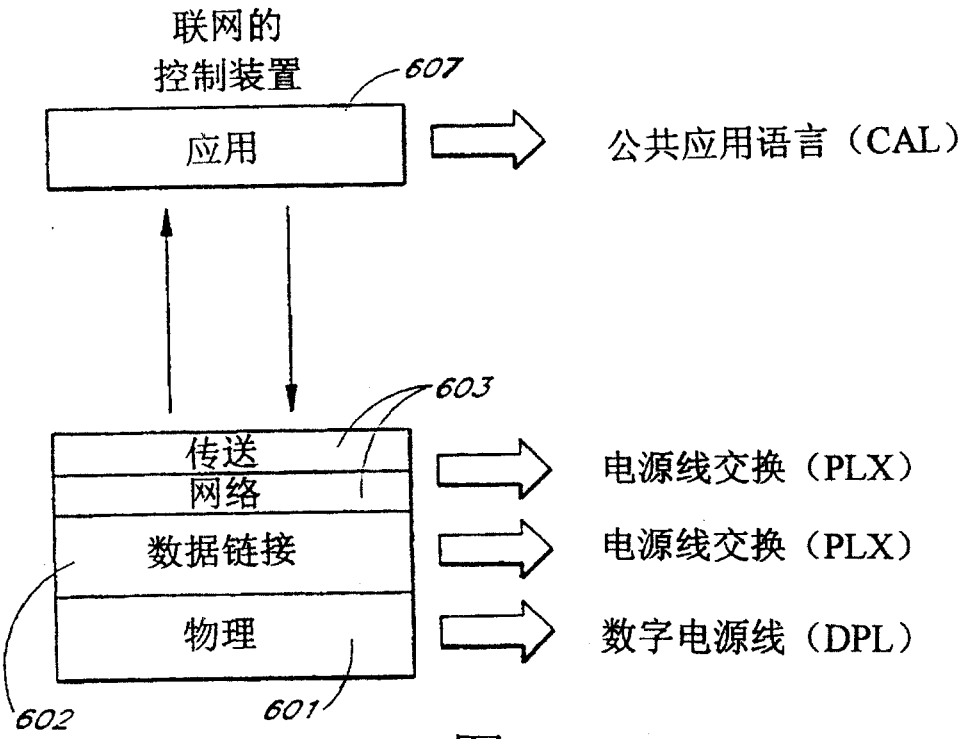
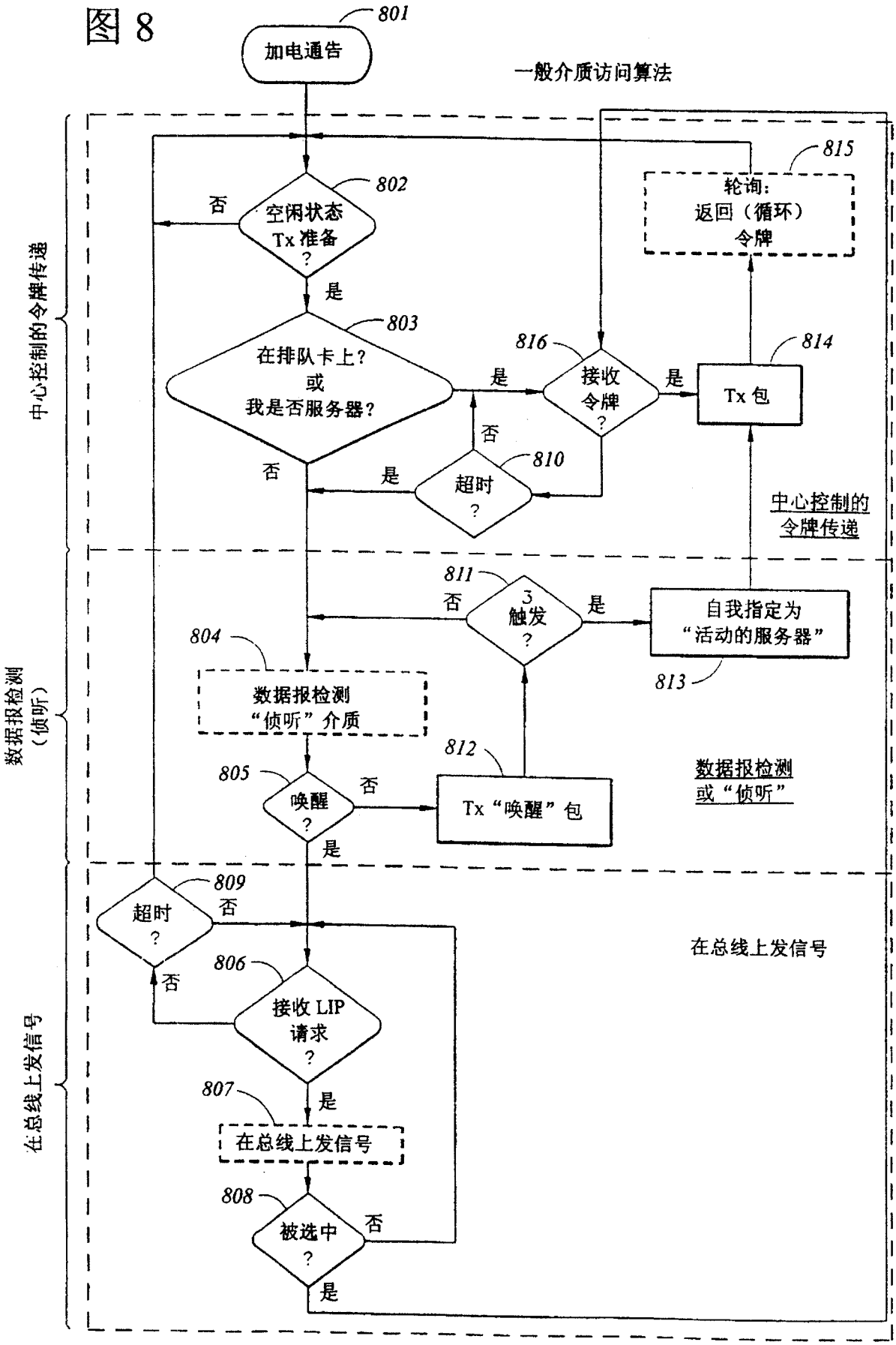

联网的
控制装置
607
应用
公共应用语言（CAL）
603
传送
网络
电源线交换（PLX）
数据链接
电源线交换（PLX）
物理
数字电源线（DPL）
602
601

图 7

图 8


801
加电通告
一般介质访问算法
802
空闲状态
Tx 准备
?
否
是
803
在排队卡上?
或
我是否服务器?
是
否
816
接收
令牌
?
是
814
Tx 包
815
轮询:
返回（循环）
令牌
否
是
810
超时
?
中心控制的
令牌传递
811
3
触发
?
否
是
自我指定为
“活动的服务器”
813
804
数据报检测
“侦听”介质
812
数据报检测
或“侦听”
805
唤醒
?
否
Tx “唤醒”包
是
809
超时
?
否
在总线上发信号
否
806
接收 LIP
请求
?
是
807
在总线上发信号
808
被选中
?
否
是
中心控制的令牌传递
数据报检测（侦听）
在总线上发信号

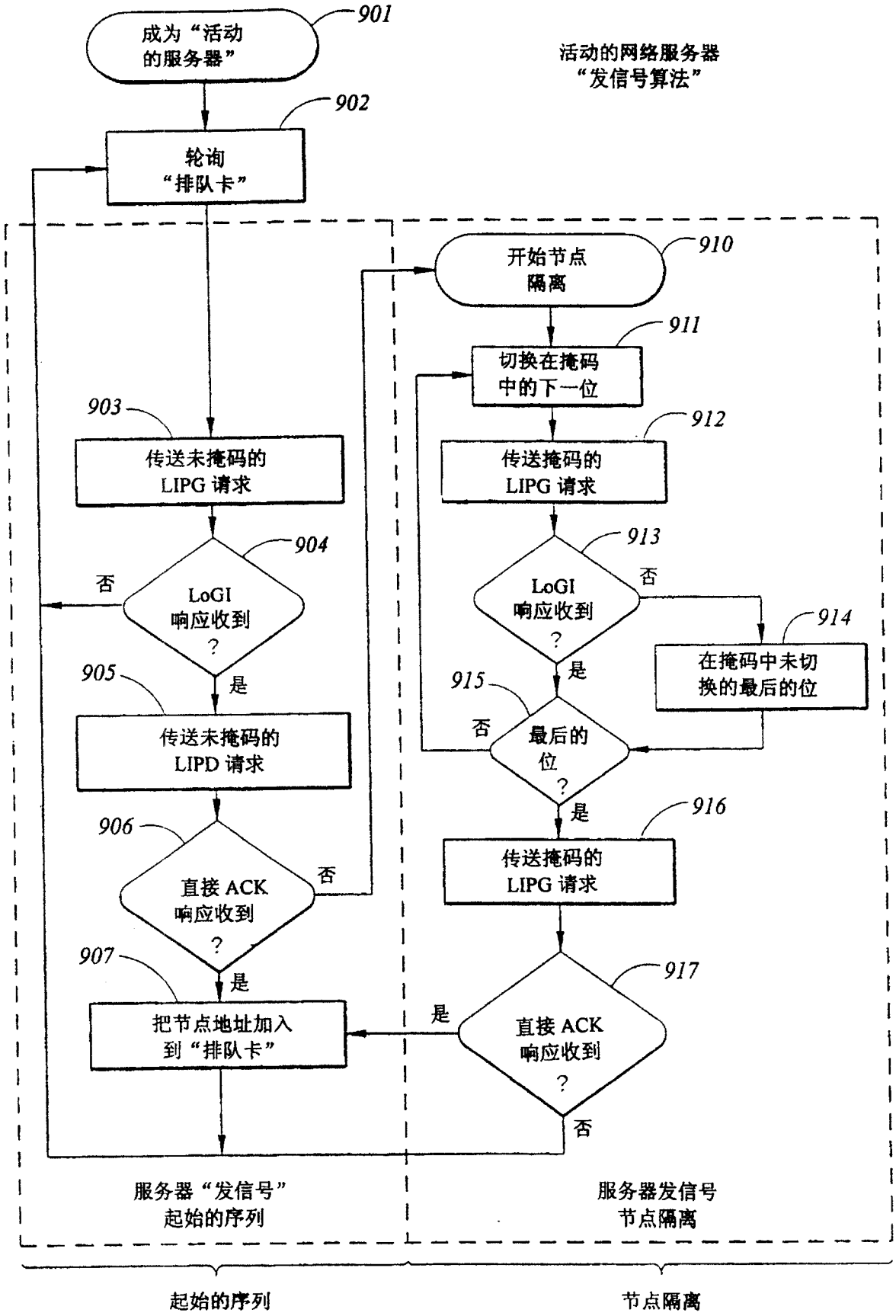

成为“活动的服务器”
901
活动的网络服务器“发信号算法”
902
轮询“排队卡”
903
传送未掩码的LIPG 请求
904
LoGI响应收到？
否
是
905
传送未掩码的LIPD 请求
906
直接 ACK响应收到？
否
是
907
把节点地址加入到“排队卡”
910
开始节点隔离
911
切换在掩码中的下一位
912
传送掩码的LIPG 请求
913
LoGI响应收到？
否
是
914
在掩码中未切换的最后的位
915
最后的位？
否
是
916
传送掩码的LIPG 请求
917
直接 ACK响应收到？
是
否
服务器“发信号”起始的序列
服务器发信号节点隔离
起始的序列
节点隔离

图 9A

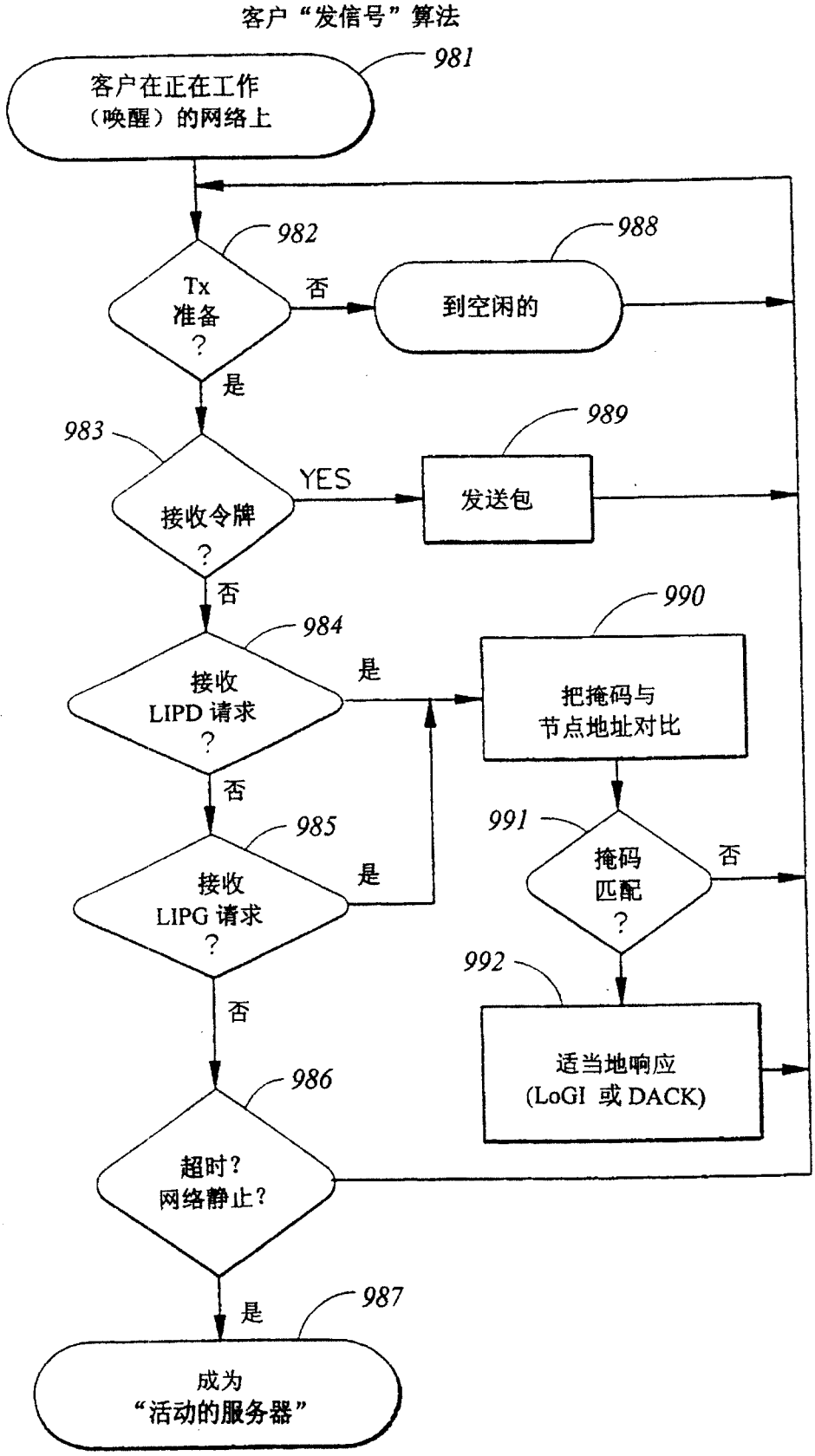

客户“发信号”算法
981
客户在正在工作
（唤醒）的网络上
982
Tx
准备
?
否
988
到空闲的
是
983
接收令牌
?
YES
989
发送包
否
984
接收
LIPD 请求
?
是
990
把掩码与
节点地址对比
否
985
接收
LIPG 请求
?
是
991
掩码
匹配
?
否
992
适当地响应
(LoGI 或 DACK)
否
986
超时?
网络静止?
是
987
成为
“活动的服务器”

图 9B

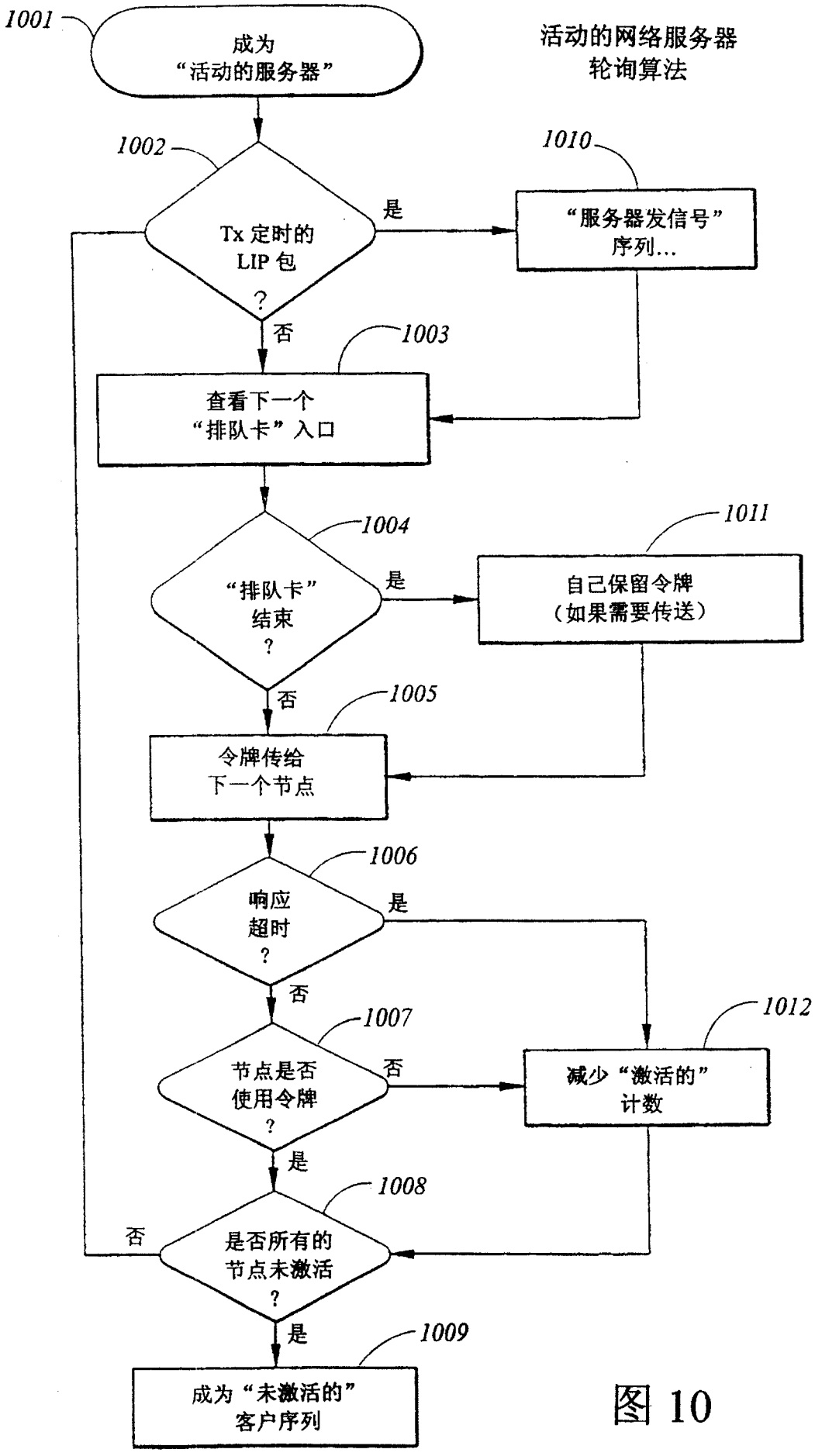

活动的网络服务器
轮询算法
1001
成为
“活动的服务器”
1002
Tx 定时的
LIP 包
?
是
否
1010
“服务器发信号”
序列...
1003
查看下一个
“排队卡”入口
1004
“排队卡”
结束
?
是
否
1011
自己保留令牌
（如果需要传送）
1005
令牌传给
下一个节点
1006
响应
超时
?
是
否
1007
节点是否
使用令牌
?
否
是
1012
减少“激活的”
计数
1008
是否所有的
节点未激活
?
否
是
1009
成为“未激活的”
客户序列

图 10

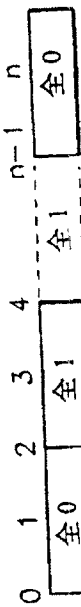

0
1
2
3
4
n-1
n
全0
全1
全1
全0
1100

图 11

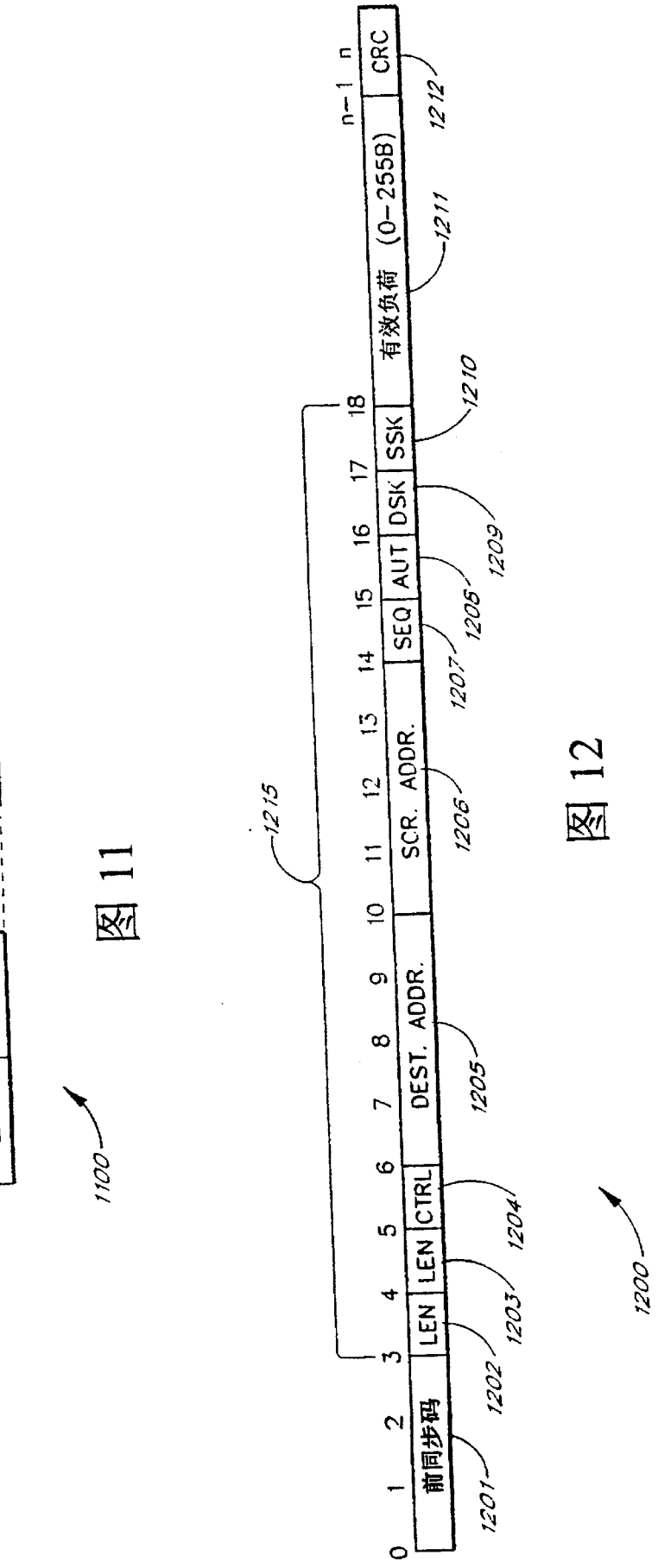

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
n-1
n
前同步码
LEN
LEN
CTRL
DEST. ADDR.
SCR. ADDR.
SEQ
AUT
DSK
SSK
有效负荷 (0-255B)
CRC
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1215
1200

图 12

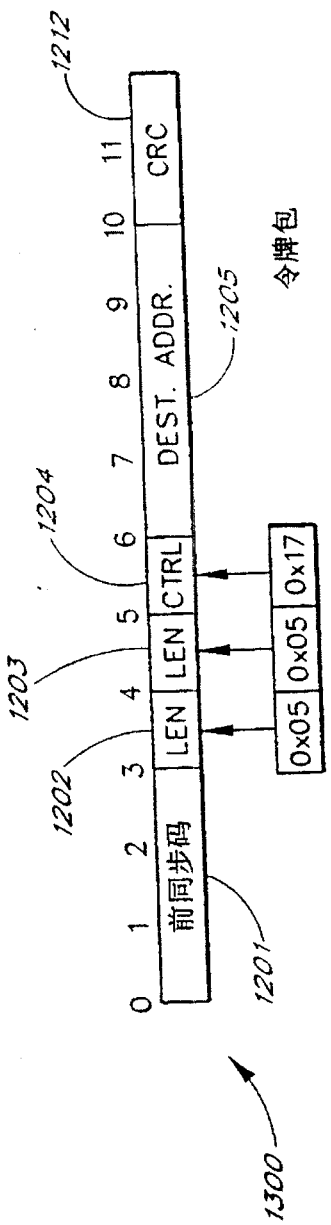

0
1
2
3
4
5
6
7
8
9
10
11
前同步码
LEN
LEN
CTRL
DEST. ADDR.
CRC
0x05
0x05
0x17
1201
1202
1203
1204
1205
1212
1300
令牌包

图 13

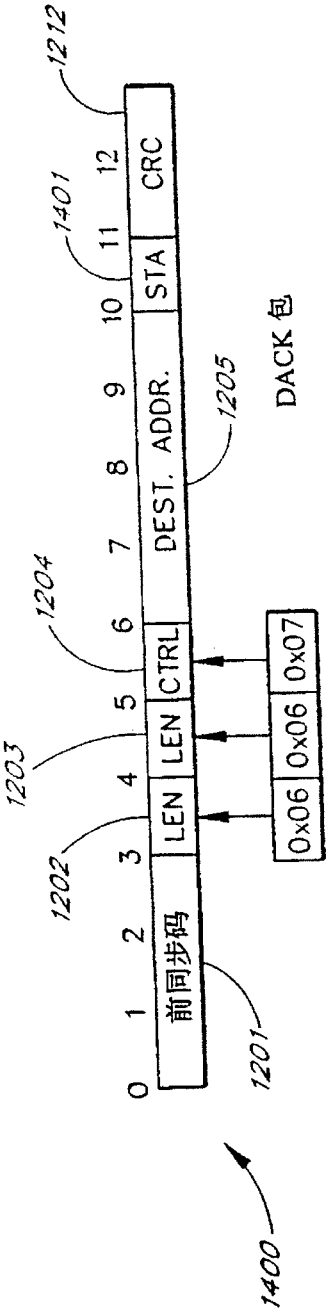

0
1
2
3
4
5
6
7
8
9
10
11
12
前同步码
LEN
LEN
CTRL
DEST. ADDR.
STA
CRC
0x06
0x06
0x07
1201
1202
1203
1204
1205
1212
1401
1400
DACK 包

图 14

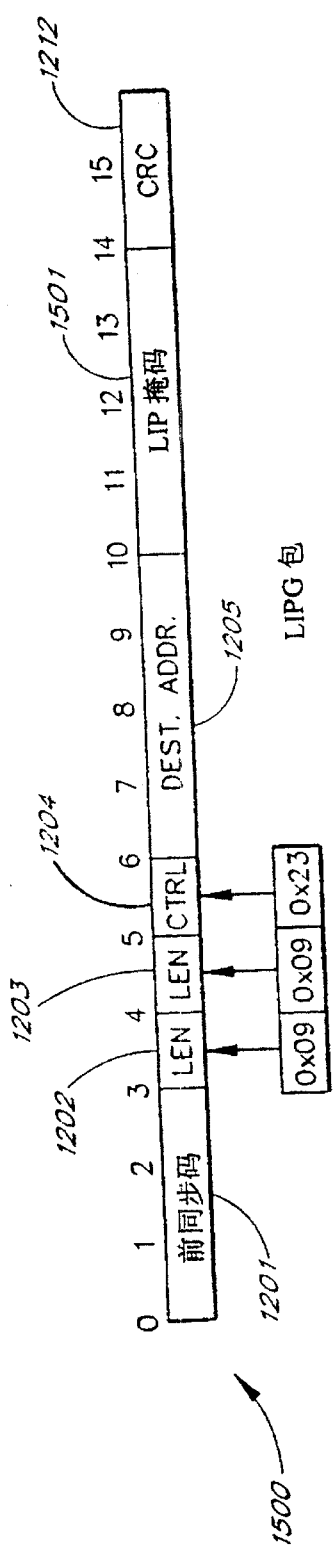

1500
1201
前同步码
1202
LEN
1203
LEN
1204
CTRL
1205
DEST. ADDR.
1501
LIP 掩码
1212
CRC
0x09
0x09
0x23
LIPG 包
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

图 15

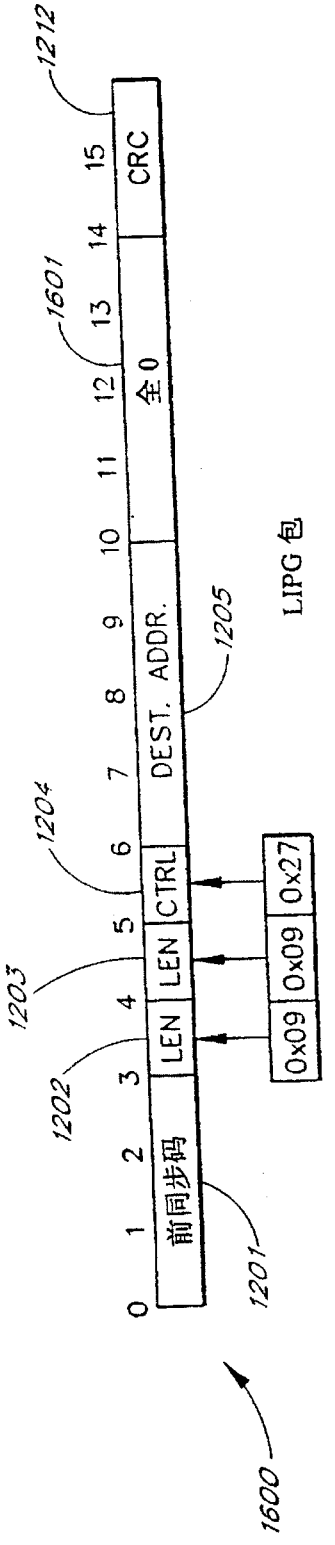

1600
1201
前同步码
1202
LEN
1203
LEN
1204
CTRL
1205
DEST. ADDR.
1601
全 0
1212
CRC
0x09
0x09
0x27
LIPG 包
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

图 16

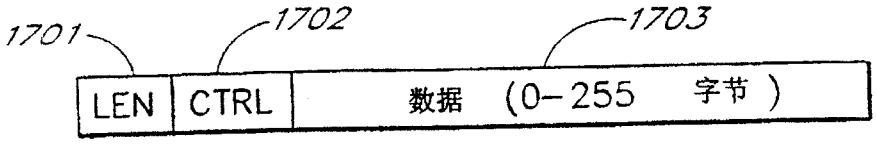

1701
1702
1703
LEN
CTRL
数据 (0-255 字节)

图 17

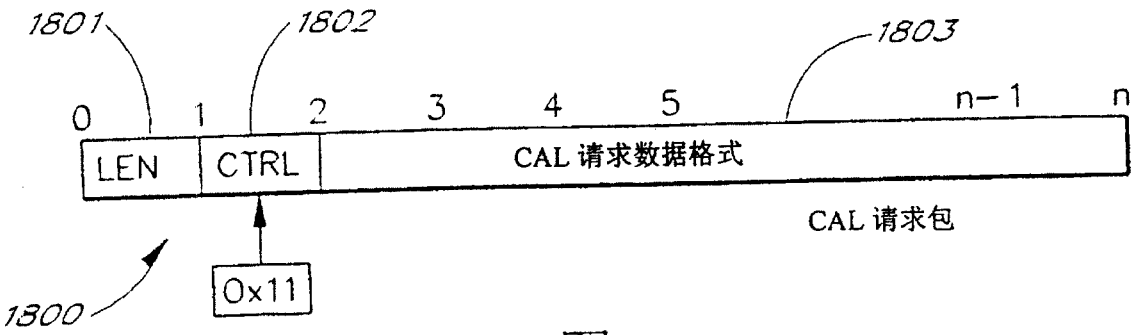

1801
1802
1803
0
1
2
3
4
5
n-1
n
LEN
CTRL
CAL 请求数据格式
CAL 请求包
1800
0x11

图 18

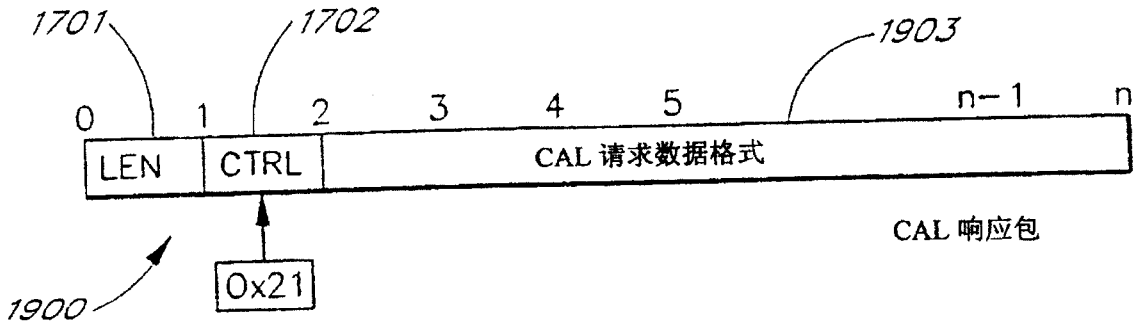

1701
1702
1903
0
1
2
3
4
5
n-1
n
LEN
CTRL
CAL 请求数据格式
CAL 响应包
1900
0x21

图 19

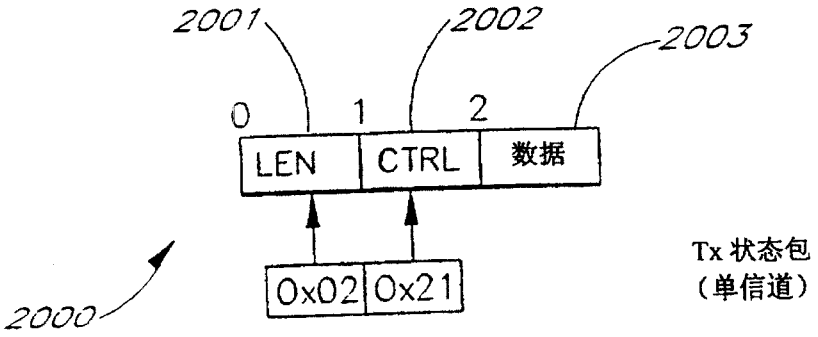

2001
2002
2003
0
1
2
LEN
CTRL
数据
0x02
0x21
2000
Tx 状态包
（单信道）

图 20

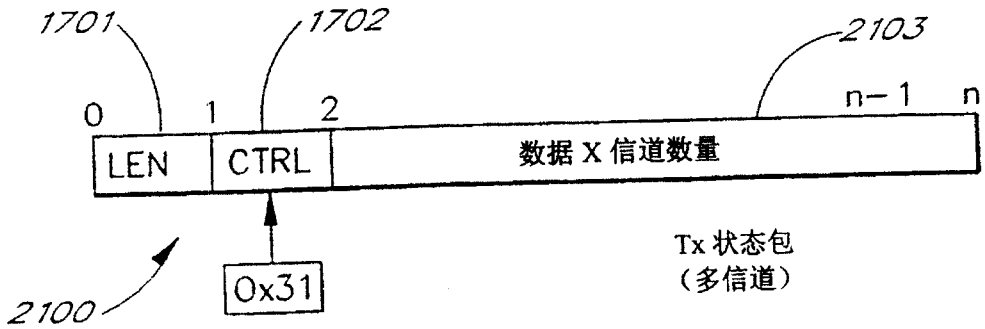

1701
1702
2103
0
1
2
n−1
n
LEN
CTRL
数据 X 信道数量
0x31
2100
Tx 状态包
（多信道）

图 21

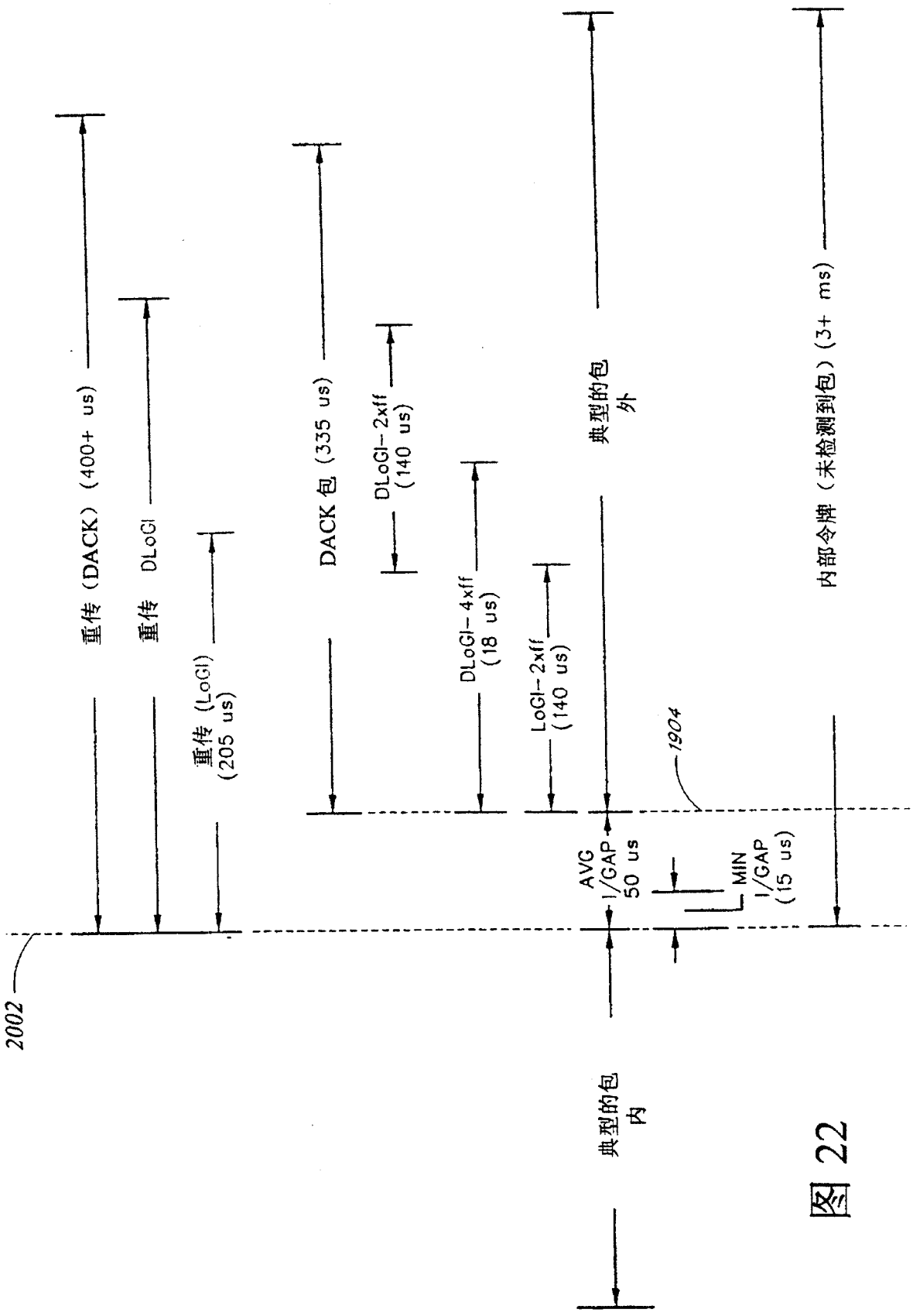

重传（DACK）（400+ us）
重传 DLoGI
重传（LoGI）（205 us）
DACK 包（335 us）
DLoGI-2xff（140 us）
DLoGI-4xff（18 us）
LoGI-2xff（140 us）
典型的包外
内部令牌（未检测到包）（3+ ms）
1904
AVG I/GAP 50 us
MIN I/GAP（15 us）
2002
典型的包内

图 22