南京邮电大学



硕士学位论文摘要

学科、专业：工科、计算机软件与理论

研究方向：软件技术及其在通信中的应用

作者：二零零七级 硕士研究生：喻勇 指导教师：孙知信 教授

题目：基于网络处理器的 BRAS 驱动关键技术研究与实现

英文题目：Research and Realization of the Key Technology about the Driver Module of BRAS based on Network Processor

主题词：BRAS 体系结构，网络处理器，业务支持数据结构，路由表，访问控制列表，通讯模块

Keywords：BRAS Architecture, Network Processors, Business Support Data 吗 Structure, Routing Tables, ACL, Communication Module

课题来源：1、国家自然基金项目，60973140
2、江苏省自然基金项目，BK2009425
3、江苏省高校自然基金项目，08KJB520005
4、江苏省青蓝工程学术带头人资助项目
5、中兴通讯基金项目

# 中 文 摘 要

随着 Internet 及相关网络技术的发展，网络的规模迅速增长，新的应用不断涌现，宽带接入服务已经成为运营商急剧增长的重要业务来源，宽带接入服务器（BRAS）是目前实现宽带接入的主流设备。为满足不断增长的用户数量和更高的服务质量，BRAS 相关技术的研究成为各主流通信设备制造商研发机构的重要课题。

为了实现线速转发和提供硬件层面的可编程环境，BRAS 体系结构已经发展为基于网络处理器的分布式系统。为对用户提供多种业务服务，BRAS 不仅要实现非协议报文的线速转发还要实现对协议报文的特殊业务处理。底层网络处理器在实现对报文线速处理的同时要实现越来越多的业务，如 QoS,AAA 等，因此 BRAS 体系结构设计的过程中设计了处理各种业务的上层业务模块。为实现 BRAS 体系中网络处理器与上层业务处理模块的交互，需要设计一个网络处理器服务模块，又称驱动模块，以处理网络处理器与上层业务模块的报文交互和微引擎转发报文时需要的业务支持数据结构的维护。驱动模块在 BRAS 中起着承上启下的作用，因此驱动模块的合理设计关系到整个系统的效率、稳定性以及可靠性。

文章首先介绍了基于网络处理器的 BRAS 基本原理与技术，并对基于 IXP2800 网络处理器的 BRAS 系统结构进行了总体分析，随后分析了 BRAS 驱动模块在系统中的功能，设计的原理，并探索了实现驱动模块功能的关键技术。第四章对 BRAS 驱动业务支持数据结构的相关技术进行了研究，主要采用了 HASH、TRIE 和 RFC 算法实现不同的业务支持数据结构，其中大部分业务表项通过采用 HASH 数据结构进行存储和查询，采用基于 TRIE 数据结构的算法实现路由表的存储和管理，访问控制列表(ACL)的实现则是使用 RFC 算法。文章详细给出了使用上述算法数据结构以实现相关业务数据结构的原理，操作流程以及实验验证过程。在驱动通讯模块设计部分，文章分别从 BRAS 驱动主 IXP2800 上送协议报文、上层协议下发给主 IXP2800 和从 IXP2800 的协议报文以及从 IXP2800 上送报文（NO ARP）三个报文流向分别给出了实现原理和流程，并在最后通过实验进行了验证和分析。

通过对实验结果进行分析，文章所采用的相关技术较好地实现了驱动模块的设计功能，为 BRAS 系统在业务数据结构支持和模块间报文传送设计了稳定且高效的实现机制。

**关键字**：BRAS 系统结构，网络处理器，业务支持数据结构，路由表，访问控制列表，通讯模块

# ABSTRACT

Broadband Remote Access Server (BRAS) is the main broadband access equipment currently, and in order to meet the growing number of users and higher demand of service quality, BRAS-related technology research became one of the most important R&D subject for the institutions of the telecommunications equipment manufacturers.

In order to achieve wire-speed forwarding and hardware level programmable environment, a new generation of BRAS architecture, as a network processor-based distributed systems, has been developed. BRAS not only need to forward the non-protocol packet with wire-speed but also to achieve the treatment of the special protocol packet to provide a wide range of business services to users. In the BRAS system, the underlying network processor module need to achieve wire-speed packet processing while also provide more and more services, such as QoS, AAA and so on, as a result,BRAS architecture need to design upper business blocks to deal with various business. To help the network processor to interact with the top business process, BRAS system need to design a service module of network processor,also known as the driver modules,to handle packet interaction between network processor and the top business, and maintain those support data structures used by the micro-engine of network processor to forward packets.Driver module in the BRAS plays a role of linking and media between the underlying network processor module and the upper operational controlling module, so the rational design of drive modules is related to the efficiency, stability, and reliability of the overall system.

This paper first describes the basic principles and techniques of network processor-based BRAS, and give an overall analysis of the BRAS system structure based on the IXP2800 Network Processor; then analyse the function of BRAS drive module in the system, design principles, and explore the key technologies to achieve drive module features. In Chapter IV , the technologies ,related to the business support data structures of BRAS driver, have been studied, mainly using HASH, TRIE and the RFC algorithm to realize different business data structures, most of which through the use of HASH table to store and query entry for business data structures, the use of TRIE data structure algorithms for routing table storage and management, and ACL implementation is based on the RFC algorithm. Articles give the principle of relevant business data structures, operating processes in detail, using the above algorithm and data structure, and experimental verification process. In the design part of the communication module, the article were driven from the BRAS to send the packets from the main IXP2800 to the upper

upper protocol module , packets from upper protocol module to the IXP2800 and packets from the assistant IXP2800 (NO ARP) to upper protocol module.The principles and processes of the 3 kinds of packets flows are given respectively, and finally verified through experiments and analysis.

As the analysis of experimental results showing, the technologies of the article achieve a more stable and efficient implementation mechanism of data structures supporting business and packet transmission for the drive module of the BRAS system.

**Key Words:** BRAS Architecture, Network Processors, Business Support Data Structure, Routing Tables, ACL, Communication Module

# 目 录

# 第一章 引 言

## 1.1 课题研究背景

目前，Internet 处于高速发展时期。这一高速的发展势头对于 Internet 的业务提供商(ISP) 来说，则意味着面临更新的要求和挑战。随着各种高速接入方式的不断推出，作为接入 Internet 的关键接口设备，目前的宽带接入服务器越来越成为发展高速、宽带和综合一体化 Internet 接入的一个“瓶颈”[1]。

宽带接入服务器（BRAS）是面向宽带网络应用的新型接入网关。它可以完成宽带用户的 IP/ATM 网的数据接入，实现多种业务的汇聚和转发，解决不同用户对传输容量、带宽利用率等要求，为用户提供 VPN 服务、构建企业内部 Internet、支持管理域向用户批发业务等应用，负责用户的管理和控制。图 1-1 说明了 BRAS 在网络中的位置和作用。

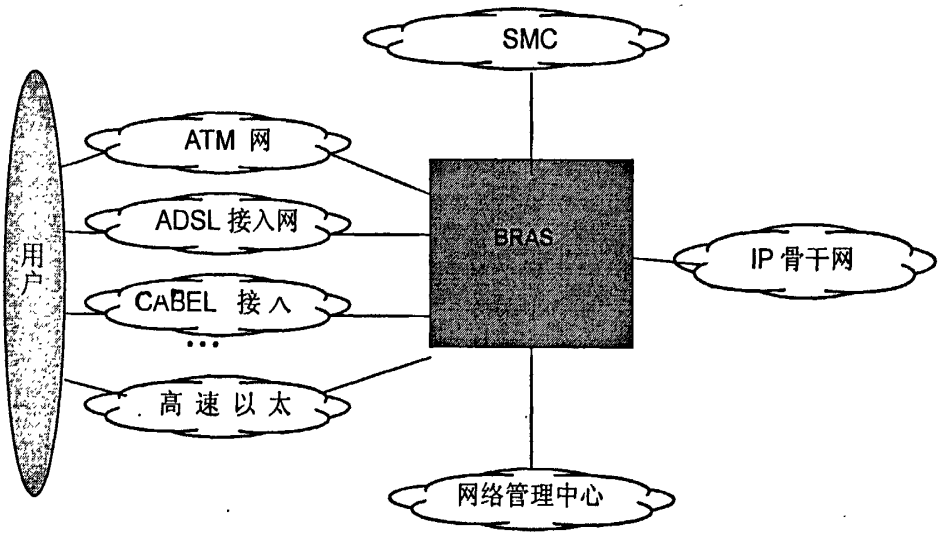


图 1-1 BRAS 在网络中的位置

BRAS 可以高速地提供大量宽带用户的接入，易于快速扩容和增加新功能，可支持 ADSL、CTMS、LAN、无线接入等多种接入方式，满足各种不同类型的运营商和服务提供商的需要。具有简单、高效、统一的用户管理模式，提供灵活的多种多种认证、计费和管理方法。

普通用户通过各种接入手段，包括以太网方式或 ADSL 方式接入到 BRAS 中，经过 BRAS 的认证之后（认证一般和 RADIUS 服务器结合进行），用户可以选择 BRAS 提供的多种上网服务，如选择 ISP，或选择上某个企业的内部网，或选择运营商提供的特殊服务，

如 VOD 等。BRAS 与 RADIUS 配合对用户的行为进行计费，如按时长计费或按流量计费。除普通用户以外，BRAS 还适用于对集团大用户的承载和管理， BRAS 支持专线、VPN 以及虚拟 ISP。

BRAS 从路由器发展而来[2]，作为接入汇聚层设备，替代了原来这个层次上的路由器（或者三层交换机），因为传统的汇聚层路由器只起到网络互联的作用，没有用户的概念，只有流的概念，不利于电信精细化管理（收费），没有对网络流量进行计费的功能。BRAS 网关不仅采用新型的网络处理器构建路由器实现网络互联功能，而且在此基础上加入了用户接入认证计费等相应的模块，在汇聚层实现对运营网络中用户的管理。BRAS 和路由器的比较如表 1-1。

表 1-1 BRAS 和路由器的比较

| | | BRAS | 路由器 |
|---|---|---|---|
| 管理平面 | 设备 | 嵌入式网管、SNMP、RMON | 嵌入式网管、SNMP、RMON |
| | 用户 | 用户管理、AAA、Radius | 无 |
| 控制平面 | 设备 | 路由协议以及其他信令 | 路由协议以及其他信令 |
| | 用户 | PPPoX 协议处理、用户认证 | 无 |
| 数据平面 | 设备 | PPPoX 接入，DHCP 接入， | 路由方式，互连互通 |

## 1.2 课题来源

在上节的背景介绍中已经提到，BRAS 是从路由器发展而来，而路由器技术已经发生的重要的变化。现今主流路由器已经发展到基于网络处理器的三层体系结构（控制层面；数据层面；管理层面），网络处理器处于底层，负责数据报文的线速转发。虽然网络处理器具有软件可编程能力也可实现对分组处理流程的优化，以满足线速处理的要求，但是也存在着诸如指令偏向底层以致不方便进行复杂业务处理、内存空间小等特点，使得在处理复杂数据结构方面存在缺陷；另一方面，网络处理器主要功能在于实现报文的线速转发，让其分担数据结构维护和与上层管理和控制层面的交互是不方便也不值得的，因此需要一个分担网络处理器上述任务的模块以实现对网络处理器需要的业务支持数据结构进行存储和维护，以及实现与上层业务进行报文交互的模块，在本文中称为驱动模块。

本课题主要来源于中兴通讯股份有限公司科技基金项目，该项目主要对电信级宽带接入服务器（BRAS）项目的研究，该项目从高端路由器的 BRAS 中的驱动模块、主备倒换模块、UM 模块、AM 模块、SVRCFG 模块、AAA 模块、TIMERMGR 模块、RADIUS 模块、NETLOG 模块、MCAST 模块、PPP 模块、SAL 模块等小模块，本文中的内容属于其中承上启下的驱动模块。本文主要目标是设计合理的驱动模块相关技术，以提高驱动 BRAS 系统的性能和稳定性，研究内容包括如下方面：

1）设计合理的算法数据结构，以实现驱动模块对于与网络处理器微引擎共享的数据结构的存储和维护。包括针对不同类型业务，以实现存储空间的高效利用，维护操作的快速和系统稳定性为目标，课题针对不同类型的业务，设计应用了包括 HASH，TRIE,RFC 算法数据结构存储相关业务数据结构，实现了相应的维护操作。

2）设计了服务于主网络处理器（负责上行报文处理），从网络处理器（负责下行报文处理）之间及与上层业务控制管理模块进行通讯的报文交互机制。

## 1.3 论文的研究内容与结构安排

第一章 首先简要介绍了本课题的研究背景、课题来源项目的研究内容、以及参与项目过程中所进行与课题相关的研究工作。

第二章 对 BRAS 驱动模块的相关概念进行了介绍，包括 BRAS 的产生背景与功能需求，对网络处理器的相关概念也进行了介绍。

第三章 描述了本课题的 BRAS 交换体系结构，着重对驱动模块在系统中的位置和功能进行了总体性的描述。

第四章 对 BRAS 驱动模块相关业务支持数据结构的设计和实现进行描述和探讨。对于主要使用的 HASH、TRIE、RFC 算法作了深入的分析和探讨，介绍了实现原理并且对实现结果进行了分析。

第五章 介绍了驱动模块的通讯实现机制，对主网络处理器（IXP2800）上送报文、从网络处理器（IXP2800）下发报文、从网络处理器（IXP2800）上送报文进行分别介绍。

最后，第六章总结了本文所作的工作，并对该课题进一步研究工作的重点和难点进行概述。

# 第二章 BRAS 驱动相关概念与技术

## 2.1 BRAS 介绍

BRAS 是面向宽带网络应用的接入网关，可以完成宽带用户的 IP/ATM 网的数据接入（目前主要接入手段主要基于 xDSL/Cable Modem/高速以太网接入技术/无线宽带接入技术等），实现多种业务的汇聚和转发，解决不同用户对传输容量、带宽利用率等要求，为用户提供 VPN 服务、构建企业内部 Internet、支持管理域向用户批发业务等应用，负责用户的管理和控制。BRAS 是宽带城域网中的重要设备之一，它的引入将使城域网成为可运营、可管理的网络。它实现了多种宽带用户的接入，快速的增值业务投放。适用的对象包括电信运营公司、接入服务提供商（ASP）、国际互联网服务提供商（ISP）。

BRAS 高速地提供大量宽带用户的接入，易于快速扩容和增加新功能，可支持 ADSL、CTMS、LAN、无线接入等多种接入方式，满足各种不同类型的运营商和服务提供商的需要。具有简单、高效、统一的用户管理模式，提供灵活的多种多种认证、计费和管理方法。其在通信网中所处位置如图 2-1 所示。

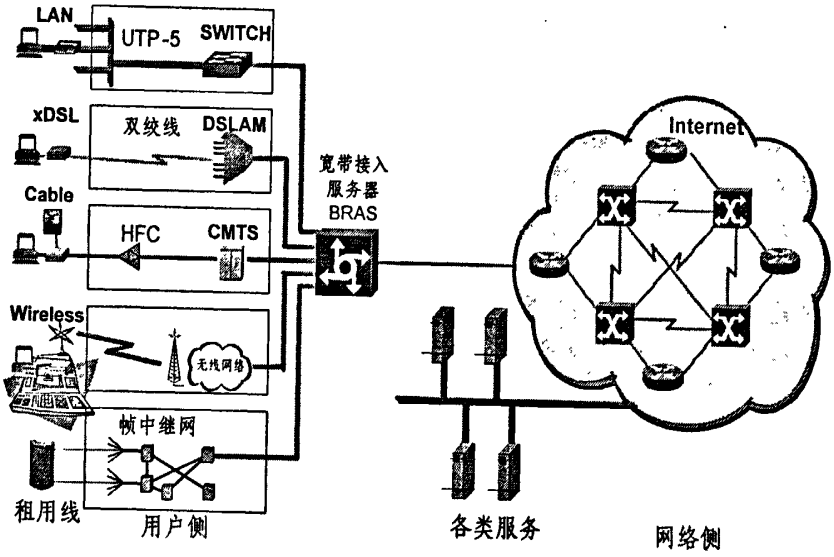


图 2-1 网络功能结构图

图 2-2 显示了 BRAS 的一个典型应用，用户通过各种接入手段，包括以太网方式或 ADSL 方式接入到 BRAS 中，经过 BRAS 的认证之后（认证一般和 RADIUS 服务器结合进行），用户可以选择 BRAS 提供的多种上网服务，如选择 ISP，或选择上某个企业的内部网，或选择运营商提供的特殊服务，如 VOD 等。BRAS 与 RADIUS 配合对用户的行为

进行计费，如按时长计费或按流量计费。

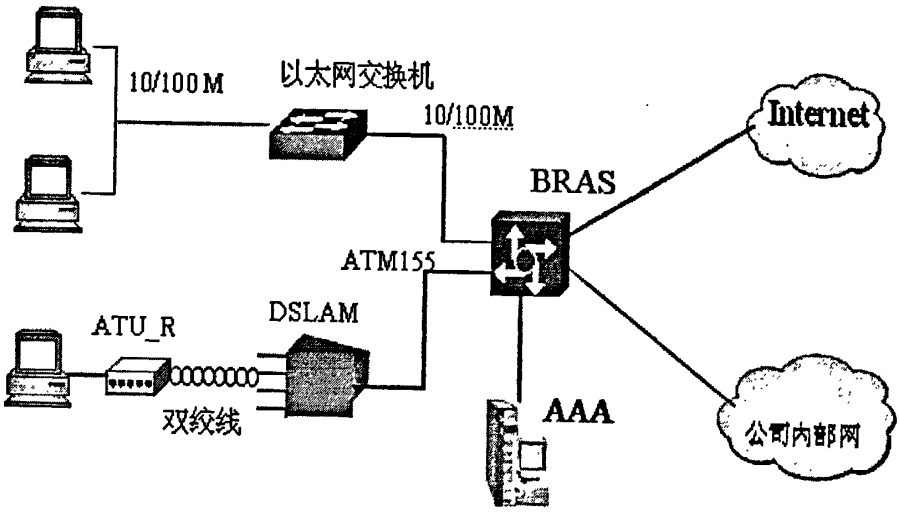


图 2-2 BRAS 的一个典型应用

## 2.2 BRAS 的功能需求

BRAS 是宽带接入网的骨干网之间的桥梁，是基于网络处理器的分布式系统[3]，提供基本的接入手段和宽带接入网的管理功能。它位于网络的边缘，提供宽带接入服务、实现多种业务的汇聚与转发，能满足不同用户对传输容量和带宽利用率的要求，因此是宽带用户接入的核心设备[4，5]。图 2-3 与图 2-4 为引入 BRAS 设备的典型网络结构。

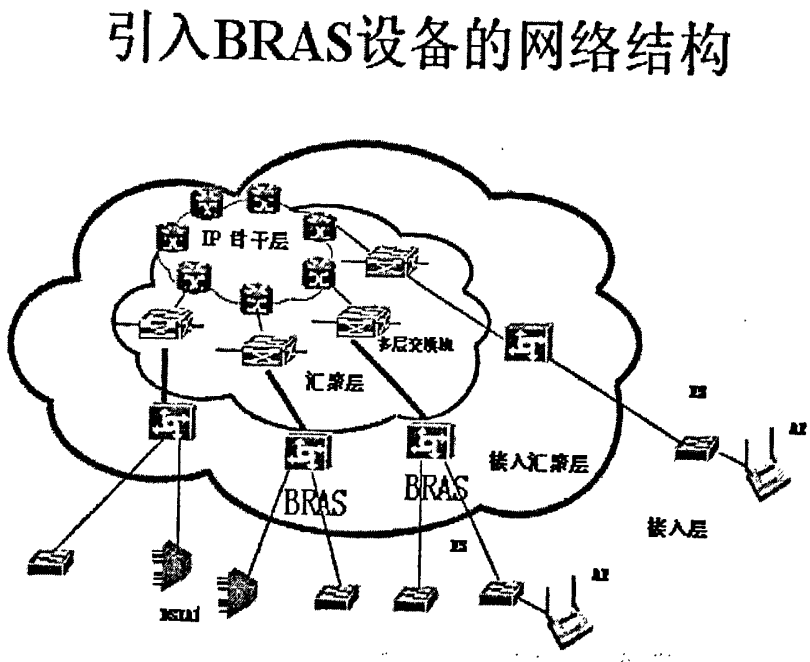


图 2-3 引入 BRAS 设备的网络结构

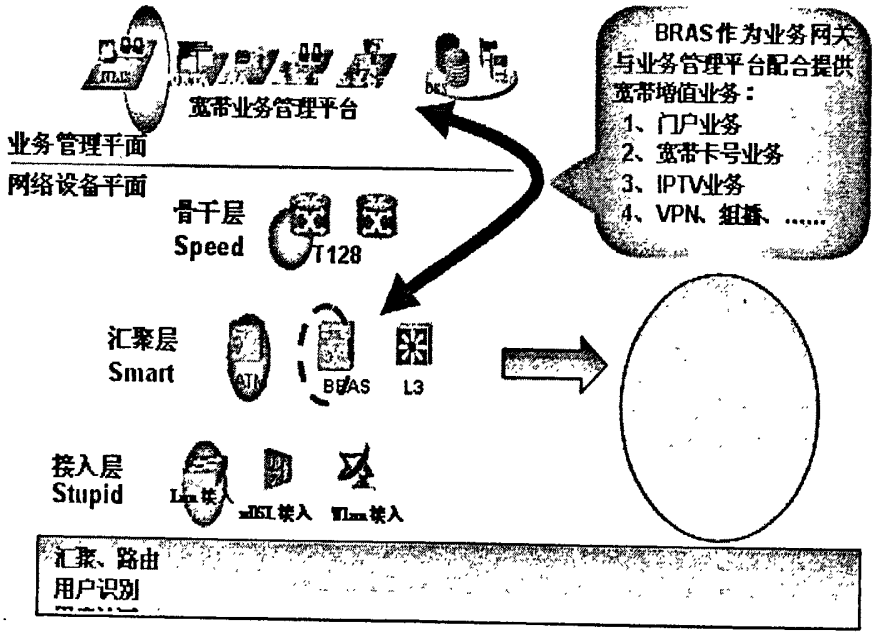


图 2-4 引入 BRAS 设备的网络结构

BRAS 的功能需求简介：

- 用户管理、认证计费、安全等方面的全部功能，如PPP接入、固定IP接入、DHCP+WEB功能、RADIUS认证计费、防攻击
- 支持RIP、OSPF、BGP、IS-IS等动态路由协议[24，27]
- 支持MPLS VPN（包括二层和三层）
- 支持VPDN，V1.0支持LAC
- 支持各种组播路由协议
- 支持限速、Diffserv、802.1P、优先级队列等QoS技术
- 主控板、交换板、电源冗余备份
- 采用分布式设计，用户接入处理在线卡上终结，大大提高了系统性能和系统可用性。

## 2.3 网络处理器

当前 BRAS 已经发展到采取高端路由器内核架构，而高端路由器内核架构的关键技术是使用网络处理器实现报文的处理和转发。本文研究的 BRAS 驱动模块的实质是网络处理器的服务模块，为网络处理器实现相关业务维护数据结构和实现网络处理器与上层业务的报文交互[6]。

网络处理器芯片是一种可编程器件，主要应用于通信领域的各种任务，如包处理、协议分析、路由查找、声音/数据汇聚、防火墙、Qos 等[7]，是一种专门用于网络的专用包处理器，网络处理器一般可以利用软件或固件对其编程，从而实现特殊的用途。网络处理器如同微处理器简化计算机设计一样，将路由器、交换机和防火墙等复杂网络设备的设计和实现变得常规化和规范化。

## 2.3.1 网络处理器的定义与特性

2.3.1.1 网络处理器的定义

网络处理器是为网络应用领域设计的专用指令处理器（Application Specific Instructure Set Processor，ASIP）[8]。ASIP 具有自己的结构特征和专门的电路设计以适应于网络的分组处理，同时它又是一块软件可编程的芯片。从功能方面来看，网络处理器可以定义如下：

网络处理器是具有以下功能的 IC：

1)它具有软件可编程能力；

2)它是对分组处理流程的优化，以满足线速处理的要求；

3)它可以接管很多原来主 CPU 完成的控制与管理功能。

2.3.1.2 网络处理器的特点

网络处理器是新一代的用来执行数据处理和转发的高速可编程处理器，其特点如下：

1)可编程性与灵活性[9，10]：网络处理器是硬件和软件的结合体。硬件方面，网络处理器芯片内部部件主要由协处理单元和网络处理单元组成。协处理单元一般运行嵌入式系统，可编程。网络处理单元一般由多个微引擎单元组成，微引擎单元中采用多线程结构，具有高速处理大容量数据的能力。软件方面由板级支持包、嵌入式操作系统、路由协议软件包和伪代码组成。其中：前三项作用于协处理单元，板级支持包记录协处理单元需要管理的硬件信息以及其配置信息，嵌入式操作系统是路由协议或其他应用协议运行的基础，协处理单元通过运行路由协议软件包可生成并维护路由表。微代码运行于网络处理单元的微引擎中，主要完成对数据的处理以及转发。网络处理器这种软硬件结合的方式一方面通过硬件芯片所集成的多处理单元、多线程结构保证其高速处理能力，另一方面通过软件所包含的专用指令集的可编程性保证其灵活性。

2)硬件并行处理能力：网络处理器具有丰富的高速 I/O 接口，如物理链路接口、交换接口、PCI 总线接口等，通过内部高速总线连接在一起，使其具有很强的硬件并行处理能力。

3)可扩展性：多个网络处理器可以互连构成网络处理器簇，以支持更大型的高速网络处理，适应网络发展。

网络处理器的出现将硬件的高效性和软件实现方案的灵活性结合起来，将传统网络数据包的“存储-转发”模型变成了“存储-处理-转发”模型，并且还提供对上层网络的运行支持，使高速智能网络环境下的流量控制机制成为可能。

## 2.3.2 网络处理器的基本架构

### 2.3.2.1 Intel IXA 架构

IXA（Intel Exchange Architecture）是 Intel 公司开发的用于网络处理器产品系列的体系结构。Intel IXA 确定了一种以软件可编程处理器为基础的新型网络架构，使网络系统从传统的固定协议逐渐演变为具有软件升级功能的可编程系统。IXA 具有强大的包处理能力，能适应高速网络中通信服务的要求。

IXA 架构是一个灵活的全方位框架，也是一套完整的网络体系解决方案，涵盖了软件和硬件两个部分。硬件部分以 Intel 网络处理器（IXP1200、IXP2400、IXP2800 等）为核心，集成了嵌入式 Intel 架构系统处理器、相应的应用程序接口（Application Programming Interface，API）以及其他主要的 Intel IXA 组件等；软件方面 Intel 公司提供了网络处理器软件的开发环境，可以实现网络系统的编辑以及仿真调试。

Intel IXA 编程架构与传统的网络编程架构有着巨大的差别。传统编程架构中将数据处理和数据包的接受以及其他处理集中在一个层面。Intel IXA 架构将处理分为以下三个层面[11]：

1)控制层面：处理各种通信协议，维护转发表及状态信息；

2)数据层面：包括慢速数据通道和快速数据通道，负责数据的转发；

3)管理层面：负责系统的管理、统计、计费等，以及 IP Qos 中策略的管理。

数据处理层面的存在很大程度上加快了数据处理的速度。而控制层面与管理层面将网络系统中的控制信息与管理信息独立出来，由专有层面负责这些信息的处理，扩充了网络系统对数据的处理能力。这种分层的处理方式符合 IP Qos 的需求，能适应网络流量控制的要求。

在硬件框架上，IXA 体系结构中使用以下的三种关键技术：Intel 微引擎技术，Intel Xscale 技术，以及多线程技术使 IXA 体系结构更高效更灵活。

### 2.3.2.2 Intel 微引擎技术

Intel IXA 体系结构中包含多个可编程的、硬件支持多线程的 RISC 处理器，称为微引擎（Microengine，ME）。微处理器可提供每秒 1G 的运算能力，可用于高速网络中的数据包处理，其硬件结构简图如图 2-5 所示。

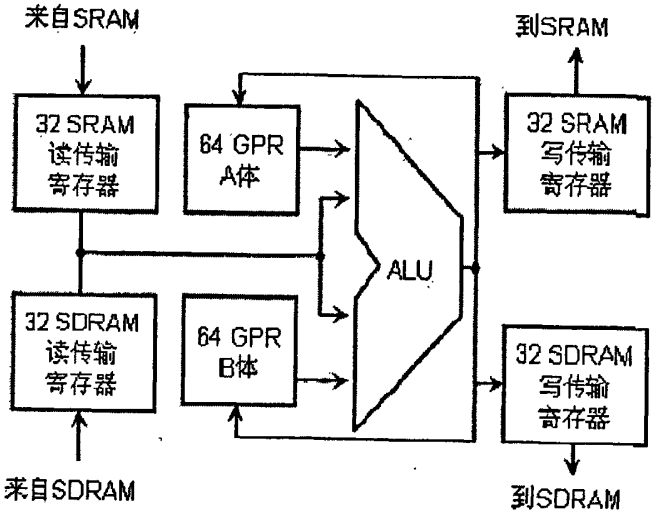


图 2-5 微引擎结构简图

微引擎中包含一个 32 位的算术逻辑单元 ALU(Arithmetic Logic Unit)和移位器部件，能够在一个周期内完成一次 ALU 操作及一次移位操作。微引擎中有 128 个 32 位通用寄存器（GPR）分为 A 体和 B 体，分别连接 ALU，用于存放程序中的数据。SRAM 与 SDRAM 用于在微引擎和其他部件中存储传输数据。

微引擎执行指令经过五级流水线：读指令、指令译码、形成源寄存器地址、从源寄存器地址读操作数、执行并将结果写入目标寄存器。微引擎指令和结构针主要是为分组处理而设计的。在网络系统中，微引擎主要用于数据包的处理，包括：包分类、包头效验、包的过滤转发、路由表查找及队列管理等[12]。

2.3.2.3 Intel Xscale 技术

Intel Xscale 技术在工业上提供最高速的能量到性能的转化率，在平面控制的应用领域中提供低动力、高密度处理的能力[13]。Intel Xscale 微架构中使用了超流水线技术，具有一个多进程且高效的指令级处理管道架构，因此其反应时间小。Intel Xscale 技术具有高性价比，低能耗的特点。在网络处理器体系架构中，Intel Xscale 技术应用在 Xscale Core 上。Xscale Core 是 Intel 网络处理器的高层控制及管理单元。主要用于控制面的任务处理及异常包的处理，如：路由表等与微引擎共享的数据结构的管理和更新，建立和控制通讯媒介和交换装置等，也可用来处理一些要求附加的需要复杂处理的异常数据包。

2.3.2.4 多线程技术

网络数据包之间存在弱相关性，它们不能被有效的高速缓存在单一处理器的 Cache 中。因此，网络处理器的微引擎单元采用了硬件多线程技术（Multi-Processing）[14]，线程之间切换的时间开销很小，可将不相关的数据包缓存在多个微引擎中，达到了并行处理不相关数据包的功能，大大减轻了过大外部存储器访问延时带来的不利影响。

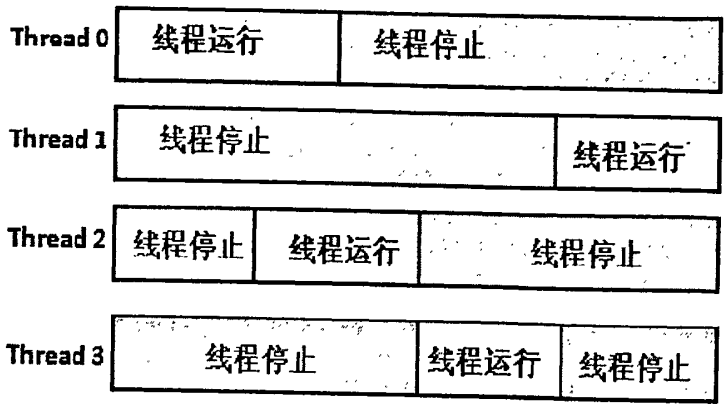


图 2-6 多线程工作方式

图 2-6 所表示的是网络处理器的多线程工作方式。网络处理器的统一微引擎上可运行多个线程（Thread）。微引擎具有 4 个程序计数器，应用于 4 个线程。线程切换时不需要保存程序计数器的值。每个线程有自己的线程表示，使得对应访存结果能够直接返回给请求的线程。在实际应用中，当微引擎中的一个线程发出访问请求时，为了避免等待，该线程可主动将自身切换出去，使得其他线程继续运行，待到存储器返回时再切换到该线程。当一个线程进行访存等慢速操作时，可以启动线程切换，从而充分利用访问间歇，提高微引擎的利用率。

### 2.3.3 Intel IXA 可移植编程架构

网络的飞速发展，促使网络处理器研究的跃进，新产品层出不穷。基于网络处理器的软件开发的可移植性及重用性[15]成为网络处理器技术发展所要关注的重点之一。Intel IXA 移动架构为基于 Intel 网络处理器的软件开发提供了应用编程接口和硬件抽象，使得运行在 XScale 核和微引擎上的软件开发具有良好的可移植性和可重用性。

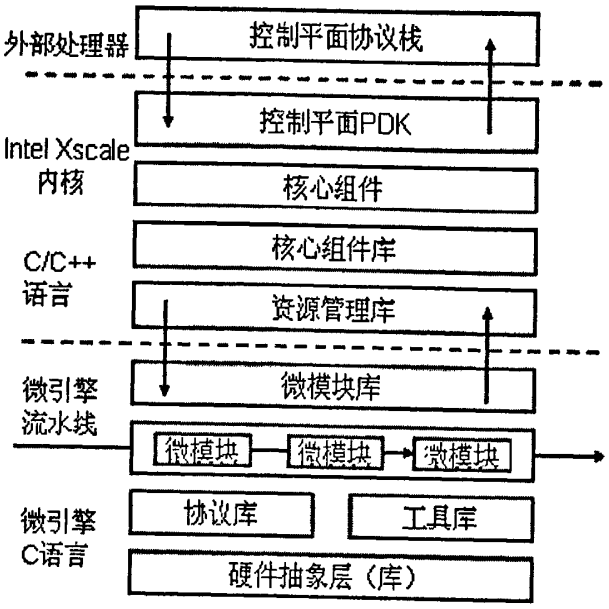


图 2-7 Intel IXA 移动架构

Intel IXA 可移植框架由底层至自上依次包括：硬件抽象层（库）、协议库及工具库、微模块、微模块库、资源管理库、核心组件库、控制面板 PDK 以及控制平面协议栈。

Intel IXA 可移植框架充分体现了构件的思想，使网络处理器在软件开发中可分模块进行编辑处理，再按一定的顺序将各模块组织起来满足特定的应用。这种构件的方式充分体现了 Intel IXA 架构的可移植性及重用性，使网络处理器的应用开发更加灵活。

硬件抽象层、协议库及工具库也可总称为数据平面优化库。库中包括了一些底层的微引擎宏指令以及微引擎 C 语言编写的函数库，可用来编写微模块及其他微引擎代码。这些代码经过 Intel IXA 的优化，效率高，占用空间小。

在网络处理器中，由微引擎上快速通道处理网络数据包的处理函数统称为微模块。微模块函数由宏指令或微引擎C语言编写，可以完成一个较为完整独立的数据包处理功能如：IPv4 转发、2 层以太网过滤、数据包分类等。

核心组件与核心组件运用 Intel Xscale 技术，主要负责系统的配置、管理、异常数据包处理等。资源管理库为 Intel Xscale Core 上的一个软件模块，为系统的硬件初始化、配置、管理、微引擎之间通信，以及微引擎与 Intel XscleCore 之间提供应用程序接口。

控制平面 PDK 提供了控制平面与数据平面的接口，使得控制平面与数据平面的通信更为方便简单。控制平面协议栈是将处理器与外部处理器连结起来的协议栈。Intel IXA 移动架构是 Intel IXA 软件开发包完整的一个部分，使网络处理器的软件开发具有了模块化，可复用的特点。

### 2.3.4 Intel 系列网络处理器

自 1997 年网络处理器的概念被提出以来，其引起了许多网络设备厂商的关注，许多厂商开始研究并生产网络处理器芯片。比较典型的产品有 Intel 公司的 IXP4XX 系列、IXP1200 系列、IXP2400 系列和 IXP2800 系列[16]，以及 Motorola 公司的 C-5，Cisco 公司的 Toaster2，IBM 公司的 NP3G4 等。其中，Intel 公司以其雄厚的技术实力，广阔的传统服务器市场以及有力的第三方支持，逐渐在在网络处理器行业中占据了一席之地。在国际网络处理器技术论坛进行的一些性能基准测试中，IXP2XXX 的产品在 MPLSVPN、数据包处理以及其他方面都有优秀的表现。本文讨论的流量管理机制便基于 Intel IXP2800 网络处理器平台实现。

#### 2.3.4.1 Intel IXP2XXX 网络处理器

Intel IXP2XXX 是 Intel 第二代网络处理器的统称，属于高端网络处理器。一般包括 IXP2400、IXP2800、IXP2850。IXP2400 是应用较为广泛的一款网络处理器，主要应用在 OC3-OC12 等中、低速率场合。IXP2800 主要应用于 OC12~OC192 等中、高速率场合。IXP2850 在 IXP2800 的基础上集成了两个加/解密单元，可达到 10Gps 的加/解密速度，因此可用于高速 IPSec 等场合。

IXP2XXX 采用了专用硬件加速设备和灵活的软件算法处理高速、复杂的网络业务。同时，采用了标准的工业接口，使 IXP2XXX 能够方便的与其他厂家的设备进行互联。在应用中，IXP2XXX 系列的网络处理器都可拆解为“控制平面”与“数据平面”。处理器中的 Xscale 负责管道处理工作，执行底层控制工作，如：信息的传输、系统内其他处理器的沟通等。数据平面建立于网络处理器中的微引擎之上。而微引擎的多少客观上反映了处理器平行处理能力的强弱。IXP2XXX 系列的产品中，IXP2400 具有 8 个微引擎单元，而 IXP2800 集成了 16 个微引擎单元，与 IXP2400 相比集成了更多的微引擎单元，更高速的存储器接口及更高时钟速率运行的处理器，因此其 IXP2800 具有更高的数据处理能力，同时更适用于现在的网络系统应用。

#### 2.3.4.2 Intel IXP2800 网络处理器

Intel IXP2800[12]是 Intel 第二代处理器中性能最强大的处理器芯片，它包括 16 个完全可编程的多线程微引擎（ME，microengine），使得 Intel Xscale 核心之间的包转发和信息管理都能在单芯片上完成。IXP2800 能达到 250 亿次每秒的运转速度。

Intel IXP2800 片内集成了 16 个 32 位可编程的微引擎，一个 700MHz 的 32 位通用 RSIC 处理器 Xscale core。Xscale 和微引擎都能访问共享资源，如：媒体交换接口、SRAM、

SDRAM。除此之外，IXP 2800 上应用了 Intel 专利技术超任务链接技术（Hyper Task Chaining Processing），可支持每秒 23.1Giga 的操作。

包转发能力和流量管理能力达到了 10Gps，能够满足现代高速智能网络应用的需要。图 2-8 给出了 IXP2800 的系统架构。

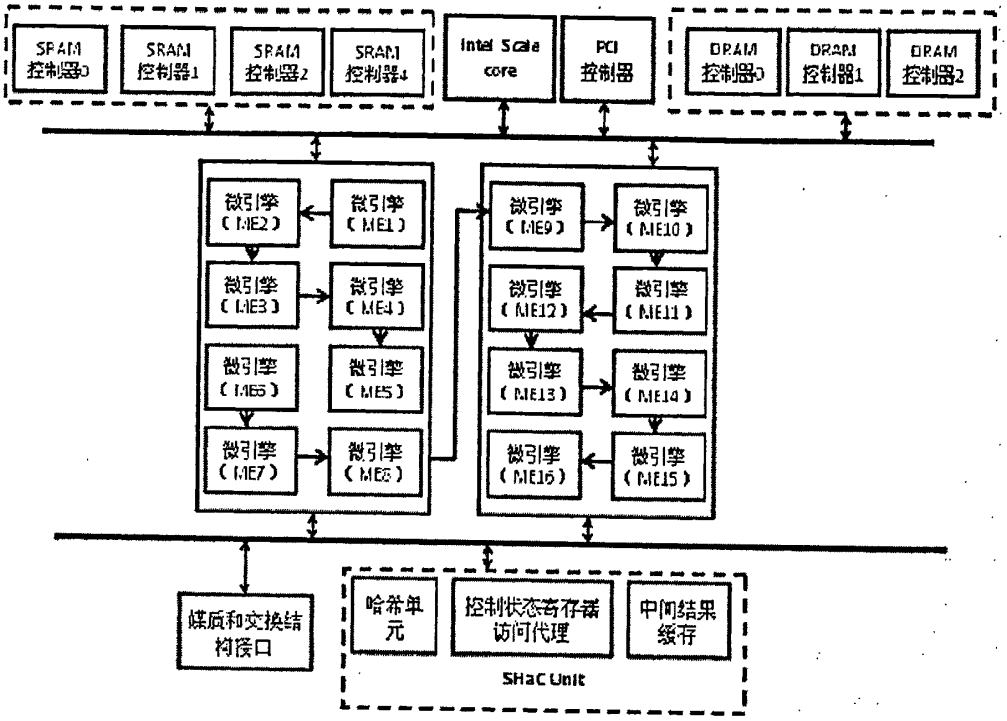


图 2-8 Intel IXP2800 基本架构

如图 2-8 所示，Intel IXP2800 的基本组成部分有微引擎（Microengine，ME）、Xscale Core 核心、媒质和交换接口（Media and Switch Fabric Interface）、SRAM 控制器（SRAM Control）、DRAM 控制器（DRAM Control）、PCI 控制器、以及由哈希单元（Hash Unit）、控制状态寄存器访问代理（CAP）、中间结果缓存（Scratch Memory）组成的 SHaC 单元（SHaC Unit）。各组成部分功能简介如下：

1)Intel Xscale Core：32 位的嵌入式 RSIC（精简指令集）处理器，负责处理复杂的算法，路由表的维护，以及系统级的管理及控制等。在数据包处理中，Intel Xscale Core 还负责异常数据包的处理。本文研究的驱动模块则是运行于 Xscale Core 上，实现的功能包括上述的各项内容。

2)微引擎（Microengine，ME）：是 IXP2800 的核心部件，负责网络中的数据面（Data Plane）处理任务，以线速（Line Rate）处理数据包。ME 又称为数据包的快通道（Fast Path）。

3)媒质和交换接口（Media and Switch Fabric Interface，MSF）：IXP2800 与外部物理层设备、交换结构的接口单元是 IXP2800 在网络中接受，发送数据包的窗口。4)SRAM 控制

器（SRAM Control）：IXP2800 中有 4 个 SRAM 控制器，用于接口 SRAM 存储设备控制，管理 IXP2800 中其他功能单元对 SRAM 存储设备的访问及操作。另外，还可以用于执行特定的复杂的数据包处理操作。

5)DRAM 控制器（DRAM Control）：IXP2800 有 2 个 DRAM 控制器，用于接口 DRAM 存储设备控制，管理 IXP2800 中其他功能单元对 DRAM 存储设备的访问及操作。DRAM 可用于存储数据包、路由表等大型的数据结构。

6)PCI 控制器：用于接口控制层面处理器（Control Plane Processor）、系统管理处理器（Management Processor）、其他 IXP 处理器，以及 PCI 以太网卡等其他设备。

7)SHaC 单元：包括哈希单元（Hash Unit）、控制状态寄存器访问代理（CAP）、中间结果缓存（Scratch Memory）。其中：哈希单元是专用于提供高速哈希运算硬件单元；中间结果缓存单元是用于微引擎之间的通信以及重要数据的内部缓存；控制状态寄存器访问代理用于对 IXP2800 中的控制、状态寄存器进行访问及操作，且也可用于设定 IXP2800 的工作模式以及采集运行的情况。

## 2.4 本章小结

本章介绍了 BRAS 驱动的相关概念与技术，首先对 BRAS 作了简要的介绍，说明了 BRAS 的作用和功能需求。其次介绍了网络处理器的产生背景和定义，讨论了网络处理器特点。再次对网络处理器的基本架构加以描述，并对 Intel IXA 架构中的核心技术：微引擎技术、Xscale 技术、多线程技术进行了讨论。证明了网络处理器开发的灵活性与高效性。本章最后以 Intel IXP2800 网络处理器为例，详细介绍了该处理器的架构，组成部件及其性能。本文的实验基于 Intel IXP2800 网络处理器。

# 第三章 BRAS 系统结构

## 3.1 BRAS 交换体系架构

BRAS 是基于高端路由器体系架构的，随着路由器技术的发展[2]，BRAS 的发展也经历了多个阶段，从单处理器到并行处理器，从共享总线到交换结构的发展过程。共享总线吞吐率和单 CPU 处理能力限制了数据交换的容量，而且这种体系结构的可扩展性比较差，很难与网络接口卡接口速率的迅速提高相适应。而采用多 CPU 和交换结构的 BRAS，多处理器可以并行处理报文路由转发，交换结构提供比共享总线高得多的带宽，支持现有的高速网络接口，从而大大提高系统的数据吞吐量。

本文的研究是基于 IXP2800 网络处理器的 BRAS 体系结构，总体架构如图 3-1。

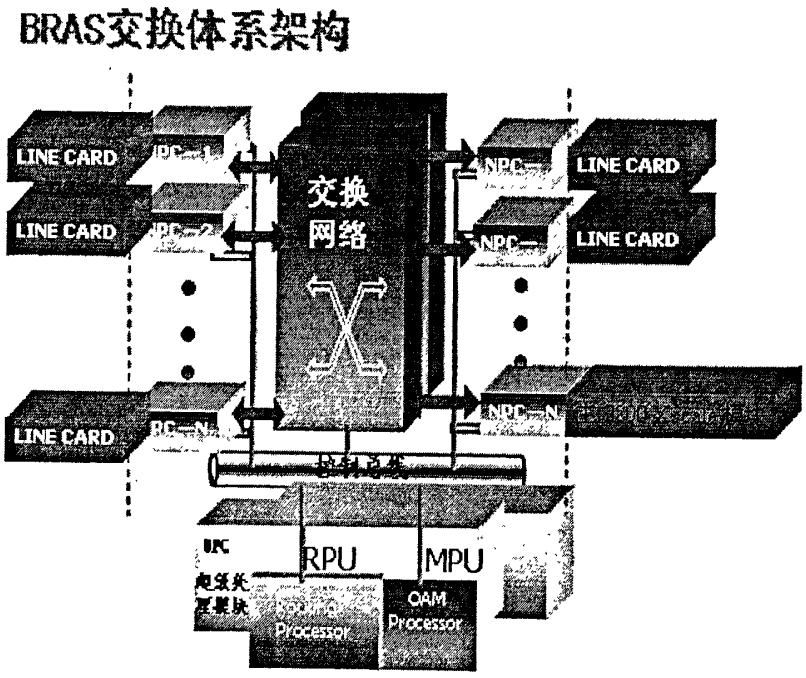


图3-1 BRAS交换体系架构

### 3.1.1 数据包处理流程

BRAS 处理的数据包主要分为两类：一类是需要直接线速转发的数据包；另一类是需要上层协议进行处理的协议包。对于普通的数据包，BRAS 通过交换网络直接转发，目前

主流的 BRAS 单块接口板都达到了 10G bps 的线速转发。对于需要上层协议处理的协议包，BRAS 需要通过驱动与上层协议进行数据包交互，以分别处理上送和下发的协议包。

数据包处理流程示意图

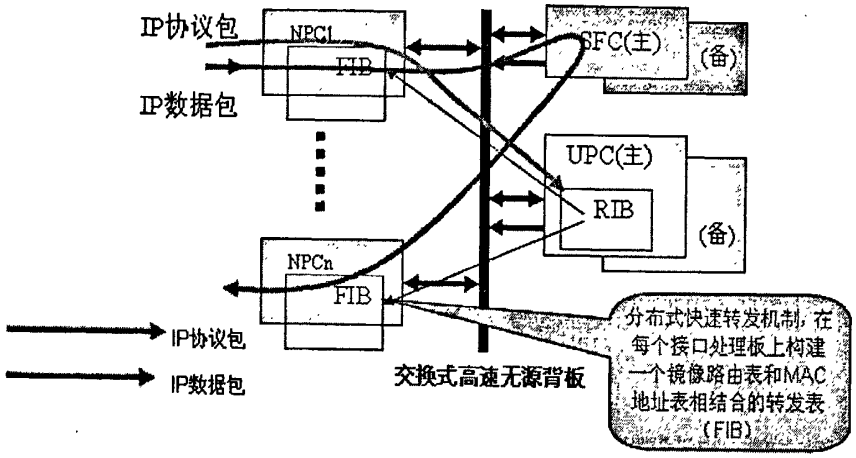


图3-2 数据包处理流程示意图

### 3.1.2 接口卡及网络处理器

BRAS 通过不同的结构卡接收不同的外部系统接入方式传送的数据包，主要的外部系统如表 3-1 所示：

表 3-1

| 外部系统 | 外部系统功能描述 | 相关接口描述 |
|---|---|---|
| ATM 网 | ATM 网的功能是汇聚 ADSL 用户的流量 | 标准的 ATM155 接口、ATM622 接口。符合 ATM 接口规范 |
| ADSL 接入网 | 向用户提供 ADSL 宽带接入，这里指的是 ADSL 的 DSLAM 直接连 UAS 的情况 | 标准的百兆以太网电口、光口或者千兆以太网口或者 ATM155 接口 |
| CABLE 接入网 | 向用户提供有线电缆（同轴电缆）宽带接入。 | 一般是百兆以太网接口 |
| 高速以太网 | 汇聚 ADSL 用户的流量，或者直接向用户提供以太网接入方式。 | 一般是千兆以太网接口，或者百兆以太网接口 |
| IP 骨干网 | 主要是 IP 城域网，负责流量传送、QOS 保证。使用户可以接入 Internet 以及各种业务中心（包括视频源、软交换中心等） | 一般是千兆以太网口，或者 POS 接口。逐渐出现 10G 以太网口。10G POS 应用估计较少。 |
| 网络管理中心 | 网络管理，包括配置管理、性能管理、故障管理、安全管理等。 | 与 UAS 物理上不一定直接相连，通过 TCP/IP 协议通讯，通过 SNMP 标准接口实现管理。 |
| SMC（业务管理中心） | 随着宽带业务的发展，可能有多种业务管理中心。典型的是认证计费设备 RADIUS，实现用户的认证、计费、授权 | 与 UAS 物理上不一定直接相连，通过 TCP/IP 协议通讯，通过 RADIUS 协议交互（RFC |

| | | |
|---|---|---|
| | 功能。 | 2865/RFC2866） |

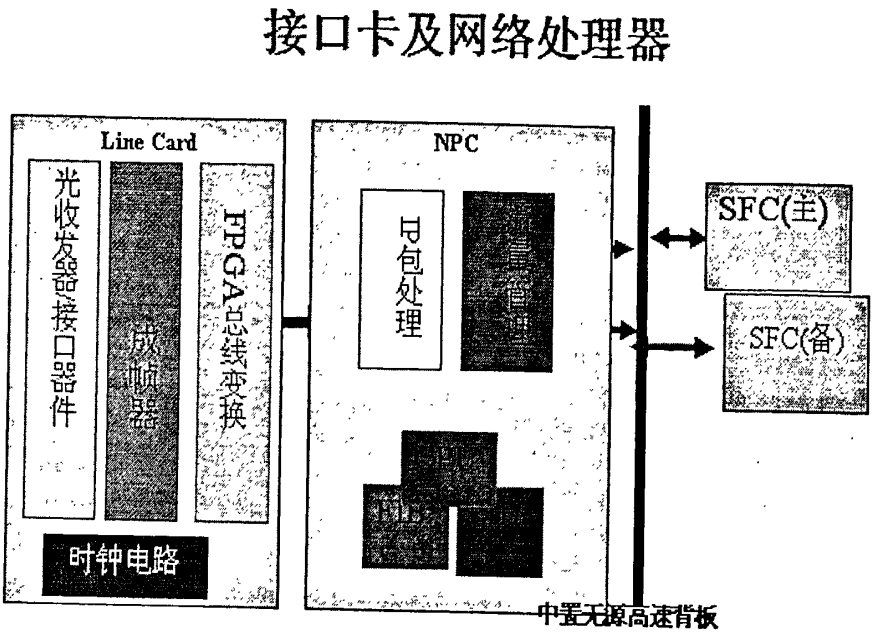


图3-3 接口卡及网络处理器

### 3.1.3 NP 单板结构

NP 单板结构如图 3-4 所示，其中 IXP2800 网络处理器处于核心位置，与外部数据交换通过接口卡进行，而内部则有 PCI、FPGA 等多种方式实现各个模块之间的通信。

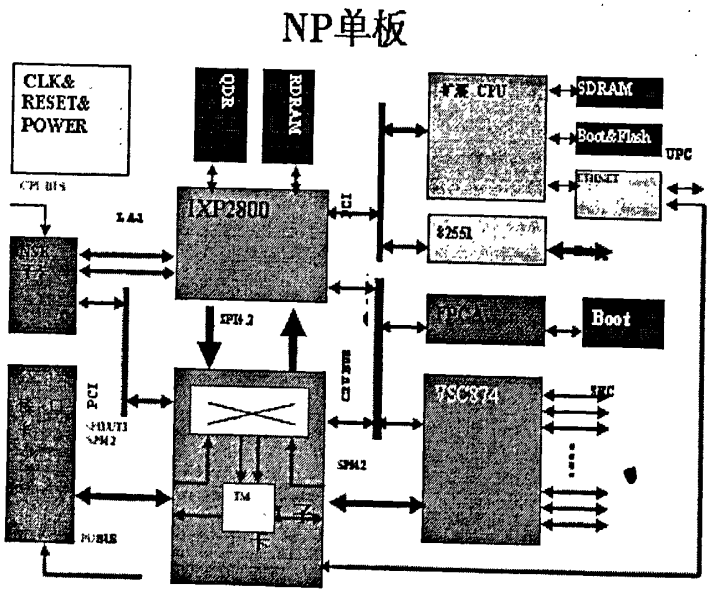


图3-4 NP单板结构

### 3.1.4 BRAS 软件子系统划分

BRAS 软件子系统包括如下部分[17]：

运行支撑系统（Running Support System）：包括平台的操作系统和系统控制子系统；

网管子系统：支持命令行管理功能，实现 SNMP 网络管理的 Agent 功能；

业务控制子系统：包括数据库/业务配置，业务总控模块，AAA 模块，IP 地址管理模块，活动用户管理模块，定制业务控制模块 。

协议子系统：协议子系统包括二层协议子系统、三层基本协议子系统、MPLS 协议子系统、IPv4 路由子系统、IPv6 路由子系统，应用层协议子系统；

承载子系统：主要负责底层微码实现，包括 NP 的微码下载，底层数据表管理，数据包接收、分析处理、查表寻路、封装和发送，用户权限控制，QOS 管理等功能 。各个系统之间的层次关系如图 3-5 所示。

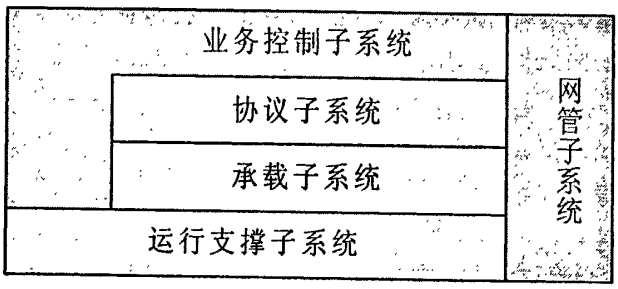


图3-5 BRAS软件子系统结构

### 3.1.5 BRAS 采用分布式处理的优势

分布式处理，具有如下几方面的优势：

1）可以提高系统容量，以及呼叫处理性能；

2）可以提高整机可用性，单块单板故障不会影响其他单板；

3）可以规避主控板主备倒换的技术难点，降低系统复杂度。

## 3.2 BRAS 系统驱动模块

驱动模块位于底层网络处理器，微码与上层协议之间的中间层。驱动模块主要的功能包括：数据结构初始化，模块间报文通讯管理，相关业务表项管理。驱动模块在整个 BRAS 软件系统中的位置如图 3-6 所示。

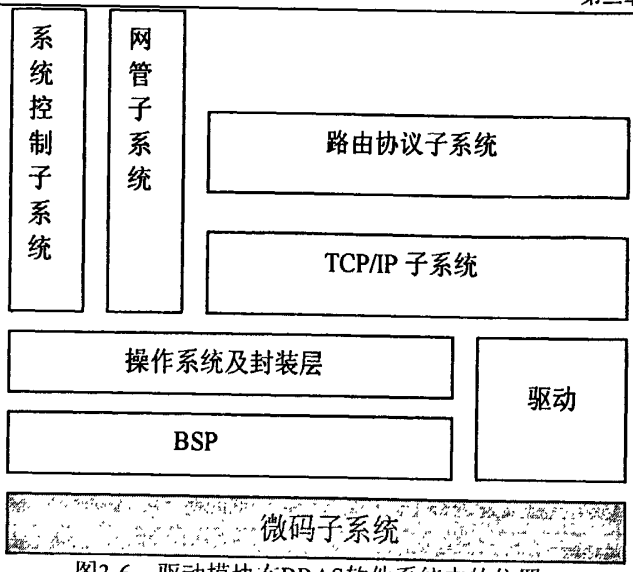


图3-6 驱动模块在BRAS软件系统中的位置

驱动模块运行在网络处理器的 Xscale core 上， 其实质是为网络处理器的报文转发处理以及与上层业务交互提供服务的模块，因此驱动模块准确的命名应该是网络处理器服务模块。

本文的研究基于 Intel 公司的 IXP2800 网络处理器，IXP2800 集成了 Xscale core 和 16 个转发微引擎。驱动模块运行在 Xscale core 中，负责对 IXP 2800 芯片上所有资源的初始化、微码转发数据包时使用的各种数据结构（表项）的初始化和管理功能、以及对微码不能处理的数据包的进一步处理。在 16 个转发微引擎中运行微码进行报文的线速处理和转发，实现 BRAS 的快速数据通道的功能。

IXP2800 的硬件结构如图 3-7 所示。

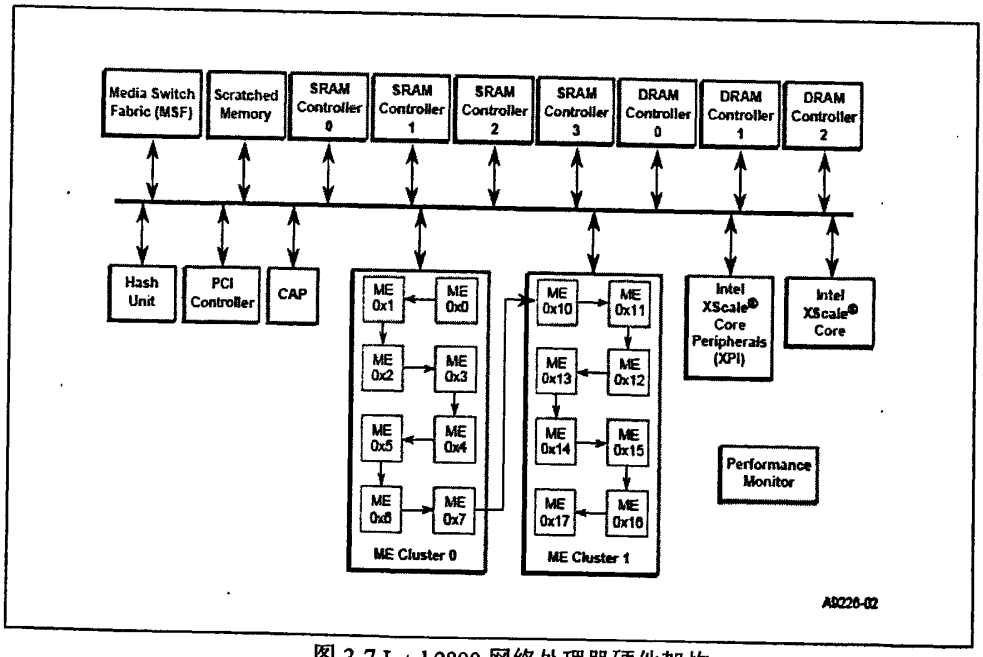


图 3-7 Intel 2800 网络处理器硬件架构

BRAS 线卡上有两种方向的数据流：

定义 3-1：上行数据流（报文），即进入 BRAS 的数据流，由 BRAS 进行处理和转发。

定义 3-2：下行数据流（报文），即 BRAS 处理完成后转发的报文，被发送到终端或者其他网络设备。

为提高转发效率，实现线速转发，NP 线卡上采用了两块 IXP2800 网络处理器分别处理上下行的报文转发，其中上行 IXP2800 作为主 NP，除了完成上行报文的转发外还进行本板资源初始化，管理，与主控板数据同步等功能，下行 IXP2800 为从 NP，主要负责向下的报文转发。主、从 IXP2800 之间通过 PCI 进行通讯：主控同步的表项由主 IXP2800 接收，发送给从 IXP2800；从 IXP2800 收到需要主控板处理的报文，先发送到主 IXP2800，再由主 IXP2800 发送给主控；从主控发来的协议报文先发送到主 IXP2800，再送到从 IXP2800，最后由从上的微引擎进行转发。主、从 IXP2800 的 Xscale core 模块还与微码进行通信，通信的内容包括：1、各种表项的下发和更新；2、协议报文的上送和下发。各个模块之间的通讯如图 3-8 所示。

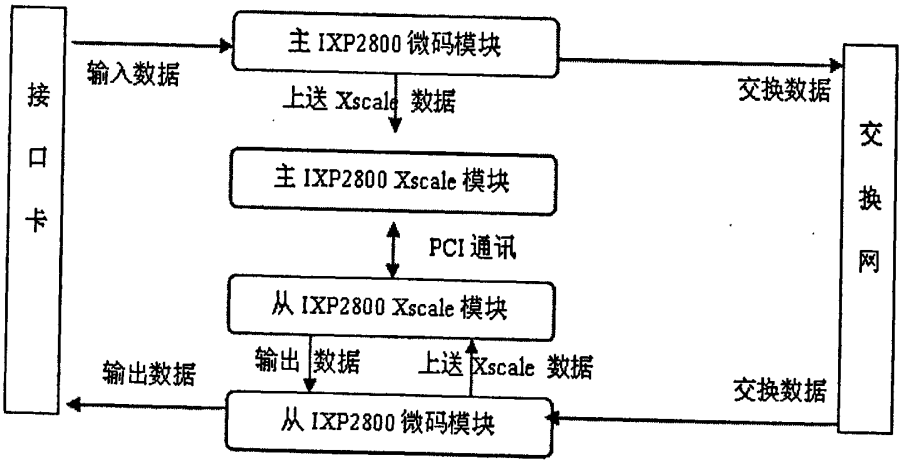


图 3-8 驱动各模块之间的通讯

## 3.3 本章小结

本章主要介绍 BRAS 体系结构，以及驱动模块在 BRAS 体系结构中的位置和与各模块的关系：首先介绍了 BRAS 中数据包的处理流程，分析接口卡及网络处理器在数据包处理过程中起的作用。然后介绍了 BRAS 体系结构中主要负责完成报文线速处理的 NP 单板的结构。再其次从软件系统的角度对 BRAS 软件系统的各模块进行了分析，说明了 BRAS 采用分布式处理方式的优点。最后，介绍了本文设计的驱动模块在系统中的作用和与各模块的交互关系。

# 第四章 BRAS 驱动业务支持数据结构设计与实现

第二章和第三章对于BRAS的相关技术和BRAS体系结构进行了总体介绍，并对BRAS驱动模块在系统中的位置和作用进行了描述和分析。作为网络处理器微引擎的服务模块，运行在Xscale core上的驱动模块的重要功能之一为存储和维护微码转发报文时需要查询的各种数据结构。驱动维护的各种数据结构是微码进行报文转发时主要的参数依据来源。各种表项数据结构被设计存储与网络处理器与运行驱动的XSCALE共享的SDRAM和SRAM中，由于SDRAM与SRAM空间有限，而为实现报文的线速转发对于访问效率的要求也很高，因此设计合理高效的驱动支撑模块数据结构对于各种业务功能的稳定发挥具有至关重要的作用。

本章开始详细探讨主要的三种数据结构和算法：1）HASH，是BRAS驱动中使用最为广泛的一种数据结构，用以实现多种业务表项的存储和查找； 2）TRIE，从节省存储空间和提高查询效率的角度考虑，本文采用TRIE数据结构进行路由表的存储和查询；3）RFC，本文使用RFC算法实现访问控制列表（ACL）的存储和查询。

## 4.1 基于HASH的相关业务表项数据结构

### 4.1.1 HASH数据结构的HASH索引表

Hash，一般翻译做“散列”，也有直接音译为”哈希“的，就是把任意长度的输入，通过散列算法，变换成固定长度的输出，该输出就是散列值，其关键优势在于根据关键字直接访问。这种转换是一种压缩映射，也就是，散列值的空间通常远小于输入的空间，不同的输入可能会散列成相同的输出，而不可能从散列值来唯一的确定输入值。由于Xscale core的内存空间十分宝贵，相关业务表项的决定参数很分散，采用HASH方法存储相关业务表项可以极大减少存储空间的开销，提高查询效率[18,19]。

为实现HASH表项的高效查找，HASH表项设计两个相关表项：HASH索引表与HASH表。为说明本文特定环境下的HASH索引表和HASH表的区别，本文给出如下定义：

定义 4-1：HASH 索引表，一组连续的存储空间，每个表项占据空间较小（一个 32 位的指针，4 个字节），通过相关的参数进行 HASH 运算得到索引号，索引到 HASH 索引

表的表项，表项中的指针指向对应冲突链的首个表项基址，如图 4-1 所示。

定义 4-2：HSAH 表，一组连续的存储空间，每个表项各个字段分为两部分：1）实现特定业务的供微码查询的各个位；2）下一个表项指针，用以在冲突链中索引到下一个冲突表项，如图 4-2 所示。

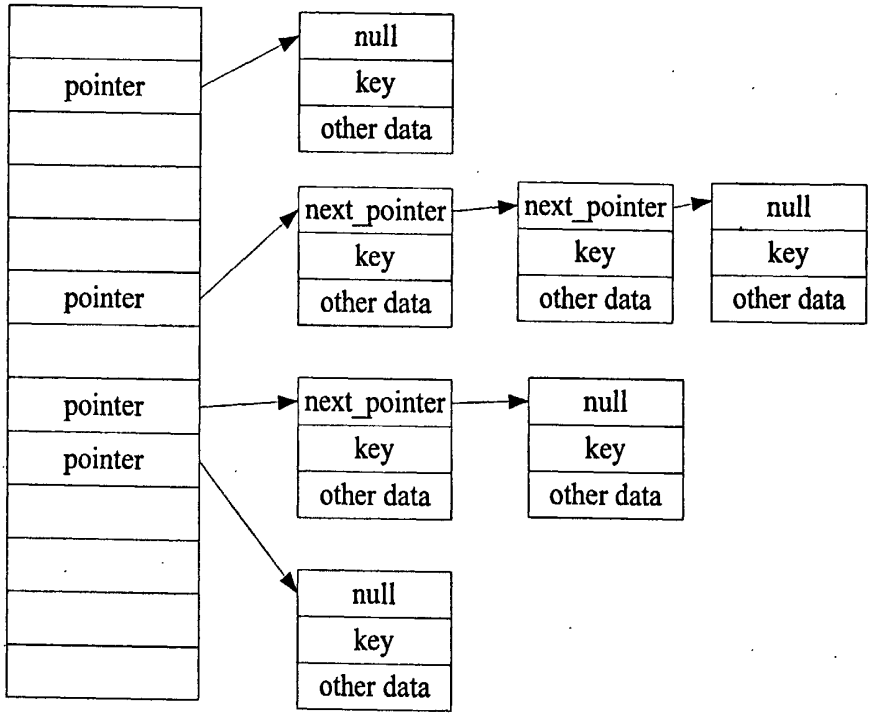


图 4-1　HASH 索引表

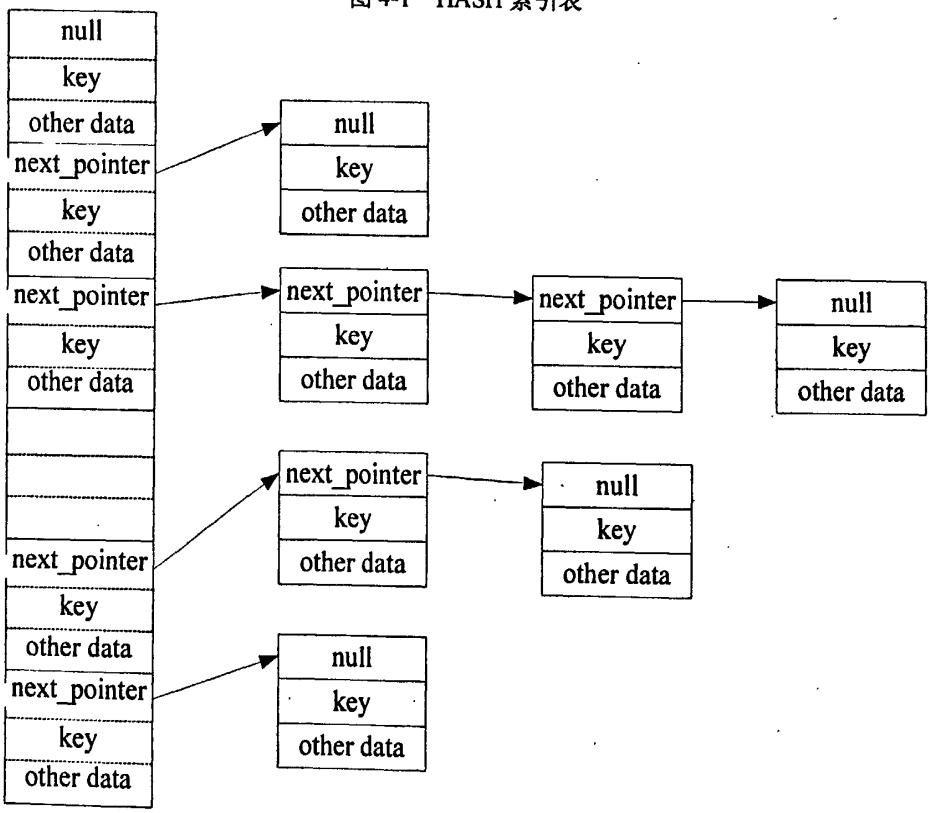

图 4-2 HASH 表结构

### 4.1.2 本文设计的基于 HASH 数据结构的业务表项

HASH 数据结构是 BRAS 网络处理器业务数据结构使用的最多的一种数据结构，有着算法简单，效率高的优点。 本文设计的 BRAS 业务驱动模块采用 HASH 算法进行存储和查找的相关业务表项主要如表 4-1 和表 4-2 所示。

表 4-1 主 IXP2800 上采用 HASH 算法进行存储和查找的的部分表项

| 业务表项 | 表项作用 |
|---|---|
| 电路表 | 存储各个电路的相关信息 |
| 接口表 | 存储各个接口的相关信息 |
| L2TP 表 | 2 层封装表相关信息 |
| MPLS 隧道电路属性表 | MPLS 隧道电路属性 |
| 用户转发表 | 各个用户相关信息 |
| 上送 Xscale 报文 MAC 限速表 | MAC 限速相关信息 |

表 4-2 从 IXP2800 上采用 HASH 算法进行存储和查找的的部分表项

| 业务表项 | 表项作用 |
|---|---|
| 用户转发表 | 用户转发表 |
| 接口表 | 存储各个接口的相关信息 |
| L2TP 表 | 2 层封装表相关信息 |
| MPLS 隧道电路属性表 | MPLS 隧道电路属性 |
| VPLS ARP 表 | VPLS ARP |
| TCP 限速表 | TCP 限速 |
| VPLS 成员表 | 各 VPLS 成员相关信息 |

表 4-1 和表 4-2 为主 IXP2800 和从 IXP2800 上采用 HASH 算法进行存储和查找的表项，相关表项结构见附录。

### 4.1.3 基于 HASH 的数据结构存储和操作实现

采用 HASH 表存储的各种表的存储与查找过程类似，本文以电路表为例，描述 HASH 表的存储与查询过程，其他由于篇幅原因，其他的表项不做赘述。

#### 4.1.3.1 基于 HASH 的数据结构初始化

各种转发表初始化的过程大致相同，首先为表项在 SRAM 或 SDRAM 中分配或指定大小的空间，将空间清零，将表基址指针保存在一个全局结构体变量 TblCtrl 中，最后将初始化完毕的表的基址下发到微引擎中去。

电路表采用如下四个键值作为 HASH 运算的参数：槽位号 (slot) 、端口号 (port)、外层 VLAN 号(outvlan)、 内层 VLAN 号(invlan)。

对电路表的操作主要包括如下二种：添加与更新、删除电路表。

4.1.3.2 添加与更新电路表过程

(1) 主收到上层协议下发添加电路表消息，主先向从发送添加电路表的消息,若从添加电路表失败，则返回，要求协议层重新发消息添加电路表，若成功则转到 2

(2) 添加主上电路表，将上层下发的电路表项结构的部分参数初始化，并依据接口板类型不同对端口号进行转化。

(3) 依据转换后的四个参数 slot,port,invlan,outvlan 计算对应的索引表索引，然后得到索引表中对应的电路表项索引的内容。

(4) 依据索引表项的有效位，判断挂接其后的 HASH 链表是否为空，若为空，则申请一个未分配的电路表项，将转化后的上层协议下发的电路表各个参数内容写入新申请的电路表项中，将该电路表项设置为对应索引表项的后继结点，再将该索引表项有效位置为 1。

(5) 若 EntryIndex 的有效位为 1，则说明该索引表项后面已经挂接了冲突链表。从头结点向后遍历 hash 链表，逐个进行匹配，匹配的依据参数为修改后的四个 hash 关键值，若找到，则将转化后的上层协议下发的电路表各个参数内容写入这个电路表项，若找到最后一个结点都未找到，则新申请一个电路表项，再将转化后的上层协议下发的电路表各个参数内容写入这个电路表项，将其作为原 hash 链表最后一个电路表项的后继结点，其后继置为空。

(6) 添加更新电路表项结束调用结束。

上述添加与更新电路表项的过程，详细流程如图 4-3。

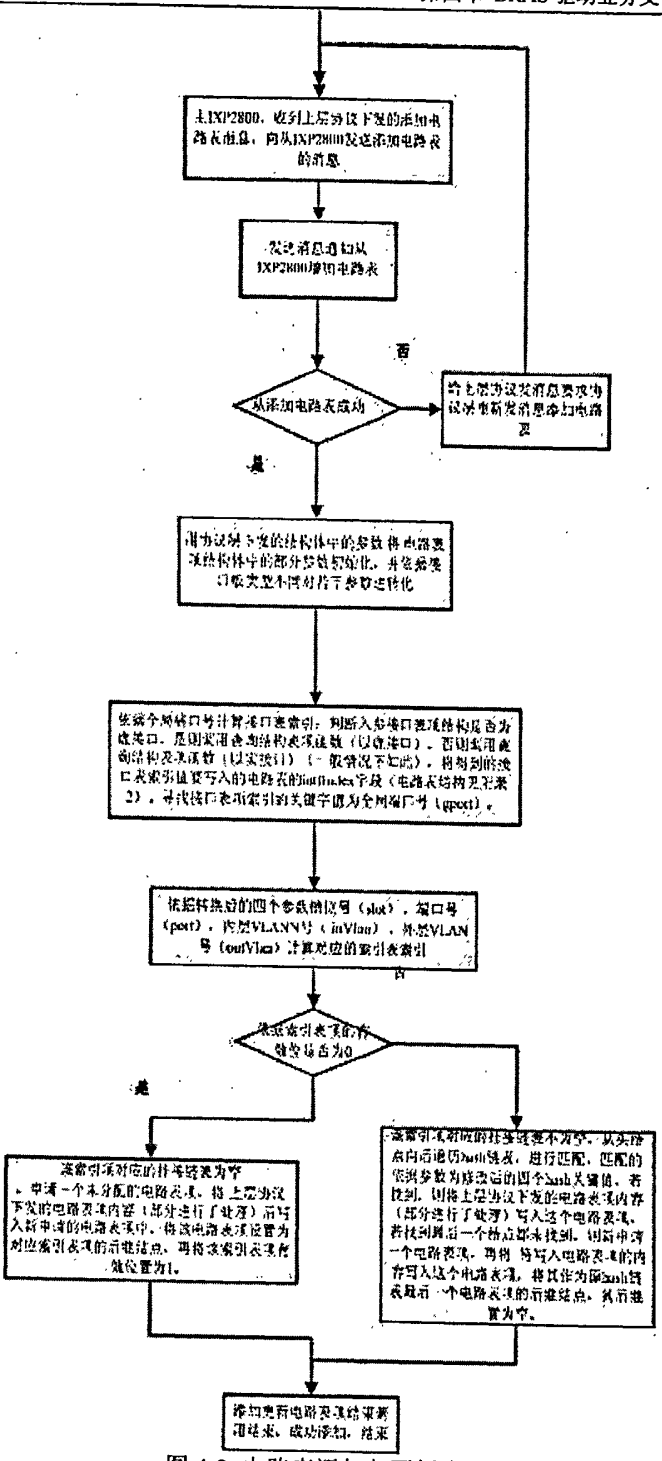


图 4-3 电路表添加与更新流程

4.1.3.3 删除电路表的过程

(1)根据上层协议下发的电路表项结构体的参数确定 slot，port，outVlan，inVlan。其中 slotId,portId 要根据接口板类型进行转换，即 slot＝ 上层下发结构体中的槽位号-1; port 的转换与接口板相关若接口板为 fei 或者 atm 板，则 portId ＝16-上层下发结构体中的端口号，若接口板为其他类型，则 portId ＝上层下发结构体中的端口号-1 ;内外层 vlan 直接由输入参数获得。转(2)。

| 外层 vlan() | | |
|---|---|---|
| Slot | port | 内层 vlan |

图 4-4 电路表 HASH 运算参数

(2)计算获取电路表项索引：以转换后得到的四个参数得到 hash 索引表的索引 hashCirEntry，然后计算得到对应的 hash 索引表项的地址 EntryIndex， EntryIndex 为相应索引表项。索引表中有一个有效位，用以判定该索引有没有挂接相应的电路表项链表。判断 EntryIndex 的有效位是否为 0，若为 0，则表示对应电路表项为空，返回。若 EntryIndex 的有效位不为 0，则转到(3)。

(3) 为操作为防止取值发生错误，设 linkIndex 为 EntryIndex 的低十六位，如果 linkIndex 与 CIR_TBL_INVALIDATE_LINK（全 0，表示无效的电路表项链接指针）相等，表示电路表中已无表可以删除，发出警告信息。否则，继续删除电路表项的过程，转到(4)。

(4)由 linkIndex 计算得到对应的电路表项 CurrentItem ，若 CurrentItem 的四个关键值 CurrentItem.slotId, CurrentItem.portId, CurrentItem.outVlan, CurrentItem.inVlan 与刚开始转换上层下发的电路表信息后得到的四个参数 slotId,portId,outVlan,inVlan 相等，表示此此 hash 链表中第一个电路表项就是我们要删除的，于是先后分两步删除：1.将待删除电路表项的后继索引赋给索引表中对应索引项 2.删除电路表项，释放该表项的存储空间。若不相等，说明 HASH 索引表后挂接的电路表项链表的表项数不止一个，转(5)。

（5）若不相等，则一直在电路表中寻找相等的表项，若找不到（即 linkIndex == CIR_TBL_INVALIDATE_LINK，表示无效的电路表项链接指针），则返回；找到，将当前要删除表项的后继表项置为前一个表项的后继表项，然后将该表项清空。详细过程过程如图 4-5 所示 。

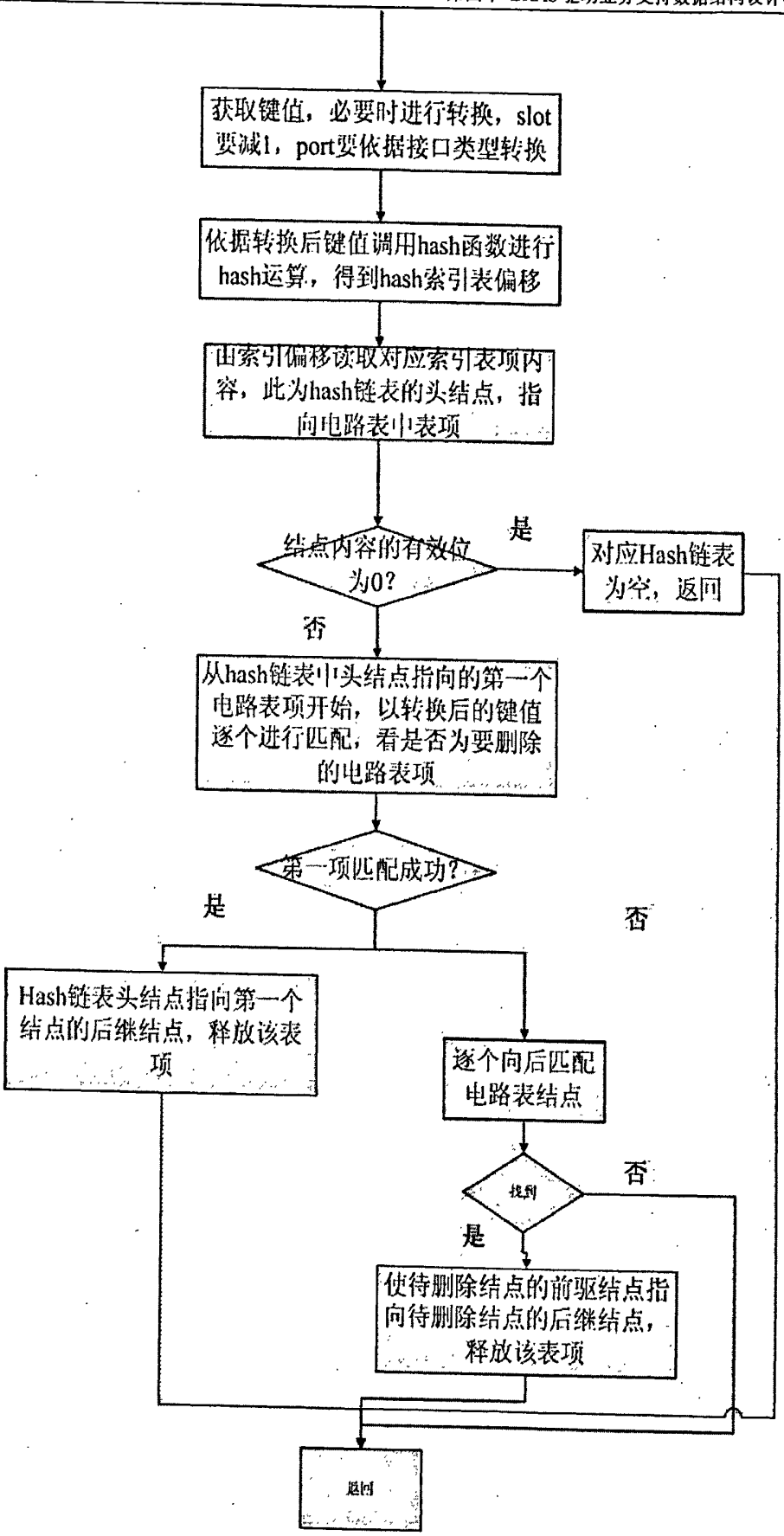


图 4-5 删除电路表项过程

## 4.2 基于 TRIE 树的路由表存储和管理

### 4.2.1 使用 TRIE 树的背景

本文在第一章课题来源中曾介绍过，BRAS 从路由器发展而来，作为接入汇聚层设备，与高端路由器一样，BRAS 也要实现依据 IP 地址的报文转发和路由表管理。随着 Internet 的发展,最初 IP 地址分配方式存在的问题逐渐暴露出来,较为突出的两个是 IP 地址空间消耗太快和路由表增长速度过快。另一方面，通信技术和计算机技术的不断发展，使得 Internet 的规模不断增大,骨干网线路的高速化,使路由器处理报文的时间要求越来越高,路由查找已成为制约路由器性能的瓶颈,其主要困难之一在于要进行最长匹配前缀[20-25]。

路由的主要任务是按照 IP 分组中的目的地址转发分组。转发分组的关键是读取报文的目标地址，在路由表中查找与之匹配的表项，最终根据该表项所指出的端口将报文转发出去。因此，路由表的组织和快速路由查找算法是实现高速分组转发的关键。

由于目前主流的 IP 地址具有 32 位，因而在进行路由表的查询时，如果以线性方式进行路由表项的存储和查询，将会导致产生不可接受的大量占用存储空间。事实上，路由表项涉及的 IP 必然存在大量的冗余，如何有效的利用大量的冗余以减少珍贵的 BRAS 驱动与网络处理器的微引擎（ME）存储空间占用是路由表项数据结构算法研究的关键。本文基于上述技术分析， 采用基于 TRIE 的数据结构和算法实现 BRAS 路由表的存储和管理。

### 4.2.2 本文使用的 TRIE 树结构结构描述

TRIE 是一种数字查找树,它来源于单词“Retrieval”,读作“Try”。TRIE 是一种有效的检索方法,它的特点是实现方法灵活,所需的存储器空间小[26]。本文基于 TRIE 数据结构和算法的路由表存储和管理总体结构[27-35],如图 4-6 所示。

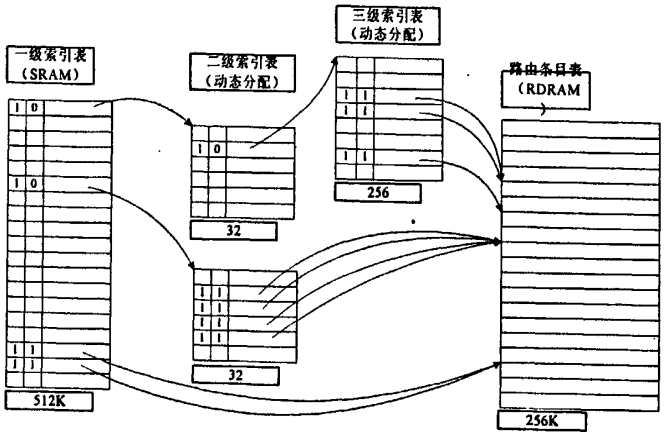


图 4-6 路由表存储结构架构

使用 TRIE 表是为了获得直接寻址的速度。对关键码范围进行均分的数据结构，可以实现动态展开和回收数组。为实现存储空间的节省，本文设计的路由索引结构通过两张 TRIE 表索引表分别进行，是依据子网掩码的长度不同而设计的两张分别具有 64K 个初始条目和 256 个初始条目的 TRIE 树结构。

Ipv4 路由表采用 TRIE 表形式的数据结构，采用索引表+ TRIE 表的方式来实现查找，执行最长匹配算法，整个路由表的结构如图 1 所示。系统采用两个 TRIE 表来覆盖 32 位的 Ipv4 地址。第一个表（Hi64K 表）用 64K 个条目作为 TRIE 表的根节点，该表用 16-4-4-4-4 的方式来构建前缀长度超过 16 位的路由条目。第二个表（Hi256 表）用 256 个条目作为 TRIE 表的根节点，该表用 8-4-4 的方式来构建前缀长度小于或等于 16 位的路由条目。索引表及 trie 中的每个条目占用两个 LWs，其结构如表 4-7 所示。

为方便描述本文的相关技术和流程，给出如下的定义：

定义 4-3：Hi64k，以 64K 个条目作为 TRIE 表的根节点的 TRIE 表。

定义 4-4：Hi256，以 256 个条目作为 TRIE 表的根节点的 TRIE 表。

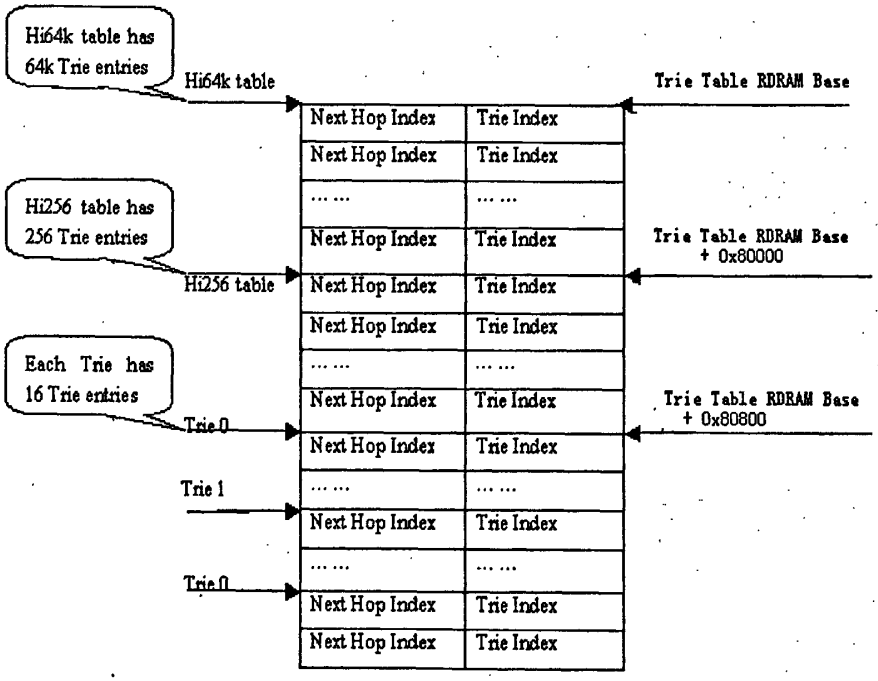


图 4-7　IPV4 路由查找表的结构示意图

表 4-3 IPV4 TRIE 查找结构

| LW | Bits | 描述 |
|---|---|---|
| 0 | 31:0 | 指向 Nexthop information 的索引 |
| 1 | 31:0 | 指向 Next Trie 表项的的索引 |

以 Hi256 表为例，其索引结构如图 4-8 所示。

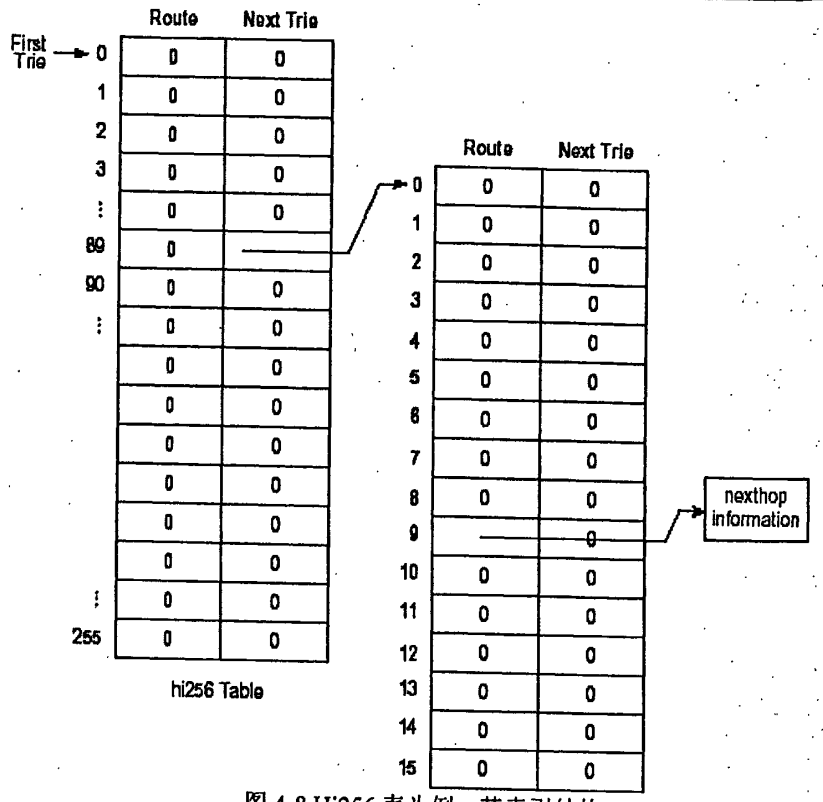


图 4-8 Hi256 表为例，其索引结构

微码的转发模块通过 TRIE 表路由查找的最终结果是返回一个下一跳表项（Nexthop Information ）的索引，该索引指向 SRAM 或 RDRAM 中的下一跳表（Next hop information Table）中的对应项。Next hop information 中包含有下一跳的接口及其链路上的重要信息，是进一步对数据包处理的依据。

为方便说明路由表的相关操作，便于流程图的说明，给出如下定义：

定义 4-3：VpnId，VPN 号。虚拟专用网(VPN)被定义为通过一个公用网络(通常是因特网)建立一个临时的、安全的连接，是一条穿过混乱的公用网络的安全、稳定的隧道。虚拟专用网是对企业内部网的扩展。

定义 4-4：mask，子网掩码。

定义 4-5：Dest_IP，目的 IP 地址。

定义 4-6：vrf，虚拟路由。虚拟路由在逻辑上，可以看做是一个独立的路由器，并有独立的路由列表。

定义 4-7：M，子网掩码长度计数吗，每 4BIT M 值增 1。

定义 4-8：N，路由条目对应起始查找表后挂接的 trieblock 的个数。

定义 4-9：trieblock，TRIE 上子树上，二级或三级索引表的一个分支根节点块，是一段连续的存储空间，每个 tireblock 有 16 个元素。

定义 4-10：Tire_ptrs 数组，各级的路由指针及 trie 指针在逐级向下搜索时都保存在这

个数组中，用于在后面逆向逐级向上填入 trie 指针及路由指针。

定义 4-11：Mask_field，mask 最低位起第一个不为 0 的 4 元组掩码段。

定义 4-12：Dest_field，目的 IP 四元组。

掩码字段 mask 中各元素指明了 Hi256 表中各表项所采用的掩码。在增加路由条目时，如果新增路由的掩码大于原掩码，才可用新路由覆盖 Hi256 中所指路由。由于 Hi256 表仅保存了 8bit 索引子树的 256 个叶子，因此在新增路由时，完全有可能发生一个长的前缀条目将一个已存在的短的前缀条目覆盖掉。若简单覆盖掉，在以后删除该最长匹配条目时，同时也将该 IP 地址的上一级较短匹配删除掉了，将导致该路由不可达。因此，每当新增路由时，都会将对应于本次新增掩码长度的路由条目指针存入相应的 7 位前缀路由条目指针、6 位前缀路由条目指针等不同的掩码段数组中。这样当删除某路由条目从而删除该路由索引时，可以将次最长掩码所对应的路由条目指针恢复到 Hi256 表中。

### 4.2.3 路由表项添加流程设计

(1) 当驱动模块收到上层业务的添加路由消息，首先须解析出相关参数信息，如 VPN 号（VpnId），目的 IP 地址（Dest_IP），子网掩码等信息。若 VpnId 不为 0，则表明需要添加的虚拟路由 (Vrf)，VpnId 为 0，则进入一般的添加路由流程。

(2) 查看目的 IP 地址（Dest_IP）和子网掩码（mask），若 Dest_IP 和 mask 都为 0，则要添加的是一条默认路由，否则继续一般的添加路由流程。

(3) 确定子网掩码（mask）4 元组个数：对 mask 从最低位开始，4 位一组进行扫描，确定全 0 的 4 元组个数 M，从而确定起始查找表。依据 M 值的不同，确定不同的起始查找表和需要的 trieblock 的个数 N，详细的分类方法见图 4-8。

(4) N 的值表示了路由条目对应起始查找表后挂接的 trieblock 的个数，循环逐次查找 trieblock,每次然后 N-1,直至 N=0 为止。其中任何一次，如果没有挂接 trieblock 则新分配一个 trieblock。然后将新查到的已有的或者新分配的 trieblock 的索引倒序写入一个数组中。

(5) 写最后一级 trieblock 或者 Hi256 表，填充路由条目指针，并且写路由表。最后，完成各个 trieblock 的级联。

上述过程如图 4-9 所示。

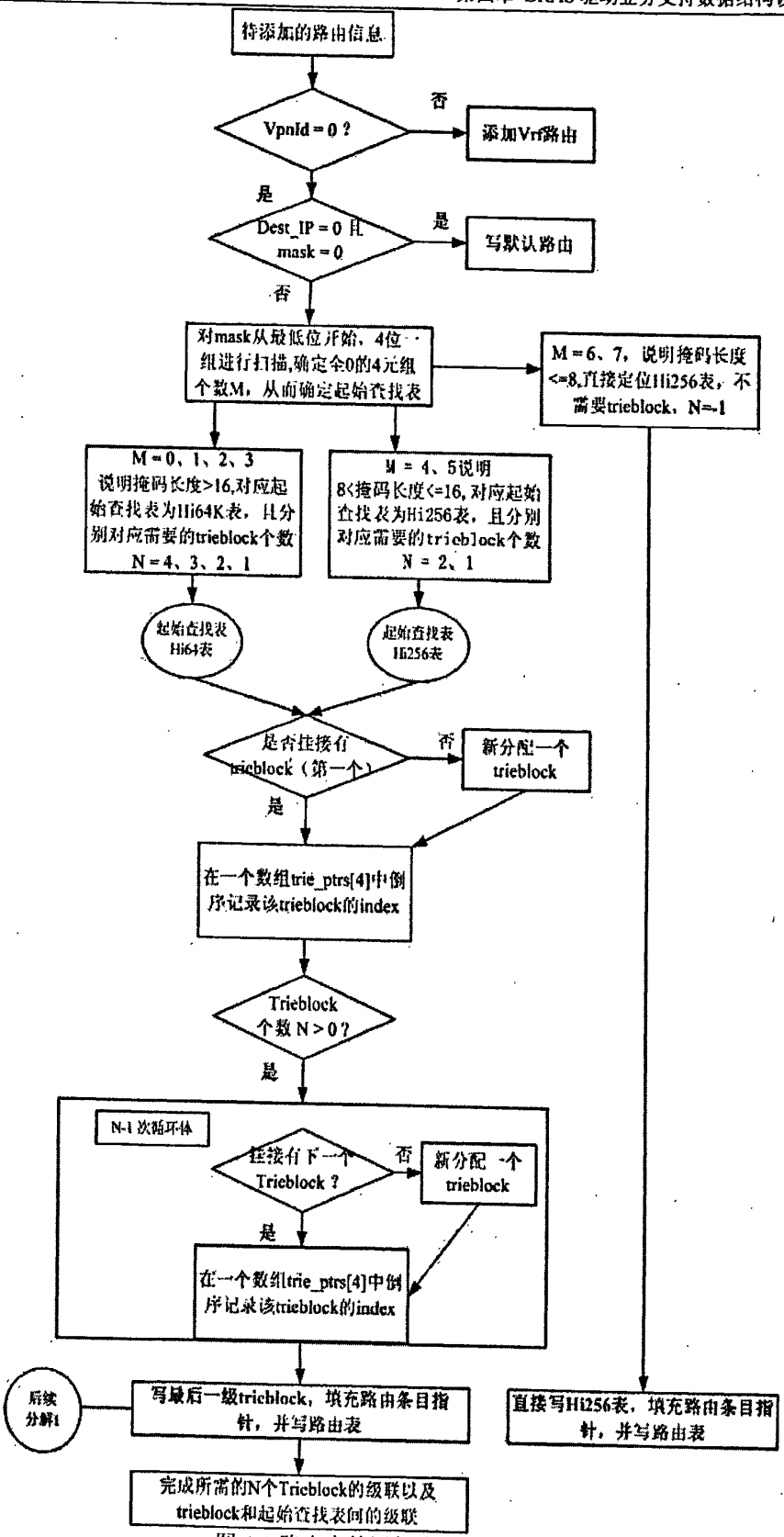


图 4-9 路由表的添加流程（1）

(6) 根据掩码长度决定本次路由增加所影响到的 trie 条目，即由 IP 地址及掩码所覆盖的所有可能 IP 地址。如 0xF 只影响到一条条目，0xE 将影响到两条条目，0xA 将影响到四条条目，0x8 将影响到八条条目。然后对所有影响到的条目进行检查写入，若条目所对应

的掩码小于本次新增掩码，则说明本次新增路由为最长匹配，则用新掩码覆盖原掩码，用新路由覆盖原路由；

(7) 将新路由存入由路由条目所指路由表的相应记录中；

(8) 现在逆向逐级向上填入 trie 指针及路由指针，各级的路由指针及 trie 指针在逐级向下搜索时都已保存起来，这时只需填入即可，最后填入 64K_table 或 256_table 对应 trie 条目内容。逆向填入可确保所有下级条目均已正确设置，确保并发路由查找可找到正确条目，转至第 10 步；

(9) 对于掩码小于 8 位的新增路由，直接在 256_table 中填入新增路由信息，此处的填写要考虑到对相关 IP 地址的覆盖影响；

(10)路由增加完毕。

上述过程如图 4-10 和图 4-11 所示。

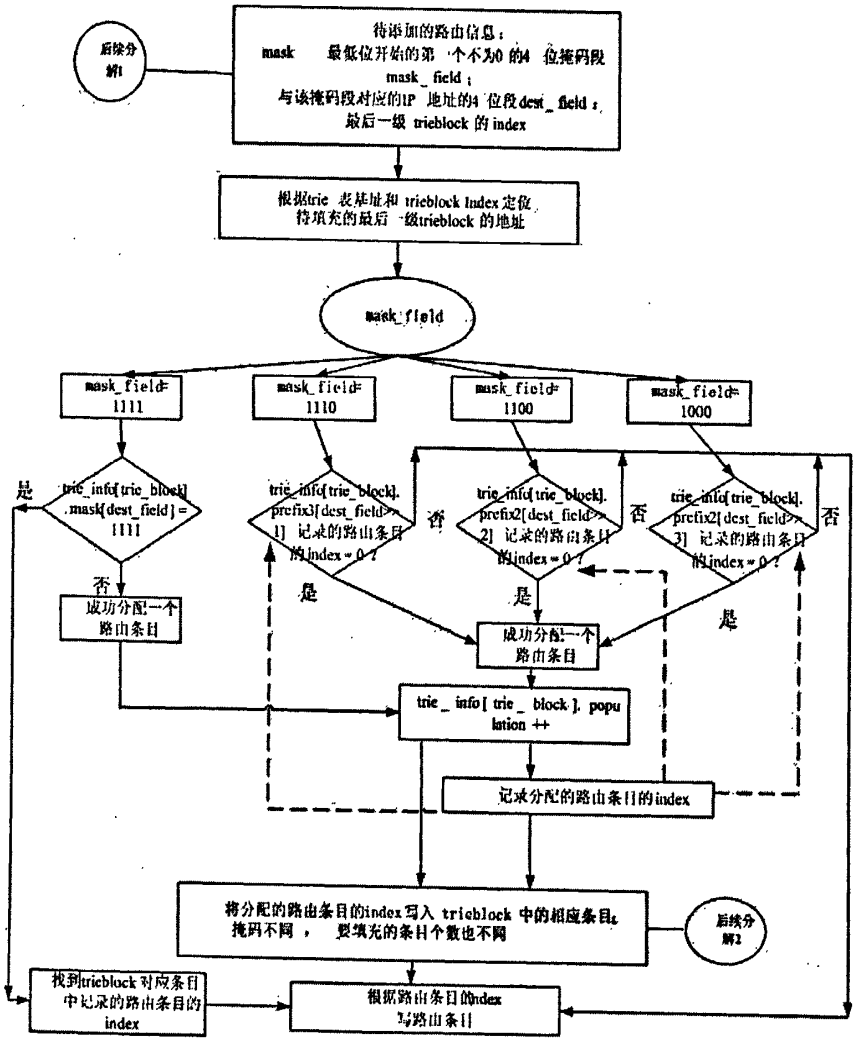


图 4-10 路由表的添加流程（2）

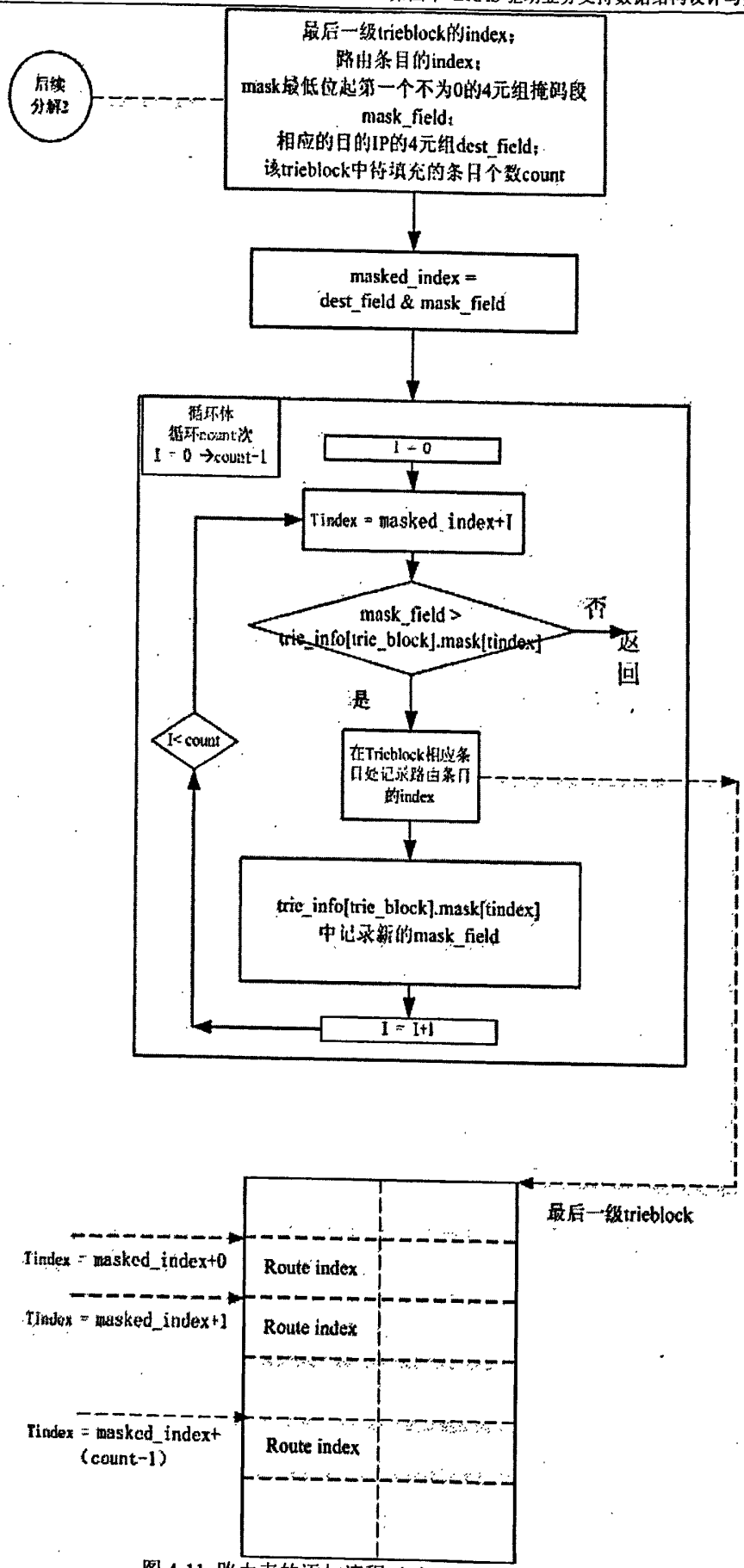


图 4-11 路由表的添加流程（3）

### 4.2.4 路由表项查找流程设计

通过以上的路由增加方法建立起路由表后,路由查找过程就非常简洁和容易。

对 Hi64k 及 Hi256 进行并行查找，循环直至 TRIE 条目中的 TRIE 指针为零，即无下级 TRIE 元素，也就是找到叶子结点。然后从叶子结点中提取出路由指针，若长路由指针(即由 Hi64k 出发所查到的路由指针存在，则它就是需匹配的路由条目，若长路由指针不存在，则短路由指针所指路由条目就是需匹配的路由条目。

### 4.2.5 路由表项删除流程设计

每个 trie block 有 16 个叶子，由 4 位 IP 目的地址索引。在一个 trie block 中，最多有 16+8+4+2 种可能的匹配。如果所有的路由组合都加入该 trie block 中，即加入 4 位掩码的 16 条路由、3 位掩码的 8 条路由、2 位掩码的 4 条路由、1 位掩码的 2 条路由，则将有 14 条隐藏条目。因此当任何一个较长掩码条目被删除，应当使用次最长掩码条目去代替它。在路由增加中将所有的隐藏条目都保存在 trie block 对应的附加信息结构的 1-3 位前缀路由条目指针中。对于 Hi256 表，也采用同样方法，其隐藏条目存储在 1-7 位前缀路由条目指针中。

在删除路由表条目的同时，必须删除相应的 trie 条目，并且找出次最长掩码条目，并将它的路由指针插入到 trie block 条目中。

每个 trie block 维护一个条目数，对 trie 指针和路由条目指针进行计数。当其中任一个增加时，均递加该计数，当任一个被删除时，均递减该计数。当该计数为 0 时，说明该 trie block 已无可用条目，则将该 trie block 回收入 trie block 空闲队列中。

路由条目删除后，也将该路由条目回收入空闲路由条目队列中。

详细的路由表删除流程如图 4-12 和图 4-13 所示。

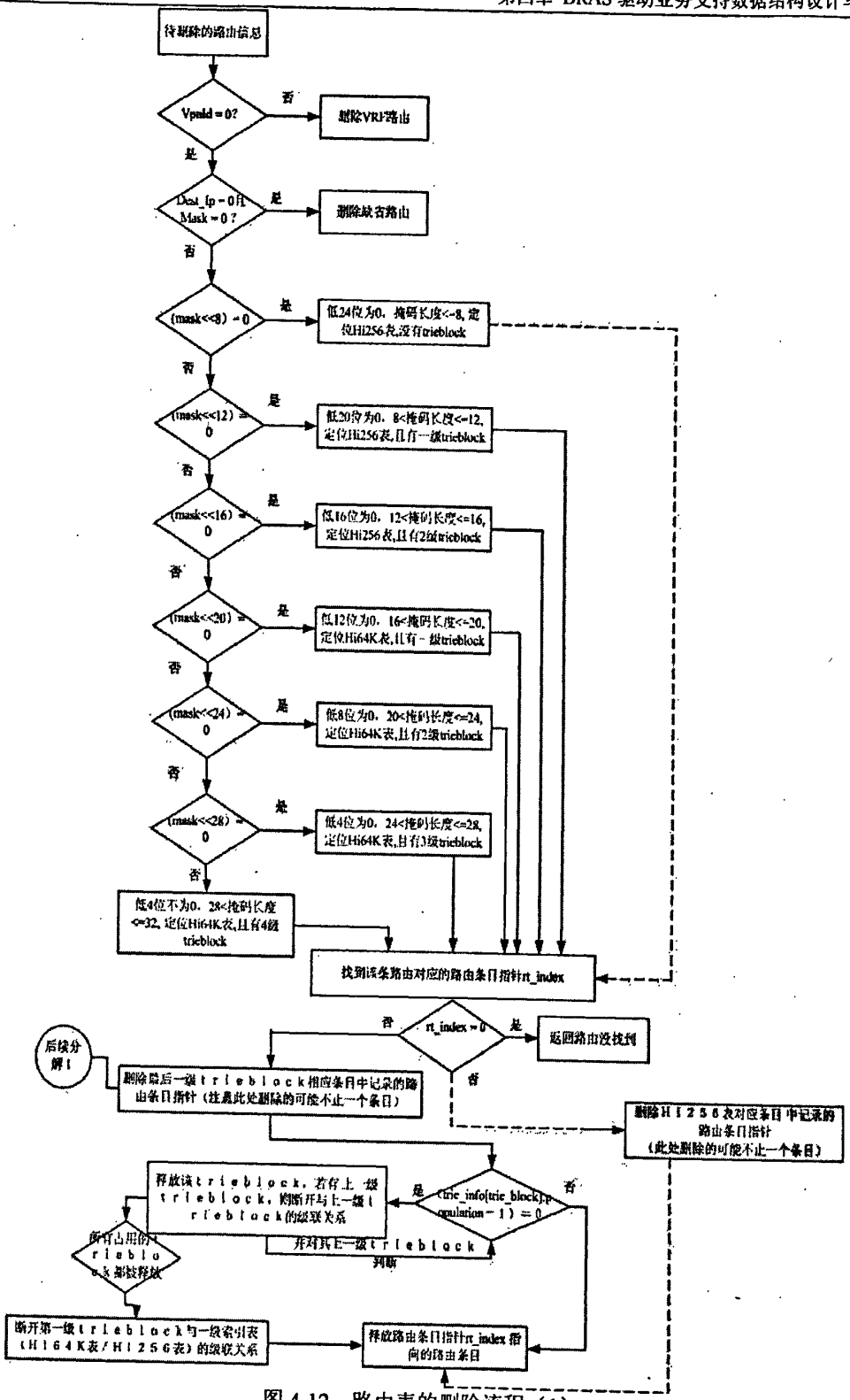


图 4-12　路由表的删除流程（1）

图 4-13 主要内容是详细介绍图 4-12 中删除最后一级 t r i e b l o c k 相应条目中记录的路由条目指针的流程，此处删除的可能不止一个条目。

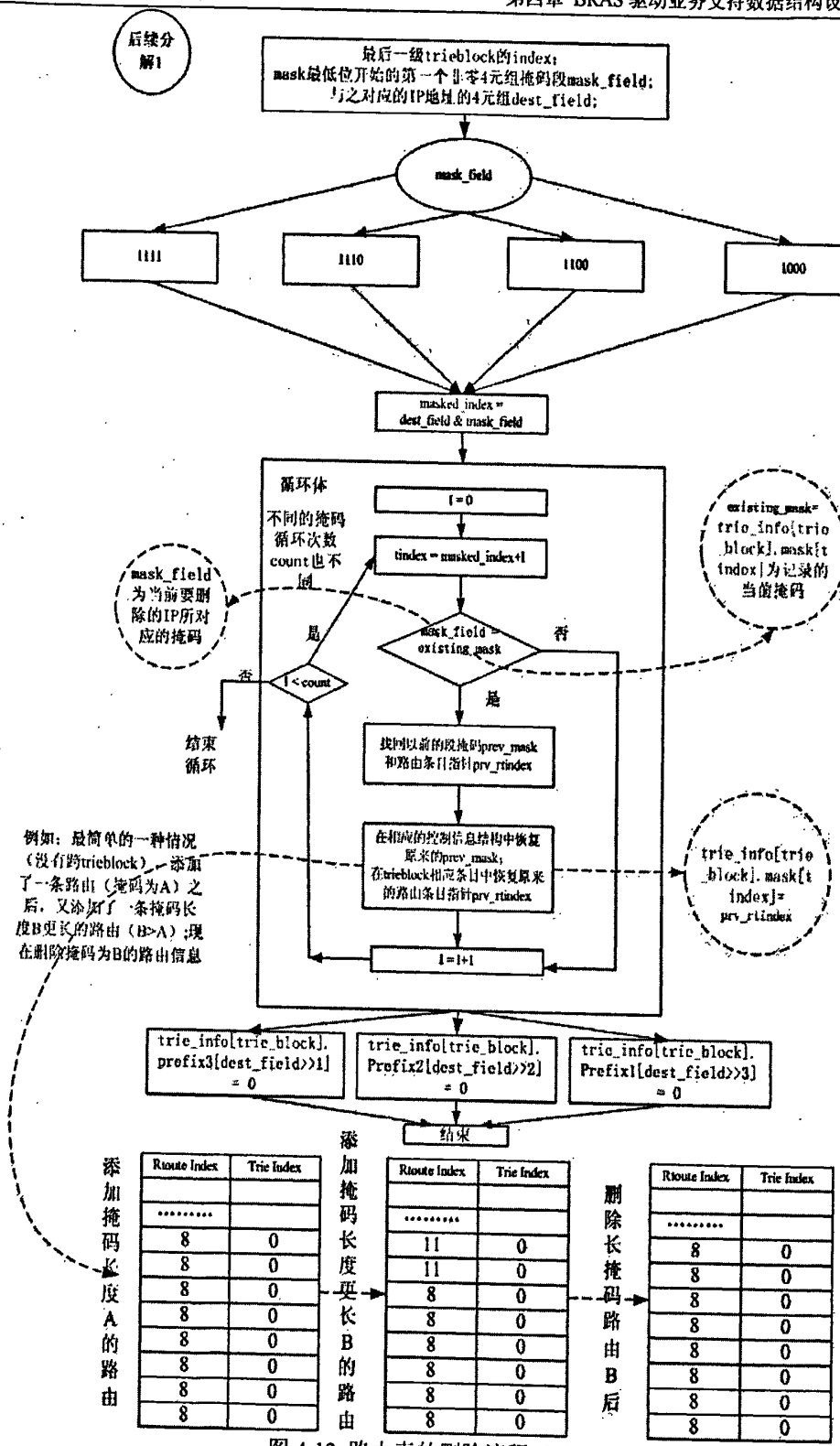


图 4-13 路由表的删除流程（2）

## 4.3 基于 RFC 算法的 ACL 流分类算法设计与实现

RFC 算法是 Recursive Flow Classification 递归流分类算法的缩写[36]，算法的主要思想就是在存在冗余配置的前提下将 IP 报文中的分类问题转化为 IP 包头的较长字段 S 比特长

度映射到一个较短的 T 比特长度的空间内，一般的一个经验公式为 T=logNS，[T<<S，其中 N 为分类规则的总数，T 为最后得到的 eqID 的长度]，在这个较小的空间内使报文方便地查找到自己的流 ID 所对应的动作。

### 4.3.1 使用 RFC 算法的实际需求

在实际的网络应用过程中，当网络访问流量较大时或者需要对于网络流量进行管理控制时,就需要对网络流量进行分类，为了控制某些特定类型的流的通过或者拒绝通过以及对于特定流识别后进行特殊处理（QOS;NAT 等）需要使用访问控制列表技术-ACL 技术，而 ACL 技术的基础是对于特定的流能够进行区分和识别。按照正常思维，对于数据进行分类的最简单的方法就是查找线性表，在线性表的结果中设置报文的流 ID，即可以达到分类的目的。但是这种情况仅对于字段很少同时查找 key 值极少的情况下才有效，对于多字段的大 key 值的查找耗费内存较大,尤其是对于 IPv4 报文五元组分类时总 key 值达到了 112bit，即使排除其他因素，按照线性表的查找方式，则最少需要 2 的 112 次方条目来存放这个表结构，表的大小绝对是天文数字，因此需要考虑其它的实现方法[37-39]。

目前报文分类的算法有多种，主要包括：有向非循环图算法；交叉乘积算法；Tuple 空间搜索算法；RFC 递归流分类算法等几种主要算法，本文实现 ACL 技术使用基于 RFC 递归流分类算法。

### 4.3.2 RFC 算法实现报文分类的基本原理

为方便说明基于 RFC 实现 ACL 的原理，对讨论中设计的相关概念给出如下定义：

定义 4-13：chunk，将报文的所有流特性字段按照所属特性拆分为不同的字段，如源 IP/目的 IP，源端口/目的端口，协议，标记等，每个字段称为一个 chunk。

定义 4-14：eqID，chunk 单独相应的线性表（已经提前处理好），得到的值为各个 chunk 独立的 eqID。

RFC 算法的原理是将报文的所有流特性字段按照所属特性拆分为多个不同的 chunk 即不同的字段(源 IP/目的 IP，源端口/目的端口，协议，标记等)。这些 chunk 在第一阶段单独查找独立的线性表，得到各个 chunk 独立的 eqID。在后续阶段，如图 4-14 所示，将这些 eqID 进行递归查找，最后实现压缩目的。

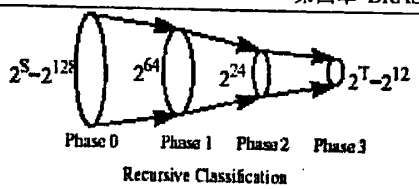


图 4-14 RFC 算法实现压缩的过程

将之前得到的各个 eqID 按照特定的规则组合，形成新的索引值，再进行内存查找从而得到新的 eqID。根据所分 chunk 的多少不同进行多次类似的递归查找，直至最后形成一个 eqID，即流 ID，这时这个流 ID 就可以查找一个较小的流分类表从而得到本流所配置的属性值，例如端口 ACL 的是否允许结果（permit 或者 deny）。

图 4-15 是一个 RFC 算法的粗略流程图。

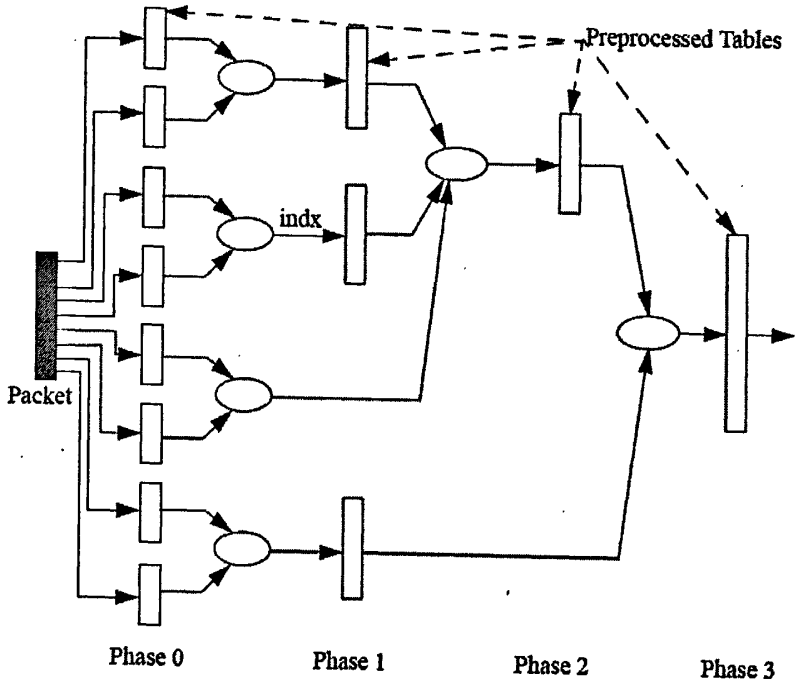


图 4-15 RFC 算法的粗略流程

### 4.3.3 RFC 算法实现压缩的原理

RFC 算法所存在的基础是存在冗余配置，RFC 算法只是将这些冗余配置进行压缩。如果不存在冗余配置，则 RFC 算法就不能实现报文的分类[40-43]。

以协议字段简单来说，假设对于协议字段（8bit）而言，如果 8bit 所代表的 256 个值的每个值都需要单独标识，即每种协议都需要有不同的策略，即需要独立的流 ID，这时的组织结构就如图 4-16 的左图所示，即对于 chunk 字段（协议字段）的每个值都需要对应一个流 ID，这样就不能实现从 S 到 T 压缩的目的，即 T＝S。但是因为在实际中所配置的规则一般只是对于特定的流进行区分识别，即不会对于每个字段的所有情况都进行配置。例如，只配置区分 ICMP 和 TCP 两种报文类型，这样就得到了图 4-16 的右图，即对于 ICMP

协议得到的 eqID 值为 1，对于 TCP 协议得到的 eqID 值为 2，对于其他的协议类型得到的 eqID 的结果都是 0，这样得到的结果就只有三种情况：0，1，2 使用 2bit 的编码就可以表示，这样就实现了从 S（8bit）到 T（2bit）的压缩映射。

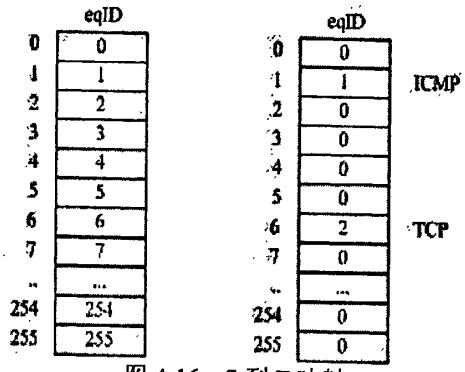


图 4-16 S 到 T 映射

关于这样是否会存在配置极端的情况而使得 RFC 算法失去意义：从理论上说这个问题是存在的，即如果需要对于每种存在的流都分配流 ID 就可能出现最后得到的 T 不远远小于 S 的情况。但是，因为 BRAS 的 ACL 规则一般都是人工配置，大量的配置除了造成维护人员的工作量增加外对于报文控制的准确性也是大有影响的，因此在实际的应用中，极少有超过 1000 条规则的使用，图 4-16 是一份调查得到的图表，从中可见基本几种在 0-100 条的规则之内。这样的话一般只需要使用较少的 bit 即可标识出常用的流 ID 的情况，从而实现从 S 到 T 的压缩映射。

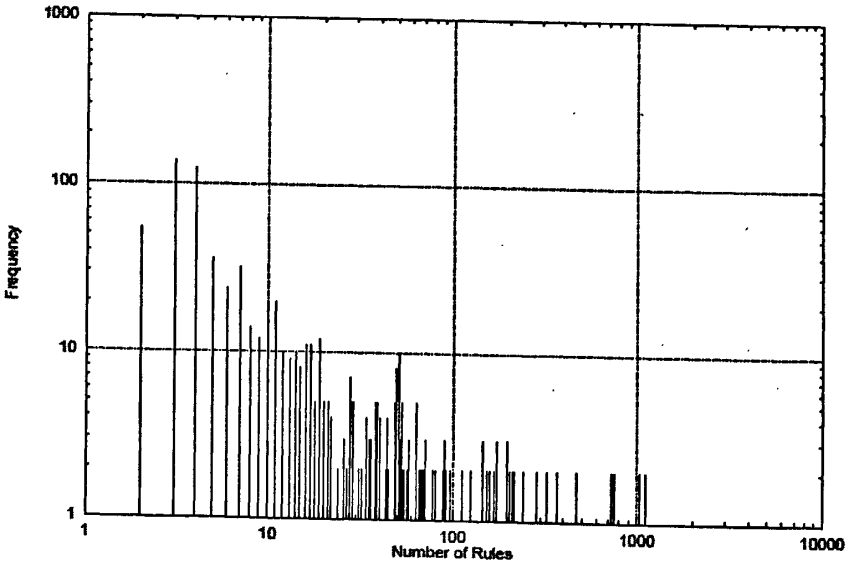


图 4-17 使用规则条数的调查统计

### 4.3.4 基于 RFC 算法的 ACL 实例

以下面的规则为例进行分析：

表 4-6 示例实现规则

| Rule# | Chunk#0 (Src L3 bits 31..16) | Chunk#1 (Src L3 bits 15..0) | Chunk#2 (Dst L3 bits 31..16) | Chunk#3 (Dst L3 bits 15..0) | Chunk#4 (L4 protocol) [8 bits] | Chunk#5 (Dstn L4) [16 bits] |
|---|---|---|---|---|---|---|
| R0 | 0.83/0.0 | 0.77/0.0 | 0.0/0.0 | 4.6/0.0 | udp (17) | * |
| R1 | 0.83/0.0 | 1.0/0.255 | 0.0/0.0 | 4.6/0.0 | udp | range 20 30 |
| R2 | 0.83/0.0 | 0.77/0.0 | 0.0/255.255 | 0.0/255.255 | * | 21 |
| R3 | 0.0/255.255 | 0.0/255.255 | 0.0/255.255 | 0.0/255.255 | * | 21 |
| R4 | 0.0/255.255 | 0.0/255.255 | 0.0/255.255 | 0.0/255.255 | * | * |

1）第一步：协议字段的压缩

上层根据配置情况为各自的 chunk 生成对应的 eqID 和 CBM[class bitmap]值。其中 eqID 是用于区分本 chunk 内部根据配置得到的不同分类情况，每个 eqID 对应一个范围或一个确定的值，如上面例子所述，如果配置识别 ICMP 和 TCP 协议则得到的 eqID 的值为 3 个，即可用 2bit 来表示出来。而 CBM 是描述各个配置的规则流对于每种 eqID 的匹配情况，如果有一条规则匹配到本 eqID 中，则 CBM 的对应位就为 1，否则为 0。CBM 的位数的长度等于配置的流的规则的长度。最后生成的格式如图 4-18 所示。

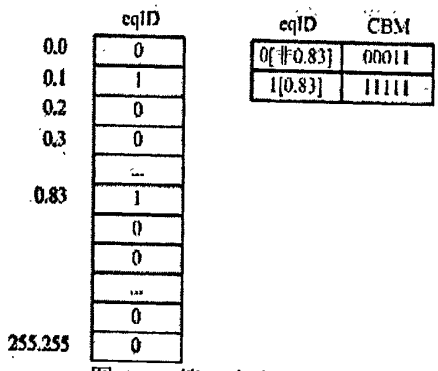


图 4-18 第一步生成格式

因为在 chunk1 中只存在两种情况，0.83/16 的情况和其他情况，因此得到的 eqID 只有两种情况：0 和 1。由于为 5 条流，则 CBM 的长度为 5，对于 eqID 为 0 的情况，即非 0.83 的情况，规则 1.2.3 均不符合，因此生成的 CBM 值为 00011，对于 eqID 为 1 的情况，即 0.83 的情况，前三条流是一定符合的，后两条流因为配置的是任意值，因此也是可以匹配到 0.83 中的，因此生长的 CBM 值为 11111。因为配置相当于定义了不同种类的流，只要不违反流的规则就认为是符合流的要求。即 CBM 每个规则所代表的流中如果可以匹配到 eqID 所标识的特性中则可以认为 CBM 的对应 bit 为 1。注意要将 eqID 和 CBM 所代表的值区分开来，eqID 是互斥的，即不同的 eqID 所对应的范围是不同的，不能够兼容的，但是 CBM 中每个 bit 所代表的规则是一个特定的流，这个流是可以包含不同的 eqID 所定义的范围的，即如果一条规则为 permit any，则所有 eqID 的对应位都为 1，因为这样定义的流可能落在 eqID 的所有范围内。

同样，按照上面的方式，生成其他的 chunk 的 eqID 和 CBM 值图 4-19。

chunk1的值

| eqID | CBM |
|---|---|
| 0[other] | 00011 |
| 1[0.77] | 10111 |
| 1[1.0] | 01011 |

chunk2的值

| eqID | CBM |
|---|---|
| 0[0.0/16] | 11111 |
| 1[other] | 00111 |

chunk3的值

| eqID | CBM |
|---|---|
| 0[other] | 00111 |
| 1[4.6] | 11111 |

chunk4的值

| eqID | CBM |
|---|---|
| 0[other] | 00111 |
| 1[17] | 11111 |

chunk5的值

| eqID | CBM |
|---|---|
| 0[other] | 10001 |
| 1[20-30除21] | 11001 |
| 2[21] | 11111 |

图 4-19 其他的 chunk 的 eqID 和 CBM 值

到这里完成了第一阶段的 eqID 的获取，前面是线性查找得到 eqID 实现第一步的冗余压缩，后面要继续进行 eqID 的冗余压缩，也就是 RFC 算法的精髓即开始递归计算。因为例子中规则较少，所以每个 chunk 中得到的 eqID 的长度也较小，但是实际中可能每个 chunk 中得到 5bit 的信息，则 6 个 chunk 得到的信息仍然为 30bit，如果按照线性查找则仍然是一个不可实现的数目，因此需要继续进行压缩。

eqID 能够实现压缩的原因：原理与起始时的压缩是一样的，还是以协议字段的压缩为例，如果配置 255 条协议，则得到的 eqID 还是不能被压下来，只是在不将该字段配满的情况下才存在信息冗余，因此才能实现压缩。同样，对于两个 eqID，以协议和端口为例：三种协议 TCP，UDP，ICMP 和三个端口 1，2，3，所能组合出的情况最多有 9 种，即三种协议分别与三个端口都需要得到对应的流 ID，但是在实际中不不会出现将各个 chunk 字段的所有元素完全进行排列组合配置的情况，这样就存在压缩空间。例如只配置了 TCP 和端口 1 和端口 2 的规则；只配置了 UDP 协议和端口 1 和端口 3 的规则；只配置了 ICMP 协议和端口 2 和端口 3 的规则，这样原来的 9 种情况就被压缩到了 6 种情况。

2） 第二步：eqID 的压缩

RFC 算法的第二步是实现 eqID 的压缩，可以将多个 chunk 的 eqID 同时进行压缩，

这样的优点是节省访问内存的次数，缺点是耗费的内存较大。因此一般采用两个 chunk 或者三个 chunk 的方式进行压缩计算。

压缩时的计算方法为：index = a * Nb * Nc + b * Nc + c[三个 chunk 进行压缩]

Index＝a* Nb + b[两个 chunk 进行压缩]

其中，Nb 表示 eqID 的数目，a 表示第一个合并表的 eqID 序号[从 0 到 eqID 的序号]，同样 b 表示另一个合并表的 eqID 序号。得到的结果是新生成的 eqID 表的 index 值，具体的 CBM 内容值为两个或者三个原来 entry 的结果相与之后的结果。之所以使用将 CBM 值相与的运算就是为了完成多个 chunk 的合并，即使得符合配置的同一条规则的所有字段都匹配才能匹配到一条流 ID 上。

以下面实例为例：

例如：需要将源 IP 的高 16bit 和目的 IP 的高 16bit 合并，如下：

Chunk0：

| EqId | CBM |
|---|---|
| 0 | 10000 |
| 1 | 11001 |
| 2 | 10100 |
| 3 | 10010 |

Chunk2：

| EqId | CBM |
|---|---|
| 0 | 10000 |
| 1 | 11001 |
| 2 | 10100 |
| 3 | 10010 |

则计算公式为：Index=chunk2 的大小[4]*C2[0-3 的不同取值]+C0[0-3 的不同取值]，每个 index 得到的结果为将对应的两个 entry 的 CBM 相与之后的结果如下：

表 4-7 entry 的 CBM 相与之后的结果

| | C0[0] | C0[1] | C0[2] | C0[3] |
|---|---|---|---|---|
| C2[0] | 0. 10000 | 1. 10000 | 2. 10000 | 3. 10000 |
| C2[1] | 4. 10000 | 5. 11001 | 6. 10000 | 7. 10000 |
| C2[2] | 8. 10000 | 9. 10000 | 10. 10100 | 11. 10000 |
| C2[3] | 12. 10000 | 13. 10000 | 14. 10000 | 15. 10010 |

依次计算，最后将所有相同的结果进行合并处理，得到

表 4-8 eqID 合并压缩处理结果

| eqID | CBM |
|---|---|
| 0 | 10000 |
| 1 | 11001 |
| 2 | 10100 |
| 3 | 10010 |

即完成了 eqID 的合并和压缩处理。按照这种方法进行压缩到最后得到最后以及 eqID 即报文所属的流 ID。

### 4.3.5 基于 RFC 算法的驱动 ACL 对应网络处理器微码实现流程

ACL 表是一个全局表，每次重新配置 ACL 时都会影响到全局 ACL 的重新计算。因此在查询 ACL 表时首先需要查询当前的 ACL 表的全局状况表，即微码中首先查找得到的 ACL_list 表，表结构如表 4-9 所示。

表 4-9 ACL 列表

| 位置 | 长度 | 长度 | 描述 |
|---|---|---|---|
| LW0 | 31:14 | 18 | 保留 |
| | 13 | 1 | 是否需要对 phase11 的结果和 flag 字段的结果进行合并查找 |
| | 12 | 1 | 是否需要对 phase10 的结果和协议字段的结果进行合并查找 |
| | 11 | 1 | 是否需要对 phase9 的结果和目的端口的结果进行合并查找 |
| | 10 | 1 | 是否需要对 phase8 的结果和源端口的结果进行合并查找 |
| | 9 | 1 | 是否需要对目的 IP 的高 16bit 和低 16bit 结果进行合并查找 |
| | 8 | 1 | 是否需要对源 IP 的高 16bit 和低 16bit 结果进行合并查找 |
| | 7 | 1 | 是否需要对于 TCP 中的 flag[ACK/RST]进行查找 |

| | | | |
|---|---|---|---|
| | 6 | 1 | 是否需要对协议字段进行查找 |
| | 5 | 1 | 是否需要进行目的端口的查找 |
| | 4 | 1 | 是否需要进行源端口的查找 |
| | 3 | 1 | 是否需要进行目的 IP 低 16bit 的查找 |
| | 2 | 1 | 是否需要进行目的 IP 高 16bit 的查找 |
| | 1 | 1 | 是否需要进行源 IP 低 16bit 的查找 |
| | 0 | 1 | 是否需要进行源 IP 高 16bit 的查找 |
| LW1 | 31:16 | 16 | coefficient IP 源地址合并时的系数<br>实际为 log2eqID 向后取整[56 对应 5.85 因此取 6，即微码的偏移位数，下同] |
| | 15:0 | 16 | coefficient IP 目的地址合并时的系数 |
| LW2 | 31:16 | 16 | coefficient IP 源地址和源端口合并时的系数 |
| | 15:0 | 16 | coefficient IP 目的地址和目的端口合并的系数 |
| LW3 | 31:16 | 16 | coefficient pahse10 结果和协议结果合并时的系数 |
| | 15:0 | 16 | coefficient phase11 结果和 flag 合并时的系数 |
| LW4 | 31:16 | 16 | coefficient pahse12 和 phase13 结果合并时的系数 |
| | 15:0 | 16 | table14's element number 表 14 的成员个数 QW 为单位，表大小均为 QW 为单位，下同 |
| LW5 | 31:16 | 16 | table8 size 即 table 表的大小 |
| | 15:0 | 16 | table9 size table9 的大小 |
| LW6 | 31:16 | 16 | Table10 size table10 的大小 |
| | 15:0 | 16 | Table11 size table11 大小 |
| LW7 | 31:16 | 16 | Table12 size table12 的大小 |
| | 15:0 | 16 | Table13 size table13 大小 |

第一阶段：

首先查找 ACL 的 list 表，得到对于上层对于每个 chunk 的分析情况，判断具体的 chunk 是否需要进行查找[如果配置项中没有将该 chunk 进行分段则不需要查找]，在需要查找的情况下，使用本 chunk 的 key 字段查找线性表得到本 chunk 的 eqID。因为每个 entry 占用 16 个 bit，而每次读取 Sdram 得到 2 个 lw 即 4 个 entry，因此需要根据最低位和次低位得到对应的 entry。

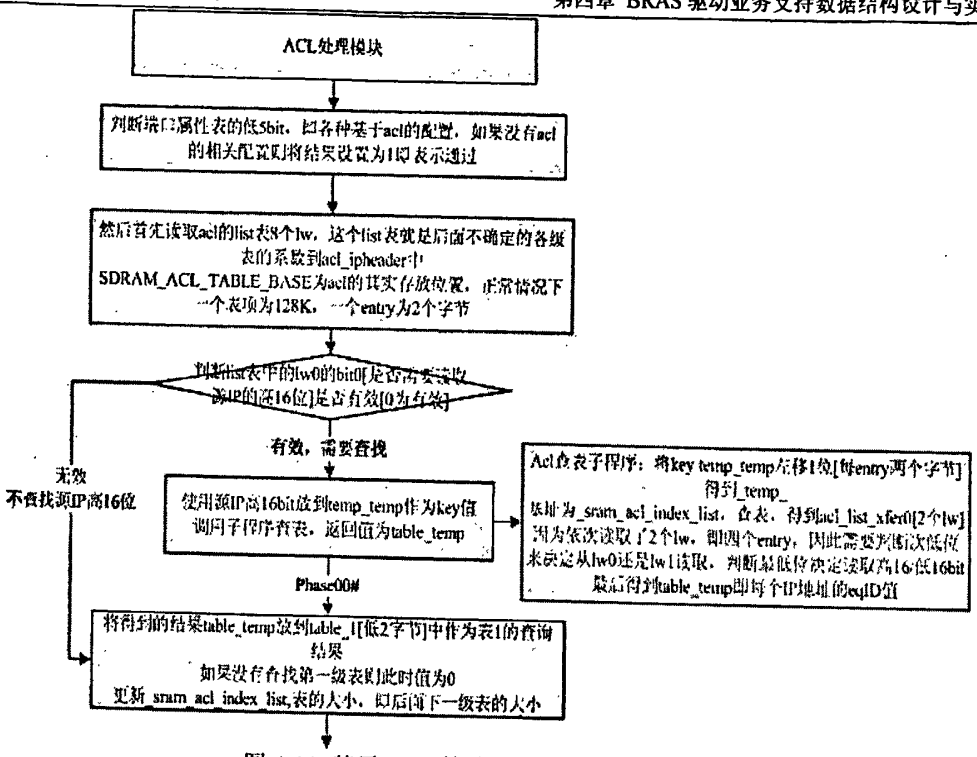


图 4-20 基于 RFC 算法的 ACL 实现第一阶段

第二阶段：

需要对于 eqID 进行合并处理，取出之前读取的 ACL_list 表中的对应系数，这些系数的含义就是待合并的两级 eqID 的后者 eqID 的大小，由上层转化为微码的移位偏移长度，便于微码进行移位处理。例如，eqID1 需要和 eqID2 进行合并，eqID1 的个数为 120 个，eqID2 的个数为 210 个，则上层经过转化为放到 ACL_LIST 表中的系数为 8（210 对应的最小的 2 的整数倍即 2 的 8 次方 256）。

经过将所有 chunk 的 eqID 进行合并以及进行递归合并最后在 TABLE14#位置处得到 phase12 和 phase13 阶段的两个 eqID，将 phase12 得到的 eqID 乘以上层所给的系数之后加上 phase13 的 eqID 就是最后的伪流 ID（本应再查一次表得到流 ID，但为了减少内存访问次数，省略了一次，因此存在冗余而称为伪流 ID）。

得到流 ID 之后，需要根据这个流 ID 进行对应的流的信息的查询，但是因为不同的 ACL 号对应不同的 ACL 应用，因此就需要多个 ACL 块，在每个块中都存放一套流 ID 的对应结果，具体组织结构如图 4-21 所示。

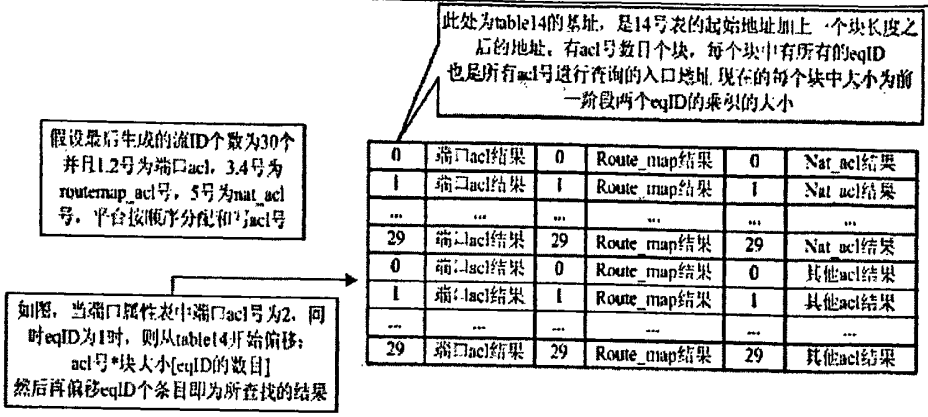


图 4-21 ACL 块组织结构图

## 4.4 实验结果分析

由于 BRAS 驱动模块在系统中处于底层，是网络处理器的服务模块，因此在系统运行过程中不能直接看到驱动模块的相关数据的状态，在 BRAS 系统的开发过程中，为方便调试和查看驱动的状态，开发了一些命令，用以显示一些驱动相关数据。本节在这些命令的基础上，调试驱动模块的相关数据，以验证本文设计的数据结构和算法是否正确运行和达到预期的功能目标。

### 4.4.1 基于 HASH 数据结构的相关业务数据结构存储显示

#### 4.4.1.1 验证驱动下发给微码的各个表项数据结构基址

在驱动模块初始化过程中，驱动模块将分配的各个业务表项数据结构的存储基址作为符号常量下发给了微引擎，以使网络处理器的微引擎在处理报文时可以找到对应的表项进行查询。IXP2800 共有 16 个微引擎（ME），其中主和从上各有 8 个微引擎，驱动须将各个表项基址下发给每个微引擎，因此需指明要查询的特定微引擎，但是实际上下发给每个微引擎的表项基址基本相同。图 4-22 显示的是主上 1 号微引擎的下发相关表项基址常量。

调试命令：npc patch-symbol　2 master 1，其中 2 表示 2 号槽位上的 NP 线卡，master 表示主 IXP2800，1 表示 1 号微引擎（ME）。

```
ZXR10(diag)#npc patch-symbol 2 master 1

The follow symbols are patched for MicroEngine          1
**********************************************
QARRAY_INIT_SRAM_BASE_CH1 : 0x00000000
QARRAY_INIT_SRAM_BASE_CH0 : 0x00000000
MULTICAST_COUNTERS_BASE : 0x00567320
MULTICAST_FWDER_TABLE_BASE : 0x02f30040
MULTICAST_ROUTE_COLLISION_TABLE_BASE : 0x004e5320
MULTICAST_ROUTE_HASH_TABLE_BASE : 0x00465320
```

```
SRAM_CAR_MAC_HASH_TABLE : 0x00463320
SRAM_CAR_POLICY_TABLE_BASE : 0x0045b320
SRAM_CAR_LIST_TABLE_BASE : 0x0045a320
SRAM_LOADBALANCE_WEIGHT_BASE : 0x00446320
SDRAM_LOADBALANCE_WEIGHT_BASE : 0x00000000
SRAM_RTMAP_BASE : 0x0043e320
SRAM_META2_BASE : 0x0040c320
SRAM_ACL_INDEX_LIST : 0x0040c300
SDRAM_ACL_TABLE_BASE : 0x02530040
SRAM_ACL_INDEX_LIST : 0x0040c300
SDRAM_ACL_TABLE_BASE : 0x02530040
SRAM_PORT_ATTRIBUTE_INFO_BASE : 0x003cb9fc
GLBPORT_LINK_LIST_BASE : 0x003c39fc
GLBPORT_INDEX_TABLE_BASE : 0x003c29fc
SRAM_PORT_ATTRIBUTE_INDEX_BASE : 0x003a29fc
IPV6_L3_NEXTHOP_TABLE_SRAM_BASE : 0x40000500
IPV6_TRIE_TABLE_SRAM_BASE : 0x40020510
INGRESS_TX_MESSAGE_BASE : 0x003a29f8
PACKET_TX_COUNTER_BASE : 0x003a2838
PACKET_COUNTERS_SRAM_BASE : 0x003a2678
SRAM_ZERO_BLOCK : 0x003a2658
SCHED_WEIGHT_CREDIT_BASE : 0x003a1558
QM_DROP_QUEUE_ENTRY : 0x00000020
QD_TOTAL : 0x00000400
QD_SRAM_BASE : 0x0039d550
IPV6_ROUTER_PREFIX_SRAM_BASE : 0x0039d520
THIS_BLADE_ID : 0x00000002
NEXTHOP_TABLE_SRAM_BASE : 0x0037d520
SRAM_INTERNAL_LABEL_64k_BASE_ADDR : 0x0037d520
SRAM_EXTERNAL_LABEL_64k_BASE_ADDR : 0x0035d520
THIS_BLADE_ID : 0x00000002
CONTROL_BLOCK_SRAM_BASE : 0x01530040
DBCAST_TABLE_SRAM_BASE : 0x01530040
NEXTHOP_TABLE_SDRAM_BASE : 0x01530040
DEFAULT_RT_SRAM_BASE : 0x00159520
VRF_LKUP_TABLE_SRAM_BASE : 0x000d9520
RT_LKUP_TABLE_SRAM_BASE : 0x00058d20
TRIE_TABLE_SRAM_BASE : 0x0015d520
IPV4_STATS_TABLE_BASE : 0x00000600
TUNNEL_INGRESS_LIST_SRAM_BASE : 0x00052d20
TUNNEL_ENCAP_NEXTHOP_SRAM_BASE : 0x00056d20
TUNNEL_DECAP_NEXTHOP_SRAM_BASE : 0x00052b20
V4V6_TUNNEL_INGRESS_LIST_SRAM_BASE : 0x00042710
V4V6_TUNNEL_ENCAP_NEXTHOP_SRAM_BASE : 0x0004eb10
V4V6_TUNNEL_DECAP_NEXTHOP_SRAM_BASE : 0x00042510
DL_REL_BASE : 0x00294040
FREE_LIST_ID : 0x00000001
BUF_SRAM_BASE : 0x00008500
BUF_FREE_LIST0 : 0x00000011
There are        55 symbols are patched for MicroEngine        1
```

图 4-22 驱动下发给微引擎表项基址常量

#### 4.4.1.2 电路表 (cirTbl)

查询各种电路表情况有两个主要的命令：1）cirtbl status 显示表建立的信息；2）cirtbl search 则用于索引主和从的电路表（包括 ETH，ATM，POS 等接口）。对于以太网（ETH）没有配 VLAN 的，用 default_vlan 代替。

显示建立的电路表的相关信息：图 4-23 表示已经正确建立一个电路表，表项中只添加了一个电路表项，目前被删除的电路表项个数为 0，以及其他的信息。

```
ZXUAS(diag)#npc cirtbl status 1

                                        Circuit Table Stat
Circuit Table Total    = 1
Circuit Table  add or update  total    = 1
Circuit Table  add or update Ok total  = 1
Circuit Table  delete total    = 0
Circuit Table  delete Ok total  = 0
Circuit Table  alloc  mainTabl number = 1
```

图 4-23 建立的电路表的相关信息

命令显示指定参数的电路表项内容：在 4.1 节已经介绍过，电路表的 HASH 参数选取的是槽位号（slotId），端口号（portId），外层（outVlanVpi），内层（inVlanVci），下面以这四个参数查询相应的电路表项的状态，如图 4-24 所示。

```
ZXUAS(diag)#npc cirtbl search 5 ma 1 eth de de

MASTER CIRTBL_ADDRESS                                  :   0x90100540
-------------------------------------------------------------------
CIRTBL_ATTRIBUTE_SLOT                                  :   0x4
CIRTBL_ATTRIBUTE_PORT                                  :   0xf
CIRTBL_ATTRIBUTE_OUTVLAN(VPI)                          :   0xffff
CIRTBL_ATTRIBUTE_INVLAN(VCI)                           :   0xffff
CIRTBL_ATTRIBUTE_GPORT                                 :   0x5010000
CIRTBL_ATTRIBUTE_VIR_FLAG                              :   0x0
CIRTBL_ATTRIBUTE_SMARTGROUP_GPORT                      :   0x0
CIRTBL_ATTRIBUTE_INTF_INDEX                            :   0x1
CIRTBL_ATTRIBUTE_CIR_RX_FRAME                          :   0x0
CIRTBL_ATTRIBUTE_CIR_RX_BYTES                          :   0x0
```

图 4-24 指定参数的电路表项内容

关于图 4-23 显示数据的几点说明：1）微码以及驱动的槽位号和端口号数据是由上层业务下发的槽位号和端口号处理后得到的，具体关系是：驱动的槽位号=上层下发槽位号-1，驱动的端口号=16-上层的端口号。因此，入参 slotId=5,portId=1,实验结果显示为 CIRTBL_ATTRIBUTE_SLOT=0x4, CIRTBL_ATTRIBUTE_PORT=0xf (15)。2）vlan 的参数选择，有 vlan 的要填 vlan 号，没有配置的 vlan 即默认 vlan 为 0xffff。

经过上述实验证明，基于 HASH 算法的电路表工作正常，运行结果正确。

4.4.1.3 接口表诊断 (interface)

接口表用于查询接口属性：status 命令显示表建立的信息，search 命令则用于索引主从接口表。如果查实接口的接口表时， 子接口号为 0 。

显示接口表建立的信息：图 4-25 显示，当前接口表已有总共 16 个表项，都是正常添加的，目前没有任何接口表项被删除。

```
ZXUAS(diag)#npc int status 5

                                        Interface Table Stat
Interface Table Total    = 16
Interface Table  add or update  total    = 16
Interface Table  add or update Ok total  = 16
Interface Table  delete total    = 0
Interface Table  delete Ok total  = 0
Interface Table  alloc  mainTabl number = 16
```

图 4-25 接口表建立的信息

显示特定接口表项内容：使用 interface 命令，其中 5 表示槽位号，10 表示 5 号槽位 NP 线卡的 10 号端口。该表项内容显示，接口没有配置 VLAN（内外层 VLAN 均为 0xffff），全局端口号为 0x5010000（由端口号，槽位号，内外层 VLAN 组合而得），以及其他的信息。

```
ZXUAS(diag)#npc int search 5 ma 1 0

MASTER INTFTBL_ADDRESS                              :   0x90510600
-------------------------------------------------------------------
INTFTBL_ATTRIBUTE_OUTVLAN(VPI)                      :   0xffff
INTFTBL_ATTRIBUTE_INVLAN(VCI)                       :   0xffff
INTFTBL_ATTRIBUTE_GPORT                             :   0x5010000
INTFTBL_ATTRIBUTE_PARENT_GPORT                      :   0x0
-------------------------------------------------------------------
INTFTBL_ATTRIBUTE_UNI_NNI_BIT                       :   0x1
INTFTBL_ATTRIBUTE_VPLS_BIT                          :   0x0
INTFTBL_ATTRIBUTE_VPWS_BIT                          :   0x0
INTFTBL_ATTRIBUTE_VRF_BIT                           :   0x0
INTFTBL_ATTRIBUTE_MULTICAST_BIT                     :   0x0
INTFTBL_ATTRIBUTE_PORTACL_BIT                       :   0x0
INTFTBL_ATTRIBUTE_ROUTEMAP_ACL_BIT                  :   0x0
```

图 4-26 接口表项内容显示

上述实验证明，基于 HASH 算法的接口表工作正常，运行结果正确。

4.4.1.4 用户表诊断 (usrtbl)

显示主从用户表信息， 在查询用户表时， 需先选择用户类型，在输入相应参数。

显示用户表建立信息：使用 usrtbl 命令，图 4-27 显示当前用户表没有任何用户表项，即当前没有任何用户通过 BRAS 进行接入。

```
ZXUAS(diag)#npc usrtbl status 5

                                        User Table Stat
User Table Total    = 0
User Table  add or update  total    = 0
User Table  add or update Ok total   = 0
User Table  delete total    = 0
User Table  delete Ok total   = 0
User Table  alloc  mailTable num = 0
User Table  add Entry But not New Enty = 0
User Table  del  Entry But not Find = 0
```

图 4-27 显示用户表建立信息

由于用户有多种方式实现接入，而不同的接入方式由于特性不同，在建立和查找 HASH 表时使用了不同的 HASH 计算参数，比如，PPPOE 或 PPPOEOA 的接入方式使用会话号（sessionId）作为 HASH 参数，而 IPOE 或 IPOEOA 接入方式则以虚拟路由号(vrfId),和用户的 IP 地址（userIpAddr）作为关键字进行 HASH 计算。

在查询特定用户表项时，要指定接入类型，以使 BRAS 知道以何种方式进行 HASH 运算和查找用户表，图 4-28 上半部分在需输入接入类型的地方使用了？（？是 BRAS 系统开发时，为方便输入使用的提示系统），随后列出了 BRAS 具有的接入方式，和各种方式的定义。图 4-28 的下半部分尝试查询一个以 PPPOE 方式接入用户号为 1 的用户，事实上在上文查询用户表状态信息时，已经知道当前没有任何用户接入，因此，此时查询必然找不到对应的用户表项，图 4-27 也印证了这一点，返回了 NO SUCH ITEMS（不存在该用户表项）。

```
ZXUAS(diag)#npc usrtbl search 1 ma ?
  ipoe        IP over-ethernet encapsulation type
  ipoeoa      IP over-ethernet over-atm encapsulation type
  pppoa       PPP over-atm encapsulation type
  pppoe       PPP over-ethernet encapsulation type
  pppoeoa     PPP over-ethernet over-atm encapsulation type
  route1483   Route1483 encapsulation type
ZXUAS(diag)#npc usrtbl search 1 ma pppoe ?
  <1-32000>  Session ID
ZXUAS(diag)#npc usrtbl search 1 ma pppoe 1

        NO SUCH  ITEMS!
```

图 4-28 查询存在的用户接入方式和尝试查询一个不存在的用户

驱动模块设计的基于 HASH 的业务数据结构有数十种（参见附录），事实上，每一种业务数据结构的存储和操作算法都基本相同，由于论文篇幅限制，这里不给出每个 HASH 相关业务表项的实验过程。通过上述实验过程，可以得出，本文设计的基于 HASH 的驱动模块相关业务可以正常的运行，实现相关数据的正确存储和维护。

### 4.4.2 基 TRIE 算法的路由表存储显示

本小节通过运行的 BRAS 系统，验证本文所设计的基于 TRIE 数据结构与算法设计的路由表存储和管理技术。

#### 4.4.2.1 显示当前路由表状态信息

使用 forwardtable 命令，显示已经存储的路由表信息。图 4-29 所示为当前 6 号槽位的 NP 线卡上的 IPv4 路由表，其中显示了 Hi64K 和 Hi256 两张 TRIE 索引表基址（64K search table addr、256 table base addr）、路由表基址（Route table base addr）、路由表容量（Route table capacity）、当前路由表条目数（Route entry number）等信息。

```
ZXR10(diag)#npc forwardtable status ipv4 6

64K search table addr     : 0x8006ca90 | 256 table base  addr     : 0x800eca90
Route table base addr     : 0x1d532880 | VPN   table base addr   : 0x800ed290
Default Route base addr   : 0x8016d290 | Trie table base  addr   : 0x80171290
Route table capacity      :    524288 | Trie  table capacity    :      69632
Route entry number        :        10 | Trie entry used number  :         17
Route entry add times     :         8 | Route entry add OK times :         8
Route entry del times     :         0 | Route entry del OK times :         0
VPN entry add times       :         3 | VPN entry add OK times   :         3
VPN entry del times       :         0 | VPN entry del OK times   :         0
Default route entry number:         0 | VPN entry number         :         3
```

图 4-29 指定槽位路由表状态信息

#### 4.4.2.2 查询指定 IP 的路由表条目

查找条目失败的情况：图 4-30 为查询一个条目失败的情况，说明不存在该输入信息对应的路由条目，其中 vrf 为可选参数（适用于三层 VPN 接入入口），用于索引 vpn 路由条目，vpn1 为 vpn 字段名，200.1.1.1 为 ip 地址，查询结果为“Have no infor for this dest IP address”，即不存在与此 IP 对应的路由表条目。

```
ZXR10(diag)#npc forwardtable search ipv4 6 vrf vpn1 200.1.1.1


Have no infor for this dest IP address
The rtmv4 hi256 search table base address is : 0x800eca90
The rtmv4 hi64k search table base address is : 0x8006ca90
The VPN search table base address is   : 0x800ed290
The route table base address is  : 0x1d532880
The Default route table base address is  : 0x8016d290
```

图 4-30 查询指定路由条目失败显示

查找条目成功的情况：图 4-31 为查询路由条目成功的实例，其中命令参数 ipv4 说明要搜索 ipv4 路由表（区别于 ipv6），6 为 vpn 号，200.1.1.1 为目的 IP 地址。查询成功，返

回了对应路由表项的地址（The route entry address），以及表项中相关参数信息（标志位 Flag ，下一跳地址：NextIP ）。

其中 NextIP 值的含义为：0Xc8010101——>0xc8.0x01.0x01.0x01——>200.1.1.1；

OutPort 含义为：0x5——>0x00,0x00,0x05——>0 号子接口（即没有子接口），16 号端口，6 号槽位（槽位转换方式：OutPort_cardNo ＋ 1；端口转换方式和接口类型有关，这里的例子是 16 口百兆接口卡，对应端口转换方式是 16 － OutPort_portnum）。

```
ZXR10(diag)#npc forwardtable search ipv4 6 200.1.1.1


The route entry address is : 0X1d5329c0
Flag            : 0X80001
NextIP          : 0Xc8010101
OutPort         : 0X5
Ext_Label       : 0X3
Int_Label       : 0X3
3rd_level_Label : 0X3
Reserved        : 0X0
Reserved        : 0X0
```

图 4-31 查询指定路由条目成功显示

4.4.2.3 显示当前 BRAS 系统中存储的所有路由条目

使用 show ip route 命令显示当前 BRAS 系统中驱动模块在 SDRAM 中已经存储的所有路由条目，如图 4-32 所示。

```
4830_44#sho ip route
Total number of routes:      49
IPv4 Routing Table:
Dest            Mask            Gw              Interface     Owner    Pri Metri
1.1.1.30        255.255.255.255 1.1.1.30        loopback30    address  0   0
1.1.1.42        255.255.255.255 44.1.1.2        gei_3/8       ospf     110 2
1.1.1.44        255.255.255.255 1.1.1.44        loopback1     address  0   0
1.20.20.44      255.255.255.255 1.20.20.44      loopback16    address  0   0
1.20.20.49      255.255.255.255 6.66.1.2        gei_3/6.3     static   1   0
2.2.2.0         255.255.255.0   2.2.2.2         loopback2     direct   0   0
2.2.2.2         255.255.255.255 2.2.2.2         loopback2     address  0   0
2.2.22.2        255.255.255.255 35.1.1.2        gei_3/5       static   1   0
5.5.5.0         255.255.255.0   5.5.5.1         fei_1/5       direct   0   0
5.5.5.1         255.255.255.255 5.5.5.1         fei_1/5       address  0   0
6.1.2.0         255.255.255.0   6.66.1.2        gei_3/6.3     static   1   0
6.66.1.0        255.255.255.0   6.66.1.3        gei_3/6.3     direct   0   0
6.66.1.3        255.255.255.255 6.66.1.3        gei_3/6.3     address  0   0
22.22.0.0       255.255.0.0     22.22.22.23     fei_1/16      direct   0   0
22.22.22.23     255.255.255.255 22.22.22.23     fei_1/16      address  0   0
40.40.40.0      255.255.255.0   40.40.40.44     loopback40    direct   0   0
40.40.40.44     255.255.255.255 40.40.40.44     loopback40    address  0   0
42.76.1.0       255.255.255.0   44.1.1.2        gei_3/8       ospf     110 2
44.1.1.0        255.255.255.0   44.1.1.1        gei_3/8       direct   0   0
44.1.1.1        255.255.255.255 44.1.1.1        gei_3/8       address  0   0
50.50.50.44     255.255.255.255 50.50.50.44     loopback50    address  0   0
61.1.0.0        255.255.0.0     61.1.1.1        gei_3/10.1    direct   0   0
61.1.1.1        255.255.255.255 61.1.1.1        gei_3/10.1    address  0   0
61.2.0.0        255.255.0.0     61.2.1.1        gei_3/10.2    direct   0   0
61.2.1.1        255.255.255.255 61.2.1.1        gei_3/10.2    address  0   0
61.3.0.0        255.255.0.0     61.3.1.1        gei_3/10.3    direct   0   0
61.3.1.1        255.255.255.255 61.3.1.1        gei_3/10.3    address  0   0
61.4.0.0        255.255.0.0     61.4.1.1        gei_3/10.4    direct   0   0
61.4.1.1        255.255.255.255 61.4.1.1        gei_3/10.4    address  0   0
61.5.0.0        255.255.0.0     61.5.1.1        gei_3/10.5    direct   0   0
61.5.1.1        255.255.255.255 61.5.1.1        gei_3/10.5    address  0   0
61.6.0.0        255.255.0.0     61.6.1.1        gei_3/10.6    direct   0   0
61.6.1.1        255.255.255.255 61.6.1.1        gei_3/10.6    address  0   0
61.7.0.0        255.255.0.0     61.7.1.1        gei_3/10.7    direct   0   0
61.7.1.1        255.255.255.255 61.7.1.1        gei_3/10.7    address  0   0
61.8.0.0        255.255.0.0     61.8.1.1        gei_3/10.8    direct   0   0
61.8.1.1        255.255.255.255 61.8.1.1        gei_3/10.8    address  0   0
61.9.0.0        255.255.0.0     61.9.1.1        gei_3/10.9    direct   0   0
61.9.1.1        255.255.255.255 61.9.1.1        gei_3/10.9    address  0   0
61.10.0.0       255.255.0.0     61.10.1.1       gei_3/10.10   direct   0   0
61.10.1.1       255.255.255.255 61.10.1.1       gei_3/10.10   address  0   0
61.9.1.1        255.255.255.255 61.9.1.1        gei_3/10.9    address  0   0
61.10.0.0       255.255.0.0     61.10.1.1       gei_3/10.10   direct   0   0
61.10.1.1       255.255.255.255 61.10.1.1       gei_3/10.10   address  0   0
66.1.2.0        255.255.255.0   66.1.2.3        gei_3/4.1     direct   0   0
66.1.2.3        255.255.255.255 66.1.2.3        gei_3/4.1     address  0   0
81.81.81.0      255.255.255.0   81.81.81.44     gei_3/10      direct   0   0
81.81.81.44     255.255.255.255 81.81.81.44     gei_3/10      address  0   0
91.91.91.0      255.255.255.0   44.1.1.2        gei_3/8       ospf     110 2
99.99.99.0      255.255.255.0   44.1.1.2        gei_3/8       ospf     110 2
192.168.0.0     255.255.0.0     192.168.1.44    fei_1/16      direct   0   0
192.168.1.44    255.255.255.255 192.168.1.44    fei_1/16      address  0   0
4830_44#
```

图 4-32 查询指定路由条目成功显示

4.4.2.4 添加一条路由

使用路由条目添加命令（ip route IP 地址 子网掩码 接口）添加一条路由，如图 4-33 所示。

```
4830_44(config)#ip route 100.100.100.100 255.255.255.255 gei_3/8
```

图 4-33 添加一个路由条目

重新显示系统中所有的路由条目，结果显示已经成功添加图 4-33 所添加的路由条目，如图 4-34 所示。

```
4830_44#sho ip route
Total number of routes:      49
IPv4 Routing Table:
Dest            Mask            Gw              Interface     Owner    Pri Metri
1.1.1.30        255.255.255.255 1.1.1.30        loopback30    address 0   0
1.1.1.42        255.255.255.255 44.1.1.2        gei_3/8       ospf    110 2
1.1.1.44        255.255.255.255 1.1.1.44        loopback1     address 0   0
1.20.20.44      255.255.255.255 1.20.20.44      loopback16    address 0   0
1.20.20.49      255.255.255.255 6.66.1.2        gei_3/6.3     static  1   0
2.2.2.0         255.255.255.0   2.2.2.2         loopback2     direct  0   0
2.2.2.2         255.255.255.255 2.2.2.2         loopback2     address 0   0
2.2.22.2        255.255.255.255 35.1.1.2        gei_3/5       static  1   0
5.5.5.0         255.255.255.0   5.5.5.1         fei_1/5       direct  0   0
5.5.5.1         255.255.255.255 5.5.5.1         fei_1/5       address 0   0
6.1.2.0         255.255.255.0   6.66.1.2        gei_3/6.3     static  1   0
6.66.1.0        255.255.255.0   6.66.1.3        gei_3/6.3     direct  0   0
6.66.1.3        255.255.255.255 6.66.1.3        gei_3/6.3     address 0   0
22.22.0.0       255.255.0.0     22.22.22.23     fei_1/16      direct  0   0
22.22.22.23     255.255.255.255 22.22.22.23     fei_1/16      address 0   0
40.40.40.0      255.255.255.0   40.40.40.44     loopback40    direct  0   0
40.40.40.44     255.255.255.255 40.40.40.44     loopback40    address 0   0
42.76.1.0       255.255.255.0   44.1.1.2        gei_3/8       ospf    110 2
44.1.1.0        255.255.255.0   44.1.1.1        gei_3/8       direct  0   0
44.1.1.1        255.255.255.255 44.1.1.1        gei_3/8       address 0   0
50.50.50.44     255.255.255.255 50.50.50.44     loopback50    address 0   0
61.1.0.0        255.255.0.0     61.1.1.1        gei_3/10.1    direct  0   0
61.1.1.1        255.255.255.255 61.1.1.1        gei_3/10.1    address 0   0
61.2.0.0        255.255.0.0     61.2.1.1        gei_3/10.2    direct  0   0
61.2.1.1        255.255.255.255 61.2.1.1        gei_3/10.2    address 0   0
61.3.0.0        255.255.0.0     61.3.1.1        gei_3/10.3    direct  0   0
61.3.1.1        255.255.255.255 61.3.1.1        gei_3/10.3    address 0   0
61.4.0.0        255.255.0.0     61.4.1.1        gei_3/10.4    direct  0   0
61.4.1.1        255.255.255.255 61.4.1.1        gei_3/10.4    address 0   0
61.5.0.0        255.255.0.0     61.5.1.1        gei_3/10.5    direct  0   0
61.5.1.1        255.255.255.255 61.5.1.1        gei_3/10.5    address 0   0
61.6.0.0        255.255.0.0     61.6.1.1        gei_3/10.6    direct  0   0
61.6.1.1        255.255.255.255 61.6.1.1        gei_3/10.6    address 0   0
61.7.0.0        255.255.0.0     61.7.1.1        gei_3/10.7    direct  0   0
61.7.1.1        255.255.255.255 61.7.1.1        gei_3/10.7    address 0   0
61.8.0.0        255.255.0.0     61.8.1.1        gei_3/10.8    direct  0   0
61.8.1.1        255.255.255.255 61.8.1.1        gei_3/10.8    address 0   0
61.9.0.0        255.255.0.0     61.9.1.1        gei_3/10.9    direct  0   0
61.9.1.1        255.255.255.255 61.9.1.1        gei_3/10.9    address 0   0
61.10.0.0       255.255.0.0     61.10.1.1       gei_3/10.10   direct  0   0
61.10.1.1       255.255.255.255 61.10.1.1       gei_3/10.10   address 0   0
100.100.100.100 255.255.255.255 44.1.1.1        gei_3/8       static  1   0
61.9.1.1        255.255.255.255 61.9.1.1        gei_3/10.9    address 0   0
61.10.0.0       255.255.0.0     61.10.1.1       gei_3/10.10   direct  0   0
61.10.1.1       255.255.255.255 61.10.1.1       gei_3/10.10   address 0   0
66.1.2.0        255.255.255.0   66.1.2.3        gei_3/4.1     direct  0   0
66.1.2.3        255.255.255.255 66.1.2.3        gei_3/4.1     address 0   0
81.81.81.0      255.255.255.0   81.81.81.44     gei_3/10      direct  0   0
81.81.81.44     255.255.255.255 81.81.81.44     gei_3/10      address 0   0
91.91.91.0      255.255.255.0   44.1.1.2        gei_3/8       ospf    110 2
99.99.99.0      255.255.255.0   44.1.1.2        gei_3/8       ospf    110 2
192.168.0.0     255.255.0.0     192.168.1.44    fei_1/16      direct  0   0
192.168.1.44    255.255.255.255 192.168.1.44    fei_1/16      address 0   0
4830_44#
```

图 4-34 添加一条路由条目后的路由表

4.4.2.5 删除一个指定路由条目

使用 no ip route 命令（no ip route IP 子网掩码），删除一个指定的路由条目，如图 4-35 所示。

```
4830_44(config)#no ip route 100.100.100.100  255.255.255.255
```

图 4-35 删除一个路由条目

使用命令（show ip route）查看删除指定路由条目后，系统中所有的路由条目，图 4-36 显示已经成功删除了图 4-33 中路由条目添加命令成功添加的一个路由条目。

```
4830_44#sho ip route
Total number of routes:      49
IPv4 Routing Table:
Dest            Mask            Gw              Interface    Owner    Pri Metri
1.1.1.30        255.255.255.255 1.1.1.30        loopback30   address  0   0
1.1.1.42        255.255.255.255 44.1.1.2        gei_3/8      ospf     110 2
1.1.1.44        255.255.255.255 1.1.1.44        loopback1    address  0   0
1.20.20.44      255.255.255.255 1.20.20.44      loopback16   address  0   0
1.20.20.49      255.255.255.255 6.66.1.2        gei_3/6.3    static   1   0
2.2.2.0         255.255.255.0   2.2.2.2         loopback2    direct   0   0
2.2.2.2         255.255.255.255 2.2.2.2         loopback2    address  0   0
2.2.22.2        255.255.255.255 35.1.1.2        gei_3/5      static   1   0
5.5.5.0         255.255.255.0   5.5.5.1         fei_1/5      direct   0   0
5.5.5.1         255.255.255.255 5.5.5.1         fei_1/5      address  0   0
6.1.2.0         255.255.255.0   6.66.1.2        gei_3/6.3    static   1   0
6.66.1.0        255.255.255.0   6.66.1.3        gei_3/6.3    direct   0   0
6.66.1.3        255.255.255.255 6.66.1.3        gei_3/6.3    address  0   0
22.22.0.0       255.255.0.0     22.22.22.23     fei_1/16     direct   0   0
22.22.22.23     255.255.255.255 22.22.22.23     fei_1/16     address  0   0
40.40.40.0      255.255.255.0   40.40.40.44     loopback40   direct   0   0
40.40.40.44     255.255.255.255 40.40.40.44     loopback40   address  0   0
42.76.1.0       255.255.255.0   44.1.1.2        gei_3/8      ospf     110 2
44.1.1.0        255.255.255.0   44.1.1.1        gei_3/8      direct   0   0
44.1.1.1        255.255.255.255 44.1.1.1        gei_3/8      address  0   0
50.50.50.44     255.255.255.255 50.50.50.44     loopback50   address  0   0
61.1.0.0        255.255.0.0     61.1.1.1        gei_3/10.1   direct   0   0
61.1.1.1        255.255.255.255 61.1.1.1        gei_3/10.1   address  0   0
61.2.0.0        255.255.0.0     61.2.1.1        gei_3/10.2   direct   0   0
61.2.1.1        255.255.255.255 61.2.1.1        gei_3/10.2   address  0   0
61.3.0.0        255.255.0.0     61.3.1.1        gei_3/10.3   direct   0   0
61.3.1.1        255.255.255.255 61.3.1.1        gei_3/10.3   address  0   0
61.4.0.0        255.255.0.0     61.4.1.1        gei_3/10.4   direct   0   0
61.4.1.1        255.255.255.255 61.4.1.1        gei_3/10.4   address  0   0
61.5.0.0        255.255.0.0     61.5.1.1        gei_3/10.5   direct   0   0
61.5.1.1        255.255.255.255 61.5.1.1        gei_3/10.5   address  0   0
61.6.0.0        255.255.0.0     61.6.1.1        gei_3/10.6   direct   0   0
61.6.1.1        255.255.255.255 61.6.1.1        gei_3/10.6   address  0   0
61.7.0.0        255.255.0.0     61.7.1.1        gei_3/10.7   direct   0   0
61.7.1.1        255.255.255.255 61.7.1.1        gei_3/10.7   address  0   0
61.8.0.0        255.255.0.0     61.8.1.1        gei_3/10.8   direct   0   0
61.8.1.1        255.255.255.255 61.8.1.1        gei_3/10.8   address  0   0
61.9.0.0        255.255.0.0     61.9.1.1        gei_3/10.9   direct   0   0
61.9.1.1        255.255.255.255 61.9.1.1        gei_3/10.9   address  0   0
61.10.0.0       255.255.0.0     61.10.1.1       gei_3/10.10  direct   0   0
61.10.1.1       255.255.255.255 61.10.1.1       gei_3/10.10  address  0   0
61.9.1.1        255.255.255.255 61.9.1.1        gei_3/10.9   address  0   0
61.10.0.0       255.255.0.0     61.10.1.1       gei_3/10.10  direct   0   0
61.10.1.1       255.255.255.255 61.10.1.1       gei_3/10.10  address  0   0
66.1.2.0        255.255.255.0   66.1.2.3        gei_3/4.1    direct   0   0
66.1.2.3        255.255.255.255 66.1.2.3        gei_3/4.1    address  0   0
81.81.81.0      255.255.255.0   81.81.81.44     gei_3/10     direct   0   0
81.81.81.44     255.255.255.255 81.81.81.44     gei_3/10     address  0   0
91.91.91.0      255.255.255.0   44.1.1.2        gei_3/8      ospf     110 2
99.99.99.0      255.255.255.0   44.1.1.2        gei_3/8      ospf     110 2
192.168.0.0     255.255.0.0     192.168.1.44    fei_1/16     direct   0   0
192.168.1.44    255.255.255.255 192.168.1.44    fei_1/16     address  0   0
4830_44#
```

图 4-36 删除一条路由条目后的路由表

## 4.4.3 基于 RFC 算法的访问控制列表（ACL）存储显示

### 4.4.3.1 显示已存在的 ACL 规则

使用 ACL 规则显示命令（acl standard number num），显示 num 对应的 ACL 列表项中的规则，如下图显示 acl standard number 99，显示 99 号 ACL 列表中的条目。其中包含三条规则：

rule 2 permit 10.10.1.5 0.0.0.0

rule 1 deny any

rule 3 permit 228.1.1.0 0.0.0.255

```
4830_42(config)#acl standard number ?
  <1-99>       Configure standard ACL number
  <1000-1499>  Configure standard ACL number (expanded range)
4830_42(config)#acl standard number 99
4830_42(config-std-acl)#sho thi
!
acl standard number 99
  rule 2 permit 10.10.1.5 0.0.0.0
  rule 1 deny any
  rule 3 permit 228.1.1.0 0.0.0.255
!
!
```

图 4-37 显示系统中已经配置的 ACL 规则

4.4.3.2 删除一条存在的 ACL 规则

删除其中一条规则，如图 4-38 所示删除了上述的三条规则中的规则 1（rule 1）。

```
4830_42(config-std-acl)#no ru
4830_42(config-std-acl)#no rule 1
```

图 4-38 删除一条 ACL 规则

显示删除规则 1 后剩下的规则，图 4-39 显示已经成功删除，只剩下规则 2 和规则 3。

```
4830_42(config-std-acl)#sho thi
!
acl standard number 99
  rule 2 permit 10.10.1.5 0.0.0.0
  rule 3 permit 228.1.1.0 0.0.0.255
!
!
```

图 4-39 删除一条 ACL 规则之后显示所有剩下的 ACL 规则

4.4.3.2 添加一条 ACL 规则

添加一条规则，使用 permit 命令，如图 4-40 所示。

```
!
4830_42(config-std-acl)#per
4830_42(config-std-acl)#permit 227.1.1.1 ?
  A.B.C.D      Wildcard bits
  time-range   Configure time range
  <cr>
4830_42(config-std-acl)#permit 227.1.1.1
```

图 4-40 添加一条 ACL 规则

添加过后的结果如图 4-41 所示。

```
4830_42(config-std-acl)#sho thi
!
acl standard number 99
 rule 2 permit 10.10.1.5 0.0.0.0
 rule 3 permit 228.1.1.0 0.0.0.255
 rule 1 permit 227.1.1.1 0.0.0.0
!
```

图 4-41 显示添加一条 ACL 规则之后的 ACL 规则库

## 4.5 本章小结

本章介绍 BRAS 驱动业务支持相关数据结构的设计与实现。首先介绍了基于 HASH 的相关业务表项数据结构，说明了实现的原理，并以电路表的相关操作举例说明了相应的方法和流程。其次，介绍了基于 TRIE 树的路由表存储和管理的算法，对添加、查找和删除指定的路由条目的过程和算法进行了详细说明。再其次，介绍了基于 RFC 算法的 ACL 流分类算法设计与实现，包括使用 RFC 算法实现报文分类和压缩的原理，并对基于 RFC 算法的驱动 ACL 列表对应的网络处理器微码实现流程进行说明。最后，在真实的 BRAS 系统配置和测试平台对上述的基于 HASH、TRIE 以及 RFC 算法的相关数据结构进行验证，实验证明，本文设计的数据结构和算法可以实现预期的目标。

# 第五章 BRAS 驱动通讯模块设计与实现

在 BRAS 系统的软件总体设计中，驱动模块运行在 NP 单板上，其任务包括：存储网络处理器处理各种业务需要的数据结构，并为网络处理器等模块提供操作借口；报文的转发和分析处理，对于需要由主控处理的软件转发报文、本地报文以及异常报文都将上交给主控系统处理，以及接受上层模块下发的协议报文。

通讯模块解决的主要是几个数据处理单元之间的信息交互问题，由于 NP 线卡的特点，上行 IXP2800 作为主 NP,负责与主控协议层的通讯和上行报文的转发处理，同时也对从 NP 与主控协议层之间的通讯起中间人的作用。同时，主从 NP 的 Xscale 还必须转发从微码上送协议处理的报文，因此通讯模块还包括 Xscale 与微码的通讯。即通讯模块包括：

(1) 主 NP 的 Xscale 与主控通讯（通过回调函数接口）

(2) 主 NP 的 Xscale 与从 NP 的 Xscale 通讯（通过 PCI）

(3) 主从 NP 的 Xscale 与各自模块的微码通讯（Xscale Ring）

如图 5-1 所示。

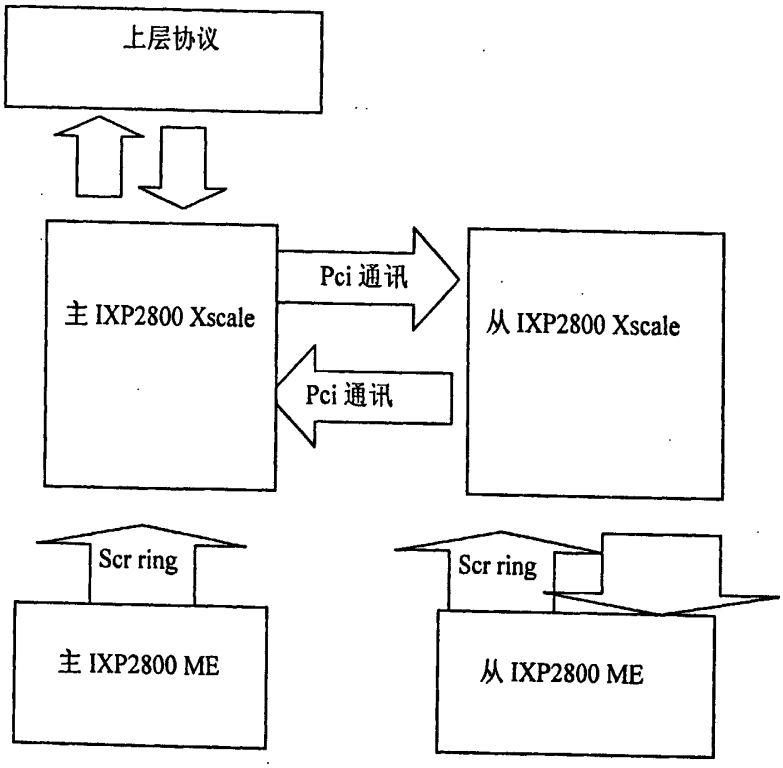


图 5-1 主从 IXP2800 通讯结构

主从 Xscale 之间通过 PCI 总线进行通讯，Xscale 与 ME 之间通过共享内存 Scratch Ring

进行通讯，主 Xscale 与上层协议之间的报文通过上层代码传递。

使用驱动通讯模块的数据流有：

(1)上层协议下发的协议报文流

上层协议—>主 IXP2800 Xscale—>从 IXP2800 Xscale—>从 IXP2800 ME

(2)主微码上送的协议处理/数据报文流

主 IXP2800 ME—>主 IXP2800 Xscale—>上层协议

(3)从微码上送的协议处理报文流

从 IXP2800 ME—>从 IXP2800 Xscale—>主 IXP2800 Xscale—>上层协议

## 5.1 主 IXP2800 上送的协议报文流

整体流程设计：主 IXP2800 Xscale 在初始化时使用 API 函数启动了一个收包任务 RmPktEng，该收包任务通过一个 while（1）循环不断轮询 Scratch Ring，发现有数据后取出 bufferhandle，然后调用函数通过 bufferhandle 从 buffer 链中拷贝出报文到 Sdram 中，然后将报文封装为指定结构后调用函数将封装好的报文以消息形式发送给上层，流程如图 5-2 和 5-3 所示。

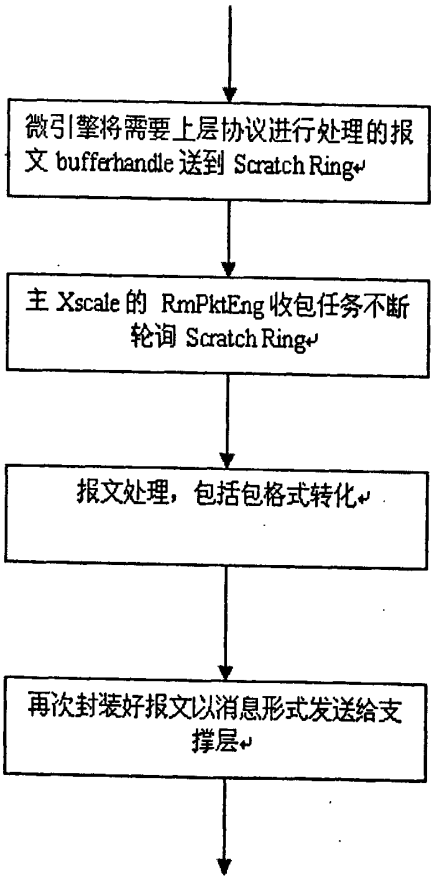


图 5-2 主 IXP2800 上送的协议报文流

主 IXP2800 Xscale 在初始化时启动了一个收包任务 SendPacket，该收包任务通过一个 while（1）循环不断轮询 Scratch Ring，发现有数据后取出 bufferhandle，通过 bufferhandle 从 buffer 链中拷贝出报文到 sdram 中，然后将报文封装为 STRU_SEND_PKT 结构，最后将封装好的报文以消息形式发送给上层。

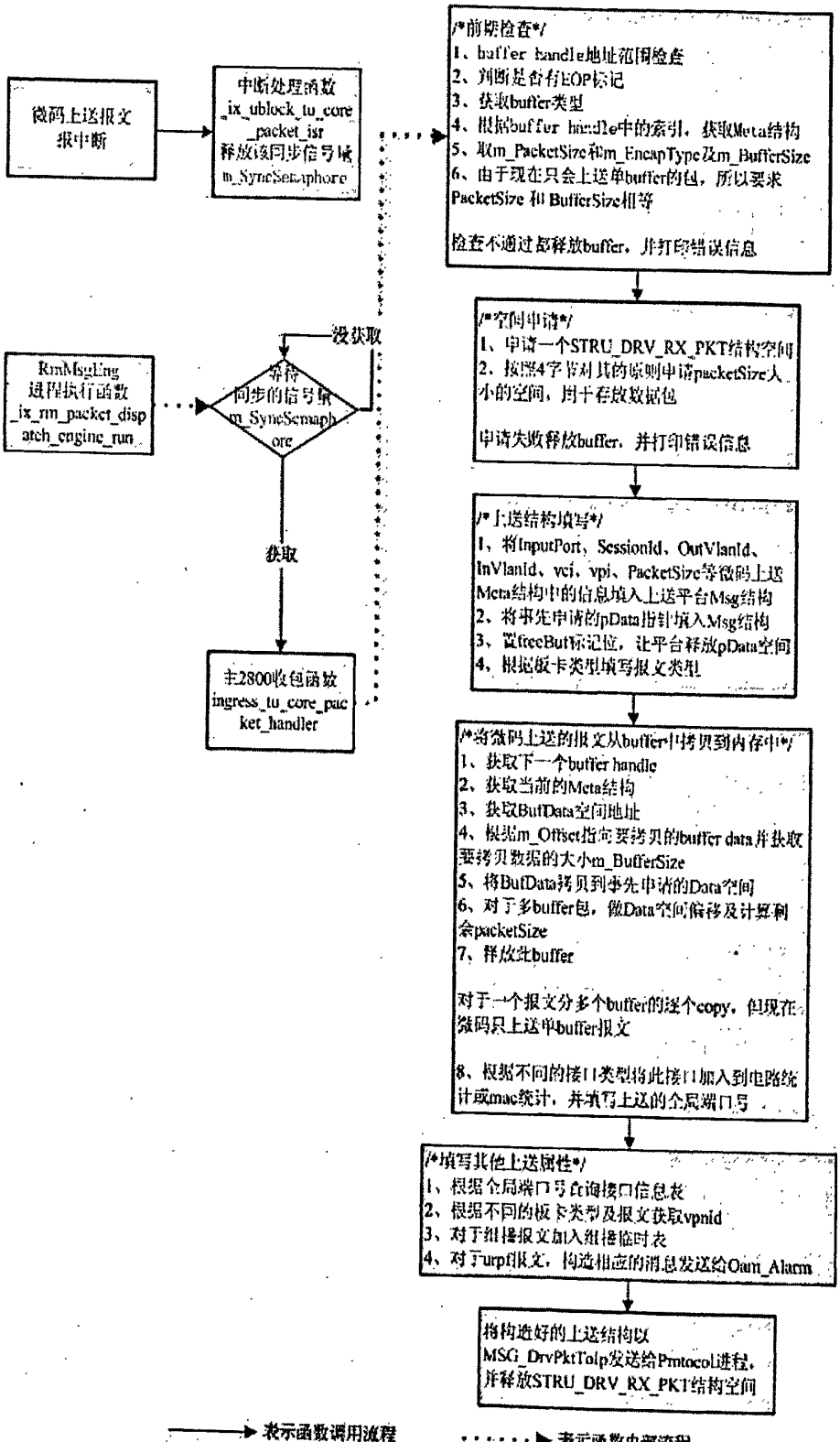


图 5-3 上送的协议报文流程

## 5.2 上层协议下发的协议报文流

下发 NP 的协议报文流在到达主 IXP2800 Xscale 后，支撑层调用驱动注册的回调接口函数将报文送到驱动。

协议报文从支撑层进入到驱动时，封装数据结构为新的 STRU_DRV_SEND_PKT 格式，然后将其转换为 Xscale 的内部报文数据结构包括（首部＋数据部），其中 pData 是整个协议报文。转换数据结构的目的是将逻辑端口号以及其他一些接口属性与实际接口统一，并符合 PCI 发送格式。

下发的包由主 IXP2800 通过 PCI 通讯发送至从 IXP2800，从 IXP2800 处理从 PCI 总线上接收到的协议报文，从为微引擎分配的 buffer 链中取出一个 buffer，将报文拷贝到该 buffer 后，将 buffer 的地址 bufferhandle 写入与微引擎通讯的 Scratch Ring 中，这样从 IXP2800 就将协议报文下发给了微引擎。

微引擎中的任务一直轮询这个与从 IXP2800 通讯的 Scratch Ring,发现有数据后取出 bufferhandle 然后访问所指向的 buffer，就可以获得协议报文将其转发。

下发 NP 的协议报文流在到达主 IXP2800 Xscale 后，支撑层调用驱动接口函数 DownSendPacket()将报文送到驱动，然后驱动对协议报文进行格式转换和发送。

协议报文从支撑层进入到驱动时，封装使用的数据结构为 STRU_SEND_PKT，函数 xscale_send()将其转换为 Xscale 的内部报文数据结构（包头＋包数据），其中包头是数据首部，包数据是整个协议报文。转换数据结构的目的是将逻辑端口号以及其他一些接口属性与实际接口统一，并符合 PCI 发送格式,最后通过 PCI 总线将报文发送至从 IXP2800.至此，主 IXP2800 对于下发报文的处理过程结束。

从 IXP2800 处理从 PCI 总线上接收到的协议报文：从为微引擎分配的 buffer 链中取出一个 buffer，将报文拷贝到该 buffer 后，将 buffer 的地址 bufferhandle 写入与微引擎通讯的 Scratch Ring 中，这样从 IXP2800 就将协议报文下发给了微引擎。

下发协议报文总体流程如图 5-4 所示。

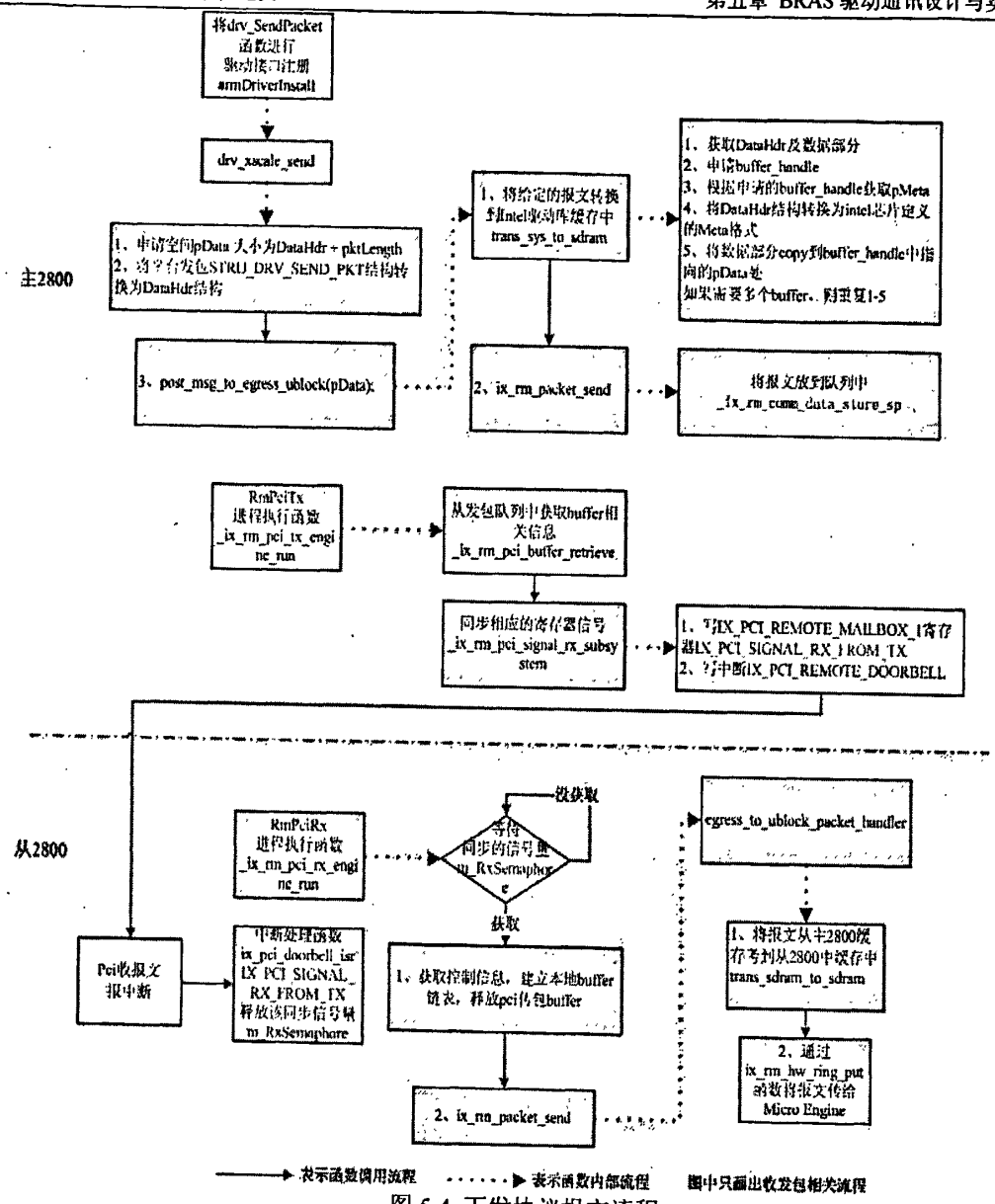


图 5-4 下发协议报文流程

## 5.3 从 IXP2800 上送的协议报文流

微引擎在报文转发过程中，遇到找不到 ARP 的情况下需要发送 no arp 报文，此报文需要上层协议进行处理，从微码到从 Xscale 的过程与上面主 IXP2800 的处理流程基本一致，收包任务对报文进行处理，将报文封装为 STRU_DRV_RX_PKT 结构,然后通过 PCI 发送给主 IXP2800，这里需要注意的是报文不再通过主 IXP2800 上的驱动处理再上送（因为在从 IXP2800 中已经处理过了），而是由上面的支撑层直接接收。从 IXP2800 上送的协议报文总体流程如图 5-5 所示。

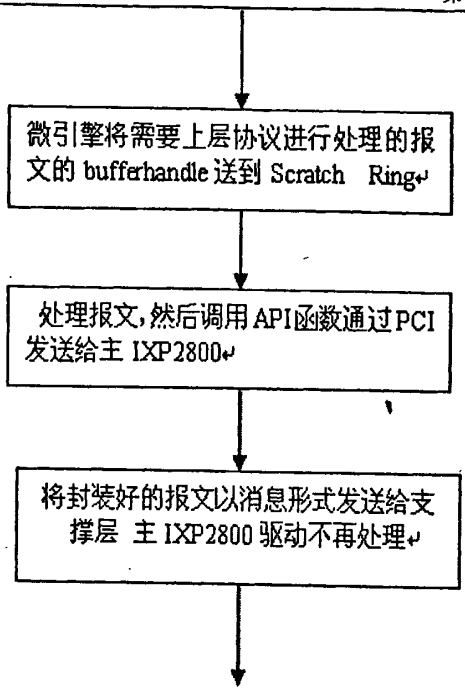


图 5-5 从 IXP2800 上送的协议报文流

微引擎在报文转发过程中，遇到 ARP 失败的情况，需要发送 ARP 失败报文交由上层协议进行处理，从微码到从 IXP2800 Xscale 的过程与上面主 IXP2800 的处理流程基本一致，该函数同样将报文封装为 STRU_SEND_PKT 结构,然后通过 PCI 发送给主 IXP2800，这里报文不再通过主 IXP2800 上的驱动处理再上送（因为在从 IXP2800 中已经处理过了），而是由上面的支撑层直接接收。

## 5.4 实验结果分析

### 5.4.1 验证当前处理的报文

显示当前处理的报文相关信息，使用 packet 命令，如图 5-6 所示。其中参数 2 为槽位号（slot），master 表示主 IXP2800，4 表示 4 号 srcatch ring（这一队列在 ETHER、POS 接口代码中是接收模块和转发模块的接口队列）。

命令结果中参数说明：

1）Ring base addr：驱动与微码通信的 Ring 在 scratch 中的地址；

2）Buffer handle：驱动与微码报文交互的 Buffer handle 在 scratch 中地址；

3）Buffer meta addr：驱动与微码报文交互 meta 结构在 sram 中的地址；

4）Packet Size：报文大小；

5）Input Port：收入报文端口号；

6）Output Port：输出报文端口；

7）Buffer addr : Buffer 基址；

8）Buffer Size : Buffer 大小；

9）Buffer Data are following：后面显示当前报文报文内容在 dram 中的地址，以及报文中的数据。

```
ZXR10(diag)#npc packet current 2 master 4


Ring base addr     : 0xca001000
Buffer handle      : 0xc0002148
Buffer meta addr   : 0x80008520
Packet Size        :         60
Input  Port        : 0x0
Output Port        : 0xff

Buffer addr        : 0xc4a88c0
Buffer Size          : 60
Buffer Data are following.....................
Address  0001 0203 0405 0607 0809 1011 1213 1415
=================================================
0c4a88c0 ffff ffff d000 ffff 207d c0d0 0100 0608
0c4a88d0 0406 0008 d000 0100 207d c0d0 0246 01a8
0c4a88e0 0000 0000 ffff 0000 0000 ffff 0000 0000
0c4a88f0 0000 0000 0000 0000 0000 0000
```

图 5-6 当前处理报文（packet）相关信息

## 5.4.2 验证报文发送与接收统计数据

使用 communications status 命令查看当前发送与接收统计数据，包括可用于主从通讯，包括主上上送， no arp 上送以及主从通讯的计数。Reset 用于清除统计。 下面对命令结果参数进行说明：

接收报文情况统计：

1）Num of pkt receive from ME:从微引擎接收的报文数；

2）Num of pkt to protocol OK:成功上送协议的报文数；

3）Num of pkt invalid buffer:无效 buffer 的报文数；

4）Num of pkt alloc memory faile：分配存储空间失败的次数；

5）Num of no arp pkt receive：接收 no arp 报文数；

6）Num of no arp pkt to protocol：上送协议的 no arp 报文数。

发送报文情况统计：

1）packet to slave mcode ：发送到从 IXP2800 的报文数；

2）packet to slave mcode ok ：成功发送到从 IXP2800 的报文数。

```
ZXUAS(diag)#npc communication status 5 re

========receive packet information========
Num of pkt receive from ME    : 854

Num of pkt to protocol ok     : 854
Num of pkt invalid buffer     : 0
Num of pkt get buffer meta    : 0
Num of pkt alloc memory fail  : 0

Num of no arp pkt receive     : 0
Num of no arp pkt to protocol: 0

========send packet information========
packet to slave mcode         : 733
packet to slave mcode ok      : 733

packet to slave mcode         : 733
packet to slave mcode ok      : 733
```

图 5-6 报文通讯计数信息

实验结果分析：

如图 5-6 所示，实验时间段内，微码模块收到了 854 个报文，全部成功上送至上层协议（protocol），无效 buffer 个数为 0，存储分配空间失败数为 0，在接受报文方面完全达到预期要求。在发送报文方面，有 733 个报文由主 IXP2800 发送至从 IXP2800，发送成功 733 个，失败率为 0，达到了主从通讯的要求。

## 5.5 本章小结

本章介绍 BRAS 驱动通讯模块的设计和实现，从驱动通讯模块处理的三个方向的报文流分别进行说明。首先介绍了主 IXP2800 上送的协议报文流，其次介绍上层协议下发的协议报文流，再其次介绍了从 IXP2800 上送的协议报文流。最后，对上述的通讯模块设计的流程进行了验证。

# 第六章 总结与展望

## 6.1 总结

基于网络处理器架构的BRAS需要设计一个网络处理器服务模块，又称驱动模块。驱动模块是介于网络处理器与上层协议之前的服务模块，肩负着传递上层协议与底层微码的报文交互、管理和维护相关业务数据结构的关键任务。作者在参与项目过程中，参与了整个驱动模块系统的研究、设计和编码开发以及维护，任何一个系统的故障或者效率不足问题在故障定位和系统性能分析研究过程中都要驱动开发人员的参与，深刻体会到BRAS驱动模块的设计是否合理直接关系到整个BRAS系统的效率、可靠性和稳定性。对本文所做的工作，进行简要的总结如下：

1、本文系统地介绍了BRAS的整体结构，驱动模块在整个系统的位置、功能定位和与其他模块交互情况。然后按照驱动模块的主要任务，对BRAS驱动模块关键技术进行了分析研究，探究了各个技术的实现原理。

2、在业务支持数据结构方面，本文详细介绍了目前分布式BRAS数据结构主要使用的业务存储数据结构，对HASH、RFC、TRIE各个数据结构在驱动模块的应用进行了详细的分析研究。

3、使用TRIE数据结构实现路由表存储与查找的优点是访问次数稳定，时间复杂度是O(1)，特别适用于基于网段的、需要最长匹配的算法，但是同时也有其缺点，如内存占用不稳定，浪费可能比较大，依赖于具体的路由条目，实现相对比较复杂。

## 6.2 展望

由于BRAS基于高端路由器的体系结构，随着路由器技术的发展到第五代，与路由器趋同，目前BRAS的发展有网络侧接口大型化、性能要求不断提高的趋势；另一方面，随着无线网络的发展，国内3G发展的如火如荼，多网融合成为大的趋势。用户接入方式多样化、服务需求多样化，导致在接入网领域也在发生的许多的技术变革。运营商之间、企业之间竞争日趋激烈，对于BRAS产品的性能要求越来越高，另一方面也要求BRAS等通信设备在功耗和设备成本方面为运营商等企业提供更多的竞争力。因此，作者认为目前关

于 BRAS 驱动的研究目前在以下领域可做进一步研究：

1、应对多网融合，BRAS 微码与驱动等模块应对更多种接入方式兼容。对于不同业务、不同接入方式微码应有相应的高效的处理流程，充分利用已有的资源以实现负载均衡。驱动作为微码服务支持结构，可在相应服务数据结构、收发报文解析方向进行研究和优化。

2、对于驱动相关数据结构、算法的研究。目前为实现业务质量的提升，部分厂家引入 TCAM 等硬件提高驱动模块的效率，这对于提升 BRAS 系统性能具有积极意义，但是这些硬件的高功率、昂贵的价格和某些不稳定性给设备商和运营商也带来了成本压力，因此对驱动模块相关数据结构算法与硬件结合进行研究对于降低成本和实现负载均衡具有积极意义。

# 致 谢

在本文即将完稿之际，我无法忘记那些直接或间接为论文做出贡献以及曾经给予我支持和帮助的人们。

首先，我由衷地感谢我的导师孙知信教授。在硕士三年的学习和毕业设计的全过程中，我始终得到了孙老师悉心的指导。孙老师深厚的理论水平、严谨的治学态度、朴素的生活作风以及积极进取的精神不仅使我在学校期间得到很大的帮助和启迪，而且对我以后的工作和学习也将有深远的影响，使我受益终身。

我还要感谢诸位朝夕相处的项目组的同事和同学们，感谢他们在我学习、生活中给予的无私关怀和帮助。在整个项目的开发过程中，整个小组同甘共苦，一起讨论攻克了很多技术难关，在此向他们表示衷心的谢意。

最后，向各位不辞辛苦审阅本论文的教授、专家们表示由衷的谢意。

# 参考文献

[1] 胡钧.宽带接入服务器.《中兴通讯技术》.2001 年第 1 期.

[2] 魏亮.高端路由器现状及新发展.信息产业部电信传输研究所 2003.

[3] Gimenez, G. ;Inigo,J. ; Lagares, J. ; Navarro, L. ; Reig, F. ; Rodriguez, G. . Ecology in Global Distributed Systems.Distributed and Networked Environments, 1994.

[4] Francois Blouin.Nalin Mistry,Tadeusz Drwiega and Tom Anschutz.Performance Evaluation of Rate Control and QoS Capabilities of BRAS.Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004. IEEE ; 2004 , Page(s): 293 – 301.

[5] Jian Wang, Jian Gu,Fang Mei , Zhe Zhang and Yanheng Liu.One New BRAS Model Based on Trusted Network.International Symposium on Electronic Commerce and Security; 3-5 Aug. 2008.

[6] 谢希.基于网络处理器的流量控制设计与实现.湖南大学工程硕士学位论文,2008.5

[7] Network Processor Conference.http://www.networkprocessors.com.2001.

[8] Nikoladis I.Network Systems Design Using Network Processor.Network IEEE,2004,Vol 18:5-5.

[9] Network Processor Conference.http://www.networkprocessors.com.2001.

[10] Xiaoning Nie,Ulf Uordqvist,Lajos Gazsi,etal.Network Processors for Access Network(NP4AN)In:SOC Conference,Seoul:IEEE,2004,265-269.

[11] 石晶林,程胜,孙江明.网络处理器原理、设计与应用,第一版.北京:清华大学出版社,2003,60-90.

[12] 张宏科,苏伟,武勇.网络处理器原理与技术,第一版.北京:北京邮电大学出版社,2004,241-245.

[13] MarkLaPedus.Intel's XScale architecture debuts in network-processor chip set. http://www.eetimes.com/story/OEG20000927S0078,2007-9-27.

[14] 陈红松,季振洲,胡铭曾.高性能网络处理器同时多线程结构设计与研究.微处理机,2005,Vol 26:17-20.

[15] Chris Ostle,Karam S.Chatha.An ILP Formulation for System-Level Application Mapping on Network Processor Architectures.EDA Consortium,IEEE,2007:67-69.

[16] Intel Corporation.Intel IXP2800 Network Processor Product Brief.2004.

[17] 港湾网络公司供稿.BRAS 面向业务接入控制层的发展.世界通信:2004.NO.8.

[18] Rodwald, P.; Stoklosa, J. .Family of Parameterized Hash Algorithms. 2008. SECURWARE '08. Second International Conference on Emerging Security Information, Systems and Technologies,Page(s): 203 – 208.

[19] Huntley, C.; Antonova, G.; Guinand, P. . Effect of Hash Collisions on the Performance of LAN Switching Devices and Networks. Proceedings 2006 31st IEEE Conference on Local Computer Networks, 2006 , Page(s): 280 - 284

[20] 梁志勇, 徐 恪, 吴建平, 徐明伟.分布式路由器中的路由管理模型. 清华大学学报(自然科学版):2003 年第 43 卷第 4 期.

[21] 陈胜宇,江美意.OSPF 协议及路由表的算法实现.湖北邮电技术, 2002 年 01 期.

[22] 张晨霞,周涛.IP 路由表生成方案研究. 齐齐哈尔大学学报, 2004 年 04 期.

[23] 吴彤,杨嗣超.诸鸿文.路由表快速查找算法.通信技术, 2000 年 04 期.

[24] 刘尉悦,王永纲,张万生, 王砚方.基于 Hash 和二叉树的路由表查找算法.中国科学技术大学学报, 2006 年 03 期.

[25] 曾斌,邢继峰,李之棠.OSPF 协议及路由表的算法实现. 计算机工程与应用, 2003 年 18 期.

[26] 张铭,赵海燕,王腾蛟.数据结构与算法.北京大学信息科学与技术学院 “数据结构与算法”教学小组.

[27] 王博文.通用类 TRIE 树及自动生成. 计算机应用: 第 20 卷第 12 期, 2000 年 12 月.

[28] 王智强,王振兴, 张定心.基于 TRIE 的快速路由查找算法. 信息工程大学学报, 2003 年 03 期.

[29] Pi-Chung Wang,Chia-Tai Chan,Wei-Chun Tseng,Yaw-Chung Chen.Fast trie-based routing lookup with tiny searchablecore. Global Telecommunications Conference,2002: GLOBECOM '02. IEEE Volume: 3.

[30] Shishibori,M.;Ando,K;Okada,M.;Jun-IchiAoe. A key search algorithm using the compact Patricia trie.Intelligent Processing Systems, 1997. ICIPS '97. 1997 IEEE International Conference on ,Volume: 2.

[31] Litwin, W.A.; Roussopoulos, N.; Levy,G; Hong, W. . Trie Hashing With Controlled Load;Software Engineering, IEEE Transactions on Volume: 17 , Issue: 7.

[32] 王旭, 顾乃杰,陈静.基于源转发树路由结构的多播路由表查找方案的比较. 计算机工程, 2003 年 06 期.

[33] 程青松, 王文鼐, 唐宝民.引入流量因素的路由表查找算法.南京邮电学院学报, 2002 年 04 期.

[34] Jing Fu; Hagsand, O.; Karlsson, G. . Performance evaluation and cache behavior of LC-trie

for IP-address lookup. 2006 Workshop on High Performance Switching and Routing, 2006.

[35] Yi Jiang,Fengjun Shang.Research on Multibit-Trie Tree IP Classification Algorithm.2006 International Conference on Communications,Circuits and Systems Proceedings:Volume: 3.

[36] 刘胤,杨世平.基于 RFC 算法的快速多维数据包分类算法.计算机工程, 2008 年 06 期.

[37] 喻中超,吴建平,徐恪. IP 分类技术研究. 电子学报,2001.

[38] 徐恪,徐明伟,吴建平,喻中超.IP 分类技术研究综述. 小型微型计算机系统,2002.7.

[39] 尚凤军,王海霞.基于完全无冲突哈希的数据包分类算法研究.计算机工程与应用,2004.34,173 页.

[40] 吴层.基于 IXP2400 的改进 RFC 算法的研究与设计.北京交通大学中国优秀硕士学位论文全文数据库,2008.

[41] Bo Xu,Dongyi Jiang,Jun Li.HSM: a fast packet classification algorithm. 19th International Conference on Advanced Information Networking and Applications, 2005: AINA 2005. Volume: 1.

[42] Yuke Pan,Bing Chen,Tao Xu. A Timesaving Recursive Flow Packet ClassificationAlgorithm. Wireless Communications and Trusted Computing, 2009. NSWCTC '09. International Conference on Networks Security, Volume 2 ,Publication Year: 2009 , Page(s): 442 – 445.

[43] Baboescu, F., Sumeet Singh, Varghese, G. .Packet classification for core routers: is there an alternative to CAMs? . Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. INFOCOM 2003,IEEE Societies Volume:1 Publication Year:2003, Page(s): 53-63 vol.1.

# 附录 驱动模块的业务表项结构

## 数据结构 1——主 IXP2800 部分业务表项结构

电路表索引表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | Validate | reserve1 | | | | | | | | | | | | | | | Index | | | | | | | | | | | | | | | |

电路表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | cirValidate | vir_port_flag | ATM_Pass | reserve1 | | | | | | | | | | | | | nextFwtLink：冲突电路表地址索引 | | | | | | | | | | | | | | | | |
| 1 | slot | | | | | | | | port | | | | | | | | outVlanVpi | | | | | | | | | | | | | | | |
| 2 | inVlanVci | | | | | | | | | | | | | | | | intfIndex | | | | | | | | | | | | | | | |
| 3 | gport | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | cirRxFrame | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | cirRxBytes | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | cirRxTrafficLasttick | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | cirRxTrafficBehave | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | cirRxTrafficDc | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | lockBit1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | cirDropPackets | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | cirTotalPackets | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | virGport | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 13 | ATM_slot | | | | | | | | ATM_port | | | | | | | | ATM_Vpi | | | | | | | | | | | | | | | |
| 14 | ATM_Vci | | | | | | | | | | | | | | | | reserved3 | | | | | | | | | | | | | | | |
| 15 | reserved2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

接口索引表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | Validate | reserve1 | | | | | | | | | | | | | | | | | | Index | | | | | | | | | | | | |

接口表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | Validate | reserve1 | | | | | | | | | | | | | | | | | | nextFwtLink | | | | | | | | | | | | |
| 1 | slot | | | | | | | | port | | | | | | | | outVlanVpi | | | | | | | | | | | | | | | |
| 2 | inVlanVci（接口表中的电路信息只有为网络侧时Xscale用） | | | | | | | | | | | | | | | | reserved2 | | | | | | | | | | | | | | | |
| 3 | reserved4 | uniNni | vplsBit | vpwsBit | vrfBit | mcastEn | portAcl | routeMapAcl | carAcl | statBit | cirCarBit | ucastCarBit | bcastCarBit | mcastCarBit | floodCarBit | macLearnEn | rawTag | bubMember | cFlag | urpf_mode | vir_port_flag | reserved3 | | | | | | | mac_offset | | | |
| 4 | portAclNo | | | | | | | | | | | | | | | | routeMapAclNo | | | | | | | | | | | | | | | |
| 5 | urpfAclNo | | | | | | | | | | | | | | | urpfAclFlag | vpnVrfId | | | | | | | | | | | | | | | |
| 6 | cirCarList | | | | | | | | | | | | | | | | ucastCarList | | | | | | | | | | | | | | | |
| 7 | bcastCarList | | | | | | | | | | | | | | | | mcastCarList | | | | | | | | | | | | | | | |
| 8 | floodCarList | | | | | | | | | | | | | | | | reserved6（主2400里临时保存用户转发表index） | | | | | | | | | | | | | | | |
| 9 | vlanRange1End | | | | | | | | | | | | | | | | vlanRange1Start | | | | | | | | | | | | | | | |
| 10 | vlanRange2End | | | | | | | | | | | | | | | | vlanRange2Start | | | | | | | | | | | | | | | |
| 11 | vlanRange3End | | | | | | | | | | | | | | | | vlanRange3Start | | | | | | | | | | | | | | | |
| 12 | reserved7（用来保存buffer offset） | | | | | | | | | | | | | | | | cirType（在主2400上用来保存电路表index） | | | | | | | | | | | | | | | |
| 13 | gport | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 14 | rxUcastFrame | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 15 | rxUcastBytes | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | rxBcastFrame | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 17 | rxBcastBytes | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 18 | rxMcastFrame | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 19 | rxMcastBytes | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 20 | ucastRxTrafficLasttick | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 21 | ucastRxTrafficBehave | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 22 | ucastRxTrafficDc | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 23 | lockBit2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 24 | bcastRxTrafficLasttick | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 25 | bcastRxTrafficBehave | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 26 | bcastRxTrafficDc | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 27 | lockBit3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 28 | mcastRxTrafficLasttick | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 29 | mcastRxTrafficBehave | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 30 | mcastRxTrafficDc | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | lockBit4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 32 | floodRxTrafficLasttick | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 22 | floodRxTrafficBehave | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 33 | floodRxTrafficDc | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 34 | lockBit5 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 35 | rxdropPackets | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 36 | uRpfAclDropCount | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 37 | uRpfAllDropCount | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 38 | virGport | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 39 | virParentGport | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 40-62 | reserved8[23] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

## L2TP 索引表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | Validate | reserve1 | | | | | | | | | | | | | | | | | Index | | | | | | | | | | | | | |

## L2TP 表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | valid | reserve1 | | | | | | | | | | | | | | | nextFwtLink | | | | | | | | | | | | | | | |
| 1 | tunnel_id | | | | | | | | | | | | | | | | l2tp_session_id | | | | | | | | | | | | | | | |
| 2 | slotId | | | | | | | | portId | | | | | | | | outVlanVpi | | | | | | | | | | | | | | | |
| 3 | inVlanVci | | | | | | | | | | | | | | | | reserve2 | | | | | | | | | | | | | | | |

**PPPOE,PPPOEOA 用户索引表**

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | Validate | reserve1 | | | | | | | | | | | | | | | | | Index | | | | | | | | | | | | | |

**IPOE,IPOEOA,IPOA,PPPOA 用户索引表**

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | Validate | reserve1 | | | | | | | | | | | | | | | | | Index | | | | | | | | | | | | | |

**用户转发表**

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | Validate | reserve1 | | | | | | | | | | | | | | | | | nextFwtLink | | | | | | | | | | | | | |
| 1 | reserve2 | | | | | | | | | | | | | | | | | | vrfId | | | | | | | | | | | | | |
| 2 | userIp | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | reserve3 | | | | | | | | | | | | | | | | | | portId | | | | | | | | | | | | | |
| 4 | uniVpi | | | | | | | | | | | | | | | | | | uniVci | | | | | | | | | | | | | |
| 5 | macHi32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | macLo16 | | | | | | | | | | | | | | | | | | pInternetLog | | | | | | | | | | internetLogEn | usrPqEn | usrCarEn | staticEn |
| 7 | outVlan | | | | | | | | | | | | | | | | | | inVlan | | | | | | | | | | | | | |
| 8 | carList | | | | | | | | | | | | | | | | | | pqList | | | | | | | | | | | | | |
| 9 | userAcl | vrfBit | usrCarAcl | specialAcl | webForceEn | macBindBit | reserve4 | userType | | | | | | | | | | | sessionId | | | | | | | | | | | | | |
| 10 | vbuiIndex | | | | | | | | | | | | | | | | | | userAclNo | | | | | | | | | | | | | |
| 11 | reserve6 | | | | | | | | | | | | | | | | | | carAclNo | | | | | | | | | | | | | |
| 12 | l2tpTunnelId | | | | | | | | | | | | | | | | | | l2tpSessionId | | | | | | | | | | | | | |
| 13 | l2tpSrcIp | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 14 | l2tpDstIp | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 15 | reserve7 | | | | | | | | | | | | | | | | | | l2tpUdpDstPort | | | | | | | | | | | | | |
| 16 | upstreamPacket | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 17 | upstreamBytes | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 18 | bps | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 19 | burstNormal | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 20 | burstMac | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 21 | reserve8 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 22 | upLasttick | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 23 | upBehave | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 24 | upDc | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 25 | lockBit1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 26-27 | reserved9[2] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 28 | rxDropPackets | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 29-31 | reserve[3] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

多播 ACL 索引表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | Validate | reserve1 | | | | | | | | | | | | | | | | | | | | | Index | | | | | | | | | |

多播 ACL 表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | valid | reserve1 | | | | | | | | | | | | | | | | | | | | | nextFwtLink | | | | | | | | | |
| 1 | sourceIp | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | groupIp | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | reserved2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

组播路由 hash 索引表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | sourceIp | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | group(dst_ip) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | bitmapSlot | | | | | | | | | | | | | | | | reserved | | | | | | | | | | | OUTPORT_VALID | UPSEND_LIMIT | ASSERT_LIMIT | FORWARD | UPSEND |
| 3 | nextIndex | | | | | | | | | | | | | | | | mrFwdIndex | | | | | | | | | | | | | | | |

组播路由转发表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | reserved | | | | | | | | | | | | | | | | Global_outport_block_id | | | | | | | | | | | | | | | |
| 1 | gport(slot_8+port_8+external_vlan_16) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | fwdCounter(转发包计数器地址，指向SRAM) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | revCounter(接收包计数器地址，指向SRAM) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | lastSourceIp（在（*，G）匹配中用当前包的源ip更新此项） | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | reserved_1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | reserved_2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | reserved_3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

路由表下一跳表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | load_blance_ptr | | | | | | | | | | | vpn_aggre_flag | vbui_inter_flag | host_route_flag | l2tp_index_flag | vpws_natpt_flag | rsvpte_frr_flag | vpls_spoke_flag | vpls_flag | sinf_decap_flag | fins_encap_flag | vpws_flag | local_addr_flag | smart_trunk_flag | ld_flag | nat_flag | loopback_flag | outlabel_valid_flag | direct_link_route_flag | local_and_forward_flag | abort_packet_flag | local_packet_flag |
| 1 | next_jump_IP | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | slotId | | | | | | | portId | | | | | | | | | outVlanVpi | | | | | | | | | | | | | | | |
| 3 | inVlanVci | | | | | | | | | | | | | | | | smartGroupId | | | | | | | | | | | | | | | |
| 4 | reserved3 | | | | | | | | | | | | | | | | MPLS_outer_label | | | | | | | | | | | | | | | |
| 5 | VPN_ID | | | | | | | | | | | | | | | | MPLS_inner_label1 | | | | | | | | | | | | | | | |
| 6 | bak_label_offset | | | | | | | | | | | | | | | | MPLS_inner_label2 | | | | | | | | | | | | | | | |
| 7 | reserved5 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

## 数据结构 2——从 IXP2800 各种转发表表项结构

用户转发索引表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | Validate | reserve1 | | | | | | | | | | | | | | | | | Index | | | | | | | | | | | | | |

用户转发表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | Validate | reserve1 | | | | | | | | | | | | | | | | | nextFwtLink | | | | | | | | | | | | | |
| 1 | reserve2 | | | | | | | | | | | | | | | | vrfId | | | | | | | | | | | | | | |
| 2 | userIp | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | uniVpi | | | | | | | | | | | | | | | | uniVci | | | | | | | | | | | | | | |
| 4 | macHi32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | macLo16 | | | | | | | | | | | | | | | | sessionId | | | | | | | | | | | | | | |
| 6 | logIndex | | | | | | | | userType | | | | | | | | portId | | | | | | | logEn | tcpRate | tcpSum | tcpDisplay | usrAcl | usrPqEn | usrCarEn | staticEn |
| 7 | outVlan | | | | | | | | | | | | | | | | inVlan | | | | | | | | | | | | | | |
| 8 | carList | | | | | | | | | | | | | | | | pqList | | | | | | | | | | | | | | |
| 9 | reserve5 | | | | | | | | | | | | | | | usrCarAcl | userAcLNo | | | | | | | | | | | | | | |
| 10 | downPacket | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | downBytes | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | bps | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 13 | burstNormal | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 14 | burstMac | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 15 | reserve3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | downLasttick | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 17 | downBehave | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 18 | downDc | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 19 | lockBit1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 20 | tcpSessionCounted | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 21 | tcpSessionTotal | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 22 | tcpSessionBandwith | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 23 | lockBit2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 24-29 | reserve[6] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 30 | txDropPackets | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | reserved4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

电路表索引表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | Validate | reserve1 | | | | | | | | | | | | | | | Index | | | | | | | | | | | | | | | |

电路表

| | 31 | 30 | 29-16 | 15-0 |
|---|---|---|---|---|
| Word 0 | cirValidate | vir_port_flag | reserve1 | nextFwtLink：冲突电路表地址索引 |
| 1 | slot | | port | outVlanVpi |
| 2 | inVlanVci | | | intfIndex |
| 3 | gport | | | |
| 4 | cirTxFrame | | | |
| 5 | cirTxBytes | | | |
| 6 | cirTxTrafficLasttick | | | |
| 7 | cirTxTrafficBehave | | | |
| 8 | cirTxTrafficDe | | | |
| 9 | lockBitI | | | |
| 10 | virGport | | | |
| 11 | cirDropPackets | | | |
| 12 | cirTotalPackets | | | |
| 13-15 | Reserved[3] | | | |

接口索引表

| | 31 | 30-13 | 12-0 |
|---|---|---|---|
| Word 0 | Validate | reserve1 | Index |

接口表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | Validate | reserve1 | | | | | | | | | | | | | | | nextFwtLink | | | | | | | | | | | | | | | |
| 1 | slot | | | | | | | | port | | | | | | | | outVlanVpi | | | | | | | | | | | | | | | |
| 2 | inVlanVci（接口表中的电路信息只有为网络侧时Xscale用） | | | | | | | | | | | | | | | | vpnVrfId | | | | | | | | | | | | | | | |
| 3 | gport | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | reserved3 | uniNni | vplsBit | vpwsBit | vrfBit | mcastEn | portAcl | routeMapAcl | carAcl | statBit | cirCarBit | ucastCarBit | bcastCarBit | mcastCarBit | reserved2 | rawTag | qInqType | | | | | | | | | | | | | | | |
| 5 | mac_offset | | | | | | | | cirType | | | | | | | | portAclNo | | | | | | | | | | | | | | | |
| 6 | cirCarList | | | | | | | | | | | | | | | | ucastCarList | | | | | | | | | | | | | | | |
| 7 | bcastCarList | | | | | | | | | | | | | | | | mcastCarList | | | | | | | | | | | | | | | |
| 8 | txUcastFrame | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | txUcastBytes | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | txBcastFrame | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | txBcastBytes | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | txMcastFrame | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 13 | txMcastBytes | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 14 | ucastTxTrafficLasttick | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 15 | ucastTxTrafficBehave | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | ucastTxTrafficDc | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 17 | lockBit2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 18 | bcastTxTrafficLasttick | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 19 | bcastTxTrafficBehave | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 20 | bcastTxTrafficDc | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 21 | lockBit3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 22 | mcastTxTrafficLasttick | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 23 | mcastTxTrafficBehave | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 24 | mcastTxTrafficDc | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 25 | lockBit4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 26 | txDropPackets | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 27-31 | reserved5[5] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

二层封装表索引表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | Validate | reserve1 | | | | | | | | | | | | | | | Index | | | | | | | | | | | | | | | |

二层封装表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | valid | reserve1 | | | | | | | | | | | | | | | nextFwtLink | | | | | | | | | | | | | | | |
| 1 | nextHopIp | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | vrfId | | | | | | | | | | | | | | | | nextHopMacHi16 | | | | | | | | | | | | | | | |
| 3 | nextHopMacLo32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

L2TP 索引表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | Validate | reserve1 | | | | | | | | | | | | | | | | | Index | | | | | | | | | | | | | |

L2TP 表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | valid | reserve1 | | | | | | | | | | | | | | | | | nextFwtLink | | | | | | | | | | | | | |
| 1 | tunnel_id | | | | | | | | | | | | | | | | | | l2tp_session_id | | | | | | | | | | | | | |
| 2 | userTableIndex | | | | | | | | | | | | | | | | | | reserve2 | | | | | | | | | | | | | |
| 3 | reserve3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

TCP 限数索引表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | Validate | reserve1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | next_index | | | | | | | | | | | | | | | | | | |

TCP 限数表

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 0 | valid | used | reserved | | | | | | | | | | | | | | vrfId | | | | | | | | | | | | | | | |
| 1 | srcIpaddr | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | dstIpAddr | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | srcPortId | | | | | | | | | | | | | | | | | | | | | | dstPortId | | | | | | | | | |
| 4 | nextFwtLink | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | nextPointer | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | prePointer | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | timeStamp | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

# 攻读硕士学位期间的学术论文

## 一、已发表的学术论文：2 篇

一种基于网络拓扑和流量特征的P2P流量识别与控制方法. 全国计算机新科技与计算机教育论文集(2009). （第一作者）

基于BRAS和PPPoE的地址池管理改进模型的性能对比研究 全国计算机新科技与计算机教育论文集(2009). （第二作者）

# 攻读硕士学位期间参加的科研项目

1、国家自然基金项目（60973140/F0208）

2、江苏省自然基金项目（BK2009425）

3、江苏省高校自然基金项目（08KJB520005）

4、江苏省六大高峰项目

5、江苏省青蓝工程学术带头人资助项目

6、中兴通讯基金项目