摘　要

随着计算机网络、通信和控制技术的发展，以太网技术在工业控制中的应用成为当前控制领域的研究热点。以太网自身存在的通信延迟不确定性是其进入工业控制领域的主要技术障碍。

EPA(Ethernet for plant automation)是现场设备层的实时以太网解决方案，它通过对ISO/IEC 8802-3 协议规定的数据链路层进行扩展，增加了一个通信调度管理实体来对数据包发送进行调度，避免碰撞从而增强实时性。

研究了 IEEE1588 精确时钟同步协议的工作原理，深入分析了影响时钟同步精度的因素，在 Linux 系统中，通过修改内核的时钟管理程序提高了时钟精度，采用在网卡中断处理程序中记录时间戳的方法来提高时间戳的准确度，并对时间偏差使用迭代滤波的方法进行补偿，从而实现了 EPA 系统中设备间的精确时钟同步，测试结果显示同步精度达到了 10μs。

研究了 Linux 内核的网路协议栈和链路层实现，在精确时钟同步的基础上，通过修改 Linux 内核的 QoS 接口和网卡驱动程序实现了 EPA 链路层实时调度规则。

组建了测试平台，并开发了上、下位机测试软件。测试平台采用 DUT5000 以太网控制模块，操作系统为 Linux。上位机测试软件采用 VC++6.0 开发，可以进行可视化的带宽配置和图形化的测试结果显示。下位机采用 Linux C 开发。通过对时钟同步精度、递交时间和调度等指标的测试对 EPA 网络的实时性性能进行了分析。

关键词：工业以太网；EPA；实时性；调度；Linux

The Reseach and Implementation of Industrial Ethernet Real-time Scheduling Technology

Abstract

With the development of computer network, communication and control technology, the application of Ethernet in the industrial control field becomes a research hotspot. However, the inherent uncertainty caused by communication delay of Ethernet highly prevents its step into the industrial control area.

EPA (Ethernet for plant automation) is a device-layer solution for the real-time Ethernet. It enhances the real-time performance of Ethernet by expanding the data link layer set in ISO/IEC 8802-3 protocol with a CSME (communication scheduling management entity) which consequently schedules the sending of packets to avoid data collision in the network.

This paper provides a comprehensive study on the principle of IEEE1588 precise time protocol and gives out analysis of the factors influencing the synchronization precision. An approach is proposed to improve the precision of clock synchronization among distributed devices by modifying Linux kernel, which includes: 1) enhancing the precision of the kernel's default clock management timer, 2) capturing time stamps in the link layer by modifying the ISR of NIC driver and 3) compensating the offset by iterated filtering. The test result indicates that a precision of 10μs is achieved.

Based upon the precise clock synchronization, the architecture of the network protocol stack and the implementation of data link layer in the Linux kernel are studied and then give an implementation of EPA's data link scheduling algorithm by modifying the QoS interface and NIC driver is given..

The test platform is established based on DUT5000 Ethernet control module and Linux OS and the test programs are developed separately on PC and in the embedded device. The PC test program, developed in VC++6.0, is used for the visual configuration of bandwidth and for the display of graphic test result. The embedded test program is developed with Linux C. A final analysis of EPA's real-time performance is given through the tests of clock synchronization accuracy, delivery time and scheduling.

Key Words: Industrial Ethernet; EPA; Real-time performance; Scheduling; Linux

独创性说明

作者郑重声明：本硕士学位论文是我个人在导师指导下进行的研究工作及取得研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写的研究成果，也不包含为获得大连理工大学或者其他单位的学位或证书所使用过的材料。与我一同工作的同志对本研究所做的贡献均已在论文中做了明确的说明并表示了谢意。

作者签名：冀朝旭 日期：2007.12.16

大连理工大学学位论文版权使用授权书

本学位论文作者及指导教师完全了解“大连理工大学硕士、博士学位论文版权使用规定”，同意大连理工大学保留并向国家有关部门或机构送交学位论文的复印件和电子版，允许论文被查阅和借阅。本人授权大连理工大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，也可采用影印、缩印或扫描等复制手段保存和汇编学位论文。

作者签名：冀朝阳

导师签名：仲梁权

2007 年 12 月 16 日

1 绪论

1.1 工业控制系统发展历史

随着微电子技术、计算机技术以及通信技术的飞速发展，工业控制系统也不断发生变革。与此同时，作为控制系统的重要组成部分的工业控制网络，也不断地向前发展。

20 世纪 60 年代，数字计算机进入控制领域，产生了第一代控制系统 CCS(计算机集中控制系统)。在 CCS 中，数字计算机取代了传统的模拟仪表，从而能够使用更为先进的控制技术，例如复杂控制算法和协调控制。从而使自动控制发生了质的飞跃。但由于控制简单，直接面向控制对象，并未形成控制网络体系。CCS 在集中控制的同时也集中了危险，系统可靠性很低。由于只有一个 CPU 工作，实时性差。系统越大，上述缺点越突出[1]。

真正意义的工业控制网络体系是七十年代出现的第二代计算机控制系统：分散型控制系统 DCS(也称集散控制系统)。典型的 DCS 可分为操作站级、过程控制级和现场仪表 3 级。这种控制系统的特点是“集中管理，分散控制”。其基本控制功能在过程控制级中，工作站级的主要作用是监督管理。分散控制使得系统由于某个局部的不可靠而造成对整个系统的损害降到很低的程度，加之各种软硬件技术不断走向成熟，极大地提高了整个系统的可靠性，因而迅速成为工业自动控制系统的主流。然而 DCS 的缺点也是十分明显的。首先其结构是多级主从关系，底层相互间进行信息传递必须经过主机，从而造成主机负荷过重，效率低下，并且主机一旦发生故障，整个系统就会“瘫痪”。其次它是一种数字——模拟混合系统，DCS 的现场仪表仍然使用传统的 4～20mA 电流模拟信号，传输可靠性差，成本高。再有各厂家的 DCS 自成标准，通讯协议封闭，极大的制约了系统的集成与应用[2]。

为了克服 DCS 系统的技术瓶颈，进一步满足现场的需要，现场总线技术应运而生，现场总线技术将专用微处理器置入传统的测量控制仪表，使它们各自都具有数字计算和数字通信能力，并按照公开、规范的通信协议，在位于现场的多个微机化测量控制设备之间以及现场仪表与远程监控计算机之间，实现数据传输与信息交换，形成各种适应实际需要的自动控制系统。现场总线系统采用全分散控制。现场设备既有检测、变换、工程量处理和补偿功能，也有运算和控制功能。通过现场总线，将传统 DCS、PLC 等控制系统复杂的控制任务进行分解，分散在现场设备中，简化了系统结构，提高了系统的可靠性、自治性和灵活性。

现场总线自20世纪80年代产生以来，由于适应了工业控制系统网络化和智能化的发展方向，受到全世界工业自动化领域的普遍关注，国际上产生了很多现场总线规范。2000年IEC颁布的现场总线国际标准IEC61158包含了FF、Profibus、ControlNet、WorldFIP、P-NET、HSE、SwiftNet、Interbus等在内的8种类型现场总线(现已增加到十种类型)[3]。

由于技术和商业利益方面的原因，很难制定一个统一的标准。标准的不统一，导致不同现场总线协议互不兼容，符合不同现场总线协议的控制系统不能实现信息的无缝集成，导致了新的“自动化孤岛”的出现，这促使人们开始寻求新的出路[4]。在这种情况下，广泛应用于商业领域的以太网悄悄的进入了工业控制领域，掀起了新一轮技术革新的浪潮。

1.2 工业以太网简介

1.2.1 以太网

以太网是IEEE 802.3所支持的局域网标准，按照国际标准化组织开放系统互连参考模型(ISO/OSI)的7层结构，以太网标准只定义了数据链路层和物理层，作为一个完整的通信系统，它需要高层协议支持。在制定TCP/IP高层通信协议，并把以太网作为其数据链路层和物理层协议之后，以太网便和TCP/IP紧密地捆绑在一起。通常将以太网和TCP/IP协议一起称作以太网技术。

与现场总线相比，以太网因其协议简单、完全开放、稳定性和可靠性好而获得全球的技术支持，目前不仅在办公室自动化领域，而且在各个企业的管理网络、监控网络也都广泛使用以太网技术，并开始向现场设备层网络延伸。其具有以下优点[5-6]：

(1) 应用广泛

以太网是目前应用最为广泛的计算机网络技术，受到广泛的技术支持。几乎所有的编程语言都支持Ethernet的应用开发，如Java、Visual C++、Visual Basic等。这些编程语言由于广泛使用，并受到软件开发商的高度重视，具有很好的发展前景。因此，如果采用以太网作为现场总线，可以保证多种开发工具、开发环境供选择。

(2) 成本低廉

由于以太网应用最为广泛，因此受到硬件开发与生产厂商的高度重视和广泛支持，有多种硬件产品供用户选择，而且硬件价格也相对低廉。目前以太网网卡的价格只有Profibus、FF等现场总线的十分之一甚至百分之一，而且随着集成电路技术的发展，其价格还会进一步下降。

(3) 通信速率高

目前以太网的通信速率为 10M、100M 的快速以太网也开始广泛使用，1000M 以太网技术也已经成熟，10G 以太网正在研究。其速率比目前的现场总线快很多，因此以太网可以满足对带宽有更高要求的需要。

(4) 软硬件资源丰富

由于以太网已应用多年，人们对以太网技术的设计、应用等方面有很多的经验，对其技术也十分熟悉。大量的软件资源和设计经验可以显著降低系统的整体成本，并大大加快系统的开发和推广速度。

(5) 资源共享能力强

利用 Ethernet 作现场总线，很容易将 I/O 数据连接到信息系统中，数据很容易以实时方式与信息系统上层资源、应用软件和数据库共享。易于与 Internet 互连，能实现办公自动化网络与工业控制网络的信息无缝集成。

因此，如果工业控制网络采用以太网，就可以避免其发展游离于计算机网络技术的发展主流之外，从而使工业控制网络与信息网络技术互相促进，共同发展，并保证技术上的可持续发展，在技术升级方面无需单独的研究投入。

1.2.2 工业以太网

所谓工业以太网，即是应用于工业控制领域的以太网技术，它在技术上与商用以太网(即 802.3 标准)兼容，但又必须满足工业控制网络通信的需求。一般来讲，工业控制网络应该满足以下要求[4]：

(1) 具有较好的响应实时性

工业控制网络不仅要求传输速度快，而且在工业自动化控制中还要求响应快，即响应实时性好，一般为毫秒到 0.1 秒级；

(2) 容错性要求

在网络局部链路出现故障的情况下，能在很短的时间内重新建立新的网络链路；

(3) 力求简洁

减小软硬件开销，从而降低设备成本，同时也可以提高系统的健壮性；

(4) 开放性好

即工业控制网络尽量不要采用专用网络。

根据所处的层次不同，工业控制网络又可分为过程监控层网络和现场设备层网络，其中，过程监控层网络主要用于过程监控、优化、调度等方面信息的传输。其特点是信息传输具有一定的周期性和实时性，数据吞吐量较大，因此要求网络具有较大的带宽，以前由专用网络如令牌网组成。而现场设备层网络处于工厂综合自动化系统的最底层，

用于连接工业现场变送器、执行机构、远程 I/O 数据采集器等现场设备。除了满足上述特点以外，应用于现场设备层网络的工业以太网还应该满足：

(1) 环境适应性要求

包括机械环境适应性(如耐振动、耐冲击)、气候环境适应性(工作温度要求为-40～85℃，至少为-20～70℃，并要耐腐蚀、防尘、放水)、电磁环境适应性或电磁兼容性 EMC 应符合 EN50081-2、EN50082-2 标准。

(2) 可靠性要求

即能安装在工业控制现场，且能够长时间连续稳定运行。

(3) 安全性要求

在易暴可燃的场合，工业以太网产品还需要具有防暴要求，包括隔暴、本质安全两种方式。

(4) 总线供电要求

即要求现场设备网络不仅能传输通信信息，而且要能够为现场设备提供工作电源。这主要是从线缆铺设和维护方便考虑，同时总线供电还能减少线缆，降低成本。最近刚推出的 IEEE802.3af 标准对总线供电进行了规范。

(5) 安装方便

适应工业环境的安装要求，如采用 DIN 导轨安装。

1.3 工业以太网的实时性问题

1.3.1 以太网进入工业控制领域的技术障碍

虽然以太网具备进入工业控制领域的条件，但以太网要想全面进入现场控制领域，甚至取代现场总线，成为工业控制领域统一的标准，还存在以下技术障碍[7-9]。

(1) 通信不确定性

以太网采用 CSMA/CD 介质访问机制和 BEB 算法处理冲突，和其他网络如令牌网、令牌环网、主从式网络等相比，这是一种非确定性或随机性通信方式，导致了网络传输延时和通信响应的不确定性。对于工业控制网络，以太网的这种通信不确定性会导致系统控制性能下降，控制效果不稳定，甚至会引起系统振荡；在有紧急事件发生时，还可能因报警信息不能得到及时响应而导致灾难事故的发生，这是以太网应用于工业控制领域的主要障碍。

(2) 不适应于恶劣的工业现场环境

由于工业现场环境与商业环境相比，条件恶劣，因此要求工业控制网络必须具备气候环境适应性，耐冲击，耐振动，防尘防水，抗腐蚀以及较好的电磁兼容性，并要求很高的可靠性。

(3) 安全性和总线供电

对于应用于工业现场的网络，还要求具有向现场仪表提供电源的能力，即总线供电。在易爆或可燃场合，还需要解决防爆包括隔爆、本质安全等问题。同时还要控制对内部控制网络的访问，防止非授权用户得到网络的访问权，强制流量只能从特定的安全点去向外界，防止服务拒绝攻击以及限制外部用户在其中的行为。

以太网的通信存在不确定性，不能满足实时性要求，成为以太网应用于工业控制领域的主要障碍。

1.3.2 实时性问题的理论研究现状

在理论研究领域，为了改造以太网的实时性，国内外学者专家提出了许多种方法。这些方法主要可以分为两类：修改以太网的 MAC 层协议来达到确定性调度；在 MAC 层之上增加实时调度层[10-12]。

修改 MAC 层协议来获取以太网确定性调度的方法，主要有英国特拉思克莱德大学开发的 TEMPRA 协议[13]，美国加州大学开发的 CSMA/DCR 协议[14-15]和佩特雷大学开发的 GIT-CSMA/CD 协议[16]。它的主要思想是通过改动以太网的 MAC 层，即改变原始的载波侦听、多路访问和冲突检测(CSMA/CD)的运行机制，来达成确定性的以太网实时通信目的。这些方案在一定程度上确实可以保证工业控制实时通信的时间要求，但其也有着不可避免缺点。由于以太网的 MAC 层协议大都固化在硬件芯片中，修改 MAC 层协议则意味着必须对网络芯片重新进行 IC 设计，从而导致与传统的以太网出现兼容性问题，以太网的一致性和互操作性都将受到挑战。严格地说这些方法，已经不能称之为“以太网”了。

增加实时调度层来获得以太网的实时确定性方法，同修改以太网 MAC 层来获取确定性实时调度的方法相比，是一种更为可取的方案，它是在保留标准以太网接口的基础上，通过在 MAC 层上增加一个实时调度软件层，来实现以太网实时性的方法。该方案基于标准的以太网 IC 芯片，仅通过修改既有的软件协议来达成目的。这种方法既可满足工业通信的实时性要求，又可保证以太网的兼容性。这方面的例子主要有 TDMA 策略，虚拟时间协议 VTCSMA，窗口协议(windows protocols)和通信平滑(traffic smoothing)等。

TDMA(Time Devision Multiple Access)策略原理是，每一个节点都预先分配一个固定的时间片来发送数据，因此可以获得一个可预测的时间行为[17-19]。但它的缺点就是不能反映每个节点的实际带宽需求，效率不高。基于此，Rajendra.Y 等人提出了PCSMA(Predictable CSMA) [20]，他假定所有的实时信息都是周期性的，是一种离线的静态调度。Found.A 在 TDMA 的基础上，给出了具有信息优先级的 P-CSMA[21]，它把时间分为 n 个时间片，节点只能在相应的时间片内发送具有最高优先级的数，因此保证了不同优先级数据不会冲突，介质访问控制的公平性得到了提高。

Molle M 和 Kleinrock 提出的虚拟时间协议 VSCSMA 可以动态的避免冲突，当介质空闲时，节点不是立即发送数据，而是延迟一个预定的时间后在发送，延迟时间是某个参数的函数值，例如截至期、松弛期和优先级等。它极大的提高了通信速率，但缺点是节点以前发送信息的状态无法记录。ZHAO.W 和 Ramamritham.K 给出了一种基于优先级的 VTCSMA，它根据不同的延迟时间分成不同的优先级，优先级越高的信息延迟时间越短[22-23]。

窗口协议(window-based protocol)通过在全网采用完全一致的放大，缩小和移动窗口等行为来有效的限制同时发送信息的个数，进而避免，减少或者解决冲突。传统的窗口协议是不考虑存在时限信息的，Zhao.W 等提出了一种考虑时限信息的方法，它基于 LS (latest time to send a message)，即最近到达的信息被发送的可能性最大[24]。

通信平滑(traffic smoothing)是在 TCP(UDP)/IP 层与 MAC 层之间增加通信滤波器，减少实时信息和非实时信息的冲突，具体又分静态平滑和自适应平滑两种。前者通过离线给每个节点分配信息发送的速率，缺点是网络利用率不高。后者是通过网络符合的在线监测对发送速率进行动态分配[25]。

1.3.3 各家实时以太网的解决方案

在实际的工程应用领域，为了满足工业以太网在引用中实时性能的需要，各大公司和标准组织纷纷提出各种提升工业以太网实时性的技术解决方案。这些方案建立在IEEE 802.3 标准的基础上，通过对其和相关标准的实时扩展提高实时性，并且做到与标准以太网的无缝连接，这就是实时以太网(real time Ethernet，简称 RTE)[26]。

为了规范这部分工作的行为，2003 年 5 月，IEC/SC65C 专门成立了 WG11 实时以太网工作组，负责制定 IEC 61784-2“基于 ISO/IEC 8802-3 的实时应用系统中工业通信网络行规”国际标准。该标准包括 Communication Profile Family 2 Ethernet/IP、CPF3 PROFINET、CPF4 P-NET、CPF6 Interbus、CPF10 VNET/IP、CPF11 TCNET、CPF12 EtherCAT、CPF13Ethernet Powerlink、CPF14 EPA(中国)、CPF15 Modbus/TCP 以及 CPF16

SERCOS 等 11 种实时以太网行规集。其中，包括我国 EPA 实时以太网标准的 6 个新增实时以太网将以 IEC PAS(publicly available specification)公共可用规范予以发表。在上述实时以太网技术中，将有 EPA、EtherCAT、Ethernet Powerlink、PROFINET、Modbus-IDA 和 Ethernet / IP 等 6 个主要的竞争者。

(1) Profinet 实时以太网

Profinet 实时以太网是由 Profibus International(PI)组织提出的基于以太网的自动化标准。从 2004 年 4 月开始，PI 与 Interbus Club 总线俱乐部联手，负责合作开发与制定标准。Profinet 构成从 I / O 级直至协调管理级的基于组件的分布式自动化系统的体系结构方案，Profibus 技术和 Interbus 现场总线技术可以在整个系统中无缝地集成。

Profinet 提出了对 IEEE 802.1D 和 IEEE 1588 进行实时扩展的技术方案，并对不同实时要求的信息采用不同的实时通道技术。Profinet 通信协议模型如图 1.1 所示。从图中可以看出，Profinet 提供一个标准通信通道和两类实时通信通道。标准通道是使用 TCP/ IP 协议的非实时通信通道，主要用于设备参数化、组态和读取诊断数据。实时通道 RT 是软实时 SRT(SoftwareRT)方案，主要用于过程数据的高性能循环传输、事件控制的信号与报警信号等。它旁路第 3 层和第 4 层，提供精确通信能力。为优化通信功能，Profinet 根据 IEEE 802.1p 定义了报文的优先级，最多可用 7 级。实时通道 IRT 采用了 IRT(isochronous real-time)等时同步实时的 ASIC 芯片解决方案，以进一步缩短通信栈软件的处理时间，特别适用于高性能传输、过程数据的等时同步传输、以及快速的时钟同步运动控制，在 1ms 时间周期内，实现对 100 多个轴的控制，而抖动不足 1μs。

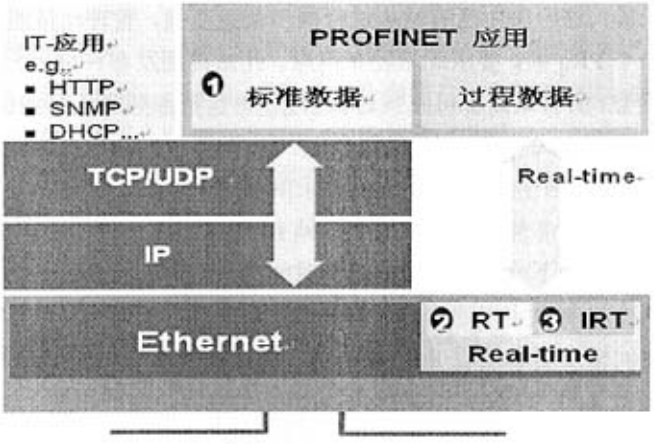

图 1.1 Profinet 通信协议模型

Fig. 1.1 Communication protocol model of Profinet

(2) Ethernet Powerlink 实时以太网

奥地利贝加莱(B&R)公司开发的Ethernet Powerlink(EPL)标准是一种可满足最苛刻实时要求(4级)、并已投入实际应用的工业以太网。该公司当初开发EPL的思路是在标准以太网基础上建立一个现场总线系统来满足控制中最苛刻的实时要求，同时克服以上介绍的传统解决方案的局限性。

为避免冲突、尽量利用带宽，EPL在时间上重新组织了网络中站间信息交换机制，在CSMA基础上引入时间槽管理机制。网络其中一个站点充当管理站管理网络通信，对其他所有站点给定同步节拍，分别分配各站发布权限，各站只能在得到发布权限后才可发布信息。EPL采用IEEE1588进行时钟同步。

一个EPL通信周期可分成4个阶段，如图1.2所示。

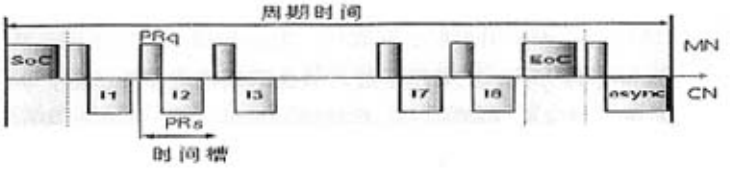

图 1.2 EPL 通信周期
Fig. 1.2 EPL communication cycle

(1) 开始阶段：管理员发布“通信周期开始(SoC)”信号，信号以广播方式发给所有站点。此信号发出后，各站点就此同步。

(2) 同步阶段：这阶段中所有站点进行同步信息交换，管理站按照一个事先定义的顺序给某站发一个PRq帧，要求此站发布信息。此站得到发布许可后，以广播形式发出一帧PRs回应信息，所有站点都可收到这帧信息，也包括那些应该得到这帧信息的站点。站点间直接横向通信方式和CAN总线很相似。

(3) 异步阶段：这个阶段是给无实时要求的信息留下的，管理站发给某站一个“邀请”帧，此站便可发布非同步信息，比如一帧IP信息。

(4) 闲置阶段：到下一个周期前的等待时间。

EPL通过两个机制来实现通信的无缝集成：实时和非实时(异步)信息可同时传输；透明地在异步时间槽发送和接收IP协议信息。通信模型如图1.3所示。

EPL接口函数API和标准以太网驱动函数完全兼容。在应用层上，基于IP的协议或软件都可不加改动就直接使用。在通信实时性不重要时，如程序下载、系统编程诊断

或参数配置过程中，用 EPL 的开放模式可使所有站点作为普通以太网站点的模式来使用（非实时模式）。在这个模式下 EPL 站点对普通以太网来说完全透明。

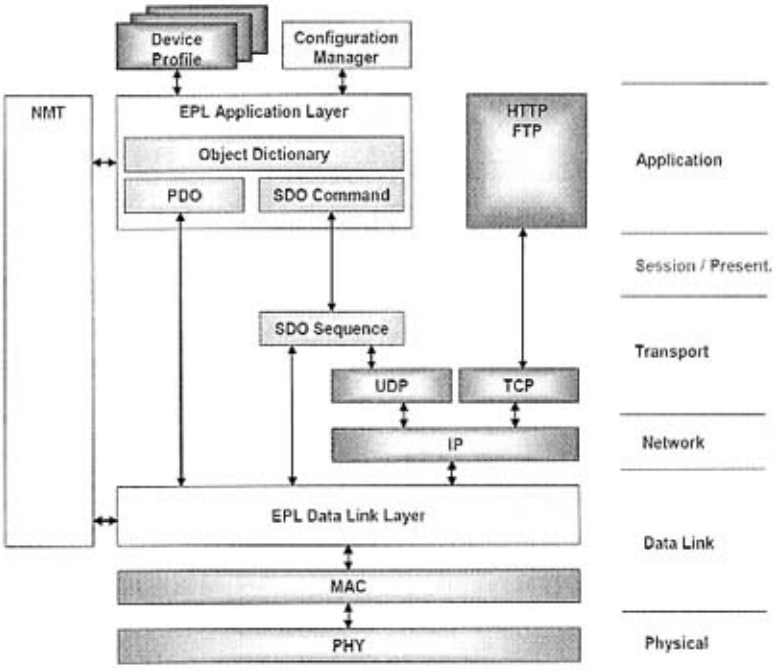

图 1.3　EPL 通信协议模型

Fig. 1.3　Communication protocol model of EPL

(3) EPA 实时以太网

EPA(Ethernet for plant automation)用于工业测量与控制系统的以太网标准在国家科技部“863”计划的支持下，由浙江大学、浙大中控、中科院沈阳自动化所、重庆邮电学院、大连理工大学、清华大学等单位联合组成的以浙江中控技术股份有限公司总裁金建祥教授为组长的标准起草工作组起草。关于 EPA 的具体情况将在第二章中详细描述。

1.4　课题主要内容与本文结构

本课题的主要内容是在深入理解了 EPA 实时性调度解决方案后，在现有的软硬件平台之上，采用 IEEE1588 精确时钟同步协议实现了设备间的精确时钟同步，通过 Linux 的 QoS 接口实现了链路层调度，并且组建了测试平台，开发了上位机测试软件和下位机测试程序，对 EPA 的实时性性能进行了测试。

本文的主要内容如下：

(1) 第一章序论部分回顾了工业控制系统发展历史，介绍了以太网和工业以太网的概念，阐述了以太网进入工业控制领域的技术障碍，并从当今理论研究和工程实际解决方案两个角度就实时性问题进行了描述。

(2) 第二章主要介绍了 EPA 的通信模型和实时性通信调度解决方案。

(3) 第三章介绍 EPA 时钟同步的实现。简要介绍了实现平台，研究了 IEEE1588 精确时钟同步协议的工作原理，深入分析了影响时钟同步精度的因素，并采取了一系列措施来提高同步精度，并在 Linux 系统中进行了具体实现。

(4) 第四章介绍了 Linux 内核中的网络协议栈，包括主要数据结构、链路层数据收发和 QoS 接口，并通过其 QoS 接口实现了 EPA 的链路层调度策略。

(5) 第五章是实时性测试，介绍了实时性测试平台和上、下位机测试软件的设计，并在测试平台上对时钟同步精度、递交时间和调度进行了测试。

最后是全文的总结。

1.5 小结

本章首先回顾了工业控制系统的发展历史，介绍了以太网和工业以太网的概念，并阐述了以太网进入工业控制领域的主要技术障碍，其中着重介绍了实时性问题，从理论角度介绍了当前国际上实时性问题的研究现状，然后从工程应用角度介绍了当前世界上主要的厂商的实时以太网解决方案。最后是本文的课题内容和组织结构。

2 EPA 实时以太网

2.1 EPA 简介

EPA(Ethernet for Plant Automation)是在国家标准化管理委员、全国工业过程测量与控制标准化技术委员会的支持下，由浙江大学、浙江中控技术有限公司、中国科学院沈阳自动化研究所、大连理工大学、重庆邮电学院、清华大学、上海工业自动化仪表研究所、机械工业仪器仪表综合技术经济研究所、北京华控技术有限责任公司等单位联合成立的标准起草工作组，经过 3 年多的技术攻关而提出的基于工业以太网的实时通信控制系统解决方案。

EPA 实时以太网技术的攻关，以国家“863”计划 CIMS 主题系列课题“基于高速以太网技术的现场总线控制设备”、“现场级无线以太网协议研究及设备开发”、“基于蓝牙技术的工业现场设备”、“监控网络及其关键技术研究”，以及“基于 EPA 的分布式网络控制系统研究和开发”、“基于 EPA 的产品开发仿真系统”等滚动课题为依托，先后解决了以太网用于工业现场设备间通信的确定性和实时性、网络供电、互可操作、网络安全、可靠性与抗干扰等关键性技术难题，开发了基于 EPA 的分布式网络控制系统，首先在化工、制药等生产装置上获得成功应用。

在此基础上，标准起草工作组起草了我国第一个拥有自主知识产权的现场总线国家标准《用于工业测量与控制系统的 EPA 通信标准》。

2.2 EPA 的通信模型

EPA 网络通信模型参照 ISO/OSI 参考模型，取其物理层、数据链路层、网络层、传输层、应用层，并在应用层之上增加用户层(采用 IEC 61499/61804 标准)，在网络层和传输层之间增加 EPA 实时通信调度接口，通信模型如图 2.1 所示[27]。

(1) 物理层和数据链路层

为 EPA 提供数据传输物理道，并描述了多个设备共享通信信道的一种机制。在 EPA 中采用了 IEEE 802 系列范围，即 IEEE802.3、IEEE802.11 和 IEEE 802.15，但在传输介质与物理接口上增加适用于工业生产现场的应用导则。

(2) EPA 实时通信管理接口

EPA 通信调度接口定义了网络层(IP 层)与数据链路层(MAC 层)之间的接口，用于控制由网络层到 MAC 层的实时数据包与非实时数据包的传输调度，以满足 EPA 通信过程的实时性。

(3) 网络层和传输层

EPA 的网络层和传输层为 EPA 应用层提供报文传输的平台，采用 TCP(UDP)/IP 协议集，其中 UDP 协议不需要在通信两端建立连接和确认，用于实时通信；对于实时性要求不高、对传输的可靠性要求高的应用，可使用 TCP 协议，也可使用 UDP 协议。

(4) 应用层

EPA 应用层为 EPA 设备之间周期和非周期的传输数据提供通信通道和服务接口。它由 EPA 实时通信规范和通用通信协议两部分组成。其中 EPA 实时通信规范是专门为 EPA 实时控制应用进程之间的数据传输提供通信通道和服务接口。而通用通信协议则主要包括 HTTP、FTP、TFTP 等互联网络中广泛使用的协议。

(5) 用户层

用户层直接面向用户，用户根据自己的控制逻辑需要，利用组态软件组态不同功能块应用进程以完成各种控制策略，也可根据自己的需要组态各种非实时性应用程序的服务。EPA 用户层采用基于 IEC 61499 定义的功能模块结构模型和 IEC 61804 定义的功能模块元素。

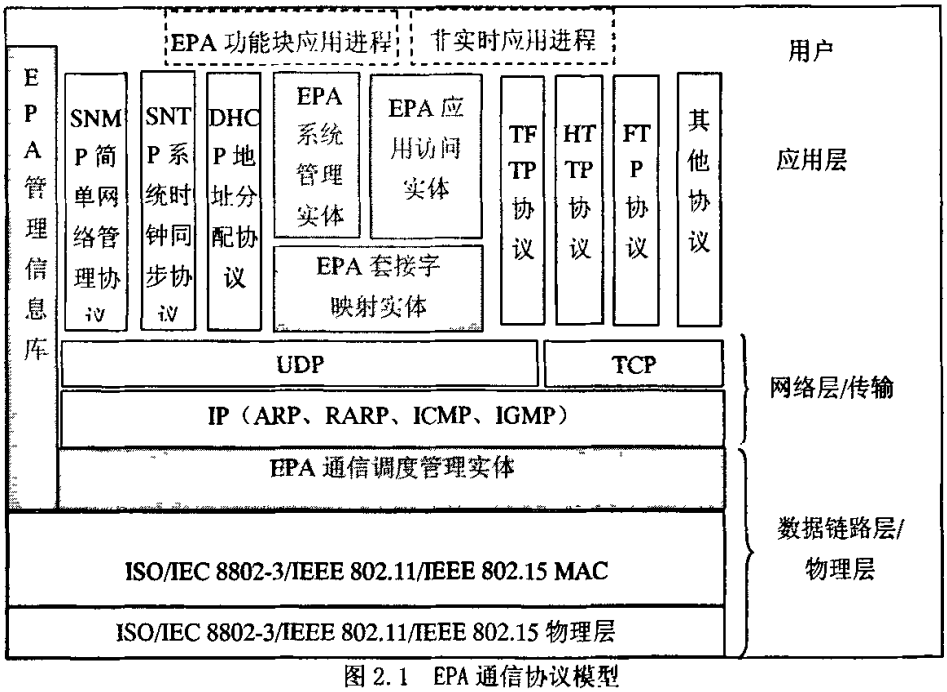

图 2.1 EPA 通信协议模型

Fig. 2.1 EPA communication protocol model

EPA 系统结构的主要组成除了 ISO/IEC 8802-3/IEEE 802.11/IEEE 802.15、TCP(UDP)/IP、SNMP、SNTP、DHCP、HTTP、FTP 等协议组件外，它包括以下几个部分：

(1) 应用进程，包括 EPA 功能块应用进程与非实时应用进程

(2) EPA 系统管理实体

(3) EPA 应用访问实体

(4) EPA 管理信息库

2.3 EPA 的通信调度

基于 EPA 的分布式网络控制系统，分为现场设备层和过程监控层两个层次，对这两层分别采用了不同的通信调度方式。对于现场设备层，采取的是一种确定性的实时通信调度；对于过程监控层则采用 CSMA/CD 这种非确定性通信调度。

2.3.1 EPA 实时通信和非实时通信

为了实现既能在现场设备之间的实时通信，又能向上与企业信息网兼容，在 EPA 系统中，将通信分为实时 EPA 通信和非实时通信。其中实时 EPA 通信的信息又可分为周期数据和非周期数据两大类。周期数据主要指与过程有关的数据，如需要按控制回路的控制周期传输的测量值、控制值，或功能块输入、输出之间需要按周期更新的数据。周期报文的发送优先级为最高。非周期数据主要指用于以非周期方式在两个通信伙伴间传输的数据，如程序的上下载数据、变量读写数据、事件通知、趋势报告等数据，以及诸如 ARP、RARP、HTTP、FTP、TFTP、ICMP、IGMP 等应用数据。

根据以太网(IEEE 802.3)协议规定，对于帧格式中的长度/类型(LENGTH/TYPE)字段，当该字段的值小于等于 1536(0x0600)时表示数据长度，当该字段的值大于 1536(0x0600)时，则表示以太网上的协议，如 0x0800 表示 IP 协议，0x0806 表示 ARP 协议，0x8035 表示 RARP 协议，0x 8137 表示 IPX 协议等。在 EPA 系统中，同样采用该字段，定义一个值表示 EPA 实时协议(如 0x8888，或 65C306/314 提案中定义的 0x8842)。利用该字段的值，在 EPA 实时通信调度接口中，对实时 EPA 通信与非实时通信进行调度，使实时 EPA 通信优先于非实时通信，从而保证 EPA 通信的实时性。

2.3.2 现场设备间的确定性通信调度

(1) 链路层模型

为了在现场设备间实现实时通信调度，EPA 对 ISO/IEC 8802-3 协议规定的数据链路层进行了扩展，增加了一个 EPA 通信调度管理实体，如图 2.2 所示。

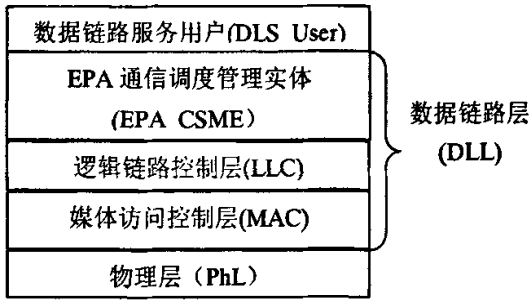

图 2.2 EPA 数据链路层模型

Fig. 2.2 EPA data link layer model

EPA 通信调度管理实体是在 ISO/IEC 8802-3 协议规定的数据链路层基础上进行的扩展，它从 DLS_User 接收数据报文，并传送给 LLC；同时解释从本地 LLC 接收的数据帧，并传送给 DLS_User。EPA 通信调度管理实体不改变 ISO/IEC 8802-3 数据链路层提供给 DLS_User 的服务，也不改变与物理层的接口，只是完成对数据报文的调度管理。

在一个 EPA 微网段内，所有 EPA 设备的通信均按周期进行，完成一个通信周期所需的时间 T 称为一个通信宏周期(Communication Macro Cycle)。

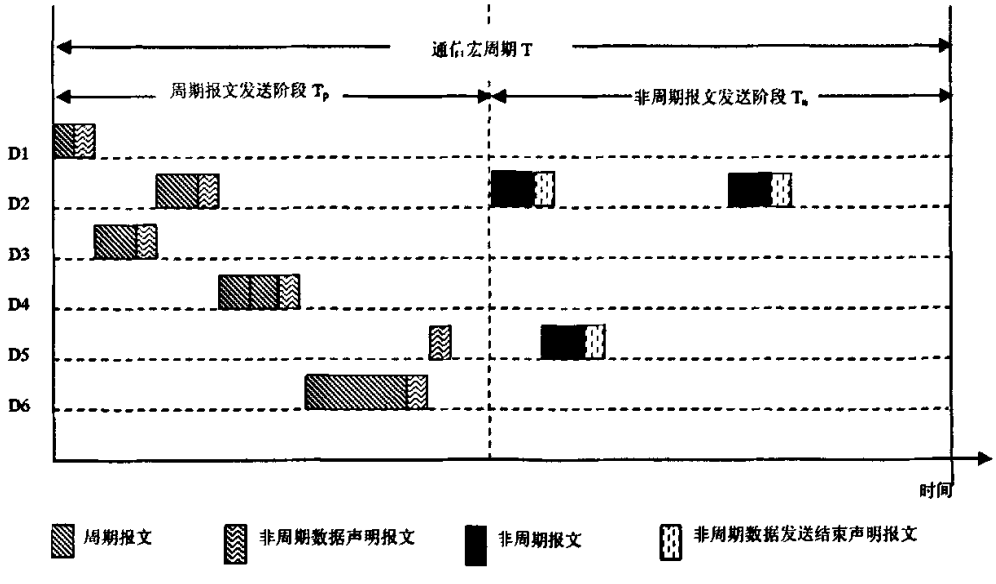

图 2.3 EPA 通信调度示意图

Fig. 2.3 EPA's communication scheduling

一个通信宏周期 T 分为两个阶段，其中第一个阶段为周期报文传输阶段 Tp，第二个阶段为非周期报文传输阶段 Tn(如图 2.3 所示)。

(2) 调度规程

一个 EPA 设备的通信调度管理协议状态机是用 4 个状态以及它们之间的转换来描述的。图 2.4 表示了这些状态之间的转换关系。

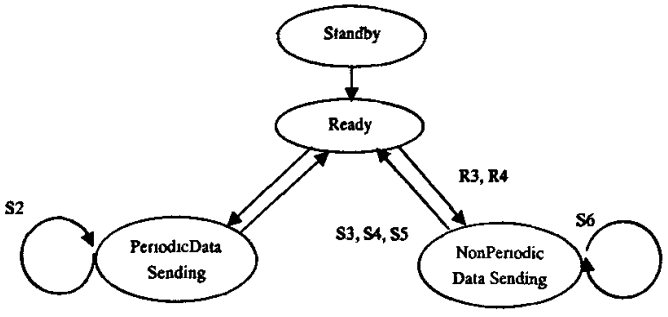

图 2.4 EPA 通信调度管理实体 EPA_CSME 状态转换图

Fig. 2.4 The state transition of EPA_CSME

(2) 状态转换与说明

① Standby 状态

EPA 设备上电后，应检测所有必需的操作参数，如未经初始化组态，EPA 通信调度管理实体 EPA_CSME 则进入 Standby 状态，直至被用户组态。否则，自动进入 Ready 状态，按下列公式计算通信宏周期的起始时间：

当前时间距离通信宏周期 T 起始时间的偏离量 = MOD(本地当前时间，T)

即本地当前时间对通信宏周期T取模所得的结果即为当前时间距离通信宏周期T起始时间的偏离量。

处于 Standby 状态时，EPA 通信调度管理实体 EPA_CSME 对所有的 DLS_User DATA 不作任何处理，直接传送给 ISO/IEC 8802-3 规定的 DLE，对来自于 DLE 的数据，也直接传送给 DLS_User。

② Ready 状态

EPA 通信调度管理实体 EPA_CSME 处于 Ready 状态时，EPA 设备处于通信调度控制状态：

a) EPA 通信调度管理实体 EPA_CSME 接收到来自 DLS_User 的 DLS_User DATA 时，先放入缓存区进行缓存；

b) EPA 通信调度管理实体 EPA_CSME 接收到来自 DLE 解析过的 DLPDU 时，首先检查其是否为“非周期数据声明报文 NonPeriodDataAnnunciation。如果是，则将其发送方的 IP 地址、非周期报文的发送优先级等参数存放在预先建立的非周期报文发送管理列表中，否则将其直接提交给 DLS_User。

③ Periodic Data Sending 状态

当 EPA 通信调度管理实体 EPA_CSME 检测到本地设备发送周期报文的时间到时，即：

MOD(本地当前时间，T)= 周期报文发送时间偏离量 SendingTimeOffset

EPA 通信调度管理实体 EPA_CSME 状态改变为 PeriodicDataSending 状态。当 EPA_CSME 处于 PeriodicDataSending 状态时，首先检查有无优先级为 0 的报文，如果没有，则发送非周期数据声明 NonPeriodDataAnnunciation 报文。否则，将需要在此时发送的 DLS_User DATA 依次传送给 DLE，由 DLE 通过 PhLE 发送到网络上，然后 EPA 通信调度管理实体 EPA_CSME 再将 NonPeriodDataAnnunciation 报文传送给 DLE，通过 PhLE 发送到网络上，并将其状态改变为 Ready。

④ NonPeriodicDataSending 状态

当 EPA 通信调度管理实体 EPA_CSME 检测到一个通信宏周期内非周期报文传输开始时间到，即：

MOD(本地当前时间，T) = NonPeriodicDataTransferOffset，或者

本地设备接收到远程设备发送的非周期数据发送结束声明报文 EndofNonPeriodDataSending 时，EPA 通信调度管理实体 EPA_CSME 状态改变为 NonPeriodicData Sending 状态。

EPA 通信调度管理实体 EPA_CSME 处于该状态时，按以下规程进行操作：

a) 本地设备有无非周期(优先级不为 0 的)数据需要发送，如没有，则进入 g)；否则进入下一步；

b) 检查本地非周期报文发送管理列表，比较本地与本网段内所有远程设备的非周期报文发送优先级。如本地设备的非周期报文优先级低于所有远程设备，则进入 g)；否则进入下一步；

c) 如本地设备的非周期报文优先级高于所有远程设备，且本次宏周期内剩余时间足够该数据的完整发送，则 EPA 通信调度管理实体 EPA_CSME 将该非周期报文传送给 DLE，由 DLE 发送到网络上，转入 f)；否则保留到下次发送，转入 g)；

d) 如本地与一个或多个远程设备的非周期报文发送优先级相同，且为最高时，若本地设备的 IP 地址较大，则进入 g)；否则进入下一步；

e) 如本次宏周期内剩余时间足够该数据的完整发送，则 EPA 通信调度管理实体 EPA_CSME 将该非周期报文传送给 DLE，由 DLE 发送到网络上，转入 f)；否则保留到下次发送，转入 g)；

f) 如果本地设备还有未发送的非周期报文，则转到 b)；

g) 如本地设备发送了至少一个非周期报文，EPA 通信调度管理实体 EPA_CSME 向 DLE 发送“非周期数据发送结束声明”报文 EndofNonPeriodDataSending;

h) 将本地通信调度管理实体 EPA_CSME 的状态改变为 Ready 状态。

2.4 EPA 的时钟同步

对于现场设备层的实时通信来说，通信调度能否实现依赖于网络上所有设备的时钟同步。只有精确的时钟同步，才能保证周期信息调度和非周期信息调度正确得以完成。因此，所有的 EPA 设备的时间必须同步。

时间同步协议可参考 RFC 2030 简单网络时间协议(SNTP)或 IEEE1588 精确时间同步协议。EPA 标准没有做另外规定。

在实现 EPA 的时钟同步时，采用了 IEEE1588 精确时钟同步协议，关于 IEEE1588 的原理与具体实现在第三章中详细介绍。

2.5 小结

本章首先对 EPA 做了简单说明，然后对从物理层和数据链路层、实时通信调度管理实体、传输层和网络层、应用层和用户层对 EPA 的通信模型做了介绍。在通信模型的基础上，着重对实时通信调度管理实体进行了详细的阐述。根据网络中数据信息类型的不同，EPA 给出了不同的调度策略，即对过程监控层采用以太网原来的 CSMA/CD 机制，而对现场设备层，采用 TDMA 形式的确定性调度。EPA 的确定性调度采用分时发送的机制对周期数据和非周期数据进行调度发送，避免了碰撞，提高了通信的确定性和实时性。

3 EPA 时钟同步实现

EPA 通信调度是 EPA 的以太网实时性解决方案，它必须建立在设备时钟同步的基础之上。在精确时钟同步控制下，根据预先组态的带宽分配，各个设备相互协作，完成相互间的通信任务。

3.1 实现平台

实现的硬件平台为 DUT5000(工业以太网控制模块)的硬件升级版本，处理器采用的是 Atmel 公司生成的 ARM9 芯片 AT91RM9200。软件平台为 ARM Linux，内核版本为 2.4.19。

3.1.1 硬件平台

DUT5000 硬件分为两部分，底层板和上层核心板。其中底层板为电源电路和基本的外围扩展 IO 接口电路，包括 AI、AO、DI、DO 和通信接口(串行通信和以太网通信)等；上层核心板包括处理器，SDRAM、FLASH 等。下面主要对核心板的处理器部分及其以太网接口进行说明。

AT91RM9200 是 Atmel 公司基于 ARM920T 核的高性能、低功耗 16/32 位 RISC(精简指令集计算机)微处理器，内部集成丰富的外设资源，适用于要求外设资源丰富、功耗低、工作严格稳定的工业控制等方面，如嵌入式工业控制、医疗设备、网络通信、移动计算等。

AT91RM9200 片内集成了非常丰富的外围功能模块，包括全功能 MMU 虚拟内存管理单元、内部 16 Kb SRAM 和 128 kB ROM、EBI 接口控制器、增强的时钟和 PMC(电源管理控制器)，带有 2 个 PLL(锁相环)的片内振荡器，4 个可编程的外部时钟信号，包括定时中断、看门狗、秒计数器的系统定时器，带报警中断的实时时钟，带有 8 级优先级、可单个屏蔽中断源的 AIC(先进中断控制器)，7 个外部中断源和 1 个快速中断源，4 个 32 位的 IO 控制器，20 通道外围数据控制器(PDC 或 DMA)，1 个 10 Mbps/100 Mbps 以太网控制器，1 个 USB 2.0 主机接口，1 个 USB 2.0 设备接口，2 个多媒体卡接口，3 个 SSC(同步串行口控制器，兼容 IIS)，4 个 UASRT(通用同步/异步串行口)，1 个主/从 SPI(串行设备接口)，1 个两线串行接口 TWI(主模式)，JTAG/ICE 接口等。

实现中用到了外设中定时器资源，中断管理器和以太网接口，在以后章节用到的地方会对其进行详细介绍。

系统的硬件模块实物图如图 3.1 所示。

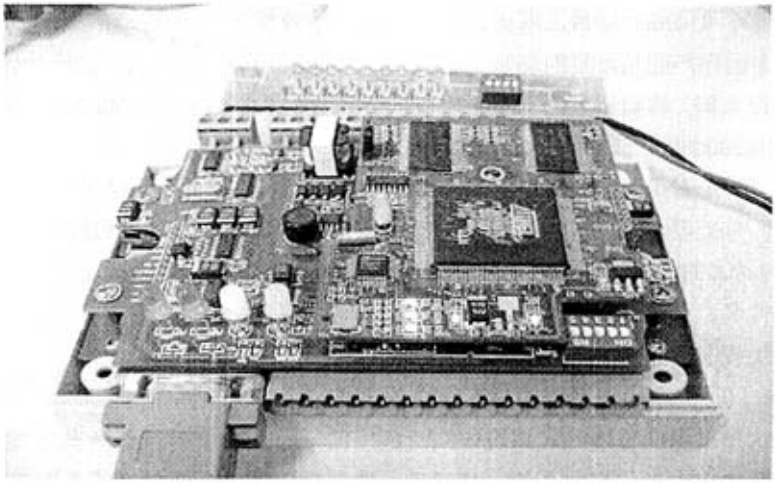

图 3.1 EPA 硬件平台

Fig. 3.1 EPA hardware platform

3.1.2 软件平台

ARM Linux 为 Linux 在 ARM 系列处理器上的移植版本，它充分考虑了 ARM 系列处理器的特性，其官方网站为 http://www.arm.linux.org.uk。近年来，Linux 因其开源、强大的网络功能等一系列特性，在嵌入式领域应用越来越广泛。

软件平台主要包括经过裁减配置过的 ARM Linux 内核(最终大约 600k)，用于加载和引导操作系统的 U-Boot，文件系统(采用 ramdisk)。

(1) 搭建开发平台

首先建立超级终端，将 PC 机上的串口与硬件平台上的调试串口连接起来，启动 PC 机上的超级终端(Windows 上为超级终端，Linux 上为 minicom)，设置串口通信参数为 115200bps，8 位数据位，无奇偶校验，1 个停止位，无数据流控制。

(2) 烧写 U-Boot

U-Boot 是嵌入式系统中用于加载操作系统的 bootloader，它的主要功能相当与 PC 机中的 bios。它负责把操作系统加载到内部 RAM 中，对操作系统的运行环境进行初始化，并且运行操作系统。

通过超级终端把 U-Boot 下载到内部 RAM 中，再又 U-Boot 自身的烧写 Flash 命令把 U-Boot 本身从内部 RAM 中拷贝并烧写到 Flash 中去，重新上电运行，U-Boot 就能启动起来。

(3) 加载操作系统内核

在烧写了 U-Boot 之后，就可以加载操作系统内核了。在调试阶段可以把内核下载在 RAM 中运行，避免反复的烧写 Flash。

通过以太网，将编译好的 Linux 内核下载到 RAM 中，在超级终端中敲入如下命令：

Tftp 0x20008000 zImage

其中 tftp 是 U-Boot 自带的命令，是 tftp 的客户端，此时需在 PC 机上的 tftp 服务器启动起来，并把要传送的文件放到 tftp 的目录下，U-Boot 和 PC 机通过 tftp 协议进行文件传输。0x20008000 是将内核下载到 RAM 中的从 0x20008000 开始的位置。zImage 是传输的文件名，此处为内核文件名。

(4) 加载文件系统

Linux 只是一个操作系统内核，还必须还有一个根文件系统才能构成一个完整的操作系统。这里采用的是嵌入式系统中广泛应用的 ramdisk 文件系统，其实就是把 RAM 中的一部分拿出来当作文件系统，相当于一块硬盘，但这个文件系统是存在与内存中，掉电丢失，要想保存，可以采用 Flash 文件系统 jffs2 等。

Tftp 0x21000000 ramdisk.gz

(5) 运行操作系统

运行内核并且挂载根文件系统，进入操作 shell：

Go 0x20008000

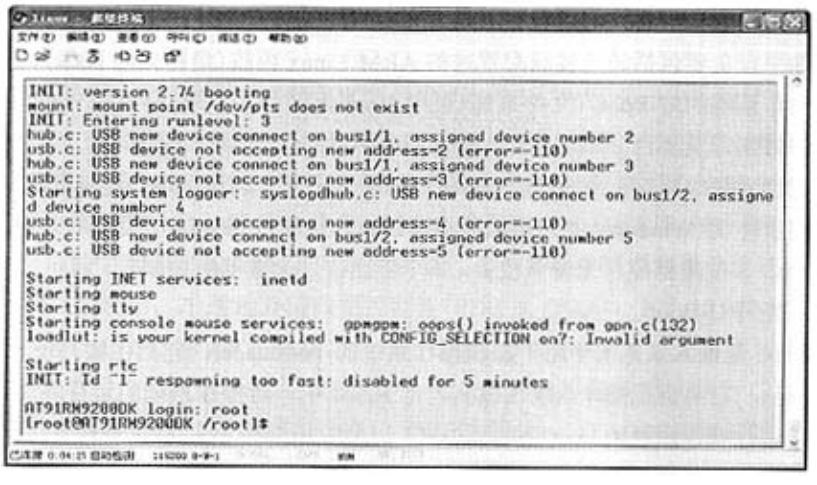

图 3.2 启动操作系统

Fig. 3.2 Run the operating system

(6) 编写、编译、下载和运行应用程序

在 PC 机上编写应用程序，用 arm-linux-gcc 交叉编译工具编译成目标板代码，然后通过 ftp 或 nfs 的方式把其下载到目标板中运行。Ramdisk 文件系统中带有 ftp 客户端命令，需要在 PC 机上启动 ftp 服务器才能传送文件。也可以通过 nfs 的方式，不过默认的 Linux 内核没有把 nfs 的支持编译进去，需重新配置内核，选中其中的如下选项：

Network File System->NFS system support 和 Provide NFSv3 client support

重新编译内核。然后启动 PC 机上的 NFS 服务器，建立共享目录，并把其挂载到目标机的文件系统中，这样在目标机的文件系统中就可以看到 PC 机上的文件了，可以直接进行各种操作。

3.2 IEEE1588 精确时钟同步协议

3.2.1 IEEE1588 简介

目前，许多应用系统都是建立在分布式网络环境上，此时，如果没有一个统一的、准确的时钟，这些应用很难正常地协调工作和运行。特别在分布式控制系统中，考虑到实时性的调度和控制，对时间统一的要求就更为严格。所以建立一个时间统一的分布式系统，是分布式网络的基本要求。

目前主要的时间同步协议有 SNTP 和 IEEE1588 两种。在 SNTP 中，同步用的时间戳都是在应用层获得的，并且它的实现假定了客户机和服务器之间的报文传输延迟相等。而报文传输延迟不仅包括报文在网络上传输的延迟，还包括报文在节点内部的处理时间(如协议封装或分解、系统中断、进程调度等)，由于客户机和服务器处理能力不同，故它们之间的报文传输时延实际上并不严格相等，从而带来时间误差，使得 SNTP 的同步精度只能达到毫秒级[28]。

IEEE 1588(网络测控系统精确时钟同步协议)最初由 Agilent Laboratories(安捷伦实验室)的 John Eidson 以及来自其它公司和组织的 12 名成员开发，后来得到 IEEE 的赞助，并于 2002 年 11 月得到 IEEE 批准。

IEEE 1588 的基本功能是使分布式网络内的最精确时钟与其他时钟保持同步，它定义了一种精确时间协议 PTP(Precision Time Protocol)，用于对标准以太网或其他采用多播技术的分布式总线系统中的传感器、执行器以及其他终端设备中的时钟进行亚微秒级同步。该协议为小型同构或异构局域网设计，设计者特别注意降低资源使用，使其可以在低成本终端设备上应用。该协议对内存及 CPU 性能没有特殊的要求，只需要最小限度的网络带宽。

3.2.2 IEEE1588 工作原理

IEEE1588 的工作原理：通过在网络内的主时钟和从时钟之间相互不断地交换信息，来确定主从时钟之间的时间偏差和网络传输延迟，从时钟利用这些信息调整本地时间从而达到与参考时钟的精确同步[29]。

IEEE1588(IEEE1588-2002)定义了五种格式的报文：同步报文，附加信息报文，延时请求报文，延时请求响应报文和管理报文[30-31]。

主时钟和从时钟之间的相互信息交换过程如图 3.3 所示[32]。

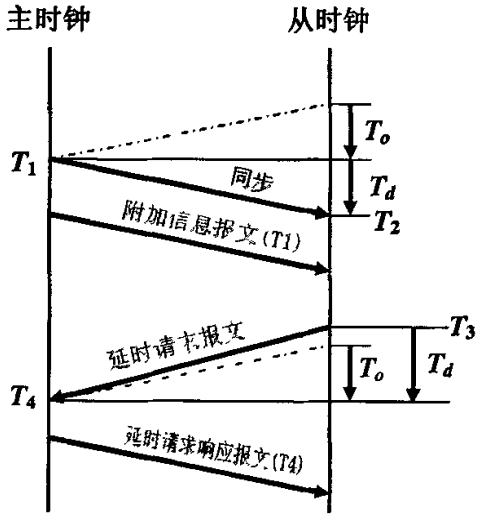

图 3.3 PTP 信息交换

Fig. 3.3 PTP message exchange

在图 3.3 中，T_1 表示同步报文的发送时间，T_2 表示从时钟接收同步报文的时间，T_3 表示从时钟发送延时请求报文的时间，T_4 表示主时钟接收到延时请求报文的时间，T_o 表示从时钟相对于主时钟的时间偏移，T_d 表示网络传输延迟。

同步过程分成两个阶段：时钟偏差测量阶段和网络延迟测量阶段。

在偏差测量阶段，主时钟周期性的向网络中以组播的形式发送同步报文，同步周期一般为 2 秒。主时钟记录下同步报文的发送时间 T_1，并发送一个附加信息报文把 T_1 传送到从时钟端。从时钟收到同步报文，记录其接收时间 T_2，则有：

$$T_1 + T_d - T_o = T_2 \qquad (3.1)$$

$$\text{即：}\ T_d - T_o = T_2 - T_1 \qquad (3.2)$$

T_d是网络传输延迟，其值在网络延迟测量阶段得到。

在网络延迟测量阶段，从时钟向主时钟发送延时请求报文。从时钟记录下延时请求报文的发送时间T_3，主时钟接收到延时请求报文，记录其接收时间T_4，并通过延时请求响应报文把T_4发送给从时钟。则有：

$$T_3 + T_d + T_o = T_4 \tag{3.3}$$

$$即：\ T_d + T_o = T_4 - T_3 \tag{3.4}$$

从时钟发送延时请求报文的周期随机分布在 2 到 30 个同步周期中，因为如果所有从时钟同时向主时钟发送延时请求报文，那么会导致主时钟工作繁忙，从而使网络延迟测量不准。

经过时钟偏差测量阶段和网络延迟测量阶段，由(3.4)-(3.2)得：

$$T_o = [(T_4 - T_3) + (T_1 - T_2)] / 2 \tag{3.5}$$

则T_o就是从时钟相对于主时钟的偏差。

从时钟根据此时钟偏差调整自己的本地时间，从而和主时钟保持一致：

$$T_s = T_s + T_o \tag{3.6}$$

3.2.3 影响时钟同步精度的因素

时钟同步精度主要取决于时间戳的测量精度，它们必须尽可能精确的反映出交换报文的接收和发送时间。

时钟精度是系统时间最小能精确到的时间单位，它是影响时间戳测量精度的一个重要方面。它与系统的硬件定时器有关，还跟操作系统的实现有关。高精度的时间戳要由高精度的系统时钟来保证。比如在一个精度只有 ms 的系统中要实现精度为μs 的时间戳是不可能的，必须修改系统的默认时钟实现以提高精度。

另一个影响因素是时间戳的记录位置。时间戳的记录时间(非时间戳的本身值)必须尽可能的反映出报文的发送和接收时间。最精确的方法是采用硬件来检测和识别 PTP 报文并记录时间戳，但采用这种方法需要硬件的支持。另外一个比较好的地方是网卡驱动里面。通过修改网卡的驱动程序，对同步报文进行识别并记录时间戳。还可以在应用层记录时间戳，它不需要对系统做任何修改，具有良好的移植性，但协议栈的抖动和任务负荷的影响使其只能达到 ms 级的精度。

另外，时钟稳定度、传输介质特性以及处理器中断响应延时等也会对同步精度有一定影响。例如随着环境的改变(比如温度)会引起晶振的频率特性改变，从而引起两个时钟工作频率的不同，造成时钟漂移[33]；传输介质的双向传输延迟并非严格相等；处理器

的中断延时会造成时间戳的测量存在偏差。必须对这些因素造成的测量值偏差进行补偿。

3.3 IEEE1588 精确时钟同步协议实现

针对影响同步精度的因素，采用了一系列的方法克服这些因素的影响。通过修改 Linux 内核的时钟管理程序提高时钟的精度；通过修改网卡驱动程序提高时间戳的准确度；通过对时间偏差进行滤波的方法进行补偿。

同步实现由两部分构成：平台相关部分和通用部分。平台相关部分主要是对 Linux 内核的修改。通用部分主要是应用层的协议实现。

3.3.1 平台相关部分实现

平台相关部分主要包括对系统时钟程序的修改，在网卡中断处理程序中记录时间戳以及时间戳的传递。

(1) 时钟精度的提高

采用的硬件平台处理器为 AT91RM9200，其上运行的操作系统为 Linux。在 Linux 中可以通过调用系统调用 gettimeofday()来得到精度为 μs 级的系统时间，但这并不意味着系统时间是以一微妙为单位在递增，这跟 gettimeofday()的实现有关。

Linux 内核在 AT91RM9200 上关于 gettimeofday()的默认实现采用的是系统定时器 RTT，它通过调用平台相关函数 at91rm9200_gettimeoffset()来获得时间值，它的默认实现为：

```
static unsigned long at91rm9200_gettimeoffset(void)
{
    unsigned long elapsed;
    elapsed=(AT91_SYS->ST_CRTR-AT91_SYS->ST_RTAR)&AT91C_ST_ALMV;
    return (unsigned long)(elapsed * tick) / LATCH;
}
```

它的实现原理是通过读取 RTT 的实时计数值来获取时间。RTT 的计数频率是 32768HZ，这样 RTT 计数器值每加一所需时间为(1/32768*1000000)μs，即 30.5μs，真正的时间精度也就是 30.5μs。为了使同步精度达到 1μs，就必须改变它的默认实现。

采用 AT91RM9200 内部的普通定时计数器 TC 来代替 RTT。

TC 包括 3 个相同的 16 位定时器/计数器通道。每个通道可独立编程，以完成不同的功能，包括：频率测量、事件计数、间隔测量、脉冲产生、延迟时间及脉宽调制。

每通道有一个 16 位寄存器，寄存器值在所选时钟的每个正沿处自减，当计数器达到 0xFFFF 并翻转为 0x0000 时，表明发生溢出，TC_SR 中的 COVFS 位置位。计数器当前值可通过计数器值寄存器 TC_CV 来实时读取，计数器由触发器复位，此时，计数器值在下一个选定时钟的有效边沿时转为 0x0000。

选择 TC0，并且使其计数频率为(MCK/32)1872000HZ，这样，计数器每递增 1 所需时间为 0.534μs，达到了精度要求。

利用 TC0 来代替 RTT，修改部分主要包括散步分：

① 初始化 TC0，在系统定时器初始化函数 setup_timer()里增加对 TC0 的初始化。初始化工作主要包括定时器的计数频率和工作模式的设置，以及中断处理的设置和开启计数器。

```
static inline void init_tc0(void) {
    /* enable Peripheral clock */
    AT91_SYS->PMC_PCER = 1 << AT91C_ID_TC0;
    /*disable tc0 clock*/
    AT91_TC0->TC_CCR = AT91C_TC_CCR_CLKDIS;
    /*disable all interrupt in tc0 */
    AT91_TC0->TC_IDR = AT91C_TC_IDR_COVFS |
                       AT91C_TC_IDR_LOVRS |
                       AT91C_TC_IDR_CPAS  |
                       AT91C_TC_IDR_CPBS  |
                       AT91C_TC_IDR_CPCS  |
                       AT91C_TC_IDR_LDRAS |
                       AT91C_TC_IDR_LDRBS |
                       AT91C_TC_IDR_ETRGS;
    /* clear any pending interrupts */
    (void) AT91_TC0->TC_SR;
    /* set mode of tc0 */
    AT91_TC0->TC_CMR = AT91C_TC_CMR_TCCLKS_TIMER_CLOCK3 |
                       AT91C_TC_CMR_BURST_NONE          |
                       AT91C_TC_CMR_WAVE;
    /* enable tc0 clock */
    AT91_TC0->TC_CCR |= AT91C_TC_CCR_CLKEN;
    /* triger tc0 */
    AT91_TC0->TC_CCR |= AT91C_TC_CCR_SWTRG;
}
```

② 修改 at91rm9200_gettimeoffset()。因为 at91rm9200_gettimeoffset()的默认实现采用的是 RTT，精度不够，因此要用 TC0 来替换 RTT。首先从 TC_CV(计数器的当前计数值寄存器)中读取 TC0 计数器的当前计数值，然后转换为 μs 值返回。修改后实现如下：

```
static unsigned long at91rm9200_gettimeoffset(void){
    unsigned long elapsed;
    unsigned long tsc;
    tsc = AT91_TC0->TC_CV;
    elapsed = (unsigned long) (tsc * 1000 / 1872);
    return elapsed;
}
```

③ 在时钟滴答中断中复位计数器 TC0。操作系统时钟滴答的周期为 10ms，而 TC0 的溢出时间为 34.996ms，因此 TC0 不会因溢出而引发中断。这样在每次时钟滴答中断处理中复位计数器，读取的 TC_CV 值就是自上次时钟滴答以来所经过的时间。

时钟滴答中断处理例程为 at91rm9200_timer_interrupt()，在其中增加复位计数器的代码使其复位，以便在新的时钟滴答周期重新开始计数。

经过改进后的时钟精度为 0.534μs，达到了精度要求。

(2) 时间戳准确度的提高

如果采用在网卡中断处理程序中记录时间戳的方法来提高时间戳的准确度，必须解决 PTP 报文的识别和时间戳传递问题。

① PTP 报文识别和时间戳的获取

PTP 报文采用 UDP 封装，以组播的方式发送。IEEE 1588 定义了两种端口号，分别被称为事件端口 319 和普通端口 320。其中事件端口用于发送或接收同步报文和延时请求报文，普通端口用于发送或接收附加信息报文和延时请求响应报文。

每种 PTP 报文都由头部和数据两步分组成，PTP 头部有一个 control 字段，用来区分 PTP 报文类型，如图 3.4 所示。

<table>
<tr><th colspan="3">以太网头部</th><th colspan="2">IP 头部</th><th colspan="2">UDP 头部</th><th colspan="2">PTP 头部</th><th>PTP 数据</th></tr>
<tr><td rowspan="2">目的
MAC</td><td rowspan="2">源
MAC</td><td rowspan="2">类型
(type)</td><td rowspan="2">IP 头部其
他部分</td><td rowspan="2">传输层
协议</td><td>源端口</td><td>目的端口</td><td rowspan="2">PTP 头部
其他部分</td><td rowspan="2">control</td><td rowspan="2">协议数据</td></tr>
<tr><td>长度</td><td>校验和</td></tr>
</table>

图 3.4 PTP 数据报文结构

Fig. 3.4 PTP package structure

首先解析数据报文，判断以太网头部中的 type 字段是否为 IP 协议号 0x0800，IP 头部中的传输层协议号是否为 UDP 协议号 17，UDP 头部中端口号是否为 PTP 注册端口

319 或 320，然后再解析 PTP 报文头部的 control 字段，识别具体的报文类型，记录相应的时间戳。

```
static int at91ether_tx(struct sk_buff *skb, struct net_device *dev)
{
    ......
    if (controlID == 0){
        do_gettimeofday(&ptp_sync_send_stamp.tm);
        ptp_sync_send_stamp.flag = 1;
    }
    else if (controlID == 1){
        do_gettimeofday(&ptp_delay_req.tm);
        ptp_delay_req.flag = 1;
    }
    ......
}
```

② 时间戳的传递

网卡驱动是属于内核层的，而利用这些时间戳是在应用层的，因此还必须把这些时间戳从内核空间传递给用户空间。

这里又分两种情况，一种是报文发送的时间戳，因为是在内核中记录的，必须想法把它重新传回用户层中；另一种是报文接收时间戳，因为 Linux 内核网络实现支持报文到达时间戳传递，因此可以在用户层得到这个时间戳，不需要特殊处理。

对于接收时间戳，通过打开套接字的 SO_TIMESTAMP 属性，内核就会对网络到达的数据包记录时间戳，并随数据包传给应用层。在应用层可以通过 recvmsg 得到这个时间戳。

对于发送时间戳，通过网卡驱动的 ioctl 来实现。

在套接字中执行系统调用 ioctl()会引起网络协议 ioctl()命令的调用。相应的符号都定义在文件/include/linux/sockios.h 中，并且通常都与特定协议实例有关。但是，当更高层协议实例的 ioctl()命令不能被处理时，那么内核就将其发送给网络设备，在这里它们可以在驱动程序方法 do_ioctl()中定义自己的命令。

为套接字实现的 ioctl()具有 16 个额外的 ioctl()命令，可以被驱动程序所使用[34]。更详细来说，这些命令是从 SIOCDEVPRIVATE 到 SIOCDEVPRIVATE+15，如果使用这些命令之一，那么就调用相关网络适配器的 dev->do_ioctl()方法。据此，可以在网卡的中断驱动程序中实现相应的 do_ioctl()方法，向应用层传递数据。

3.3.2 通用部分实现

通用部分与操作系统平台无关，主要实现了 IEEE1588 的报文交换过程以及时间戳的处理等。

(1) 主时钟端的同步过程

时钟服务器部分主要功能是周期性的向网络中组播同步报文，并通过附加信息报文把同步报文的发送时间戳传给各个设备；同时它还接收延迟请求报文，并通过延时请求响应报文把延时请求报文接收时间戳传回各设备。

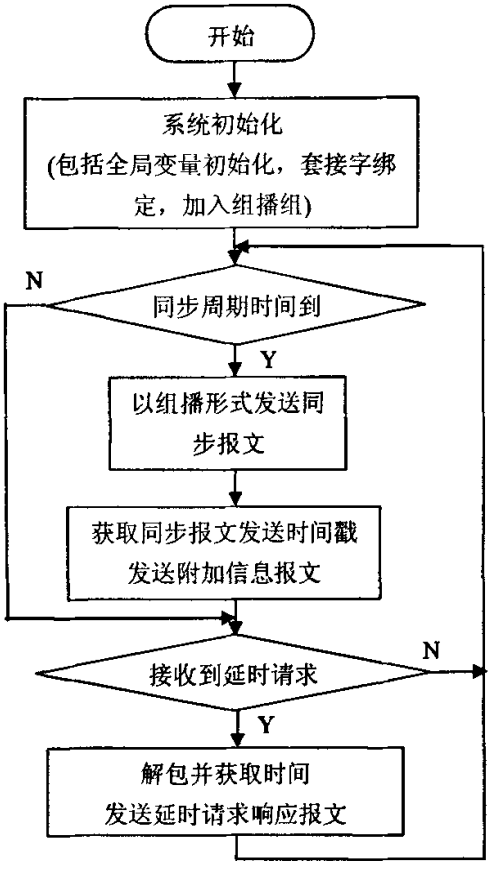

图 3.5 主时钟同步过程

Fig. 3.5 Master clock's synchronization routine

(2) 从时钟端的同步过程

从时钟部分主要功能是接收主时钟的同步请求和附加信息报文传过来的时间戳，随机向服务器发送延时请求报文并接收延时请求响应报文传过来的时间戳。最后根据这四个时间戳来调整本地时间，达到与主时钟的同步。每一次同步报文都对应一个 ID，只有附加信息报文的 ID 和同步报文 ID 相同时才有效。延时请求报文和延时请求响应报文也是如此。这样就保证了时间戳的可靠传递。

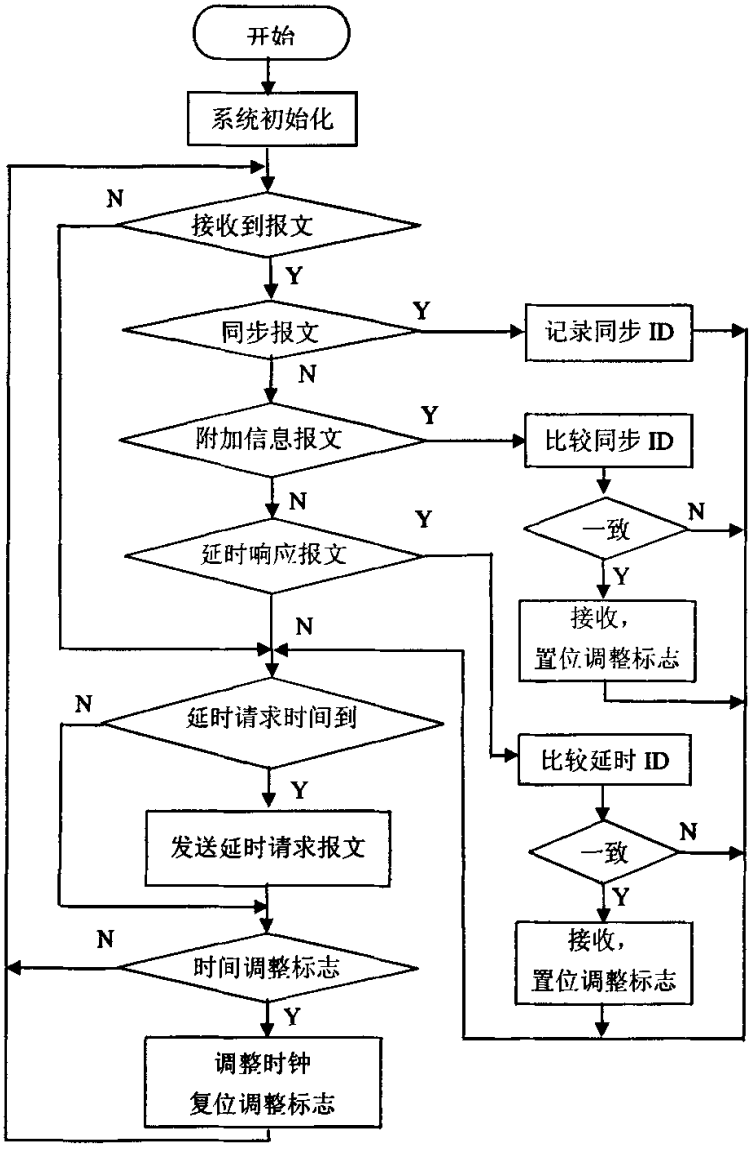

图 3.6　从时钟端的同步过程

Fig. 3.6　Slave clock's synchronization routine

(3) 时间偏差补偿

晶振频率的精度不同导致不同设备的时间度量不一样，从而导致经过测量得到的时间偏差不准。同时，CPU 的繁忙程度，任务负荷都将影响中断响应时间，从而影响网络延迟测量，带来时间抖动。采用对测量值进行滤波补偿的方法来提高精度。

测量值总共有两种，一种是时钟偏差测量，另一种是网络延迟测量，针对这两种情况，这里采用了统一的迭代滤波算法[35]。

$$y(n)=\begin{cases}\dfrac{n-1}{n}y(n-1)+\dfrac{1}{n}x(n) & n<N\\ \dfrac{N-1}{N}y(n-1)+\dfrac{1}{N}x(n) & n>=N\end{cases} \tag{3.7}$$

滤波算法不仅考虑了当前输入对当前输出的影响，同时还考虑了上次输出的影响。N 是一个常量，通常取 15～25，这里取 15。当 n 小于 N 的时候，考虑了采集次数 n 的影响，当 n>N 时，则不考虑 n 的影响，直接令其为常量值 N。

但实际情况并非这么简单，这里只是一种简化。实际上，对时钟偏差的测量值可以采用这种方法，因为它是周期测量的，但对网络延迟的测量，因为它是随机测量的，两次测量值之间的关系很复杂，采用这种方法并非是最理想的，这是以后工作的重点方向。

3.4 小结

在本章中，首先介绍了实现所用的硬件和软件平台，并对平台的工作流程和步骤作了简单说明。然后，介绍了 IEEE1588 的工作原理，深入分析了影响时钟同步精度的因素，并采取了一系列措施来提高精度。同时介绍了 IEEE1588 的应用层实现，针对时钟服务器和普通设备的同步流程分别作了描述。最后介绍了时间偏差补偿滤波算法。

4 EPA 链路层实时调度实现

在第三章中，对 EPA 的链路层调度作了详细的描述。EPA 通过在链路层增加一个通信调度管理子层，对发送到网络上的数据包进行调度，从而避免碰撞，增强实时性。

EPA 采用 TDMA(Time division multiple access)机制，将整个网络带宽划分为等间隔的通信周期，称为通信宏周期。在一个通信宏周期内，带宽又被划分为两个阶段：周期性数据发送阶段和非周期性数据发送阶段。在周期性数据发送阶段，整个周期数据发送带宽被划分给各个设备，每个设备只能在自己的带宽范围内发送数据，这样，就避免了碰撞。周期数据主要是指 EPA 系统中现场设备间过程变量的传输(如传感器检测数据发送给控制器，控制器数据到执行器等)，它的特点是数据是周期产生的。在非周期数据发送阶段，采用的是虚拟令牌的方式，每个设备内部维护一个全网络数据发送列表，这个列表的每一项代表网络上每个设备将要发送的数据的优先级，每一个设备在发送非周期数据的时候，首先遍历该优先级列表，如果自己将要发送的数据比其他设备的优先级都高，那么它就发送，否则就缓存起来。优先级列表的更新发生在两个时候，其中一个是在周期数据发送阶段，每个设备在发送完周期数据时发送一个非周期数据发送声明，通知其他设备自己将要发送的非周期数据的优先级以更新其他设备的优先级列表，另一个时候是在非周期发送阶段，每个设备在发送完一个非周期数据后，发送一个非周期数据发送结束声明告知其他设备以更新优先级列表。非周期数据主要指用于过程报警、趋势、组态、监控等的信息，他们根据重要性划分为不同的优先级别。

根据 EPA 通信的特点，研究了 Linux 内核中网络协议栈的框架，并根据协议栈的 QoS 接口实现了 EPA 的实时性调度。

4.1 Linux 内核中的网络协议栈

4.1.1 协议栈框架

Linux 是开放源代码的操作系统，它具有强大的网络功能。Linux 的网络实现是以 BSD4.4 为模型。它支持 BSD Sockets(及一些扩展)和所有的 TCP/IP 网络。

Linux 下的网络协议栈是通过一系列互相连接的层次结构来实现的，具体结构层次如图 4.1 所示。从整体上看，Linux 的 TCP/IP 网络系统基本上可以分为网络接口层、网络层(IP)、传输层(TCP/UDP)、INET 套接字层、BSD 套接字层和应用层 6 部分。其中，在 Linux 内核中实现的网络协议栈包括了前五个层次，在应用层和 BSD 套接字层之间的应用程序接口以 BSD4.4 为模板。

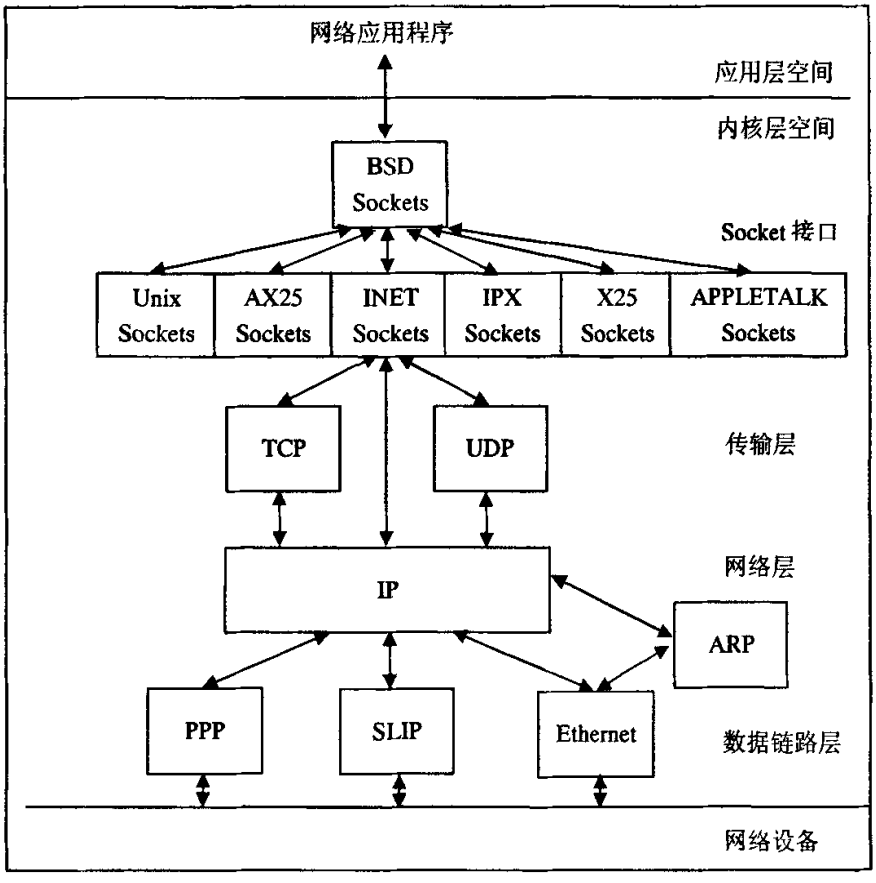

图 4.1 Linux 内核网络协议栈分层结构图

Fig. 4.1 Linux kernel protocol stack architecture

(1) BSD 套接字层

是应用程序和内核之间的接口，实现操作系统提供的套接字系统调用，为用户进程提供统一的套接字编程接口，每个套接字在内核中以 socket 结构体现。

(2) INET 套接字层

与 INET 套接字层同层的是以 BSD 套接字为接口的各协议栈的入口，不同的协议栈如 IPX、UNIX 等有自己的地址结构。INET 套接字层给 TCP 和 UDP 协议提供了一个统一地址族的接口，对应的结构是 sock 结构。

(3) TCP/UDP 传输层

主要为两台主机上的应用程序提供端到端的通信。实现传输层协议(TCP/UDP 协议)的操作，UDP 是无连接的，而 TCP 是面向连接的，所以 TCP 协议比 UDP 复杂的多。

在传输层给将要发送的数据添加 TCP 或 UDP 头，或将传给本机的数据送到相应的套接字队列中去。

(4) IP 网络层

IP 层是 TCP/IP 协议栈的关键部分。它的主要功能是将数据从一台主机经过网络传送的另一台主机，它的主要工作有 IP 数据包的分片和重组，路由查找，IP 转发等。网络层协议包括 IP、ICMP 以及 IGMP 等。

(5) 网络接口层

包括 OSI 参考模型的硬件层和数据链路层。对应的是各种网络设备及其驱动程序。它将从 IP 层送来的数据封装成数据帧并通过物理介质发送出去，并从物理链路取得数据然后交给 IP 层处理。

4.1.2 主要的数据结构

Linux 网络设计采用了面向对象的设计思路，使用一组数据结构来代表协议对象，这里主要介绍描述网络设备的 net_device 结构和描述协议栈中表示数据包的 sk_buff 结构[36-38]。

net_device 结构定义在/include/linux/netdevice.h 中，系统中的每一个网络设备都用一个 net_device 结构表示。它采用了面向对象的设计理念，里面既包含了描述网络设备属性的数据成员，也包括了网络设备的操作，用函数指针来表示。

在这个数据结构里面，主要定义的成员变量是 init 函数指针，这个函数指针初始化为设备驱动程序中提供的用来初始化 net_device 结构的过程，这个过程其实就是用来检测和驱动网络设备的。在进行检测之前，没一个网络设备的 init 函数都指向对应的初始化函数；检测时对每个网络设备调用其 init 函数，如果成功则返回并将改结构加入到内核维护的设备链表中。

在这个结构里还定义了对硬件设备的打开和关闭函数指针(open 和 close)、硬件头的建立函数指针(hard_header)、硬件上数据的传输函数指针(hard_start_xmit)。当网络设备打开的时候，就可以通过这个网络设备开始传输数据了，传输出来的数据存放在 sk_buff 结构里面。hard_start_xmit 函数指针是和某一种具体的硬件相关的，通过 dev_queue_xmit 这个外部函数调用相应设备的 hard_start_xmit 函数指针发送数据。

net_device 用来抽象一个数据链路层的具体网络设备接口卡，为 IP 层访问该设备提供接口。对 IP 层来说，它能看到的就是该设备的 net_device 结构，它就像是 IP 层与链路层交流的一座桥梁。

另外一个重要的数据结构就是描述数据包的 sk_buff(套接字缓冲区)结构。Linux 网络各层之间的数据传送都是通过 sk_buff 结构完成的。它的定义在/include/linux/skbuff.h 中。sk_buff 提供了一套管理缓冲区的方法，是 Linux 系统网络高效运行的关键。每个 sk_buff 结构包括一些控制方法和一块数据缓冲区，这个区域存放了网络传输的实际数据包。

sk_buff 是用来定位和管理一个报文在内核中被处理的整个周期的。当应用程序向一个 socket 传输数据后，该 socket 就创建相应的 sk_buff 并将有效数据的地址存入该结构的变量中。在报文穿越协议栈的过程中，每一层的报文的头信息被插入到有效的数据之前，由于为报文头申请了足够的空间，所以避免了在报文头之后对有效数据的多次拷贝。有效数据只被拷贝了两次：一次是当它从用户地址空间被传输到内核地址空间时，第二次是当报文被传送到网络适配器的时候。当从网络适配器接收到一个报文的时候，在中断处理中动态申请一个 sk_buff 结构，用来保存数据，并向上层传输。关于 sk_buff 的详细描述见文献。

4.1.3 链路层数据的接收和发送

Linux 网络协议栈链路层体系结构如图 4.2 所示[39-41]。

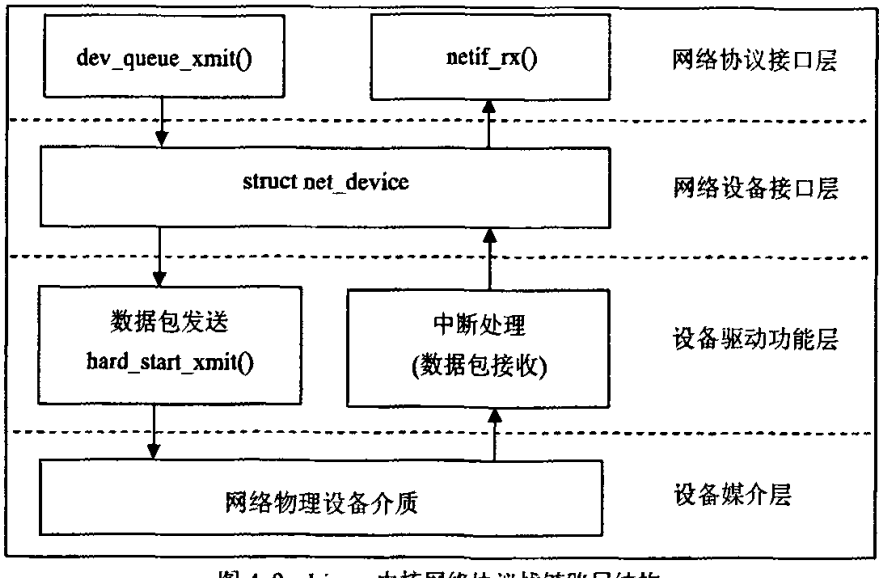

图 4.2 Linux 内核网络协议栈链路层结构

Fig. 4.2 Linux kernel protocol stack link layer architecture

链路层是网络设备和高层网络协议之间的连接层，它的主要功能是将上层协议穿过来的数据通过网络设备发送到网络上，以及通过网络设备从网络上读取数据并向上层协议传递。

(1) 数据包的发送

dev_queue_xmit()是链路层向高层协议提供的数据发送接口，所有上层协议传下来的数据都要通过调用 dev_queue_xmit()来发送数据。要发送的数据包用 sk_buff 来描述，它内部的一个成员指定了了该数据包应该通过那个网络接口被发送，该网络接口在内核中是用 net_device 结构来表示的，也就是说 sk_buff 的一个成员指向了该 net_device 结构。然后 dev_queue_xmit()通过调用该网络接口的发送函数 hard_start_xmit()将数据包发送给网络硬件并发送到网络上。如图 4.2 所示。

(2) 数据包的接收

当网络设备正确接收到一个数据的时候，该网络设备就会触发接收中断，在其中断处理程序中处理数据包。首先为该数据包分配一个 sk_buff 结构，从硬件中读出数据并放置到申请好的缓冲区里，接下来填充 sk_buff 中的一些控制信息，调用 netif_rx()接口函数将接收到的数据存放在系统的接收队列里，最后触发 NET_RX_SOFTIRQ 软件中断，进行下一步的处理。这时硬件中断处理就完成了，因为在硬件中断处理中不能处理太繁重的操作，它的主要任务就是接收数据包并放到接收队列中，剩余的处理交给内核的软中断处理。NET_RX_SOFTIRQ 软中断的处理函数为 net_rx_action()，它从接收队列中摘取数据包，并向上层协议传输。

4.1.4 QoS 接口

QoS(Quality of Service)即服务质量的意思。在传统 TCP/IP 网络中，所有的 IP 传输都采用 FIFO 的尽最大努力的传输机制，这在当今网络数据量剧增和关键业务数据增多的今天，已经不能满足要求。为此出现了服务质量的概念，也就是针对各种不同需求，提供不同服务质量的网络服务功能。Linux 利用 TC 模块实现了对 QoS 的支持[42-43]。

没有 TC 时，每个数据包的发送都会调用 dev_queue_xmit()，最终通过调用网卡驱动程序的硬件发送函数将数据包发送出去。发送机制采用的就是 FIFO 机制，一旦出现拥塞，协议栈只是尽最大努力去调用网卡发送函数，这种方法存在着很大的弊端。

为了支持 QoS，Linux 内核中在发送数据包的代码中增加了 TC 模块，从而可以对数据包进行分类、管理、检测拥塞和处理拥塞。QoS 有很多的拥塞处理机制，如先入先出队列，优先队列，定制队列，加权公平队列等等。QoS 还要求能够对每个接口分别采用不同的拥塞处理。为了能够实现上述功能，Linux 采用了基于对象的实现方法。

Linux 设计了两个数据结构来实现 QoS 的功能：队列调度器 Qdisc 和队列掉队规则 Qdisc_ops。其中队列调度器是 Qdisc_ops 是具体的数据包队列调度规则，如先入先出规则、优先队列调度规则等。Qdisc 是网络设备与具体调度规则连接的桥梁，通过它把具体的调度规则关联到网络设备上。Linux 中的 QoS 接口接口如图 4.3 所示。

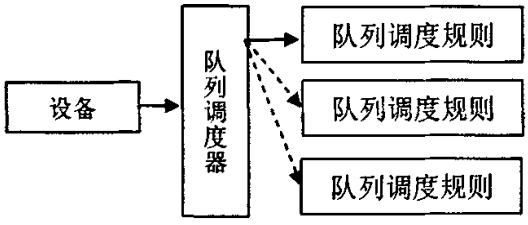

图 4.3 Linux 内核 QoS 接口

Fig. 4.3 Linux kernel QoS interface

在 Linux 内核中，每个网络设备用一个 net_device 结构来表示，它其中一个指针成员指向一个 Qdisc 结构，而 Qdisc 结构指向某种队列调度规则 Qdisc_ops。Qdisc 结构在内核中的定义如表 4.1 所示。

表 4.1 Qdisc 结构体定义

Tab. 4.1 Qdisc structure definition

类型	名称	说明
int	(*enqueue) ()	入对操作
struct sk_buff *	(*dequeue) ()	出队操作
unsigned	flags	标志位
struct Qdisc_ops*	ops	调度规则结构体指针
Qdisc*	next	下一个 Qdisc 指针
u32	handle	句柄
atomic_t	refcnt	引用计数
struct sk_buff_head	q	数据包队列
struct net_device*	dev	设备结构体
struct tc_stats	stats	Tc 状态
int	(*reshape_fail)	用于更复杂的流量整形
struct Qdisc*	__parent	父 Qdisc 指针
char	data[0]	私有数据

Qdisc 是一个接口单元，它为网络数据包的发送提供了一系列的方法。其中，ops 成员指向了一个 Qdisc_ops 结构，代表其具体的调度规则；dev 指向调度器关联的网络设备；enqueue 和 dequeue 提供了数据包入队和出队的操作接口，dev_queue_xmit 调用调度器提供的 enqueue 和 dequeue 进行数据包的调度。在调度器内部，enqueue 和 dequeue 调用 ops 指向的具体的调度规则 ops 的 enqueue 和 dequeue 方法。Qdisc 结构如表 4.2 所示。

表 4.2 Qdisc_ops 结构体定义

Tab. 4.2 Qdisc_ops structure definition

类型	名称	说明
struct Qdisc_ops*	next	下一个调度规则
struct Qdisc_class_ops*	cl_ops	类规则
char	Id[IFNAMSIZ]	掉队规则名字
int	priv_size	私有数据
int	(*enqueue) ()	入队操作
struct sk_buff	(*dequeue) ()	出队操作
int	(*requeue) ()	重新入队操作
int	(*drop) ()	丢弃操作
int	(*init) ()	初始化操作
void	(*reset) ()	复位数据包缓冲队列
void	(*destroy) ()	释放调度规则资源
int	(*change) ()	改变调度规则参数
int	(*dump) ()	输出排队规则配置参数和统计数据

Qdisc_ops 代表具体的调度规则，它也采用面向对象的方法设计，提供了一系列的方法。init 成员用于初始化新的、实例化的排队队列调度规则；requeue 和 dequeue 分别用于数据包的入队和出队操作等。

加上调度器之后的数据包发送流程如图 4.4 所示。

安装了调度器和调度规则的网络发送流程应该是这样的：(1) 上层协议发送数据包；(2) 获得当前设备的调度器和对应的调度规则；(3) 通过调度器接口的 enqueue 接口调用调度规则的 enqueue 方法将数据包压入队列；(4) 通过调度器接口的 dequeue 调用调度规则的 dequeue 方法，按照一定的调度规则从队列中取出一个数据包；(5) 调用网卡驱动的发送函数将数据包发送出去。

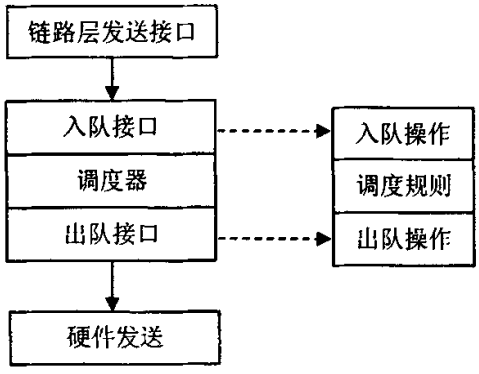

图 4.4 队列调度器和队列调度规则

Fig. 4.4 Queue scheduler and queue scheduling algorithm

4.2 链路层调度实现

EPA 通过在链路层增加实时性调度实体来对要发送的数据包进行调度，这和 Linux 内核的 QoS 接口所提供的功能是一致的，因此，这里采用 QoS 接口来实现 EPA 的链路层调度。

4.2.1 安装 EPA 调度器和 EPA 调度规则

根据 QoS 接口的要求，专门为 EPA 调度规则创建一个调度器。

设备启动之后，此时当前设备缺省的 qdisc->ops 是 pfifo_fast_ops。如果需要采用不同的 ops，那么就需要为设备安装其他的 qdisc。本质上是替换掉 dev->qdisc 指针。在 Linux 内核中，qdisc_create_dflt ()函数用来创建一个 Qdisc 结构，它的原型是：

Qdisc* qdisc_create_dflt(struct net_device* dev, Qdisc_ops* ops);

其中，参数 dev 表示该调度器关联的设备，ops 表示该调度器对应的调度规则。穿了创建调度器，首先还必须创建一个调度规则。EPA 调度规则定义如下：

```
struct Qdisc_ops epa_csme_ops {
    Null;
    Null;
    "epa_csme";
    PRI_NUM * sizeof(struct sk_buff_head);
    epa_csme_enqueue,
    epa_csme_dequeue
    epa_csme_requeue,
    NULL,
```

```
    epa_csme_init,
    epa_csme_reset,
}
```

其中的宏 PRI_NUM 表示优先级的个数，这里定义为 6；epa_csme_enqueue 等函数指针是对 qdisc_ops 结构函数指针成员的赋值，这些函数实现了 EPA 的调度规则。

创建了 Qdisc 和 Qdisc_ops 结构，接下来就是安装他们。在 Linux 内核中，通过函数 qdisc_graft()来安装调度器。它的原型是：

```
int qdisc_graft(struct net_device *dev, struct Qdisc *parent, u32 classid,
        struct Qdisc *new, struct Qdisc **old);
```

其中，dev 表示关联的设备，new 表示要安装的调度器，old 表示原来的调度器，安装一个新的调度器，必须先把旧的换掉，为了保存原来的默认的调度器，定义了一个变量 dflt_qdisc(Qdisc 类型的指针)用来保存原来的调度器。这样，就可以为设备安装新的 EPA 调度器了：

```
Qdisc_graft(dev, NULL, 0, epa_qdisc, &dlft_qdisc);
```

4.2.2 EPA 调度规则实现

实现 EPA 调度规则，就是根据 EPA 调度的规程来实现 epa_csme_ops 里的一系列成员函数。其中，最重要的是入队函数 enqueue 和出队函数 dequeue。

(1) 数据结构设计

为了实现 EPA 调度规则，主要设计了数据包缓冲队列、非周期数据调度管理队列、核心控制结构等数据结构。

① 数据包缓冲队列

数据包进入调度器后，首先要用队列缓冲起来，然后根据调度规则，选择数据包进行发送。

这里，为每种优先级的数据分别设立了缓冲队列，采用的数据结构是 Linux 内核的双链表结构 sk_buff_head。数据包进入调度器后，根据其优先级将其分别缓冲到各自对应的队列中，等待调度算法的选择发送。队列的创建和初始化工作在 epa_csme_init()中完成。

② 非周期数据调度管理列表

在 EPA 中，周期数据的调度采用的是 TDMA 机制，设备在自己所属的带宽内发送周期数据。而对于非周期数据，EPA 采用的是虚拟令牌机制，即网络上的所有设备根据其要发送的非周期数据的优先级竞争发送。为了有效的进行非周期数据的调度，每个设备内部都必须维护一个当前网络上各个设备要发送的非周期数据的优先级，并且根据数

据的发送实时更新维护。非周期数据调度管理列表中的每一项描述了网络中一个设备将要发送的非周期数据的优先级和该设备的 IP 地址，定义如下：

```
struct sched_queue {
    int flag;
    unsigned long remote_ip;
    unsigned long remote_pri;
}
```

flag 成员用来表示该表项是否有效。

定义的非周期数据调度管理列表如下：

```
static struct sched_queue non_period_sched_q[MAX_DEV_NUM];
```

MAX_DEV_NUM 宏表示设备数据最大值。

有了这个数据结构，就可以对网络上的非周期数据发送进行管理和调度了。

③ 核心控制结构

为了使调度器各个部分协调一致的工作，设计了核心控制数据结构，如表 4.3 所示。

表 4.3 核心控制结构定义

Tab. 4.3 Core control structure definition

类型	名称	说明
int	config_state	配置状态：调度/不调度
unsigned	macro_cycle	宏周期
unsigned	np_offset	非周期偏移
unsigned short	p_begin	设备周期开始偏移
unsigned short	p_end	设备周期结束偏移
int	annun_requeue	声明数据包重新入队
struct sk_buff*	annun_skb	声明数据包
int	annun_flag	声明标志
int	end_annun_flag	结束声明标志
int	send_finish_flag	发送结束标志
spinlock_t	queue_lock	队列锁：防止并发访问
int	queue_pos	非周期数据管理队列当前位置
int	last_annun_pri	上一个声明的非周期数据优先级
unsigned long	local_ip	本地 IP 地址
struct timer_list	wd_timer	定时器

核心控制结构主要包括三部分内容：关于组态信息的，例如宏周期、非周期发送偏移、周期带宽范围等；关于非周期数据发送声明和发送结束声明的部分；关于非周期数据管理队列部分，如队列锁、要发送的非周期数据的优先级等；

(2) EPA 协议状态机规定功能

EPA 协议状态机规定的功能主要包括：数据包优先级的判断；发送非周期数据发送声明和发送结束声明；周期数据和非周期数据发送操作；获取宏周期内时间偏移量等操作。

数据包根据优先级缓冲到不同的队列。EpaDataPriority() 函数用来得到数据包的优先级。首先根据以太网头中的协议号字段判断数据优先级，如判断是非周期数据发送声明或结束声明报文、arp 包还是 IP 报文；其次，如果是 IP 报文，再根据 EPA 应用层协议定义的服务号来判断优先级。最后，数据包的优先级存储在 sk_buff 的 priority 字段。

非周期数据发送声明和发送结束声明用来更新非周期数据调度管理列表。在发送完本周期数据后发送非周期发送数据声明，通知其他设备自己要发送的非周期数据优先级，各个设备收到通知后更新管理列表。在发送完一个非周期数据后发送非周期数据发送结束声明，其他设备收到后同样更新管理列表。EpaAnnunSending() 函数实现了发送非周期声明的功能，它根据报文类型(非周期数据发送声明或发送结束声明)和当前设备要发送的最高级别的非周期数据优先级来构造报文，然后直接发送。

周期数据和非周期数据发送操作完成了数据发送功能。EpaPeriodicDataSending() 实现了周期数据发送功能，它从周期数据缓冲队列中取得一个数据包，然后直接发送出去。EpaNonPeriodDataSending() 完成非周期数据发送功能，它首先遍历非周期数据缓冲队列，优先从高优先级队列中取得数据，然后发送。

宏周期内时间偏移量是判断发送何种类型数据的基础。EpaGetOffsetInCycle() 用来获得当前宏周期内时间偏移量。它首先取得本地时间，然后对宏周期取余，所得的余数就是偏移量的值，据此可以判断发送周期数据还是非周期数据，以及发送时间剩余。

(3) 调度器入队 enqueue

调度器的入对操作功能是根据数据包的优先级分别缓冲到不同的队列中。首先调用 EpaDataPriority() 计算得到数据包的优先级，然后存放到相应的缓冲队列。enqueue 是调度器提供给上层数据发送的接口，它具体调用的是 EPA 调度规则的入对函数 epa_csme_enqueue()。

(4) 调度器出队 dequeue

调度器的出队操作是数据包调度的核心，完成整个的调度算法。它根据调度算法从数据队列中选择数据包发送。

调度规则首先判断设备是否组态(即是否打开了 EPA 调度)，如果没有，则进行正常的数据包发送，否则进入 EPA 调度操作。然后，取得当前宏周期时间偏移量，判断是处于周期数据发送阶段还是非周期数据发送阶段。如果是周期数据发送阶段，则进行周期数据发送，当周期数据发送完毕，则发送非周期数据发送声明，通知其他设备更新非周期数据管理列表。如果是非周期数据发送阶段，如果有非周期数据要发送并且取得了发送权，则执行发送操作，发送完毕发送非周期数据发送结束声明，否则不执行发送操作。dequeue 流程如图 4.5 所示。

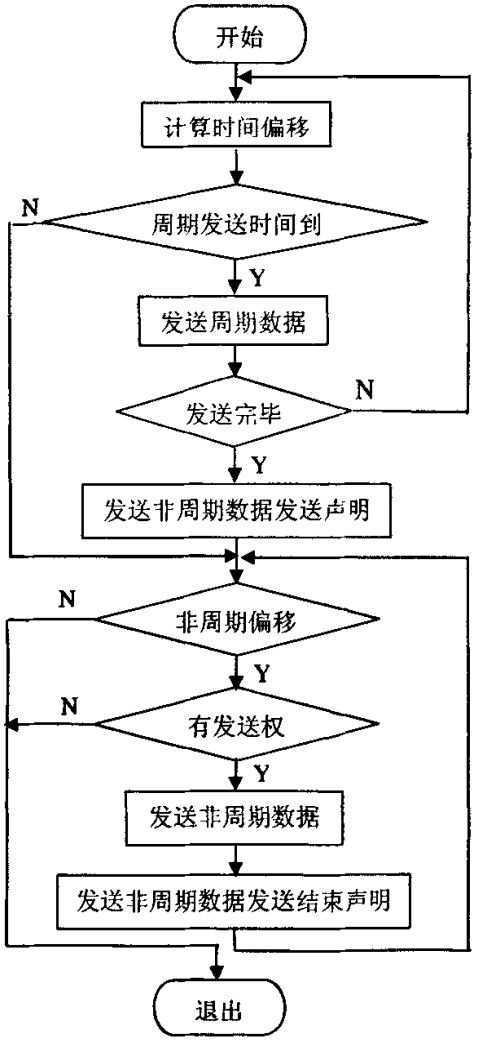

图 4.5 EPA 调度算法出队操作

Fig. 4.5 EPA scheduling algorithm dequeue operation

数据包的发送处理前面介绍的正常的发送流程外，还有另一选择。它由方法netif_schedule()标记为执行(CPU 软中断)。只要有一个套接字缓冲区在正常传输进程中不能得到发送的话，netif_schedule()就会得到调用。这样，如果 NET_TX_SOFTIRQ(网络发送软中断)被 netif_schedule()标记为执行，那么它会在对 CPU 调度程序的下一次调用中启动。net_tx_action()是 NET_TX_SOFTIRQ 软中断处理例程，它的主要任务就是调用 qdisc_restart()以启动网络设备上的数据包传输。qdisc_restart()会开启调度器的工作。

这样，图 4.5 中的调度程序会被不停的调度执行，从而完成数据包的调度。

4.2.3 组态参数传递和调度切换

EPA 调度所需的参数需要通过应用层程序来提供，这些参数包括本地 IP 地址、宏周期大小、非周期发送偏移，周期发送范围等。

这又涉及到用户空间和内核空间数据传递的问题，这里同样采用网卡的 ioctl()方法来实现。传递给 ioctl()的参数结构如表 4.4 所示。

表 4.4 组态参数结构

Tab. 4.4 Configuration data format

类型	名称	说明
unsigned long	msk	标志信息
unsigned short	macro_cycle	宏周期
unsigned short	np_offset	非周期偏移
unsigned short	P_begin	设备周期开始偏移
unsigned short	p_end	设备周期结束偏移
unsigned long	ip	本地 IP 地址

epa_csme_set_config()用来对调度的数据结构进行初始化。在 ioctl()里调用它，并用上层传过来的数据对调度数据结构进行相应的初始化。

调度的切换采用内核函数 qdisc_graft()函数来实现，它的功能主要是将网络设备上的原来的调度器去掉，并且添加一个新的。这里，采用定义的全局变量 dlft_qdisc 和 epa_qdisc 分别指向默认的调度器和 EPA 调度器，调用 qdisc_graft()实现切换。

4.3 小结

本章首先描述了 Linux 内核中的网络协议栈，介绍了协议栈的框架，主要用的数据结构（net_device 和 sk_buff），链路层向上层协议层和下层硬件层提供的数据发送和接收接口，和为了数据包调度而实现了 QoS 接口。接着，根据 QoS 接口，设计了 EPA 调度器和调度规则，并根据 EPA 调度规则实现了 EPA 链路层数据调度。

5 EPA 网络实时性测试

在时钟同步的基础上，实现了 EPA 的链路层调度。为了对实时性性能进行测试，组建了测试平台，开发了相应的测试软件，包括上位机和下位机两部分。

5.1 实时性测试平台

测试平台硬件部分由 5 台 DUT5000 工业以太网控制模块、一台 100M 以太网交换机、一台 PC 机组成。系统结构如图 5.1 所示。

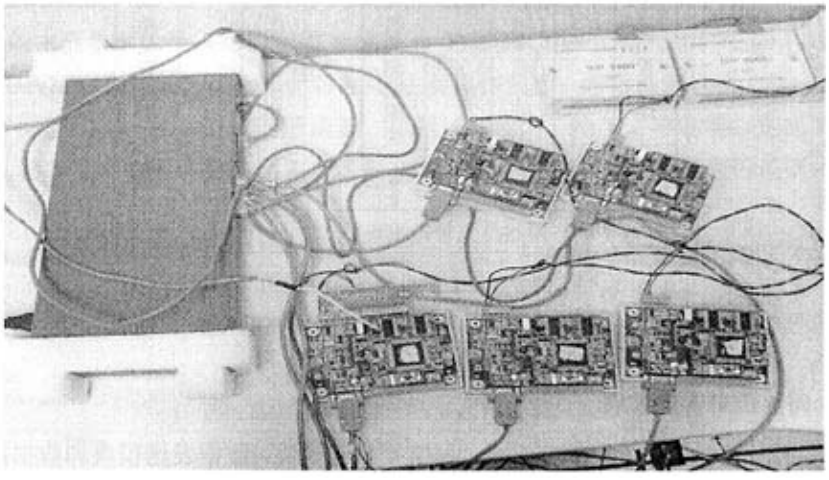

图 5.1 测试平台结构图

Fig. 5.1 Test platform structure

DUT5000 中运行嵌入式 Linux 操作系统，设备 IP 地址分别为 192.168.0.1－192.168.0.5。PC 机安装 Windows XP 系统，上面运行上位机测试软件。

测试项目包括：时钟同步精度测试、递交时间测试和调度测试等。

5.2 上位机实时性测试软件开发

上位机测试软件采用 VC++6.0 开发。主要功能是组态信息的配置下载、发送测试请求命令、接收测试响应并将测试结果以图形方式显示出来。软件结构如图 5.2 所示，主要分为网络信息管理模块、带宽配置模块、时钟同步测试模块、递交时间测试模块、调度测试模块、界面管理模块和测试结果管理模块。

网络信息管理模块是负责测试软件与测试网络的交互，各个测试模块发送的测试命令由它发送到网络中，设备的回应数据通过它派发到不同的测试模块进行处理和显示，测试结果管理模块管理测试结果数据。每个测试模块各自负责一项测试任务，它们之间的切换由界面管理模块来管理。

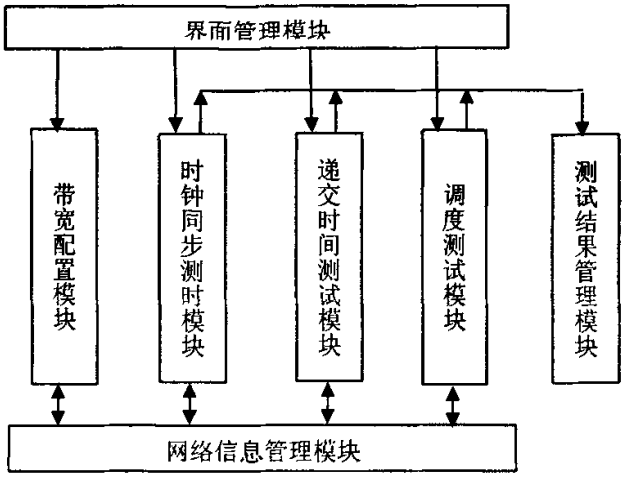

图 5.2　上位机测试软件结构

Fig. 5.2　PC test software architecture

5.2.1　网络信息管理模块

网络信息模块主要由三部分组成：网络初始化模块，数据发送模块和数据接收和派发模块。

网络初始化模块 InitNetwork() 主要完成 Winsock 套接字库加载，发送和接收套接字的创建和初始化，接收套接字的绑定和模式设置。接收套接字被设定为异步非阻塞模式，调用 WSAAsyncSelect() 函数注册网络消息 RTE_MSG_FROM_NETWORK，并注册网络事件 FD_READ。RTE_MSG_FROM_NETWORK 是用户自定义消息，它的消息处理函数为网络数据接收函数 OnMessageFromNetwork()，一旦网络上有关心的数据发送，接收函数就会被自动调用[44-46]。

数据发送模块 SendMessageToNetwork() 被各个测试模块所调用完成测试请求发送。它接收测试模块的数据报文，调用 sendto() 发送数据。

数据接收和派发模块 MessageDispatch() 负责接收并派发数据给各个测试模块。它被关联为 RTE_MSG_FROM_NETWORK 的消息处理函数。接收网络数据后，解析并判断是去往那个测试模块的，然后分别调用这些模块的接收函数来进一步处理数据。

5.2.2 带宽划分模块

带宽划分模块是一个可视化的带宽配置模块，它包括两部分：图形化带宽配置界面和配置信息的下载。

(1) 图形化带宽配置界面

图形化带宽配置界面实现带宽的可视化配置操作。它的主要功能为采用形象化的视图来表示带宽，利用鼠标拖动来配置非周期偏移和设备的周期带宽。采用面向对象的分析与设计方法，抽象出四个类 CBandDivideView、CBandDivFrame、CBandDivDev 和 CBandDivDevList 来描述和实现可视化操作功能。

CBandDivideView 派生自 MFC 类库中的 CScrollView，它是整个可视化配置界面的载体，它的主要数据成员包括图形框架成员 m_Frame(CBandDivFrame 对象)和设备列表成员 m_DevList(CBandDivDevList 对象)。主要操作包括：图形绘制操作 OnDraw()，负责整个界面的绘制；鼠标消息处理函数，包括鼠标按下、抬起，移动等，完成用户的操作功能；设备上线处理，接收设备声明报文添加设备信息；检查配置是否正确，如是否有重叠区域等。

CBandDivFrame 是可视化配置界面框架的抽象，它主要包括图形的矩形区域框架和带宽标尺，在它的内部绘制各个设备带宽条。它的主要成员如表 5.1 所示。

表 5.1 CBandDivFrame 类主要成员

Tab. 5.1 CBandDivFrame's main members

类型	名称	说明
CRect	m_rectFramArea	框架区域
CRect	m_rectNpSplitSenseArea	周期非周期分割条敏感区域
int	m_nComCycle	宏周期
int	m_nNonPeriodOffset	非周期区域偏移量
Bool	m_bDraggingNpSpliter	非否正在拖动分割条
void	DrawFrame()	绘制框架函数

其中，数据成员是用来描述框架的位置、大小信息和周期相关信息，DrawFrame()用来完成框架的绘制工作。

CBandDivDev 是设备带宽的抽象，它代表一个图形带宽的所有信息，包括位置、大小、名称、颜色等。每个设备的带宽都用一个 CBandDivDev 来表示。它的成员如表 5.2 所示。

表 5.2 CBandDivDev 类主要成员

Tab. 5.2 CBandDivDev's main members

类型	名称	说明
CRect	m_rectDevWholeArea	整个矩形块区域，包括左右边块
CRect	m_rectDevArea	中间块区域
CRect	m_rectDevLeftArea	左边块区域
CRect	m_rectDevRightArea	右边块区域
CRect	m_rectBeforeArea	前边未用区域
CRect	m_rectAfterArea	后边未用区域
CRect	m_rectNonPeriodArea	非周期带宽区域
BOOL	m_bIsInMid	鼠标点击在中间块
BOOL	m_bIsInLeftArea	鼠标点击在左边块
BOOL	m_bIsInRightArea	鼠标点击在右边块
int	m_nOrigL	整体移动时记录左值
int	m_nOrigR	整体移动时记录右值

上述主要变量的表示意义可用图 5.3 来表示。

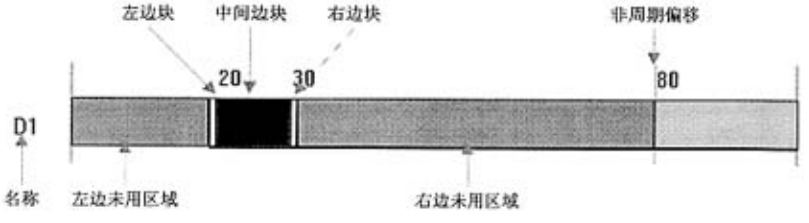

图 5.3 CBandDivDev 示意图

Fig. 5.3 CBandDivDev illustration

绘图操作 DrawDev()绘制设备带宽条图形的各个组成部分。

CBandDivDevList 类用来管理在线设备信息，它接收设备发送的设备声明报文并将其信息添加到设备维护链表中。CBandDivDevList 的成员为 CBandDivDev 对象。

绘制图形时，CBandDivideView 的 OnDraw()被调用。它首先调用 m_Frame 的绘图成员 DrawFrame()绘制整个框架和标尺，然后遍历 m_DevList，调用链表中每个设备的绘图成员 DrawDev()绘制每个设备带宽条。

当用户操作时，首先在鼠标左键按下消息处理函数中判断鼠标点击点是否位于该设备条，并且判断位于哪个区域块。根据第一次鼠标点击时落入的区间块，进行不同的操作。例如，如果第一次落入的是周期非周期间隔条敏感区域，那么当拖动时，进行的就是非周期偏移量调整的操作；如果第一次落入的区域是左边块或右边块，那么拖动的时候，进行的就是设备周期带宽的调整。

进行拖动操作时，首先在鼠标按下时记录其坐标位置，判断其操作类型，在鼠标移动处理函数中根据鼠标移动不断改变坐标信息，在鼠标按键抬起处理函数中记录最后的位置，据此改变各个区域的相对位置信息，并且调用 Invadilate()使整个区域无效进行重绘制。

可视化配置实现的操作包括：拖动周期和非周期间隔条改变非周期带宽；拖动设备带宽条的左边块或右边块来改变设备带宽分配；拖动设备带宽块整体左移或右移；显示移动过程中的标尺信息；宏周期设置等。

最终实现的界面效果如图 5.4 所示。

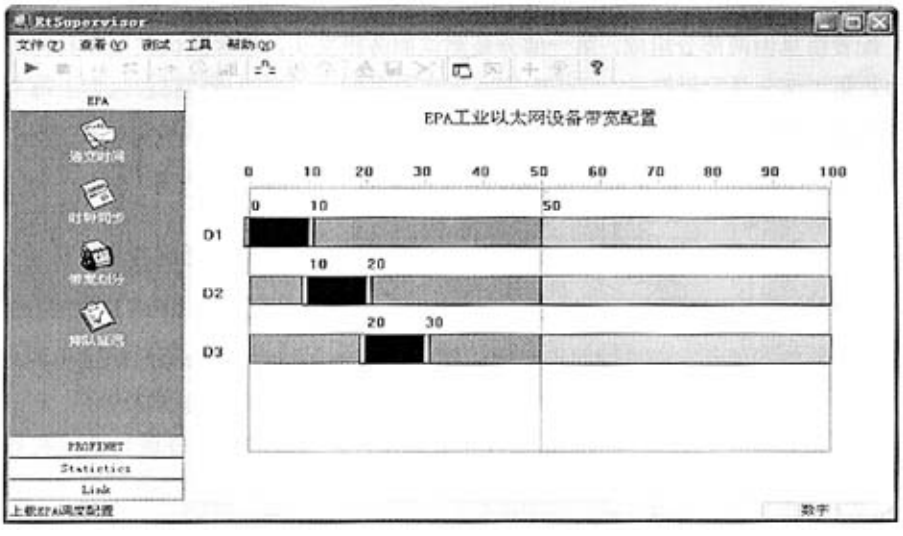

图 5.4 可视化带宽配置界面

Fig. 5.4 Visual bandwidth configuration interface

(2) 配置信息下载

带宽配置完成后，配置信息通过网络下载到各个设备中。配置信息报文格式如表 5.3 所示。

表 5.3 带宽配置信息报文格式

Tab. 5.3 Bandwidth configuration information format

组成	名称	字节数	说明
报文头	service_id	1	测试服务标识
	reserved	1	保留
	type	1	测试项目类型
	cmd	1	测试命令
配置信息	macro_cycle	4	宏周期
	np_offset	2	非周期偏移
	dev_num	2	设备个数
	ip	4	设备 IP(第一个设备)
	to_ip	4	目的发送 IP(第一个设备)
	pb	2	周期起始偏移(第一个设备)
	pe	2	周期结束偏移(第一个设备)

配置信息由两部分组成，第一部分是测试服务报文头，包含测试类型和具体的测试命令；第二部分是配置信息，它由配置头部和设备配置信息组成，前者包含了每个设备共享的宏周期、非周期偏移等共享信息，并且之处了后面的设备配置信息的个数，后者是每个设备的具体配置信息，配置信息的个数取决于配置头中的设备个数。

CheckConfig()函数被调用来检查配置信息是否有误，例如设备带宽区域有无重叠等。然后，调用网络信息管理模块的数据发送函数将配置信息发送到各个设备当中。

5.2.3 时钟同步测试模块

该模块的功能是完成时钟同步测试任务。测试软件发送测试命令给测试设备，各个被测设备接收到测试请求，在链路层记录接收时间戳并将其返回给测试设备，测试设备搜集所有数据后发送给上位机测试软件。测试软件计算同步精度并进行曲线和列表显示。它主要由两部分组成：图形显示部分和数据接收处理部分。

图形显示部分由类 CSyncView 实现，它派生于 CView。它包含一个 CSplitterWnd 成员，用于将视图进行分割为上下两步分：曲线显示部分和列表显示部分。曲线显示部分采用 NtGraph 控件(一个曲线控件)实现，列表显示部分用 ListView 实现。最终界面如图 5.5 所示。

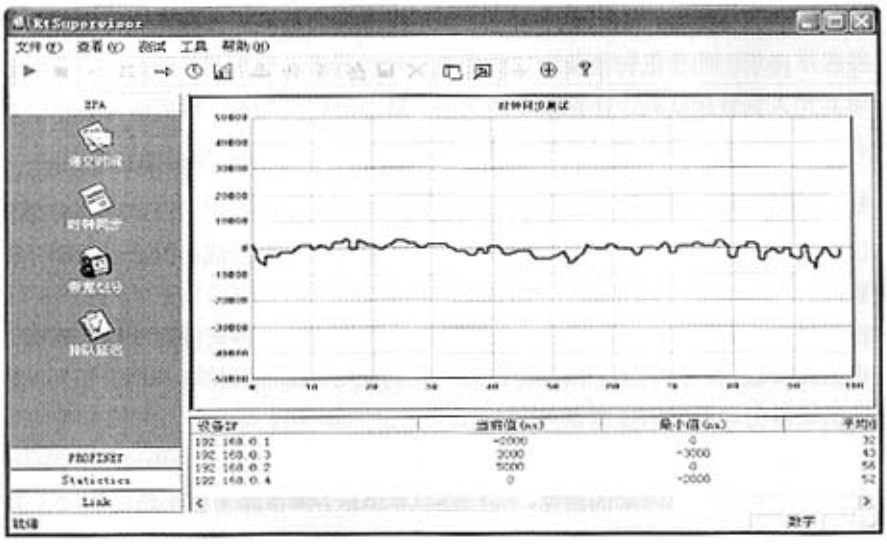

图 5.5 时钟同步测试软件界面

Fig. 5.5 Time synchronization test software interface

当网络信息管理模块接收到同步测试报文时，OnReceiveSyncData()回调函数被调用接收测试设备发送回来的数据，进行处理并显示。测试结果报文结构如表 5.4 所示。

表 5.4 同步测试结果报文格式

Tab. 5.4 Synchronization test result package format

组成	名称	字节数	说明
报文头	reserved	1	保留
	service_type	1	测试报文类型
	service_id	2	测试项目
	service_version	2	测试项目版本
	length	2	长度
测试数据	flag	1	有效性标志
	device_num	1	设备号
	test_seq_id	2	测试序列号
	drift_second	4	偏差
	stamp_send	8	测试发送时间
	stamp_recv	8	测试接收时间

OnReceiveSyncData()首先根据 lenth 字段获得测试数据的长度，利用指针强制转换将 length 字段后面的数据转换成测试数据类型结构指针，解析其中的成员，获取时钟偏差并将其加入到管理队列，计算偏差最大值、最小值和平均值。然后调用显示函数进行曲线和列表显示，并调用测试结果管理模块将结果保存到文件中。

5.2.4 递交时间测试模块

该模块的功能是完成递交时间测试任务。主要由两部分组成：图形显示部分和数据接收处理部分。

图形显示部分由类 CRtDeliveryView 类实现，实现和时钟同步测试部分类似。显示部分也由曲线显示和列表显示两部分组成，分别采用 NtGraph 控件和列表视图实现。递交时间测试界面和始终同步测试界面类似。

网络信息管理模块接收到递交时间测试报文时，OnReceiveDeliveryData ()回掉函数被调用来接收设备发送回来的数据，进行处理并显示。测试结果报文结构如表 5.5 所示。

表 5.5 递交时间测试结果报文结构

Tab. 5.5 Delivery test result package format

组成	名称	字节数	说明
报文头	reserved	1	保留
	service_type	1	测试报文类型
	service_id	2	测试项目
	service_version	2	测试项目版本
	length	2	长度
测试数据	ip	4	设备 IP 地址
	stamp_t1	8	应用层发送时间戳
	stamp_t2	8	链路层调度入时间戳
	stamp_t3	8	链路层调度出时间戳
	stamp_t4	8	应用层接收时间戳

OnReceiveDeliveryData ()解析报文，获得时间戳并将其加入到队列中去，计算最大值、最小值和平均值，进行曲线和链表显示，并保存测试结果。

5.2.5 调度测试模块

该模块用来测试 EPA 调度规则的正确性：周期信息按照预先组态的顺序发送，非周期数据按照优先级竞争机制发送。也由测试显示模块和数据接收处理模块组成。

测试报文以广播形式发送，测试软件接收到后识别其是周期报文还是非周期报文，并将其按照接收顺序显示在列表中，这样就能观察出数据包是否按照 EPA 调度策略来发送。

5.2.6 界面管理模块和测试结果管理模块

界面管理模块用来管理视图界面间的切换，采用单文档多视图结构和切分窗口的方法实现。

窗口被分割为左右两步分，左边采用的视图为一个从 CWnd 派生的实现抽屉窗口功能的开源控件 CGfxOutBarCtrl，实现导航的功能，右边为测试视图共享的区域。

右边共享视图区域任何时候都只能显示一个视图，并且显示视图的 ID 是唯一的。为了实现多视图共享一个显示区域，在框架窗口类中定义了指向每一个视图对象的指针，当需要切换到某个视图时，先调用 SetDlgCtroID()把原来正在显示的那个视图的 ID 设为 0，并调用 ShowWindow(SW_HIDE)隐藏起来，然后把将要显示的那个视图的 ID 设为显示区域 ID 值，并调用 ShowWindow(SW_SHOW)显现出来。视图关系如图 5.6 所示。

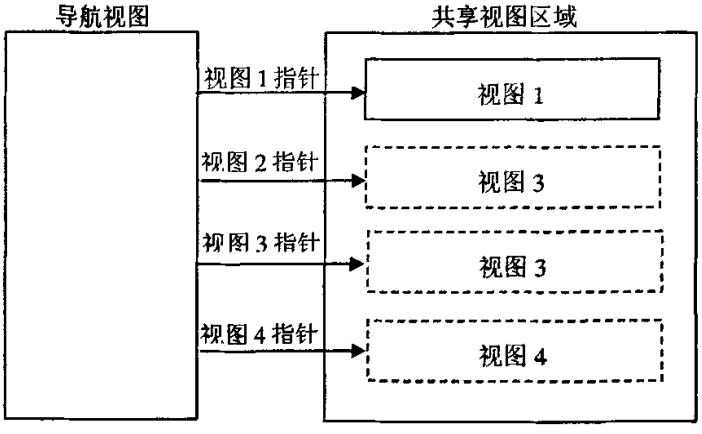

图 5.6 视图关系

Fig. 5.6 Views relationship

测试结果管理模块用来管理所有的测试结果。在工程中目录中建立根文件夹 prj，再在 prj 里为每项测试建立二级文件夹 sync、delivery 和 shedule，分别保存各自的测试结果。

5.3 下位机测试软件开发

下位机测试软件用来接收上位机发送的测试命令，执行测试任务，工作流程如图 5.7 所示。

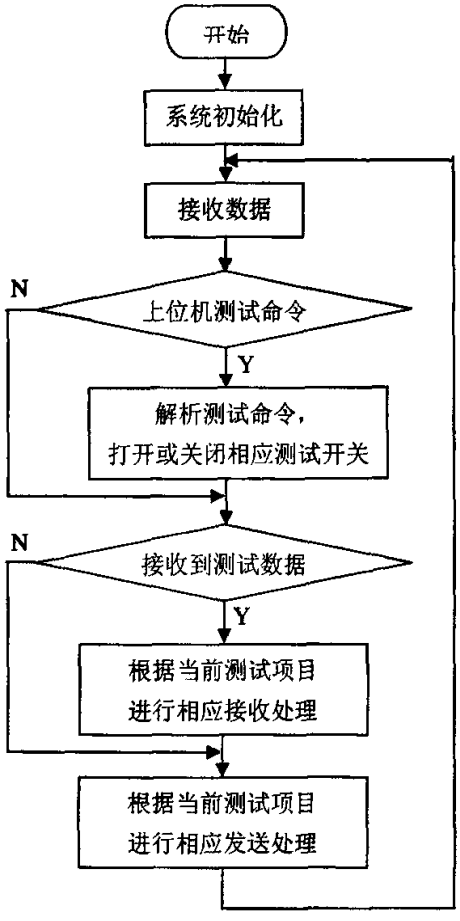

图 5.7 下位机测试软件流程

Fig. 5.7 Embedded test software working flow

接收到数据后，如果是上位机发送的测试命令，则打开或关闭相应的测试项目开关；如果接收到数据，则根据当前测试项目进行处理。如果为时钟同步测试，测试设备端接收从被测设备发送过来的测试请求接收时间戳，被测设备端接收测试请求，记录接收时间戳并将其发送给测试设备；如果为递交时间测试，则记录测试报文接收时间戳，并将测试结果发送给上位机测试软件。

在发送处理中，根据当前测试项目进行不同操作。如果为时钟同步测试，如果测试周期时间到，在测试设备端，首先将上个周期的测试结果发送给上位机测试软件，然后再发送新的测试请求，被测试设备端没有此处理；如果为递交时间测试，如果测试周期时间到，则发送新的测试报文；如果为调度测试，则发送模拟数据报文。

5.4 实时性测试

实时性指标集中反映了 EPA 网络的实时通信能力，这些指标主要有时钟同步精度、递交时间、调度测试、网络吞吐量和非实时通信带宽等，重点进行了前三项测试[47-49]。

5.4.1 时钟同步测试

时钟同步精度是指 EPA 网络上各个 EPA 设备的本地时钟与网络上主时钟的最大偏差值，是 EPA 确定性通信调度实现的前提，时钟同步精度指标反应了网络中不同设备时钟的准确性。

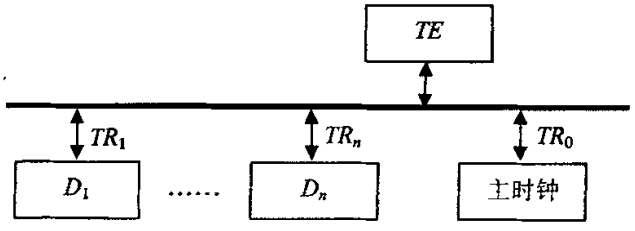

图 5.8 时钟同步精度测试示意图

Fig. 5.8 Test principle of clock synchronization accuracy

时钟同步精度测试如图 5.8 所示，测试设备 *TE* 以组播形式发送测试请求报文，同一网段的主时钟和被测设备 $D_1 \sim D_n$ 接收到测试报文后，分别在数据链路层记录接收到测试请求报文的本地时间 TR_0 和 $TR_1 \sim TR_n$，由于主时钟和被测设备 $D_1 \sim D_n$ 处于同一微网段，因此测试请求报文从 *TE* 传送到主时钟和被测设备 $D_1 \sim D_n$ 的网络延时可以被认为是相等，将网络传输延时记为 T_d，而被测设备 $D_1 \sim D_n$ 与主时钟的时间偏差记为 $TO_1 \sim TO_n$，测试设备 *TE* 与主时钟之间的时间偏差记为 TO_0，则：

$$TO_1 = TR_1 - TR_0$$

.....

$$TO_n = TR_n - TR_0$$

主时钟和被测设备 $D_1 \sim D_n$ 接收到测试请求报文后,分别将 TR_0 和 $TR_1 \sim TR_n$ 加入到测试响应报文中发送回 *TE*，由测试设备完成时钟同步精度的计算。

图 5.9 为测试 100 次的测试结果，结果表明同步精度达到了 10μs。

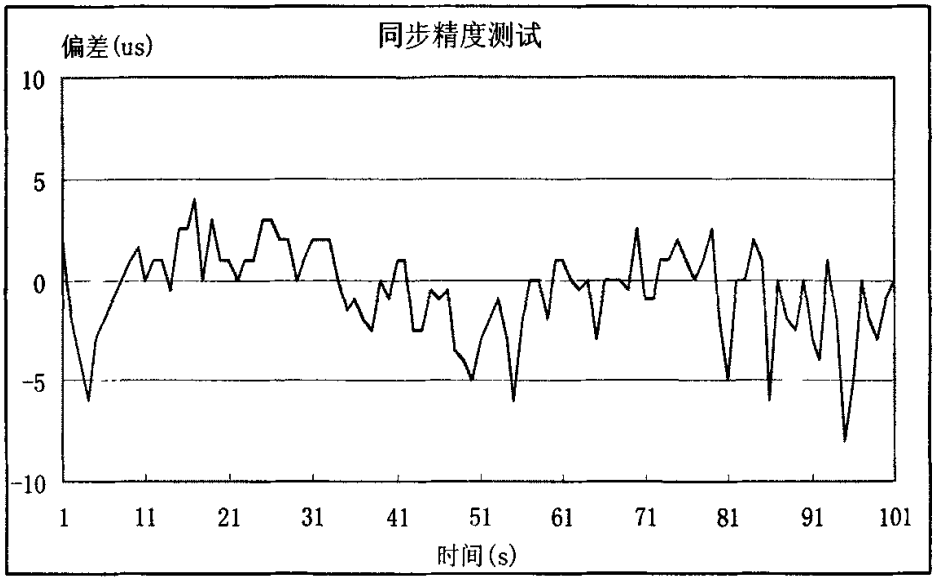

图 5.9 时钟同步精度测试结果

Fig. 5.9 Clock synchronization test result

5.4.2 递交时间测试

递交时间指标反应了不同设备应用进程之间通信所需要的时间，通过记录数据传输所经过不同网络分层的时刻来完成对通信栈、物理链路传输延时的测试。

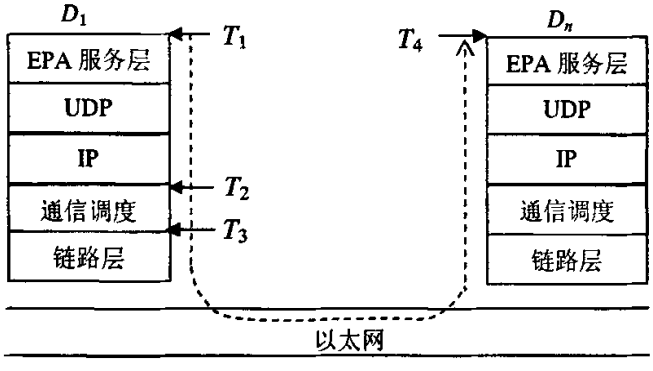

图 5.10 递交时间测试示意图

Fig. 5.10 Principle of delivery time test

递交时间测试示意图如图 5.10 所示。被测设备 D_1 发送数据，并记录发送时刻 T_1，报文自上而下经过 UDP 层、IP 层、数据链路层并经过物理链路传到 D_n。根据 EPA 规范，当报文经过数据链路层时，需根据时间组态和报文优先级对报文发送进行调度，而通信调度的延迟随组态的不同而不同，因此通信调度的延迟在测试时需要被排除。当数据进入和离开调度实体时，分别记录时间 T_2 和 T_3，则通信调度延迟为 T_3- T_2。D_n 在应用层接收到 D_1 发送的数据，记录接收时刻 T_4。D_1 和 D_n 需要实现时钟同步。则递交时间可以通过计算得到：

$$TD_1=T_2-T_1$$
$$TD_2=T_4-T_3$$
$$TD=TD_1+TD_2$$

式中，TD 为递交时间，TD_1 为通信协议栈发送延迟，TD_2 为物理层传输和通信栈接收延迟。

表 5.6 为递交时间测试结果。

表 5.6　递交时间测试结果

Tab. 5.6　Delivery time test result

发送设备 IP 地址	目的设备 IP 地址	递交时间最小值(μs)	递交时间最大值(μs)	递交时间平均值(μs)
192.168.0.1	192.168.0.2	562	1203	856
192.168.0.2	192.168.0.1	581	1215	887
192.168.0.2	192.168.0.3	590	1250	894
192.168.0.3	192.168.0.2	584	1312	873
192.168.0.3	192.168.0.4	579	1220	879
192.168.0.4	192.168.0.3	588	1227	886
192.168.0.1	192.168.0.4	576	1140	854
192.168.0.4	192.168.0.1	595	1298	852

5.4.3　调度测试

调度测试是用来验证 EPA 实时性调度的正确性。

调度测试原理：在网络时钟同步的基础上，用上位机测试软件对带宽进行配置。各个设备根据 EPA 调度策略发送数据，上位机测试软件收到后根据其数据分类(周期还是非周期)将其显示在列表中的不同区域，从而观察调度结果是否正确。

表 5.7 为调度测试结果。

表 5.7 调度测试结果

Tab. 5.7 Scheduling test result

测试 ID	周期数据 IP 地址	周期数据 发送偏移(μs)	非周期数据 IP 地址	非周期数据 优先级
0	192.168.0.1	4		
0	192.168.0.1	151		
0	192.168.0.1	262		
0	192.168.0.1(发送结束)	490		
0	192.168.0.2	20004		
0	192.168.0.2	20168		
0	192.168.0.2(发送结束)	22985		
0	192.168.0.3	40005		
0	192.168.0.3	40136		
0	192.168.0.3(发送结束)	40372		
0	192.168.0.4	60004		
0	192.168.0.4	60132		
0	192.168.0.4(发送结束)	63023		
0			192.168.0.1	1
0			192.168.0.1	1
0			192.168.0.3	1
0			192.168.0.3	1
0			192.168.0.4	2
0			192.168.0.2	3
0			192.168.0.2	3
0			192.168.0.2	3
.....	.....	.....	.....	.....
121	192.168.0.1	4		
121	192.168.0.1	135		
121	192.168.0.1	245		
121	192.168.0.1(发送结束)	476		
121	192.168.0.2	20004		
121	192.168.0.2	20130		
121	192.168.0.2(发送结束)	20356		
121	192.168.0.3	40004		

表 5.7 调度测试结果(续)

Tab. 5.7 Scheduling test result(Con.)

121	192.168.0.3	40126		
121	192.168.0.3(发送结束)	40360		
121	192.168.0.4	60004		
121	192.168.0.4	60137		
121	192.168.0.4(发送结束)	60377		
121			192.168.0.3	2
121			192.168.0.3	2
121			192.168.0.1	3
121			192.168.0.1	3
121			192.168.0.4	3
121			192.168.0.2	4
121			192.168.0.2	4
121			192.168.0.2	4
.....	.....	.....	.....	.....

表中周期数据组态发送先后顺序为 192.168.0.1、192.168.0.2、192.168.0.3、192.168.0.4。周期数据发送偏移为每个周期数据被调度发送时刻的宏周期偏移量，非周期数据为随机产生数据。

测试结果表明，数据发送符合 EPA 调度规则。

5.5 小结

本章首先介绍了实时性测试平台的硬件组成。然后详细叙述了上位机测试软件的软件结构和各个模块的设计，还介绍了下位机测试软件的工作流程。最后介绍了实时性性能测试方法，并给出了时钟同步精度、递交时间和调度的测试结果。

结　论

随着计算机技术、通信技术、网络技术和控制技术的发展，以太网在工业控制领域得到了越来越广泛的应用，并不断向现场设备层延伸。以太网通信的延迟不确定性是其进入工业控制领域的主要技术障碍。本文围绕工业以太网的实时调度问题，研究了实时性问题的理论研究现状和工程领域的实时以太网解决方案，并以 EPA 为对象，研究并实现了其时钟同步和链路层调度，并对其性能进行了测试，具体工作主要如下：

(1) 深入研究了 EPA 的通信模型和链路层调度规则。EPA 通过对 ISO/IEC 8802-3 协议规定的数据链路层进行了扩展，增加了一个通信调度管理实体来对数据包发送进行调度：对于周期数据的发送，采用 TDMA 机制给每个设备分配发送带宽；对于非周期数据，采用基于优先级的集中式调度，从而避免碰撞，增强实时性。

(2) 研究了 IEEE1588 精确时钟同步协议的工作原理，深入分析了影响时钟同步精度的因素，主要包括时钟精度、时间戳记录位置等，在 Linux 系统中，通过修改内核的时钟管理程序提高了时钟精度，采用在网卡中断处理程序中记录时间戳的方法来提高时间戳的准确度，并对时间偏差使用平均值滤波的方法进行补偿，从而实现了 EPA 系统中设备间的精确时钟同步。

(3) 研究了 Linux 内核的网路协议栈和链路层对 QoS 的支持，在精确时钟同步的基础上，通过 Linux 内核的 QoS 接口实现了 EPA 链路层实时性调度规则。

(4) 开发了上下位机测试软件。上位机测试软件采用 VC++6.0 开发，主要包括网络信息管理模块、实时性测试模块和界面管理模块等。下位机测试软件采用 Linux C 开发。

(5) 组建了测试平台，对 EPA 网络的时钟同步精度、递交时间和调度性能进行了测试。

以太网实时性问题是一个很复杂的领域，目前研究还处于探索阶段，虽然本文已经进行了较为深入的研究，但明显还存在着很多不足和待提高的地方，主要包括：

(1) 时钟同步方式还有待改进，目前是指定一个设备为主时钟，下一步可以采用 IEEE1588 中的 BMC(最佳主时钟算法)，以竞争方式选择主时钟。

(2) 时钟同步精度还有待提高，目前采用的偏差滤波算法还较为简单，下一步可以根据测试记录的偏差数据研究滤波算法以得到好的滤波效果。

(3) 调度算法还可以深入研究，现在采用的是 EPA 的调度规则，下一步可以研究更加高校的实时性调度算法。

(4) 上位机测试软件还有待改进，目前的测试项目仅包括时钟同步精度、递交时间和调度，以后可以增加新的测试项目，如非周期带宽、网络吞吐量的。

(5) 目前的测试软件是单独开发了，下一步可以考虑和 DUT_configer 集成，采用统一的软件结构。

(6) 测试方法还有待改进，目前所采用的测试方法并非最优。

相信随着实时性问题的深入研究和逐步解决，以太网在工业控制领域将会发挥越来越重要的作用。

参 考 文 献

[1] 杨朋. 工业以太网的发展及其技术特点. 微计算机信息(测控自动化), 2006, 22(2):32-33.

[2] 贾东耀, 汪仁煌. 工业控制网络结构的发展趋势. 工业仪表与自动化装置, 2006, (5):12-14.

[3] 魏庆福. 现场总线技术发展的新动向. 工业控制计算机, 2003, 13(1):1-4.

[4] 冯冬芹, 彭圣鼎, 李杰等. 工业以太网最新进展. 自动化博览二十周年纪念, 2003, (S1):102-105.

[5] 冯冬芹, 金建祥, 褚健. "工业以太网及其应用技术"讲座 第 2 讲 以太网与 TCP/IP. 自动化仪表, 2003, (5):65-70.

[6] 冯冬芹, 金建祥, 褚健. Ethernet 与工业控制网络. 仪器仪表学报, 2003, (2):23-26.

[7] 李炳宇, 萧蕴诗. 以太网在网络控制系统中的应用与发展趋势. 微型机与应用, 2002, (11):35-37.

[8] 李继容, 鲍芳. 以太网在工业自动化领域的应用及研究. 计算机应用研究, 2002, (9):126-128.

[9] 熊育悦, 赵哲分. 工业以太网在控制系统中的应用前景. 自动化仪表, 2002, 23(9):1-5.

[10] 刘明哲, 徐皑冬. 确定性实时以太网通信协议研究. 仪器仪表学报, 2005, 26(8):505-507.

[11] 王智, 王大然. 工业实时通信网络(现场总线)的基础理论研究与现状(上). 信息与控制, 2002, 31(2):146-152.

[12] 王智, 于大然. 工业实时通信网络(现场总线)的基础理论研究与现状(上). 信息与控制, 2002, 31(3):241-249.

[13] Pritty D W. A real-time upgrade for ethernet based factory networking. Proceedings of the IEEE IECON 21st international conference on industrial electronics, control and instrumention, Orlando, FL, USA, 1995:1631-1637.

[14] Lann, Gerard, Rivierre. Real-time communications over broadcast networks:the CSMA-DCR and the DOD-CSMA-CD protocols. Proceedings of the IEEE real-time systems symposium, San Juan, Puerto Rico, 1994:67-84.

[15] Ouni S. Hard and soft real time message scheduling on ethernet networks. The IEEE international conference on systems, Man and Cybernetics, Taipei, Taiwan, 2006:6-10.

[16] Kapsalis V D. Implementation of a MAC-layer protocol(GIT-CSMA/CD) for industrial LANs and its experimental performance. Tranctions on the industrial electronics, 1997, 44(6):825-839.

[17] Junghoon L, Heonshik S. A bandwidth reallocation scheme for ethernet-based real-time communication. Proceedings of the second intertional workshop on real-time computing and applications, Tokyo, 1995:28-33.

[18] Junghoon L, Seungjun P. An error control scheme for ethernet-based real-time communication. Proceedings of the third internation workshop on real-time computing and applications, Seoul, South Korea, 1996:214-219.

[19] Junghoon L. Design of a communication system capable of supporting real-time RPC.

Proceedings of the second IEEE international conference on engineering of complex computer systems, Montreal, Que., Canada, 1996:99-102.

[20] Yavatkar R. A reservation-based CSMA protocol for integrated manufacturing networks. The IEEE transactions on Systems, Man and Cybernetics, 1994, 24(8):1247-1258.

[21] Tobagi F. Carrier sense multiple access with message-based priority functions. The IEEE transcations on communications, 1982, 30(1):185-200.

[22] Molle M, Kleinrock L. Virtual time CSMA: why two clocks are better than one. The IEEE transactions on communications, 1985, 33(9):919-933.

[23] Zhao W, Ramamritham K. A real time CSMA protocols for hard real time communications. IEEE transactions on software engineering, 1987, 13(8):938-952.

[24] Zhao W. A window protocol for transmission of time-constrained messages. IEEE transactions on computers, 1990, 39(9):1186-1203.

[25] 陈垒, 冯冬芹, 金建祥等. 以太网在工业应用中的实时特性研究, 浙江大学学报(工学版), 2004, 38(6):670-675.

[26] 缪学勤. 论六种实时以太网的通信协议. 自动化仪表, 2005, 26(4):1-6.

[27] 用于工业测量与控制系统的 EPA(Ethernetforplantautomation)系统结构和通信标准, 2005.

[28] 桂康, 杨佃福, 谷海波. 工业测控 EPA 网络系统的时间同步分析与实现. 武汉理工大学学报, 2006, 28(2):65-69.

[29] Weibel H. High precision clock synchronization according to IEEE 1588 implementation and performance issues. Proceedings of embedded world, Nurenberg, 2005:22-24.

[30] ANSI/IEEE Standard 1588-2002, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, 2005.

[31] Correll K, Barendt N. Design considerations for software only implementations of the IEEE 1588 precision time protocol. Conference on IEEE-1588 standard for a precision clock synchronization protocol for networked measurement and control systems, Winterthur, 2005:353-359.

[32] Weibel H. IEEE 1588 implementation and performance of time stamping techniques. Proceedings of conference on IEEE 1588, Gaithersburg, 2004:27-29.

[33] 桂本烜, 冯冬芹等. IEEE158 的高精度时间同步算法的分析与实现. 工业仪表与自动化装置, 2006, (4):20-23.

[34] Klaus W. Linux 网络体系结构. 汪青青等译. 北京:清华大学出版社, 2006.

[35] 董西路. 变点检测研究及其在沸腾状态检测中的应用:(硕士学位论文). 大连:大连理工大学, 2005.

[36] 孙天泽, 袁文菊, 张海峰. 嵌入式设计及 Linux 驱动开发指南. 北京:电子工业出版社, 2005.

[37] 倪继利. Linux 内核分析及编程. 北京:电子工业出版社, 2005.

[38] Daniel P 著. 深入理解 Linux 内核. 陈莉君译. 北京:中国电力出版社, 2004.

[39] 李炳龙, 陈性元. Linux 网络设备驱动程序分析与设计. 计算机应用研究, 2001, (10):89-91.

[40] 王星,胡爱群.Linux下网络设备驱动程序的研究与开发.微计算机应用,2005,26(5):624-627.
[41] 郭学礼,潘松,韦智.Linux下网络驱动程序分析.计算机应用,2001,21(11):23-24.
[42] 祝琳,王光彩,顾君忠.Linux网络系统对QoS的支持.计算机工程,2002,28(2):177-179.
[43] 夏涛,周敬利,余胜生.Linux中的服务质量机制.计算机工程与应用,2001,(5):19-21.
[44] 王艳平,张越.Windows网络与通信程序设计.北京:人民邮电出版社,2006.
[45] 张越.Visuall C++网络程序设计实例详解.北京:人民邮电出版社,2006.
[46] 孙晓刚.面向软件工程的Visual C++网络程序开发.北京:清华大学出版社,2004.
[47] 李卓函,仲崇权.工业以太网EPA实时性测试方法研究.自动化仪表,2006,27(10):4-7.
[48] 张艳芳,王平.EPA时间同步测试方法与实现技术.计算机工程与应用,2007,43(16):146-148.
[49] 桂本烜.EPA现场控制器通信模块的设计与开发:(硕士学位论文).杭州:浙江大学,2006.

攻读硕士学位期间发表学术论文情况

[1] 冀朝阳, 仲崇权. IEEE1588 精确时钟同步协议在 Linux 中的研究与实现. 大连理工大学研究生网络学刊, 2007.

在论文中的相关章节:第三章.

致　谢

完成这篇论文的同时，我要感谢所有关心、帮助、指导过我的老师和同学，在此向他们表示由衷的谢意！

首先感谢我尊敬的导师仲崇权教授。本论文是在仲老师的亲切关怀和悉心指导下完成的。在研究生期间的学习中，仲老师渊博的学识，严谨的治学态度，敏捷深刻的洞察力以及平易近人的长者风范深深地影响和激励着我。从课程选排到论文选读；从毕设开题到项目规划；从论文撰写到无数次精心修改，这期间我在工作学习中取得的每一个进步都凝聚着仲老师的心血和汗水，我所获得的将成为一生中最宝贵的财富和资本。在此，谨向仲老师表示最诚挚的谢意！

感谢教研室的杨素英副教授、张利副教授、冯毅副教授、张立勇老师、李卓函老师和李丹老师，他们在思想、生活、工作和科研上给了我很多的帮助和指导，使我受益菲浅。

感谢教研室的刘宁博士，在我的论文完成过程当中，给了我很多建设性的意见和帮助。

感谢我的师兄师姐：郭福帅、鲁辛凯、俞瑞富、王超、吕凌欧、付易朋、包艳妮、苏再焘，感谢他们在生活和科研上的帮助，是他们对我耐心的指导，使我在科研能力上有了快速进步。

感谢与我并肩作战的同学：周倩、吕晓、班伟、张钰、张晓亮、曹苏雷、李亚男、郭文楠、邱盈、高磊、王俊，感谢他们在生活中对我的帮助与关怀，感谢他们在工作上对我的配合和支持。

感谢我的师弟师妹：王恒、江俊、王慧莉、宋本杰、黄成刚、陈晨、薛旭、刘洁、钟威、刘阳等，谢谢他们对我的关心与帮助。

最后，感谢我的父母和家人，感谢他们对我始终如一的关怀和鼓励，他们的关心和爱护永远是我最坚强的后盾，他们的幸福是我永远为之努力奋斗的不竭动力！