

FPGA Accelated Switching Platform

FAST 开源平台原理与应用 (2017. 12)

注：此版本只用于了解 FAST 平台原理，参与 Fast 平台设计或 FAST 用户可访问 www.fastswitch.org 获取最新版本。

目 录

1.FAST 概述	1
1.1FAST 产生背景	1
1.1.1 网络设备平台化趋势与开源技术	1
1.1.2 对数据平面开源项目的需求	2
1.1.3FAST 项目起源——iRouter	4
1.2FAST 的目标与意义	6
1.2.1FAST 的目标	7
1.2.2FAST 项目的意义	7
1.3FAST 平台组成	9
1.3.1 总体架构	9
1.3.2FPGA OS	10
1.3.3FAST 支撑软件	11
1.3.4 用户相关软硬件	12
1.4FAST 规范	13
1.4.1UM 接口规范	13
1.4.2FAST API 规范	13
1.4.3FAST 分组格式	14
1.4.4 硬件地址空间划分	14
2.FAST 分组处理模型	15
2.1 功能模块	15
2.1.1 功能模块定义	15
2.1.2 模块的实现模型	16
2.2 分组处理流水线	17
2.2.1 分组处理流水线定义	17
2.2.2 流水线等价类	19
2.2.3 软硬件紧耦合流水线	19
2.3FAST 软硬协同处理模型	20
2.3.1 模型的基本原理	21
2.3.2 分组处理流水线的实现	24
2.3.3 分组处理流水线的动态扩展	24
3 元数据	28
3.1 元数据格式	28
3.1.1 元数据包含的内容	28
3.1.2 流水线中的元数据	29
3.1.3 元数据的定义	29
3.2 元数据的处理要求	30

3.2.1 输入端口号	31
3.2.2 流水线生存时间	31
3.2.3 分组长度	31
3.2.4 源模块号	31
3.2.5 目的模块号	32
3.2.6 输出端口指示	32
3.2.7 分组目的标识	32
3.2.8 分组来源标识	32
3.2.9 分组序列号	33
3.2.10 分组流标识	33
3.2.11 分组丢弃标识	33
3.2.12 分组优先级	33
3.2.13 分组时间戳	34
3.2.14 应用定义字段	34
3.3 基于元数据的分组转发控制	34
3.3.1 组播与单播	34
3.3.2 统计信息生成	36
3.3.3 分组丢弃	36
4 分组特征向量	37
4.1 分组特征向量的组成	37
4.1.1 协议标准头	37
4.1.2 协议类型标识	38
4.1.3 用户定义标识	39
4.2 协议标准类型的应用	40
4.2.1 关键字提取	40
4.2.2 分组处理流程定向	41
4.3 用户定义标记的使用	42
4.3.1 多维分组分类	42
4.3.2 简化查表关键字	43
4.3.3 应用加速	43
5.Fast 硬件流水线	44
5.1 与 FPGA OS 的接口	44
5.1.1 信号接口	44
5.1.2 接口分组格式	45
5.1.3 查表接口	46
5.1.4 接口信号与时序定义	47
5.2.流水栈组成	48
5.2.1 用户定义解析器 (UDP)	50

5.2.3 用户定义关键字提取 (UKE)	51
5.2.3 通用查表引擎 (GME)	55
5.2.4 用户定义动作 (UDA)	56
5.2.5 通用输出引擎 (GOE)	57
5.3 匹配动作表	58
6. 用户应用 (UA) 编程	59
6.1 硬件抽象	59
6.1.1 虚拟地址空间	59
6.1.2 FAST 分组	63
6.1.3 重定向表规则	63
6.2 编程模型	64
6.2.1 ODP 编程模型的特点	64
6.2.2 与现有编程模型比较	65
6.3 编程 API	66
6.3.1 UA 管理 API	66
6.3.2 规则管理 API	67
6.3.3 硬件管理 API	67
6.3.4 订制功能 API	68
6.4 其他应该考虑的问题	68
6.4.1 UA 在设备中的角色	68
6.4.2 UA 的地址	69
7. FAST 原型系统实现与应用	70
7.1 iRouter 实现	70
7.1.1 iRouter 系统组成	70
7.1.2 iRouter 的实现	74
7.1.3 软硬件交互的实现	75
7.2 对 LISP 协议的扩展支持	78
7.2.1 LISP 协议工作原理	78
7.2.2 LISP XTR 功能的实现	81
7.2.3 LISP 转发功能的应用	84
7.3 对假冒 IP 源地址识别的扩展支持	85
7.3.1 假冒源 IP 地址识别原理	85
7.3.2 假冒源 IP 地址识别的实现	87
7.3.3 假冒源 IP 地址识别的应用	89
7.4 对精准测量服务的扩展支持	92
7.4.1 精准测量服务的原理	92
7.4.2 AMS 的实现	93
7.4.3 AMS 的评估与应用	96

8.结束语	98
-------------	----

1. FAST 概述

1.1 FAST 产生背景

1.1.1 网络设备平台化趋势与开源技术

随着软件定义网络（SDN）和网络功能虚拟化（NFV）等技术的出现和快速发展，使得网络设备的实现架构逐渐呈现出平台化的发展趋势。由于开源具有鼓励设计重用，避免“重复发明轮子”，摆脱用户对单一开发商的依赖，降低开发成本和缩短开发周期等优点，开源方法也迅速成为推动网络平台发展，提升其技术成熟度的重要手段。

目前网络平台相关的开源项目主要体现在管理控制平面，包括 SDN 控制器软件（ONOS，ODL），NFV 管理软件（OPNFV）等。这些开源项目主要由工业界主导，发展迅速，对 SDN，NFV 的快速发展和落地应用起到了重要的作用。

而 SDN、NFV、5G 等技术的发展，急需网络数据平面的创新。当前网络设备数据平面相关的开源项目相对较少，典型的是 OpenVSwitch（OVS）。OVS 完全由软件实现数据平面 SDN 交换，不但性能较低，难以在千兆以上的网络环境中应用，而且对基于硬件的分组交换技术研究参考价值不大。斯坦福大学在十多年前发起了 NetFPGA 项目，基于 Xilinx FPGA 实现了路由器，交换机和 openflow 交换机等功能，但近年来发展十分缓慢，特别在技术支持和服务方面难以满足科研人员的需求。

1.1.2 对数据平面开源项目的需求

新型交换技术的研究对网络数据平面开源项目具有强烈需求。例如有状态的 SDN 数据平面处理，基于 SDN 交换的安全防护，以及段路由与 SDN 的结合都是当前路由交换技术研究的热点，都需要对现有的 SDN 交换机数据平面进行扩展。如果没有相关开源平台进行支撑，研究人员很难从零实现一个全新的，支持上述创新功能的数据平面。

下一代网络处理器芯片和交换芯片体系结构研究对数据平面开源项目也有强烈的需求。下一代网络处理器和交换芯片是 SDN 和 5G 网络设备研制的核心，也是当前学术界最前沿的领域之一。由于缺少支撑研究的开源平台，芯片架构和关键技术的创新只能依赖网络芯片和网络设备制造商进行，大量科研机构中研究人员的创新思想难以得到验证和应用。

因此，当前需要一个针对网络数据平面的开源项目，该项目的主要特点包括：

（1）基于多核 CPU 和 FPGA 的开源平台

能够同时兼顾分组交换处理的可编程性与性能，以及实现平台易于获得是数据平面开源项目的首要要求。近年来，FPGA 发展迅速，不但可以与 CPU 通过 PCIe 接口高速通信，其自身内部的逻辑容量，内嵌 RAM 容量，以及 DDR 和网络高速接口 IP 都可以满足 40Gbps 以上网络处理的需求。特别是 NetFPGA 的成功经验表明，FPGA 平

台要比网络处理器平台更加易于获得，更加易于编程，而且 FPGA 编程的 Verilog 实现代码跨平台移植方便，易于满足不同用户的需求。

（2）实现基本的数据平面交换功能

数据平面的开源项目需要一个可扩展的框架，但不能仅仅停留在框架上。必须有先行者基于开源项目框架首先实现一些基本的网络交换功能，例如 L2 交换，L3 转发和 openflow 交换等。这样可以吸引更多用户关注和使用开源项目代码搭建自己的网络环境。这些用户在使用中的反馈是推进开源项目走向成熟的关键。

（3）支持用户基于开源平台实现自己的创新工作

用户可以在开源平台基础上实现自己的创新工作，而不用重新设计一个完整的数据平面。例如用户可以在硬件中增加定制模块来支持新的协议处理，或是在硬件逻辑中设置一个“钩子”，将指定流量定向到 CPU，通过编写软件代码实现对特定流量施加特定处理。

（4）用户得到更好的技术支持

由于参与新型交换技术的研究人员在工程实现方面的经验差距很大。例如电子工程专业的可能擅长硬件设计，而计算机专业的可能擅长软件编程；以创新研究和撰写高水平论文为目标的研究人员更加注重原理的快速验证，而一些企业希望利用开源项目能够快速实现产品原型。不论目的如何，用户都希望能够在开源社区获得更加专业，甚至是定制的服务。

此外，开源项目不应涉及太多的商业利益纷争，开源实现的代码不能绑定具体的 FPGA 型号和 CPU 类型。因此开源项目在软硬件设计时需要显式划定平台相关实现与平台无关实现的界限，定义清晰的平台相关实现与平台无关实现间的接口规范。从而支持开源项目适合更多的平台，给用户更多地选择。

1.1.3FAST 项目起源——iRouter

2016 年，在国家 863 项目“新型动态网络关键技术及验证”的研究过程中，国防科技大学等单位基于通用 CPU 加 FPGA 平台研制了可编程网络交换平台（iRouter），iRouter 基于 FPGA 实现了万兆接口上分组的解析，关键字提取，查表转发和输出调度等操作，通过 CPU 与 FPGA 协同实现数据平面功能的动态扩展，包括对 LISP（位置标签分离协议）转发的支持，对 IPv6 路径 MTU 发现的支持以及对 DDOS 攻击检测的支持等。该平台在北京，南京和长沙的多个科研单位部署应用，这些平台通过 CNGI 网络连接，基于 LISP 协议构建了跨广域的 IPv6 新型编址与路由实验网。实验网在支撑 IPv6 新型编址与路由的规模验证方面发挥了重要作用。

iRouter 基于多核 CPU 与 FPGA 的协同处理实现了可扩展的分组转发功能，具有以下特点：

（1）在 FPGA 中实现了包含分组解析，查表关键字生成，匹配查表和输出调度等模块的分组处理流水线，该流水线可以线速处理万兆接口到达的分组。根据预先配置的规则，分组可以直接交换到输出

接口发出，也可以定向到 CPU 进行进一步处理；

(2) CPU 上的软件分为数据平面功能扩展软件和控制平面软件。硬件流水线可以根据规则的动作 (Action) 确定将分组送特定的数据平面扩展软件还是控制平面软件；

(3) CPU 上的数据平面扩展软件或控制平面软件可以将发出的分组通过 DMA 再次送到硬件流水线进行转发；

(4) 用户可以通过对硬件流水线的配置，将特定分组送给 CPU 用户空间的指定进程中，而该进程对分组处理后，还能再次将分组送回硬件流水线。因此用户可以在用户空间编写一个进程，实现特定的网络功能，并通过将该功能嵌入到分组处理流水线中，实现数据平面功能的扩展。

此外，FPGA 内部的交换流水线也可以进行扩展。例如，我们在动作执行模块后传接了一个流计数器模块，根据配置，对特定数据流双向到达的分组数目和字节数目进行统计，通过检测流量的不对称性来检测 DDOS 攻击的发生。

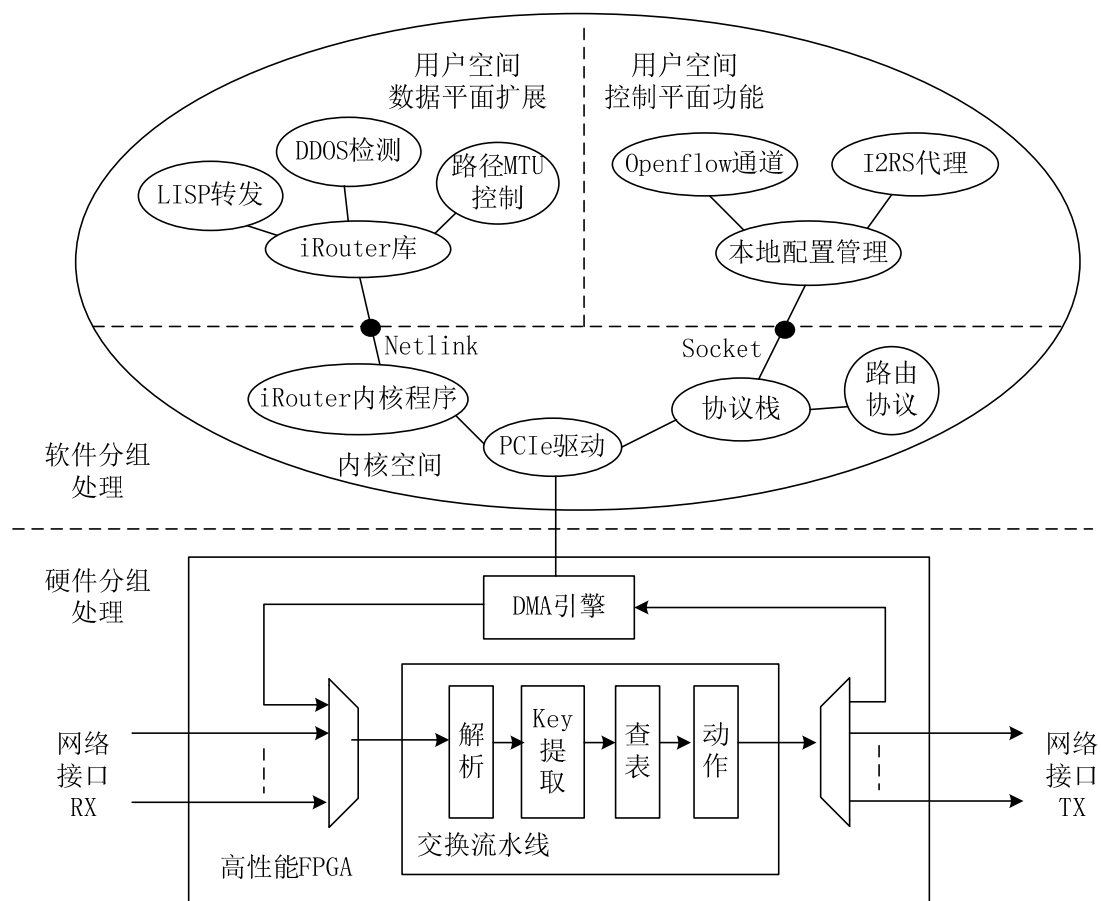


图 1-1 iRouter 的软硬件协同处理实现结构

因此，iRouter 除了实现基本的 openflow 交换功能外，还提供了一种数据平面功能扩展的机制，支持用户通过对 FPGA 中的交换流水线扩展，或者增加新的数据平面处理进程来扩充交换平台的分組处理功能。

为了进一步完善 iRouter 的设计并支持更多用户使用，我们公开了 iRouter 实现架构的细节，FPGA 中交换流水线的 Verilog 实现代码，以及 iRouter 内核程序和 iRouter 库的代码，形成了 FAST 开源项目。

1.2 FAST 的目标与意义

1.2.1 FAST 的目标

FAST 项目一方面基于 iRouter 平台采用的多核 CPU 加 FPGA 的实现架构，能够充分发挥多核 CPU 和 FPGA 两者结合带来的灵活性和高性能的优点。另一方面，FAST 还需进一步清晰的描述其软硬件协同的开放架构，规范其软硬件开发接口，激励第三方研究人员开发和贡献更多的案例，以及吸引更多的用户。

具体的说，FAST 需要在 iRouter 代码基础上，进行三方面的工作。

一是明确 FAST 软硬件协同分组处理的模型，支持在标准的硬件流水线中嵌入新的硬件处理模块，或将用户空间的用户应用相关的软件进程插入流水线，扩充流水线的功能；

二是建立一套完整的 FPGA 流水线设计规范，用户根据规范进行各种自定义模块的设计，易于插入并集成到现有的流水线中，支持最大程度实现模块的重用性，模块内部的逻辑实现不依赖其他外部模块的需求；

三是建立一套用户应用进程的编程 API，提供并不断扩充 FAST 编程库，支持用户按照类 socket 编程的方式，在用户空间实现转发平面的处理功能。

1.2.2 FAST 项目的意义

FAST 开源项目对推动当前新型路由交换技术研究具有重要意义，主要表现在以下三方面。

(1) 提供新型交换设备的解决方案

基于通用多核 CPU 和 FPGA, FAST 项目已经实现了基本的路由交换功能, 并且软硬件代码完全开源。因此用户可以直接基于 FAST 构建自己的交换设备。

FAST 目前提供的解决方案包括:

- L2 交换机
- L3 路由器
- SDN 交换 (openflow1.3)
- 精准测量服务
- LISP XTR 路由器

(2) 支撑新型交换技术的创新研究与实验

FAST 采用可扩展的软硬件紧耦合分组处理模型, 支持用户在平台已有实现基础上, 通过对 FPGA 硬件流水线的扩充, 或基于 FAST 库编写用户空间程序实现对网络设备数据平面的扩充, 从而实现对各种新型的路由交换机制进行实验研究。

FAST 架构支持的新型交换技术创新研究包括:

- SDN 交换设备研究
- NFV 硬件加速研究

-
- 智能网卡研究
 - 下一代网络处理器研究
 - 下一代交换芯片研究

(3) 新型交换技术研究的交流平台

FAST 社区将为新型路由交换技术的研究群体提供交流的平台。2016 年 10 月，第一次 FAST 开源项目工作会议在长沙举行，来自工业界，学术界和教育界的 20 个单位的 33 名研究人员成为 FAST 项目的成员。经过 1 年多的发展，目前已经有超过 30 个单位或研究组使用 FAST 进行科研和教学，主要涉及的方向包括：

- 新型网络体系结构研究和原理验证
- 下一代交换技术研究
- 可重构网络设备实现技术研究
- 低年级本科生路由交换技术实验教学
- 高年级本科生计算机网络综合实验
- 研究生的网络空间安全创新实验

1. 3FAST 平台组成

1.3.1 总体架构

FAST 定义了平台相关逻辑/代码和平台无关逻辑/代码之间的接口规范。其中 FPGA 中的硬件流水线的实现与具体的平台无关，又称

为用户模块（UM）。用户可对 UM 进行扩充，在硬件流水线中定制自己的功能，也可利用标准的 FAST UM，通过编写软件程序扩展分组处理功能。用户基于 FAST API 编写的用户空间程序称为用户应用（UA）。

FAST 平台的组成如图 1-2 所示，主要由平台相关软硬件，用户相关软硬件以及 FAST 支撑软件三部分组成。

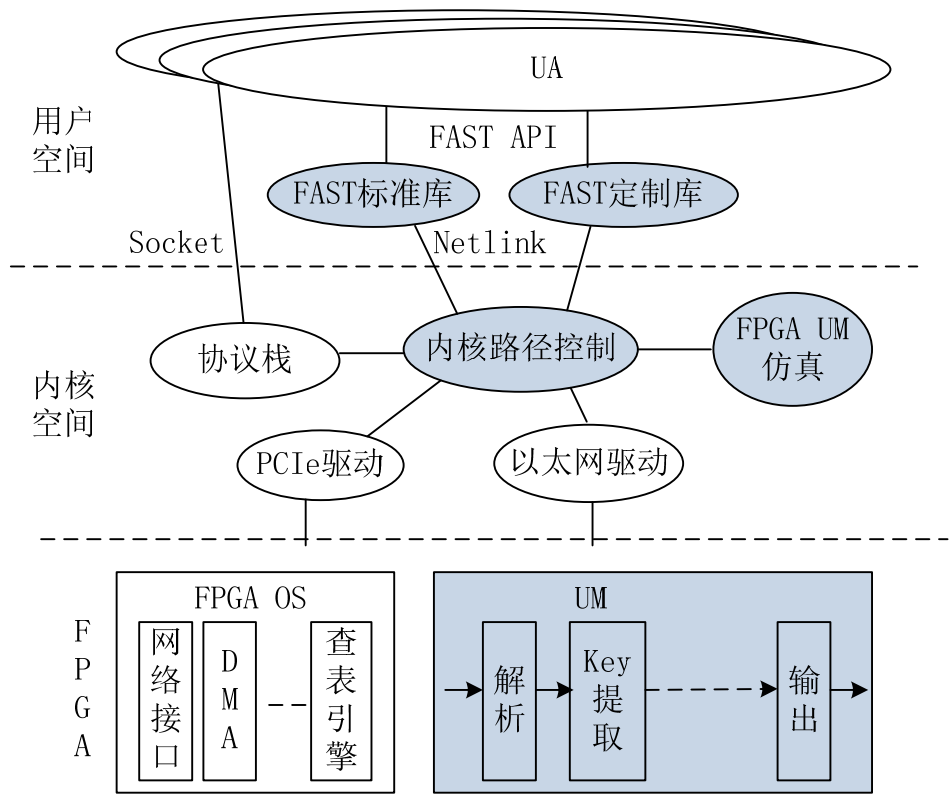


图 1-2 FAST 平台的总体组成

1.3.2FPGA OS

FAST 平台的 FPGA 中，除了平台无关的 UM 外，还有平台相关的逻辑，称为 FPGA OS。FPGA OS 主要有两个功能，一是屏蔽平台

的异构性，使得 UM 的 verilog 代码具有更好的跨平台移植能力；二是为 UM 的实现提供共性的服务。

FPGA OS 屏蔽不同 FPGA 平台的异构性主要包括五方面。一是不同的 FPGA 型号，如来自不同厂商，具有不同的开发工具，FPGA 具有不同的容量和速度等级等；二是不同的网络接口，如千兆以太网接口和万兆以太网接口，以及不同的接口数目等；三是不同的外部接口类型，如有的连接 TCAM 协处理器，有的连接 DDR 存储器，有的连接 SRAM 存储器等；四是不同的接口逻辑实现方式，例如有的以太网 MAC 逻辑在 FPGA 内嵌的 IP 核实现，有的 MAC 逻辑由 FPGA 外部的 PHY/MAC 一体的芯片实现。不同的接口逻辑实现方式获取接口状态，如 up/down，的方式也不同；五是 FPGA 与 CPU 不同的通信方式，如果 FPGA 与 CPU 在同一个板卡上或机箱内，FPGA 可能通过 PCIe 总线与 CPU 进行通信，否则，FPGA 可能通过以太网与物理位置相分离的 CPU 进行通信。

FPGA OS 为简化 UM 设计提供各种通用服务，如分组的接收和发送，外部存储器的访问，与 Cpu 通信的高速 DMA，以及分组处理中需要的规则匹配等。

1.3.3FAST 支撑软件

FAST 支撑软件包含 Linux 用户空间实现的 FAST 库，在 Linux 内核中实现的 FAST 内核路径控制，以及内核中的 UM 仿真软件三部分。

FAST 库又分为 FAST 标准库和 FAST 定制库。FAST 标准库对应标准的 FAST UM 实现，为 UA 提供分组收发，规则配置，平台信息获取（如接口计数器）等功能；FAST 定制库实现对 UM 中用户扩充模块的管理和访问。FAST 库通过 Netlink 机制与内核中的 FAST 路径控制软件通信。

FAST 内核路径控制软件主要在内核中实现多个 UA 与 UM 通信的 MUX/DMUX 功能。内核 UM 仿真软件主要实现对 FPGA 中 UM 功能的仿真，在不包含 FPGA 的标准终端和服务器上支持 UA 的运行。

1.3.4 用户相关软硬件

用户相关软硬件是用户根据自己实验需求开发的，与平台无关的软件 UA 和硬件 UM。用户功能可根据开发时间和性能要求等限制条件在 UA 和 UM 中划分。通常功能的主体在 UA 中实现，UM 实现对特定功能，如规则匹配，的加速。

FAST 编程库为 UA 开发提供调用平台资源的接口。用户的 UA 设计也完全不用考虑底层 FPGA 的实现方式，只是调用 FAST API 提供的函数接收分组，发送分组，对 FPGA 中抽象出来的 UM 进行编程控制等。除了 FAST API，UA 也可调用 Linux 平台提供的标准系统调用，如 socket，进行编程。

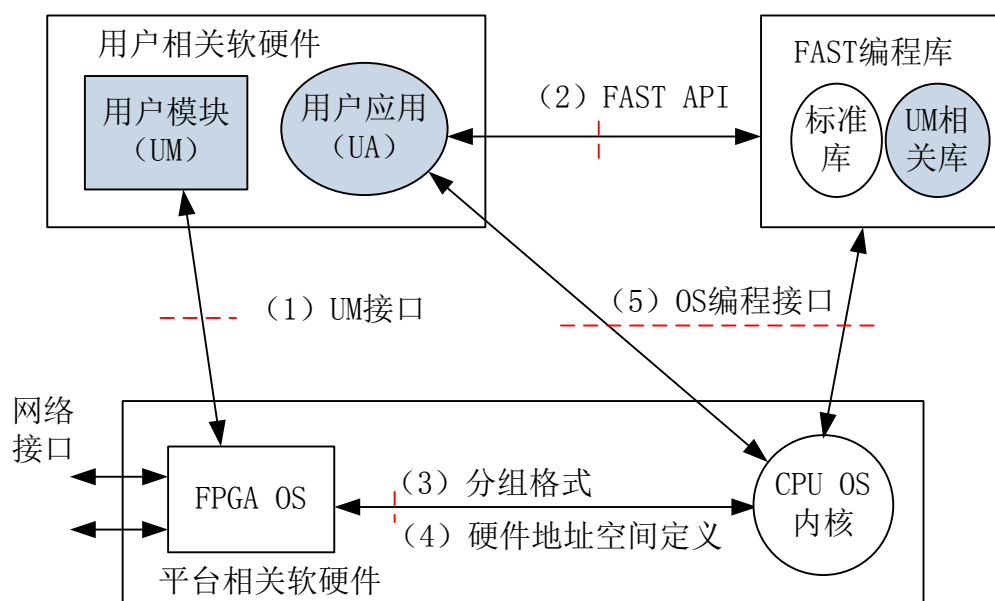


图 1-3 FAST 用户相关软硬件的开发接口

1.4FAST 规范

用户基于 FAST 平台进行实验开发需要遵循相关的规范，主要包括 UM 接口规范，FAST API，FAST 分组穿越软硬件界面时的格式，以及 FAST 平台硬件空间地址的定义等。

1.4.1UM 接口规范

UM 接口规范定义了 UM 逻辑与 FPGA OS 的接口信号和工作时序，通常 UM 与 FPGA OS 的接口包括接收和发送分组的接口，接收 UA 管理的接口，调用 FPGA OS 提供其他各类服务的接口等。

1.4.2FAST API 规范

FAST API 规范定义了 UA 访问底层资源使用的关键数据结构和相关函数的定义等。FAST 平台为 UA 开发提供了分组收发，规则管理等标准的 API，实现这些标准 API 的库称为标准库。如果用户在

UM 中需要增加一些特殊的处理功能，如深度内容检测（DPI），对这些特殊资源的管理使用就需要开发相应的 API 实现这些 API 的库称为 UM 相关库。

1.4.3FAST 分组格式

FAST 分组格式定义了网络分组在穿越软硬件界面时需要携带的元数据类型和格式定义等。

1.4.4 硬件地址空间划分

硬件地址空间规范定义了 FPGA OS 和 UM 硬件中软件可访问资源，如寄存器，计数器和存储器等，在独立的硬件地址空间中的定义。唯一的硬件地址空间定义是 UA 和 FAST 库跨硬件平台可移植的前提。

2. FAST 分组处理模型

FAST 面向通用 CPU 加 FPGA 的异构处理平台实现, 采用了软硬件协同的处理模型。对处理性能要求较高的通用处理功能由 FPGA 硬件流水线实现, 用户特定需求相关的处理功能由用户空间的软件编程实现。

2.1 功能模块

2.1.1 功能模块定义

FAST 中的模块是能够实现特定网络处理功能的, 相对独立的一个软件程序或硬件逻辑块。这些模块能够按需连接, 组成上下游关系, 形成分组处理流水线。FAST 模块具有一些共有的属性。

(1) 用户可见性

每个模块需实现特定的, 用户关心并可见的分组处理功能, 如分组协议解析和输出调度等。一些用户应用不可见且不关心的功能实现不能称为模块, 例如实现软硬件通信的 DMA 引擎, 内核驱动程序, FPGA 中的 DDR 接口控制器等不能作为模块。

(2) 不可拆分性

从网络功能实现的角度, 每个模块都不需再划分成多个更小的模块。例如可将 CPU 内核中的协议栈看成一个模块, 因为用户编程时只需考虑把接收的分组 (如 ARP 分组, ICMP 分组等) 送协议栈处理, 而不需要考虑这些分组由协议栈中具体哪个处理程序实现。

（3）自包含性

模块间是相互独立的，在功能实现上没有明确的功能依赖或调用关系。模块之间只存在上下游关系，不存在父模块和子模块关系。即不存在一个模块由多个其他模块组成。

（4）可重用性

模块的实现只需遵循一定的规范即可，不能对其上游或下游其他模块的实现提出要求。不同的模块可能来自不同的开发人员，但这些模块可以根据需要实现不同的组合，有效的减少分组流水线开发复杂度。

FAST 模型中，不论是软件模块还是硬件模块，是硬件流水线中的固定模块，还是动态插入的模块，每个模块都用唯一的模块 ID（MID）标识。

2.1.2 模块的实现模型

FAST 模块的实现模型如图 2-1 所示。每个模块输入包括分组信息 P 和元数据 M（Metadata），经过处理后，将修改后的分组 P 和元数据 M 传递给下一个模块。

每个模块具有自己的 MID，称为本地 MID（LMID）。分组元数据 M 中携带接收分组的模块 ID，即 DMID。每个模块接收到分组后，首先判断 M 中的 DMID 是否等于 LMID。DMID 等于 LMID 表示本地模块需要处理该分组，否则本地模块直接将分组 P 及其元数据 M

FAST 的分组处理流水线具有无回路,可服用和可定制三个特征。

(1) 无回路特性

每个序列的处理采用模块顺序执行的方式,不形成回路,即序列 $P(i) = \langle M_{i1}, M_{i2} \dots M_{iN} \rangle$ 的处理顺序为 $M_{i1} \rightarrow M_{i2} \rightarrow \dots \rightarrow M_{iN}$, 不会出现处理完 M_{i2} 又跳转回处理 M_{i1} 的情况。若一个分组处理中需要经过某个模块多次,那么这个模块的 MID 将多次出现在这个流水线序列中。因此模块序列中不存在 goto 跳转问题,序列中任何两个模块 MID 不相等,处理序列中不形成回路。

(2) 可复用性

同一个处理模块可以位于不同的分组处理序列中,即序列 $P(i)$ 和序列 $P(j)$ 中可能存在相同的模块。例如以太网协议分析模块可能出现在 IPv4 和 IPv6 的分组处理序列中。

(3) 可定制性

处理序列可根据用户需求进行动态调整,即:某一时刻序列 $P(i)$ 的处理序列为 $\langle M_{i1}, M_{i2} \dots M_{iN} \rangle$, 但下一刻用户根据需求可以改为 $\langle M_{i1}, M_{i7} \dots M_{iN} \rangle$, 即跳过了 M_{i2} 到 M_{i6} 共 5 个模块,或者变成 $\langle M_{i1}, M_{ix}, M_{i2} \dots M_{iN} \rangle$, 即在模块 M_{i1} 到 M_{i2} 之间插入模块 M_{ix} 。

以 IPv6 分组转发流水线 $P(i) = \langle M_{i1}, M_{i2} \dots M_{iNi} \rangle$ 为例,该分组处理流水线主要由支持分组接收、以太网分组解析、IPv6 协议解析、查表及转发执行模块等组成。若该流水线功能需要扩展,还需要支持

VxLAN 协议，那么流水线 $P(i)$ 只需在原有流水线的基础上增加 VxLAN 协议解析模块和 ~~VxLAN 协议处理模块~~，改变模块之间的连接的路径即可。

2.2.2 流水线等价类

FAST 的流水线等价类 (FEC) 是指对于分组 i 和分组 j ，若 $P(i) = P(j)$ ，则分组 i 和分组 j 属于相同的流水线等价类。

流水线等价类 (FEC) 是使用相同处理流水线的分组集合。例如具有相同五元组标识的所有分组可属于一个流水线等价类。某些具有相同协议的分组，如所有的 ARP 分组，也可以是一个流水线等价类。

在网络交换设备中，需要实现的分组处理流水线的个数与需要支持的流水线等价类的数目相等。因此网络交换设备所支持的功能主要由其包含的所有的分组转发流水线的集合确定，对外反应了其支持的流水线等价类的类型。

若网络中要支持新的协议，如 VxLAN，需要转发平面支持一个或多个新的流水线等价类，即转发平面需要构建新的转发流水线。构建新的转发流水线有两种方法，一是将现有的模块进行重新组合，二是增加一个新的功能模块。

2.2.3 软硬件紧耦合流水线

在 FAST 实现模型中，设硬件所有模块的集合为 M_H ，软件所有模块的集合为 M_S 。对于流水线 $P(i) = \langle M_{i1}, M_{i2}, \dots, M_{iN_i} \rangle$ ，若存在

$x, y(x \neq y), M_{ix} \in M_H, M_{iy}$, 那么则称这条流水线是软硬件协同的。

例如在 $P(i) = \langle M_{i1}, M_{i2}, \dots, M_{iNi} \rangle$ 流水线中, $M_{i1}, M_{i2}, M_{i3}, M_{i4}, M_{i6}$ 和 M_{i7} 属于硬件模块, M_{i5} 属于软件模块。那么 $P(i)$ 为软硬件协同流水线。

软硬件协同处理的流水线将数据平面单一的硬件处理或者软件处理进行了有效结合。目前网络处理器, 交换芯片甚至一些支持虚拟机交换的智能网卡都采用软硬协同的处理方式, 由硬件进行快速路径的处理, 软件进行慢速路径的处理。

与上述设备实现方式不同, FAST 采用软硬件模块交互更加紧密的紧耦合的流水线技术, 定义如下。

硬件所有模块的集合为 M_H , 软件所有模块的集合为 M_S 。对于流水线 $P(i) = \langle M_{i1}, M_{i2}, \dots, M_{iNi} \rangle$, 若存在

$x, y, z, u(x < y < z < u), M_{ix} \in M_S, M_{iy} \in M_H, M_{iz} \in M_S, M_{iu} \in M_H$

, 那么则称这条流水线是软硬件紧耦合的。

软硬件协同处理的过程中, 两个或两个以上的软硬件模块的输入与输出之间存在紧密配合与相互影响, 并通过相互作用从一侧模块向另一侧模块的传输数据。

2.3 FAST 软硬协同处理模型

2.3.1 模型的基本原理

FAST 平台软硬件协同的分组处理模型如图 2-2 所示，包含 n 个由 FPGA 实现的硬件处理模块，以及 m 个运行在 CPU 上的软件处理模块。其中 FPGA 硬件中的 n 个处理模块组成 FAST 硬件流水线，到达 FPGA 硬件的分组依次由硬件流水线的第 1 级（模块 X1），第 2 级（模块 X2）...第 n 级（模块 X n ）进行处理。其中 P 代表完整的分组，MD 代表元数据，PFV 代表分组特征向量。

每个分组在硬件流水线中传递时，会携带元数据（MD）和分组特征向量（PFV）。元数据包含分组处理的中间结果，用于上下游模块之间交换分组处理的控制信息；分组特征向量主要描述分组自身的特性，由分组头部 128 字节以及分组的协议类型等。

FAST 流水线的第一个模块 X1 将完整的分组送缓冲区缓存，同时生成分组的 PFV，后续模块根据 PFV 对分组进行处理，如提取查表关键字等；FAST 流水线的倒数第二级 X $n-1$ 模块将完成的分组读出，根据需要修改分组内容，如更换目的 MAC 地址等，然后送最后一级模块 X n 进行输出调度。

软硬件接头实现 FAST 硬件流水线与各软件模块 Y1...Y m 之间的分组及其元数据的数据交换，主要由 FPGA OS 和 CPU OS 内核中相应的软硬件实现。

每个软件模块一般为用户空间独立运行的进程，又称为用户应用（UA），UA 实现用户定义的处理功能。

FAST 流水线与 UA 之间交换的信息包括完整的分组以及分组的元数据信息。

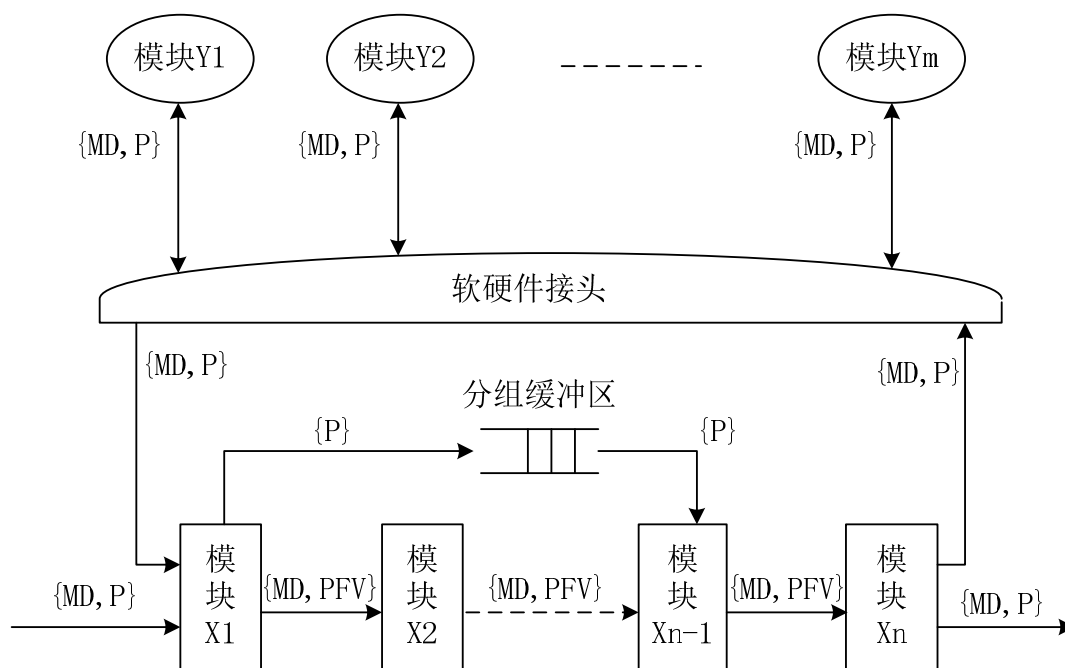


图 2-2 FAST 的软硬件协同处理模型

在 FAST 软硬件协同处理模型中，软硬件模块基于目的模块 ID (DMID) 进行通信，如图 2-3 所示。其中硬件流水线中的模块 M1, M2... Mn 位置相对固定，软件模块 Mx 和 My 可以根据需要动态加载。

网络接口接收的分组首先进入硬件流水线处理，硬件流水线中最后一个模块 Mn 处理完成后，根据处理结果，通过设置分组元数据中的 DMID 值，指示 FPGA OS 将分组直接从网络接口输出，或者送软件模块 Mx (DMID=x) 或 My (DMID=y) 进行处理。

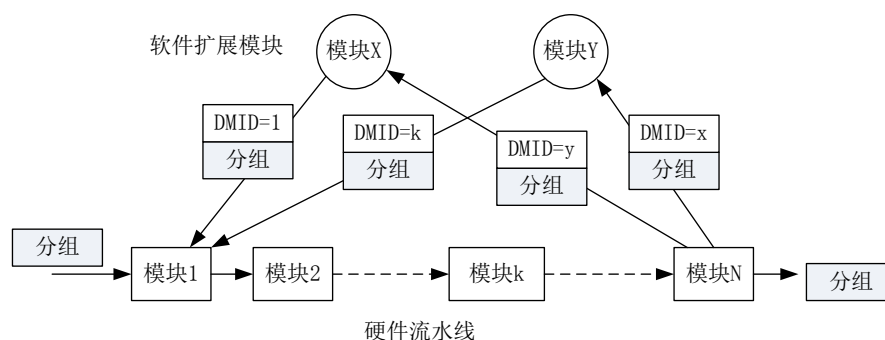


图 2-3 FAST 软硬件模块的通信

每个软件模块在处理完成分组后，将分组发送回硬件流水线，分组首先送到流水线入口模块，然后再依次经过每个硬件模块进行处理。根据模块实现模型，每个模块首先比较分组的 DMID 是否为本地 MID，若不等于，该模块不会处理分组，会将分组直接送流水线的下游模块进行处理。

例如图 2-3 中，软件模块 Y 将发送给硬件流水线分组的元数据中将 DMID 标记为 k， $1 < k < n$ ，则硬件流水线中模块 k 之前的其他模块不会对分组进行任何处理，**直接将分组送到下游模块，直到模块 K。**

典型的，软件模块 X 为内核协议栈，而软件模块 Y 为数据平面的功能扩展模块。硬件流水线如果需要将分组送控制平面处理，如目的 IP 地址为本地 IP 的分组，**则将分组的 DMID 设置为 X**，该分组会送协议栈和控制平面处理；如果需要将分组送数据平面的扩展功能模块 Y 处理，则将分组的 DMID 设为 Y。同样的，如果控制平面需要发送一个分组而无需硬件流水线处理，则将该分组的 DMID 设置为 n。该分组进入硬件流水线后，会旁路掉所有的硬件处理模块，最后经模

块 n 发给 FPGA OS 输出。

2.3.2 分组处理流水线的实现

由于网络接收的不同分组可能具有不同的属性，属于不同的流水线等价类，因此数据平面需要支持不同的分组处理流水线。在 FAST 架构中，根据模块的处理模型以及对 DMID 的设置，可以精确控制分组处理的模块，以此多个流水线可以方便的映射到 FAST 分组处理架构中。

2.3.3 分组处理流水线的动态扩展

分组处理流水线是由多个功能独立的功能模块组成，分组处理路径也可根据用户需求进行改变，不但用户可根据需求任意的增加或删除软件功能模块，还可以通过离线加载的方式重新配置硬件功能模块，满足了系统设计者对于网络功能可扩展的需求。

根据模块定义可知，每个模块中都有 NMI 表项，基于 NMI 表项的流水线可通过增删改 NMI 表项内容扩展和替换原有流水线。FAST 中每个模块具有自己的 MID，称为本地 MID (LMID)，分组元数据 M 中携带接收分组的模块 ID，即 DMID。按照 FAST 模块的定义，每个模块接收到分组后，首先判断 M 中的 DMID 是否等于 LMID，若等于则表示该模块需要处理该分组，处理完成后需查找模块内部 NMI 表项，通过规则匹配定向到下行分组处理模块中；若不等于则表示该模块不需要处理该分组，直接查找模块内部 NMI 表项，通过规则匹配定向到下行分组处理模块中。

基于 NMI 表的流水线扩展技术正是利用了 FAST 模块中 NMI 表项可灵活配置的特点。用户若需要增加新功能，只需按照 FAST 模块定义生成模块，并选定需要插入流水线的位置，修改与新增模块相连的上行模块的 NMI 表项规则即可实现对新功能的增加。用户若需要替换和删除功能，首先定位需要删除或替换的功能模块，然后修改其连接的上行模块的 NMI 表项，即可实现对功能模块的删除和替换。

FAST 流水线是由多个功能独立的功能模块组成，分组处理的路径也可根据用户需求进行改变，用户不但可以根据需求任意的增加或删除软件功能模块，还可以通过离线加载的方式重新配置硬件功能模块，从而满足系统设计者对于网络功能可扩展的需求。

FAST 流水线的分组处理主要通过每个模块内部的 NMI 表项进行控制信息的传输，并通过 Metadata 进行数据信息的传输。Metadata 和表项相结合，可表示各模块之间数据和控制转移的关系，以及各模块之间的依赖关系。

流水线动态增加的模块有两种方式获取自己的模块号。一种是向平台的流水线管理逻辑注册，由管理逻辑动态产生并分配一个全局唯一的模块号给新增模块，如图 2-4 所示。还有一种是每个模块根据实现的功能，离线分配一个固有的模块号。一般对于集成多方模块的平台采用前一种分配方法，对集成单一开发者的平台可采用第二种相对简单的静态分配方法。

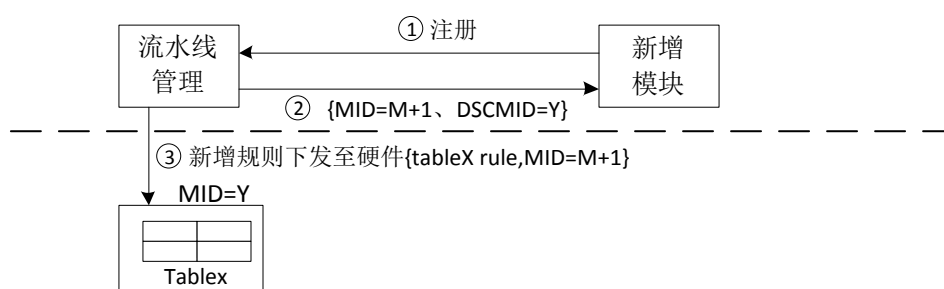


图 2-4 扩展功能模块硬件注册流程

基于 NMI 的流水线扩展原理如图 2-5 所示，流水线 $P(i) = \langle IE, x, y, z, OE \rangle$ 新增模块 s 后变为 $\langle IE, x, y, z, s, OE \rangle$ 。若 R 代表规则全集，即所有的分组均满足规则 R ，每个输入的分组都使用相同的处理流水线 $IE-x-y-z-OE$ 。需要注意的是，虽然有的分组会单播，有的组播，有些会丢弃，还有的会送到控制平面，但每个分组经过的流水线模块是一致的，因此他们属于相同的流水线等价类。

若交换机需要扩充新的处理功能，例如支持隧道的发起和终结功能（支持 LISP 和 VxLAN 等隧道的封装和解封装）。由于原有的硬件流水线不支持隧道的封头和去头功能，在隧道起点由于封装隧道头长度超过 MTU 而带来的分片问题，以及隧道端点的重组问题，因此需要增加相应的软件模块进行相应的处理。处理流水线之间传递的分组 P 的处理行为由二元组 $\langle Rule, DMID \rangle$ 控制， $Rule=R1$ 表示分组 P 的属性满足规则 $R1$ ， $DMID$ 为流水线中下一个处理分组的模块号。

新增模块 s 在流水线的处理流程为：1、分组进入 $LMIDz$ 模块；2、 $LMIDz$ 对分组信息进行查表，通过匹配 NMI 表项规则将分组转发至

LMIDs 中；3、LMIDs 模块对分组进行处理；4、LMIDs 将处理完成的分组通过查 NMI 表转发至后续分组处理模块 (LMID_x)；5、LMID_x 模块对分组进行处理。

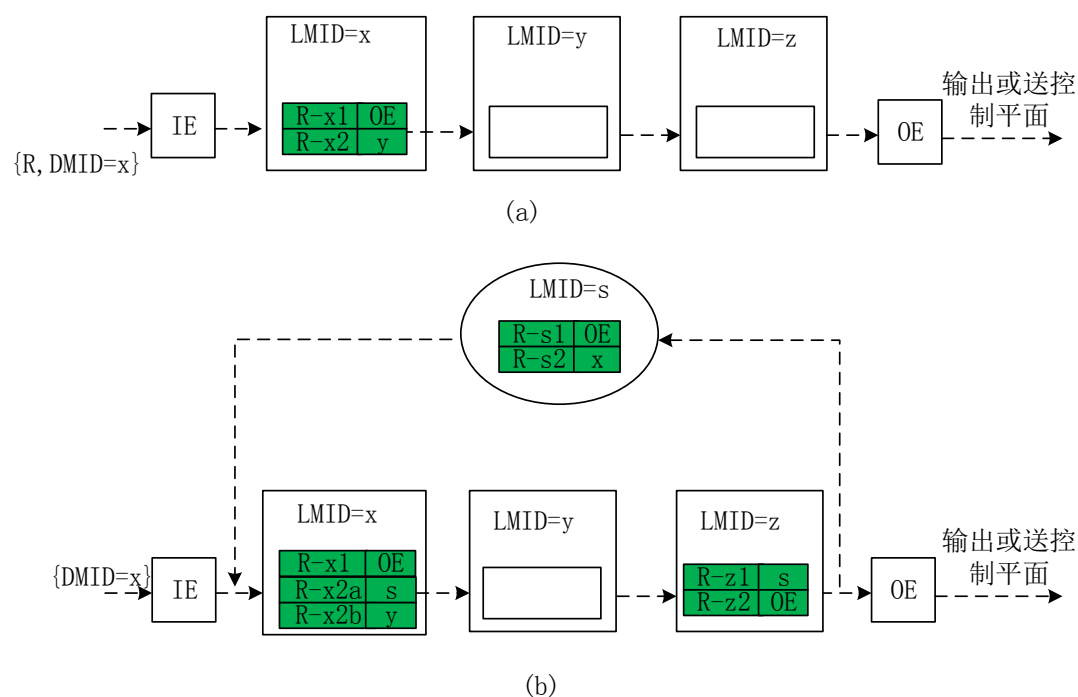


图 2-5 基于 NMI 的 FAST 流水线扩展原理

FAST 通过流水线管理模块对硬件模块的转发规则进行配置，硬件对分组进行查表匹配后根据匹配的规则进行转发。软硬件模块之间交换使用 metadata 数据结构，该结构定义了分组传输的目的 MID 以及源 MID，根据这个信息，分组便可以在任意的模块之间进行数据传递。

3 元数据

元数据（Metadata）是随分组一起在 FAST 平台软硬件协同的分组处理流水线中传递的，包含分组当前处理状态的数据结构。

3.1 元数据格式

UM 设计中的元数据主要包含报文分组处理的中间状态信息，每个分组在 UM 处理流水线模块之间，以及 UM 和 UA 之间传递时，都需要携带 Metadata 信息。

3.1.1 元数据包含的内容

FAST 采用 32 字节固定长度的元数据，包含的主要内容定义如表 3-1 所示。

表 3-1 元数据包含的内容

位置	信号名	含义
Metadata0		
[127:124]	TTL	分组在流水线中的生存周期
[123:120]	InPortNum	分组的输入端口号
[119:108]	Length	不包含 MetaData 字段的分组长度
[107:100]	SMID	最近一次处理分组的模块 ID
[99:92]	DMID	下一个处理分组的模块 ID
[91:80]	Seq_Num	分组接收序列号
[79:64]	OutportBM	分组输出端口的 bitmap
[63]	PktSRC	分组的来源，0 为网络接口输入，1 为 CPU 输入

[62]	PktDes	分组目的, 0 为网络接口输出, 1 为送 CPU
[61]	Discard	丢弃位
[60:58]	Priority	分组优先级
[57:44]	flowID	流 ID
[43:0]	TS	时间戳
Metadata1		
[127:0]	ADI	应用定义的信息

3.1.2 流水线中的元数据

元数据在硬件流水线中的格式如图 3-1 所示。在分组进入和输出硬件流水线时，元数据的位置位于每个分组之前的 32 字节。即每个分组进出 UM 的第 1 拍 16 字节为 Metadata0，第二拍数据为 Metadata1。

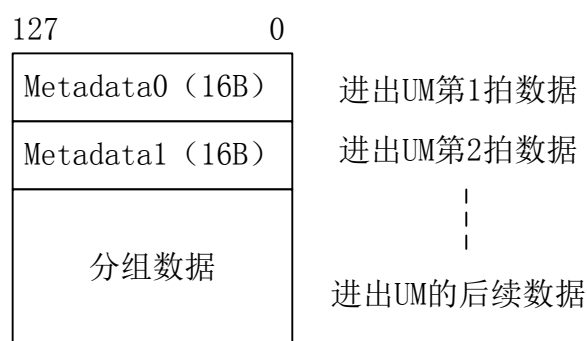


图 3-1 硬件流水线中的元数据格式

根据表 3-1 定义的元数据格式，元数据在硬件流水线中传递时，Metadata0[127:124]为 TTL 字段。

3.1.3 元数据的定义

FAST 编程库中定义了 um_metadata 数据结构，如表 3-2 所示。

UA 软件根据该数据结构对每个分组的元数据进行处理。

表 3-2 um_metadata 数据结构

```
struct um_metadata{  
    u64 ts:44, /**< 报文接收的时间戳 如果用户需要使用表示更大的  
    时间，建议存储在第二拍数据中（user[2]字段）*/  
    flowID:14, /**< 流 ID 号*/  
    priority:3, /**< 报文优先级*/  
    discard:1, /**< 指示报文是否丢弃 默认为 0，表示不丢弃，置 1 时  
    表示丢弃*/  
    pktdst:1, /**< 报文的输出目的方向 0 表示输出到网络端口，1 表示  
    输出到 CPU*/  
    pktsrc:1, /**< 报文的输入源方向 0 表示网络端口输入，1 表示从  
    CPU 输入*/  
    u64 outport:16, /**< 报文输出端口号 以 bitmap 形式表示，1 表示  
    从 0 号端口输出；8 表示从 3 号端口输出*/  
    seq:12, /**< 报文接收时的序列号 每个端口独立维护一个编号*/  
    dstmid:8, /**< 报文下次处理的目的模块编号*/  
    srcmid:8, /**< 报文上次处理时的模块编号*/  
    len:12, /**< 报文长度 最大可表示 4095 字节，但 FAST 平台报  
    文缓存区最大为 2048，完整以太网报文的 MTU 不要超过 1500*/  
    inport:4, /**< 输入端口号 取值：0——15，最多表示 16 个输入端  
    口*/  
    ttl:4, /**< 报文通过模块的 TTL 值，每过一个处理模块减 1*/  
    u64 user[2]; /**< 用户自定义 metadata 数据格式与内容 此字段由可  
    用户改写，但需要保证数据大小严格限定在 16 字节*/  
};
```

3.2 元数据的处理要求

3.2.1 输入端口号

FPGA OS 在分组接收时填写接收端口号，用于转发控制。UM 和 UA 进行分组转发时，不得修改该字段。UA 产生的分组用 PktSrc=1 标识，InPortNum 值无意义。

PU

3.2.2 流水线生存时间

FPGA OS 接收分组时，将 TTL 设为 0xFF 或其它阈值，代表允许分组在流水线中存在时间；如果 UM 模块对分组进行处理，将 TTL 减 1；如果模块旁路对分组的处理，保持 TTL 不变；UA 处理分组时，将 TTL 减 1；如果 TTL 值为 0 时，将 Discard 位设为 1，DMID 设为 5（GOE）

3.2.3 分组长度

FPGA OS 接收分组时或 UA 产生分组时，将 Length 值设置为分组长度，单位是字节。除非发生分片和重组，Length 的值设置后不再修改，即使发生因隧道封装和解封装会产生分组长度变化。需要将 Length 值设置为分组长度，单位是字节。

GOE 模块利用该值进行令牌桶限速处理和输出调度等操作。

3.2.4 源模块号

FPGA OS 不设置，不修改和不使用 SMID 的值；UM 模块或 UA 模块处理分组时，将 SMID 设置成本地 MID；UM 模块旁路分组时，不修改 SMID 的值。

在某种处理场景下，SMID 会参与分组处理决策，例如在多 UA 场景下，SMID 代表上一个处理该分组的应用，使用 SMID 字段有利于增加服务编排的灵活性。

3.2.5 目的模块号

FPGA OS 接口接收分组后，将 DMID 设置为 1（UDP 模块）。每个 UM 模块或 UA 处理完分组后，将下一处理分组的模块 ID 设为 DMID；旁路分组处理的模块不修改分组的 DMID，DMID=0，表示丢弃该报文

3.2.6 输出端口指示

FPGA OS 根据 OutPortBM 将分组送到不同的输出接口，输出接口最多为 16 个。多个输出端口对应的 bit 为 1 代表输出为组播。当 Discard=0（不丢弃）和 PktDes=0（不送 CPU 处理）时，UM 必须保证 OutPortUM 中至少有一个 bit 为 1。

3.2.7 分组目的标识

为 0 表示该分组在硬件流水线输出后，FPGA OS 不需将其送 CPU 处理，1 表示要送 CPU 处理。

3.2.8 分组来源标识

为 0 表示该分组来自网络接口，为 1 表示该分组由 CPU 上的软件产生。当来自端口的分组被硬件流水线送 CPU 软件处理完成，又发回硬件流水线时，该分组的 PktSrc 不应置为 1，只有 UA 或控制平

面自身产生的分组才会将 PktSrc 标记为 1。

3.2.9 分组序列号

FPGA OS 中，每个物理接口接收到分组后，为其分配一个序列号，这个序列号是按端口分配，每次增加 1。序列号增加到全 F 时清零，再从零开始逐一递增。UM 和 UA 不修改分组的序列号，UA 产生分组的序列号固定为全 0。

3.2.10 分组流标识

FlowID 标识分组所属的流。通常情况，硬件流水线通过对分组进行查表匹配后，将匹配表项的 ID 填写到 FlowID 域；在硬件流水线或 UA 的后续处理中，可以根据 FlowID 查找处理该分组的上下文状态。

UA 也可以指定其发出分组的 flowID 值。UA 支持对 flowID 赋值支持将转发表划分成硬件转发表和软件转发表两部分。UA 转发完成并设置 flowID 后，可将分组 DMID 设置成 4 (UDA)，支持硬件流水线基于软件查表得结果进行相应的处理动作和输出调度。

3.2.11 分组丢弃标识

为 1 表示丢弃该分组，为 0 表示不丢弃该分组。

3.2.12 分组优先级

优先级 Priority 的值由 UM 流水线模块或 UA 设置。FPGA OS 需要明确是否支持优先级调度，FPGA OS 在输出调度发生拥塞时，需

要根据优先级进行调度。

3.2.13 分组时间戳

FPGA OS 在分组输入时标记的分组接收时间，FAST 流水线和 UA 不会修改 TS 值。FPGA 用于产生本地时钟计数器的时钟与 UM 使用的时钟必须是同一时钟。

3.2.14 应用定义字段

由 UM 流水线模块或 UA 定义的信息，用于模块之间应用相关信息的交换；如果存在多个 UA，为避免多个 UA 定义 ADI 信息的冲突，可进一步将 ADI 划分，例如，ADI[63:32]分配给 UA1，ADI[31:0]分配给 UA2；

标准 UM 流水线模块不负责解析 ADI 的信息，只有在流水线中插入的应用相关模块或者虚拟端口实现的应用相关逻辑才支持对 ADI 的解析，或生成 ADI 信息发送给 UA。

3.3 基于元数据的分组转发控制

3.3.1 组播与单播

通过设置元数据中的 DMID，PktDes，Discard 和 OutPortBM 等字段，硬件流水线可以控制输出分组的方向，分为 4 种类型，如下表所示。

输 出 类型	转发流程	控制方法

1	硬件单播	DMID=xxx, OutPortBM 对应单播输出接口为 1, PktDes=0; Discard=0;
2	硬件组播	DMID=xxx, OutPortBM 对应组播输出接口为 1, PktDes=0; Discard=0;
3	定向到 CPU	DMID=UA MID, OutPortBM 设为全 0, PktDes=1; Discard=0;
4	硬件单播/ 组播同时定向到 CPU	DMID=UA MID, OutPortBM 对应单播/组播输出接口为 1, PktDes=1; Discard=0;
5	丢弃分组	DMID=xxx, OutPortBM=xxxxx, PktDes=x; Discard=1;

其中类型 3 将分组送到软件处理，例如需要送控制平面的分组，需要送数据平面 UA 的处理分组等。类型 4 在正常转发分组的同时，将分组的副本送相应的软件 UA 处理。例如当软件 UA 实现特定 IPTV 数据流传输质量监测，或入侵检测功能时，需要不影响正常数据流


转发的同时，根据规则获取特定数据流的分组。

3.3.2 统计信息生成

在分组从 UM 硬件流水线最终向接口发出时，分组的 Metadata 中保存了最完整的分组处理信息，因此 Metadata 可用于网络管理和检测应用，如生成 Netflow 信息，实现各种标准或应用定义的统计计数器。

3.3.3 分组丢弃

对于硬件流水线需要丢弃的分组，统一送给硬件流水线中最后一个模块，由最后一个模块统一进行丢弃并计数。



4 分组特征向量

4.1 分组特征向量的组成

由于不同分组的长度不同，而且硬件流水线实现分组的高速转发不需要利用分组 payload 内容，为简化分组硬件流水线的处理，减小流水线数据传输带宽的需求，通常只有包含分组 L2-L4 头的前 M 个字节，而不是整个分组进入硬件处理流水线。

分组特征向量（PFV）通常包含分组的前 M 个字节和长度为 L 字节的分组类型（协议解析器的结果）等信息。通常分组进入 UM 的第一个处理模块为协议解析模块，该模块会根据分组内容和 Metadata 生成该分组的 PFV。

4.1.1 协议标准头

协议标准头是接收分组从以太网头开始的 128 字节，不含以太网尾部的 CRC 字段。以太网帧长度不足 128 字节时，后部由全 0 补齐。例如，图 4-1 是 ARP，VXLAN 和 IPv6/LISP 分组的标准头格式。

Eth (14)	ARP (28)	PAD (86)
-------------	-------------	-------------

ARP分组

Eth (14)	IPv4 (20)	UDP (8)	VxLAN (8)	Eth (14)	IPv4 (20)	TCP (16)	PAD (28)
-------------	--------------------------	------------------------	--------------	-------------	--------------	-------------	-------------

VXLAN分组

Eth (14)	IPv6 (40)	UDP (8)	LISP (8)	IPv6 (40)	TCP (16)	PAD (2)
-------------	--------------	------------	-------------	--------------	-------------	------------

IPv6/LISP分组

图 4-1 协议标准头示意图

4.1.2 协议类型标识

每种格式的 PSH 对应一个 8 比特的分组标准类型编码 (PST)，其中 00XX XXXX 编码对应 IPv4 相关协议，10XX XXXX 编码对应 IPv6 相关协议。0000 0000 表示未识别的协议。

FAST UM-RM 支持的协议标准类型 (PST) 定义如表 4-1 所示。

表 4-1 协议标准类型定义

分组类型	PSH 组成	PST 编码
IPv4 TCP	Eth/IPv4/TCP	0000 0001
IPv4 UDP	Eth/IPv4/UDP	0000 0010
ARP	Eth/ARP	0000 0011
IPv6 TCP	Eth/IPv6/TCP	1000 0001
IPv6 UDP	Eth/IPv6/UDP	1000 0010
IPv6 LISP	Eth/IPv6/UDP/LISP/IPv6	1100 1111
未识别		0000 0000

UDT 的格式与 PST 完全一致，用来表示标准协议解析不支持，但用户定义解析支持的协议。UDT 的定义与每个解析器中用户定义

的解析模块的实现相关，支持的协议也由 UDT 表定义。

在 PST 和 UDT 的解析结果选择中，UDF 的优先级高于 ~~PST~~。例如一个基于 IPv6 的 LISP 报文，PST 的解析结果为 1000 0010 (Eth/IPv6/UDP)，而 UDT 的解析结果为 1100 1111 (Eth/IPv6/UDP/LISP/IPv6)，显然 UDT 的结果更加精确。

4.1.3 用户定义标识

每个用户定义标记 UDF 长度为 1 个比特，表示分组是否满足用户定义的特殊的属性，解析器最多支持 8 个用户定义的属性识别和标记。

FAST 流水线使用的 PFV 包含分组的前 128 字节，以及 4 字节的分组类型信息，这些信息组成了一个长度为 1056 比特的向量。

用 Verilog 语言定义的 PFV 为：

```
Reg [1055:0] Packet-Feather-Vector;
```

根据上述定义，只需一个时钟周期，PFV 就可以从流水线的上游模块传递到下游模块。

每个分组在硬件流水线中的流动表现为<Metadata, PFV>信息在流水线上下游模块中的传递。每个硬件模块从上游模块中接收分组的 PFV 和 Metadata 信息，进行本地处理后，修改 Metadata 和 PFV 的相关信息，并将这些信息发送到下游模块。

解析器输出的分组特征向量 PFV (Packet feature vector) 包含后

续分组处理需要的分组相关的所有信息。主要包括协议标准头（PSH），协议标准类型（PST）和用户定义标记（UDF）等，详细见 4.2 节。

4.2 协议标准类型的应用

4.2.1 关键字提取

需要注意的是，当分组的 PST 确定时，提取的每个关键字在 PSH 中是提前确定不会发生改变的。

例如，对于提取 IPv4/TCP/UDP 报文的五元组，可离线计算得：IPv4 源 IP 地址到以太网帧起始的偏移量为 26 字节，目的 IP 偏移量为 30 字节。协议域偏移为 23 字节，TCP/UDP 源和目的端口号分别为 34 和 36 字节。因此，KE 的相关代码如下：

显然，如果在关键字提取时需要 TCP 的 SYN 等标志位，可以计算这些标志位的偏移量，直接赋值即可。

用 Verilog 描述的 IPv4 分组五元组关键字提取的代码如图 4-2 所示。

```
If (PST=8'h00000001 || PST=8'h00000010 ) //IPv4 TCP/UDP
Begin

    Src_IP<=PSH[239:208]; //26*8=208

    Des_IP<= PSH[271:240]; //30*8=240
```

```
Pro_type<=PSH[191:184];//23*8=184

Src_port<=PSH[287:272];//34*8=272

Des_Port<=PSH[303:288];//36*8=288

End
```

图 4-2 基于协议标准类型的关键字提取代码

4.2.2 分组处理流程定向

流水线中的处理模块可根据分组协议类型，尽早确定分组的处理流程。图 4-3 描述的一个只支持 IPv6 协议的流水线工作流程，主要分为 3 种情况。

第一种情况是接收的为 IPv4 分组，或者分组协议类型未识别。流水线需要丢弃该分组，因此将分组元数据中的 Discard 位标记为 1，将目的 MID 号设为流水线中统一处理丢弃分组的模块的 MID 号(5)。这样后续的其他非 MID 号为 5 的模块，如转发查表模块不会处理该模块，因此减小了处理资源开销。

第二种情况是接收的分组为 IPv6 分组，但根据 L4 UDP 的端口号判别为 LISP 协议分组，由于后续硬件流水线不支持 LISP 分组的处理，因此将 Pktdes 设置为 1，表明需要送 CPU 软件处理，同时将其 DMID 设为 200。而 MID 为 200 的软件 UA 是系统预先加载好的支持 LISP 转发的软件模块。

第三种情况是其他 IPv6 分组，这些分组的 DMID 设置为全 0，表示将分组默认送到流水线的下一个模块处理。

1	If (PST=8' b0xxxxxxx) //未识别协议或 IPv4 协议分组
2	DMID<=8' d5; //送到 MID 为 5 的硬件模块处理;
3	Discard<=1' b1;
4	Else if (PST==8' b1001111) //分组是 IPv6 LISP 类型
5	DMID<=8' d200; //送 MID 为 200 的软件 UA 处理;
6	Pktdes<=1'b1;
7	Else
8	DMID<=8' d0; //送流水线的下一个模块处理;

图 4-3 IPv6 分组转发流水线处理流程

4.3 用户定义标记的使用

4.3.1 多维分组分类

用户定义标记可以进行多维度的分组分类。例如基于 UDF 可以标记：

- 长度为 100 字节的以太网帧
- 长度大于 1200 字节的报文
- RTP 分组
- Http post 类型分组
- 分组 payload 中包含正则表达式 X 描述的字段

因此 UDF 可以弥补标准协议解析的不足，增加灵活性。特别是在 FPGA 平台上，可以针对不同的应用场景设置不同的 UCF 标记逻辑，能够满足网络功能定制的需求。

4.3.2 简化查表关键字

使用 UDF 标记可以缩短查表关键字的长度，减小查表匹配的复杂度。

4.3.3 应用加速

UDF 逻辑还可以实现特定的分组处理功能。例如计算 IP 分组的校验和，判断 TTL 的值等，若校验和错误或者 TTL 为 0，可设置相应的标记，通过设置查表规则，丢弃相应的分组。又例如，UDF 还可以判断特定数据流是否超出令牌桶的限速等。

5. Fast 硬件流水线

5.1 与 FPGA OS 的接口

FAST 定义了 UM 规范。UM 规范是 FAST 流水线和 FPGA OS 的接口。

5.1.1 信号接口

FAST UM 定义了 8 类接口: FromPort, ToPort, FromCPU, ToCPU, Ctrl, AUX, FromMatch 和 ToMatch, 支持 FAST 的 FPGA OS 需要支持前 6 个接口, 带有 TCAM 或专用查表实现逻辑（如基于 SBV 算法的查表逻辑）的平台可根据规范的定义提供额外的 FromMatch 和 ToMatch 两个接口。UM 逻辑通过调用 FrmMatch/ToMatch 接口实现转发查表中的匹配操作。

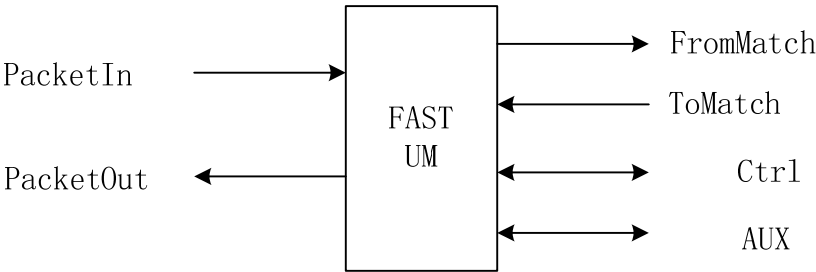


图 5-1 FAST UM 接口逻辑结构

FAST UM 的接口如图 5-1 所示，各接口简要说明如表 5-1。

表 5-1 FAST UM 的接口说明

接口名	说明
-----	----

PacketIn	FAST UM 从网络接口或 CPU 接收的报文
PacketOut	FAST UM 向网络接口或 Cpu 发送的报文
Ctrl	CPU 根据虚拟地址空间的定义，通过该接口提供的读写操作原语，对 FAST UM 内部的寄存器、各类控制表进行维护管理
AUX	FAST OS 向 UM 提供的时钟、复位、时间戳等信息，以及 UM 发出的控制平台相关外设（如显示屏）的信号等。
ToMatch	发往 Match 协处理器的查表请求，包括 288 位的 key 和 288 位的掩码；
FromMatch	从 Match 查表协处理器返回的查表结果

5.1.2 接口分组格式

FAST UM 在 PacketIn、PacketOut 两个接口采用相同的分组传输格式，数据宽度为 134 位，其中 128 位为报文数据，包括 32 字节的 Meatdata 和分组数据，最高 6 位为控制信息。如图 5-2 所示。

，报文数据的格式为接口收发的以太网报文格式。其中以太网报文格式中不包含最后的 4 字节 CRC 字段，接收时，FPGA OS 负责接收时进行 CRC 校验和剥离，发送时，FPGA OS 会计算分组的 CRC 字段并附加在报文最后。

数据通路的[133:132]位为报文数据的头尾标识。01 标识报文头部，11 标识报文中间数据，10 标识报文尾部。由于不同报文具有不同长度，因此在报文数据最后一排可能存在一些无效的字节。数据报文的最后一拍的[131:128]位用来标识无用字节的个数。其中 0000 表示 16 个字节全部有效；0001 标识最低 1 个字节无效，最高 15 个字节有效；以此类推，1111 表示最低 15 个字节无效，最高 1 个字节有效。

带外控制信息		报文数据
133	132:131	128:127
01	0000	Metadata_0
11	0000	Metadata_1
11	0000	报文前16个字节
11	0000	报文17-32字节
⋮	⋮	⋮
11	0000	⋮
10	Vbyte	报文尾部数据 (1-16个字节)

图 5-2 传送报文的数据格式

设 UM 的时钟频率为 xMhz，因此数据通路理论传输带宽为 128bit*xMHz。例如当 x=125 时，通路带宽为 16Gbps。

5.1.3 查表接口

UM 通过 ToMatch 接口向查表协处理器提交 288 位的 key 数据，从 FromMatch 接口接收返回的查表结果，包括是否匹配，以及匹配的规则 ID 等。

UM 可以通过 ToMatch 接口连续地向查表协处理器提交查表请

求，查表协处理器必须保证这些查表结果按照 UM 提交的顺序通过 FromMatch 接口返回查表结果。

ToMatch 和 FromMatch 接口设计均采用简单的 FIFO 写接口，避免了 UM 和查表协处理器由于同步而带来的复杂性。

不同的平台提供查表的规则宽度为 288 比特，每比特都可以带掩码。支持的规则数目不限，设为 2^X 条（X 为正整数），则 FromMatch 返回的 ruleID 宽度为 X 位。

平台通过虚拟地址空间的定义，支持软件访问（读写）查表协处理器中的规则，即对查表协处理器的管理无需通过 UM。

5.1.4 接口信号与时序定义

UM 的各接口信号的详细定义如表 5-2 所示。

表 5-2 UM 接口的信号定义

信号名称	方向	宽度	信号描述
PktIn 接口			
pktin_data_valid	in	1	向 UM 写入来自端口的分组数据有效
pktin_data	in	134	向 UM 写入来自端口的分组数据
pktin_ready	out	1	UM 可以接收一个来自接口的完整分组
PktOut 接口			
pktout_data_valid	out	1	UM 向 port 写分组数据有效
pktout_data	Out	134	UM 向 port 写的分组数据
pktout_ready	in	1	port 可接收一个来自 UM 的完整分组

Ctrl 接口			
Ctrl_valid	in	1	Ctrl_cmd/addr/DataIn 等信号有效，表示一次控制操作开始
Ctrl_cmd	in	1	0 表示写操作，1 表示读操作；
Ctrl_Addr	in	32	地址
Ctrl_DataIn	in	32	向 UM 写的的数据；
Ctrl_DataOut	out	32	从 UM 输出的数据；
Ctrl_DataOut_valid	out	1	DataOut 返回数据有效
ToME 接口			
um2me_key_valid	out	1	UM 向 me 的查表关键字有效
um2me_key	out	288	UM 向 ME 的查表关键字；
Um2me_ready	In	1	me 可以接收查表请求。
FromMatch 接口			
me2um_id_valid	In	1	me 向 UM 返回的匹配 id 有效；
match2um_id	In	14	me 向 UM 返回的匹配 id
Match2um_cookie	In	32	Me 向 UM 返回的匹配规则的 cookie
其他接口			
Um_Clk	In	1	UM 的时钟信号，125MHz
Um_reset_n	in	1	UM 的复位信号，低有效
Um_TimeStamp	in	64	平台提供的本地时间戳

5.2. 流水栈组成

UM-RM 主要由 7 部分组成，包含用户定义解析（UDP），用户定义关键字提取（UKE），通用查表引擎（GME），用户定义动作（UDA）

和通用输出引擎（GOE）5个流水线模块，如图 5-3 所示。

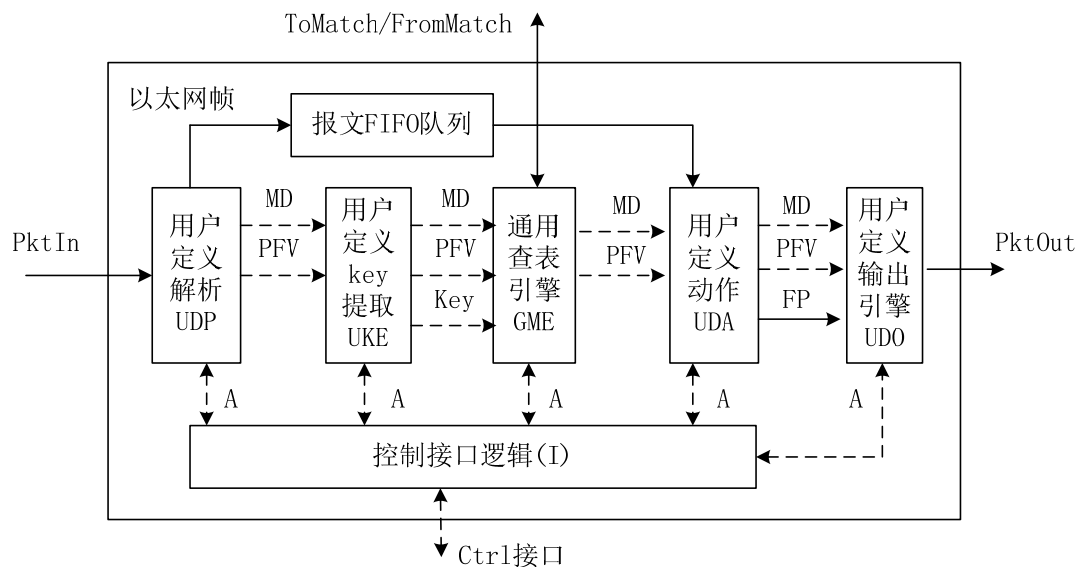


图 5-3 UM-RM 的总体架构

各部分主要功能：

- 用户定义的解析

UDP

根据用户特定的处理需求，分析分组的协议，以及分组是否满足用户定制的属性分类，生成后续控制分组处理的分组特征向量（PFV）。

- 用户定义关键字提取

根据分组的特征向量，获取分组查表得关键字。

- 查表引擎

根据产生的关键字查表，获取控制分组处理动作和输出控制的信息。

- 用户定义动作（UDA）

实现用户定义的对分组的特定处理，如修改分组头，分组分片与重组，添加二层分组头等。

- 输出引擎（OE）

基于令牌桶实现对分组输出的整形，如限制发往特定 CPU 软件 UA 或协议栈的流量，限制特定流发往特定端口的流量等。

FAST UM 中五级流水线的 MID 编号如表 5-3 所示。

表 5-3 FAST 流水线中模块的 MID 定义

流水线模块	模块 ID
UDP	1
UKE	2
GME	3
UDA	4
GOE	5

5.2.1 用户定义解析器（UDP）

分组解析器主要功能包括：（1）支持标准协议的解析，用户定义协议的解析，用户定义的标记产生；（2）从 FAST 格式报文中提取转发使用的 Metadata；（3）将分组前 128 字节转换成 1024 位的额向量，简化后续关键字提取模块 KE（Key Extractor）的操作。

分组解析器的组成如图 5-4 所示。

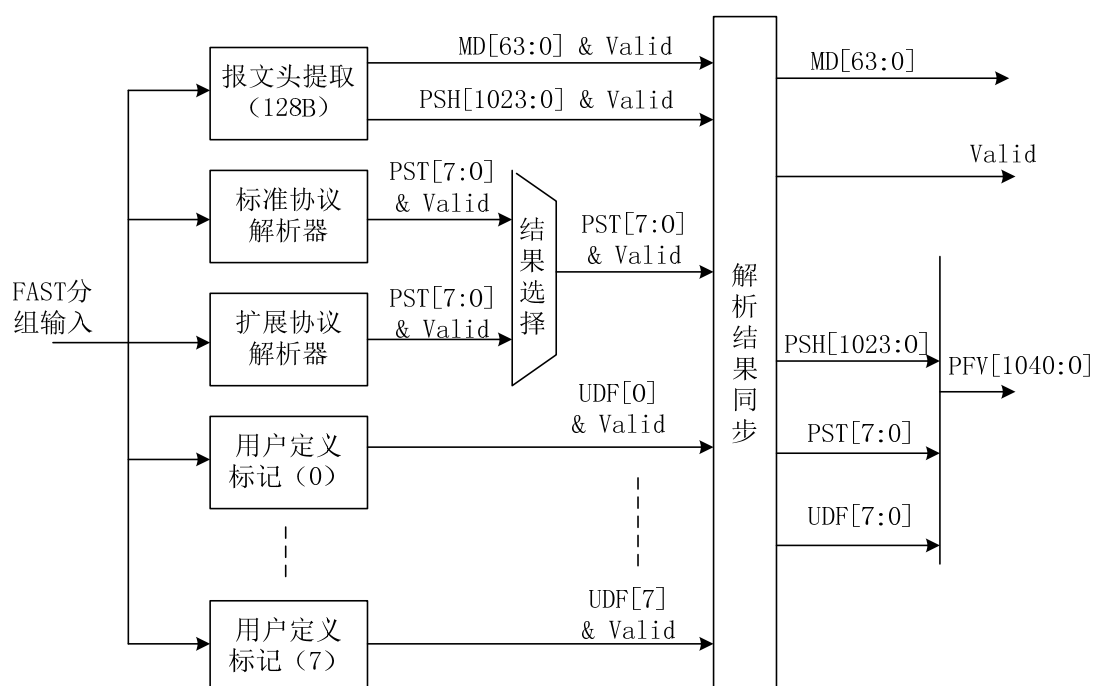


图 5-4 解析器实现架构图

5.2.3 用户定义关键字提取 (UKE)

KE 逻辑与特定的场景和用户特定的交换需求密切相关。即面向不同实验场景的 UM 设计需要定制不同的 KE 逻辑。例如工作在 IPv6 环境下的 UM，无需支持对 IPv4 分组的关键字提取。

查表关键字产生逻辑根据分组的 Meatadata，以及解析器产生的 PSH、PST 和 UDT，根据分组处理的需求产生查表得关键字。

FAST 平台提供的查表关键字长度为 288 位，规则中可以在任意位置设置掩码。

因此 UKE 和 GME 接口信号 Key 的定义为 Key[287:0]，通常包

含以下几个部分：

- **Metadata**：确定分组的来源（源端口号，CPU），目的输出接口（来自 CPU 的可能标识输出接口）；
- 基于 PST 从 PSH 中提取的字段
- 基于 UDF 标志位。

具体的关键字格式应根据应用的需求设计。例如，IPv4 五元组 key 定义如下：

Key<={ 184'h0, src_IP, Des_IP, Pro_type, src_port, des_port};

UKE 模块提取的 Key 按照 512bits 进行封装,对不同类型的分组,定义的查表关键字如表 5-4 到 5-6 所示。

表 5-4 IPv4 分组的 key 格式

位置	标志	含义
[511:272]	Reserve	保留
[271:256]	Flag	TCP 标志位
[255:240]	<u>Dport</u>	目的端口号
[239:224]	Sport	<u>源端口号</u>
[223:192]	Dip	目的 IP 地址
[191:160]	Sip	源 IP 地址

[159:156]	Inport	输入端口号
[155:152]	Frag	分片标识
[151:144]	TTL	生存时间
[143:136]	Tos	服务类型
[135:128]	IP_proto	协议号
[127:112]	Eth_type	帧类型
[111:96]	TCI	vLan 标识
[95:48]	Smac	源 MAC 地址
[47:0]	Dmac	目的 MAC 地址

表 5-5 ARP 分组的 key 格式

位置	标志	含义
[511:320]	Reserve	保留
[319:272]	Tha	目的硬件地址
[271:224]	Sha	源硬件地址
[223:192]	Dip	目的 IP 地址
[191:160]	Sip	源 IP 地址
[159:156]	Inport	输入端口号

[155:152]	Frag	分片标志
[151:144]	TTL	生存时间
[143:136]	Tos	服务类型
[135:128]	Ip_proto	协议号
[127:112]	Eth_type	帧类型
[111:96]	TCI	vLan 标志
[95:48]	Smac	源 MAC 地址
[47:0]	Dmac	目的 MAC 地址

表 5-6 IPv6 分组的 key 格式

位置	标志	含义
[511:480]	Reserve	保留
[479:464]	Dport	目的端口号
[463:448]	Sport	源端口号
[447:416]	Label	Ipv6 标签
[415:288]	Dip	目的 IP 地址
[287:160]	Sip	源 IP 地址
[159:156]	Inport	输入端口号

[155:152]	Frag	分片标志
[151:144]	TTL	生存时间
[143:136]	Tos	服务类型
[135:128]	Ip_proto	协议号
[127:112]	Eth_type	帧类型
[111:96]	TCI	vLan 标志
[95:48]	Smac	源 MAC 地址
[47:0]	Dmac	目的 MAC 地址

5.2.3 通用查表引擎（GME）

GME 将 288 位的关键字送外部匹配引擎进行匹配查表，返回是否命中的标识，以及宽度为 14 位的命中的规则 id。

GME 模块将返回的 id 填写到 MD[63:50]，若未命中，则需要设置 MD[37]=1（丢弃分组），或者 MD[36]=1（送 CPU 处理）。

需要注意的是，FAST 库通过板级相关的虚拟地址空间，而不是 UM，实现对 ME 规则的管理。

如果用户需要验证自己的查表匹配逻辑，或者 FPGA OS 提供的查表引擎无法满足应用的需求，例如规则的关键字长度大于 288 位，用户可在 UM 内部实现自己的查表引擎，同时将规则表映射到 UM 中

GOE 的虚拟地址空间。UA 通过调用定制功能 API 对规则表进行配置。

5.2.4 用户定义动作（UDA）

UDA 模块包含转发处理的 Action 表，该模块根据输入 MD 中的 FlowID 进行 Action 表的查表操作。UDA 模块需要根据 UM 实现转发机制额不同设置不同的 action 表。例如：

- 以太网交换机：FlowID-〉输出端口&优先级
- 路由器：FlowID-〉输出端口，下一跳 MAC 地址等
- 防火墙： FlowID-〉当前流状态和 action
- 入侵检测： FlowID-〉 Action（输出端口+送 UA 标志）

Action 表需要映射到 UM 中 UDA 的虚拟地址空间。

UDA 从分组缓冲 FIFO 中读取分组，可能根据 MD 中信息和 action 表的内容对分组进行操作，典型操作包括：

- 封装隧道头
- 去封装隧道头
- 分片
- 分片重组
- 修改目的 MAC 地址

UDA 在对分组进行操作时，也可能需要查表，如保存下一跳 MAC 地址的邻接表，这些表格也需要在 UM 中 UDA 的虚拟地址空间中进行定义。

UDA 会根据查表结果，修改 Metadata 中的相关信，如修改分组的输出端口，分组的优先级信息等。

5.2.5 通用输出引擎（GOE）

UDA 模块最终确定的输出分组的格式，用户定义的输出引擎不会修改输出的分组格式，主要进行令牌桶限速，计数统计，用户定义的输出调度等功能。

UDO 的位置十分适合实现用户定义的功能。一方面，经过一些列上游模块的处理，已经具有比较完善的 MD 和 PFV 信息，如 flowID 等；另外，复杂的分组内容修改等操作已经完成，简化了 UDO 模块实现的复杂性；此外，UDO 还可通过修改 Metadata 将原本输出的分组重新定向到 CPU 处理，是各类用户相关 UA 向硬件释放“钩子”，获取特定类型分组的合适位置。

典型的用户定义功能包括：

UDP

- 流量的分析，可通过各类计数器实现对流量进行分析，如通过计算流量的非对称性预测 DDOS 攻击的发生，实现 Netflow 协议等；
- 流量的控制，可根据用户的配置，确定具有特性属性分组的输

出带宽。例如可以通过参数配置控制发往 CPU 特定 UA 的分组带宽和数量。

- 输出时间控制，可以控制分组序列精准的发送，即精确控制序列间任何两个分组发送的间隔，同时将序列中第一个分组的发送时间报告 UA，这样 UA 就可以计算得到每个分组发送的精确时间，增加了主动网络测量的精度。

UDA 模块根据规则 rule 对报文进行处理，例如：报文输出方向和端口/丢弃等（目前仅实现报文输出方向和端口及丢弃这两项）：

GOE 模块根据 Dst_MID 的值决定分组报文是否转发，当 Dst_MID 的值不等于该模块的 Local_id 时，则直接转发否则执行丢弃操作。

5.3 匹配动作表

FAST 硬件流水线中，在不同的模块中可以有 1 个或多个控制平面软件，如 openflow 通道程序，可以配置的表。但在逻辑上，分组在流水线处理中遇到的第一个表应该是 UA 可以配置的分类定向表。

6. 用户应用（UA）编程

6.1 硬件抽象

6.1.1 虚拟地址空间

FAST 硬件平台中 FPGA OS 和 UM 中都存在需要软件参与管理的寄存器、计数器和 RAM 等资源。虚拟地址空间（Virtual Address Space，简称 VAS）实现了对这些硬件资源的抽象，是 FPGA 硬件与 CPU 软件间的重要界面。

FPGA OS 是平台相关的，不同平台中 FPGA OS 实现的方式可能存在较大差异。但这些平台需要为软件的管理操作提供相同的抽象，例如接口状态寄存器、接口控制寄存器，接口计数器等。

每个 FPGA OS 都需要提供一个物理接口状态寄存器，每个物理端口的状态对应其中的一个比特位。比特位为 0 表示该接口处于 down 状态，为 1 表示接口处于 up 状态。CPU 相关软件通过定时轮询该寄存器获取端口状态的变化，当发现某个物理端口对应的状态位由 0 变成 1 时，表示该接口状态由 down 变成 up。对于标准的 openflow 交换机，此时会触发向控制器发送一个异步消息。控制器获取交换机特定接口由 down 变成 up 后，会立刻触发 LLDP 协议，启动通过交换机的接口进行拓扑发现过程。

FPGA OS 中还需要提供其他软件可管理的资源，例如每个接口对应的分组收发计数器，接口的配置寄存器等。如果平台还有一些软

件通过 FPGA 进行控制的外部设备（如 PHY 芯片，面板指示灯等），也需要在 FPGA 中提供相关寄存器，由软件配置这些寄存器进行管理。

如果平台提供了查表协处理器，支持 FAST UM 通过 ToMatch 和 FromMatch 接口实现关键字的查表，那么 FPGA OS 必须向软件提供管理查表协处理器的接口，即软件通过 FPGA OS 实现对查表规则的管理，而不需要 FAST UM 的干预。

以 FPGA OS 中的接口状态寄存器为例，一些平台的 FPGA OS 可能通过芯片管理引脚（MDC/MDIO）从物理层芯片中读取对应端口的 up/down 状态，而另外一些平台可能从内嵌的 MAC 核中获取接口的 up/down 状态。如何获取寄存器信息对软件是透明的，软件只需使用 VAS 空间定义的接口状态寄存器地址就可以获取该寄存器的值，因此是可以跨平台迁移的。

FAST UM 内部也有大量的状态和配置信息需要软件进行管理，典型包括以下几类。

- （1）UM 内部控制分组转发的数据结构。
- （2）UM 内部的各种计数器
- （3）UM 逻辑工作的参数表，如令牌桶、输出调度参数表等；
- （4）UM 逻辑的工作状态，如存储分组的 FIFO 队列长度等。

与 FPGA OS 中可管理的硬件资源的数量和种类相对固定不同，

FAST UM 中需要软件管理的资源在类型和数量上随着 UM 功能的不同而不同。例如，同样是 UM 内部的 RAM，可能包含的是 openflow 的流表、标准二层交换机的转发表，路由器的 FIB 表，网络接口的邻接表等，这些表具有不同的语法和语义，这对软件的管理提出了较大的挑战。

虚拟地址空间就是将 FPGA OS 和 FAST UM 中需要软件访问的资源（寄存器、计数器、控制表等）统一编址到一个 32 位的地址空间（4GB），为 FAST 平台软件（如表管理软件、设备管理软件）和各类用户定义软件(UA, 如 FAST 白皮书中提到的精准测量服务 AMS 软件）访问这些硬件资源提供统一的接口和方法。

FPGA OS 中的端口状态寄存器、端口计数器和外设控制寄存器分别映射到 OAddr_x，OAddr_y 和 OAddr_z 地址。而 FAST UM 中流表项分别映射到 UAddr₀，UAddr₁ 和 UAddr_{N-1} 等。

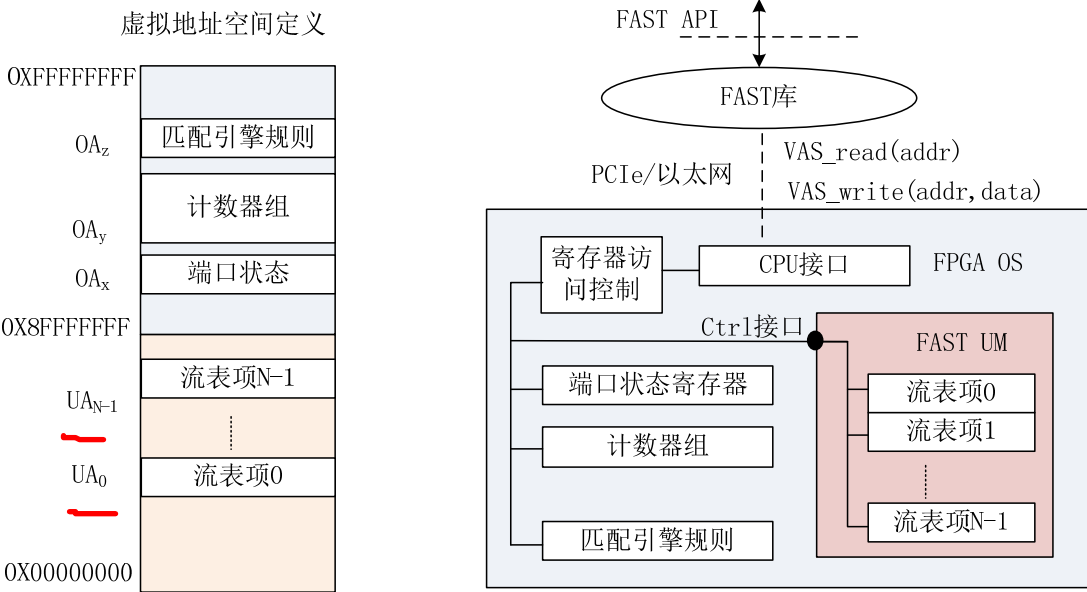


图 6-1 虚拟地址空间原理

FAST 将 4GB 的虚拟地址空间的低 2GB 空间划分为 UM 定义的空间。UM 定义空间的划分如表 6-1 所示。

表 6-1 FAST 流水线的虚拟地址空间范围

空间范围	名称
0x0000 0000-0x0fff ffff	UDP 空间
0x1000 0000-0x1fff ffff	UKE 空间
0x2000 0000-0x2fff ffff	GME 空间
0x3000 0000-0x3fff ffff	UDA 空间
0x4000 0000-0x4fff ffff	UDO 空间
0x5000 0000-0x5fff ffff	保留
0x6000 0000-0x6fff ffff	保留
0x7000 0000-0x7fff ffff	控制接口逻辑空间

FAST 将 4GB 虚拟地址空间的高 2GB 空间定义为 FPGA OS 定义的空间。例如 FAST 规定 FPGA OS 中接口状态寄存器，接口控制寄存器一级时间戳寄存器的定义如表 6-2 所示。

表 6-2 FPGA OS 中固定的寄存器定义

地址	含义
0x80000000	接口状态寄存器

0x80000001	接口控制寄存器
0x80000002	本地时间戳寄存器（63-32）
0x80000003	本地时间戳寄存器（31-0）

详细的 FPGA OS 中虚拟地址空间的定义见具体的平台说明文档。

6.1.2 FAST 分组

fast_packet 定义了 FAST 平台内部数据报文规范，格式如图 6-1 所示。

```
struct fast_packet{
    struct um_metadata um; /*UM 模块数据格式定义*/
    u16 flag;              /*2 字节对齐标志*/
    u8 data[0];            /*完整以太网报文数据*/
}__attribute__((packed));
```

图 6-1 FAST Packet 的定义

6.1.3 重定向表规则

fast_rule 是匹配表规则的数据结构，实现硬件对于报文的转发功能，定义如图 6-2 所示。

```
struct fast_rule{
    struct flow rule;      /*规则内容存储结构*/
    struct flow mask;      /*规则掩码设置结构,与上面一一对应*/
    u32 priority;          /*规则的优先级设置*/
    u32 valid; /*规则的有效标志设置,要删除一条规则只需要将此标记置为0,并更新到硬件*/
    u32 action;            /*规则所对应的执行动作*/
    u32 pad[29];           /*目前,根据硬件规则空间做的保留*/
}__attribute__((packed));
```

图 6-2 FAST Rule 的定义

6.2 编程模型

6.2.1 ODP 编程模型的特点

FAST 编程模型称为 ODP(Open DataPath)模型。ODP 编程 API 主要实现 UA 向平台注册，配置硬件流水线，从硬件流水线接受分组以及向硬件流水线发送分组等功能，如表 6-4 所示。

表 6-3 ODP 主要的编程 API

```
int fast_ua_init(int mid,fast_ua_recv_callback callback) //UA 注册  
  
Int fast_add_rule(struct fast_rule *rule)//配置规则  
  
void fast_ua_recv()//启动接受分组  
  
int fast_ua_send(struct fast_packet *pkt,int pkt_len)//发送分组
```

ODP 有两个重要特点。第一个特点是支持用户直接对网络设备的硬件流水线进行编程。通过 fast_add_Rule() API，UA 可以直接配置硬件流水线中的流表规则。规则的内容包含匹配的关键字值，匹配的掩码，匹配后执行的动作等。其中匹配执行的动作中包含 DMID，PktDes，OutPortBM 等字段，根据这些字段的组合，可以控制分组在硬件流水线输出后的流向。

例如，某个 UA 可进行所有 ARP 分组的处理，对特定视频流的传输质量进行监测。针对 IPTV 传输质量监测需求，RFC4445 定义了

两个关键指标，一是延时抖动，二是丢包率。媒体传送 指数 MDI（Media Delivery Index）指标，对流媒体在网络中传输的延时特性 DF（Delay Factor）和丢失率指标 MLR（Media Loss Rate）进行测量。

IPTV 提供商进行 MDI 指标测量在网络中一方面可以评估 ISP 提供的传输服务质量，另一方面可以对用户满意度体验 QOE 进行评估。

第二个特点是支持软硬件协同处理，即用户 UA 可以通过 Metadata 直接访问到 FPGA 的处理结果，因此可以将一些适合硬件处理的复杂操作，如分类，卸载到硬件中处理。

6.2.2 与现有编程模型比较

FAST ODP 编程 API 与其他用户空间网络编程 API 的比较见表 6-4。

表 6-4 ODP 与其他编程 API 的比较

编程模型	面向对象	主要应用
Socket	通信对端	实现网络中应用程序之间的通信，通常基于 TCP 或 UDP 协议
NetLink	内核模块	用户空间程序与内核中的模块进行通信，例如 OVS 用户控件程序通过 NetLink 与内核中的 datapath 模块通信

libpcap/Libnet	网络接口	用户控件模块直接从网络接口驱动接收以太网分组或发送分组，旁路 TCP/IP 协议栈
DPDK	网络接口	用户空间程序直接从网络接口收发分组，旁路掉接口驱动，内核的缓冲区管理等机制
ODP	数据平面	对数据平面进行编程

FAST-UA 是 FAST 构架下软件部分的用户应用（User Application），可以实现平台无关的核心管理软件，转发面扩展软件以及配置管理关键等功能，由用户在 FAST 构架下根据编程手册自行开发。本文将详细指导用户在 FAST 构架下，开发用户应用模块（FAST-UA）。

6.3 编程 API

FAST 提供三类通用编程 API，包括 UA 管理 API，规则管理 API 和硬件管理 API。

6.3.1 UA 管理 API

UA 管理 API 用于 UA 向平台注册，送平台接收 FAST 分组，向平台发送 FAST 分组，以及打印接收的 FAST 分组等。

- fast_ua_init(): UA 模块初始化函数

-
- `fast_ua_destory()`: UA 模块注销函数
 - `fast_ua_send()`: UA 发送报文功能函数
 - `fast_ua_recv()`: UA 启动报文接收线程
 - `print_pkt()`: 打印 fast 结构报文数据

6.3.2 规则管理 API

规则管理 API 负责配置 FPGA OS 中嵌入匹配引擎中德规则，包括规则的增加，删除，修改和打印等。

- `print_hw_rule()`: 打印配置到硬件的规则；
- `print_sw_rule()`: 打印软件缓存的规则
- `init_rule()`: 初始化规则模块
- `fast_add_rule()`: 添加一条规则
- `fast_modify_rule()`: 修改指定位置的规则
- `fast_del_rule()` : 删除一条规则
- `read_hw_rule()`: 从硬件读取一条指定的规则

6.3.3 硬件管理 API

硬件管理 API 提供了硬件资源初始化，直接读写虚拟地址空间中的数据等。

- `int fast_init_hw()`: 初始化硬件

-
- `fast_distory_hw()`: 释放硬件资源
 - `fast_reg_rd()`: 读取指定寄存器
 - `fast_reg_wr()`: 往硬件写入一条指定的规则内容

6.3.4 订制功能 API

订制功能 API 实现 UA 对 UM 中用户定制功能管理的 API。

例如,当用户在 UM 用户定义输出引擎(UDO)中实现对 Netflow 支持时,需根据管理员的配置,基于分组的 MD 和 PFV 等信息生成 netflow 分组,向外部发送相关的统计信息。因此 UDO 中需要保存外部 Netflow 数据包采集软件所在的服务器地址 `Netflow_Server_Addr`。虽然 UA 可通过硬件管理 API 通过写寄存器的方式直接配置服务器地址(`Netflow_Server_Addr` 的地址寄存器已经在虚拟地址空间中定义),但不直观,易于出错。

因此,可设计专门的定制功能库,其中包含用于读写 `Netflow_Server_Addr` 寄存器的 API,UA 直接调用这些 API 对 `Netflow_Server_Addr` 进行管理,不但可读性好,而且不易出错。

需要注意的是,由于 `Netflow_Server_Addr` 寄存器映射到虚拟地址空间中的位置是确定的,因此对应的订制库中相应的 API 实现是完全与具体的硬件平台无关的,具有很好的可移植性。

6.4 其他应该考虑的问题

6.4.1 UA 在设备中的角色

UA 在网络设备中有两种角色。一是实现分组处理流水线中特定的分组处理功能，作为嵌入硬件流水线的“协处理器”；二是实现相对独立的，动态加载的数据平面服务。例如 UA 可以在 FAST 平台上实现相对独立的服务或 Middlebox 功能，例如实现异常流量监测功能，web 防火墙，以及可编程的流量产生等功能。

6.4.2UA 的地址

每个 UA 可以使用 socket 机制，独立的与远端的实体进行通信。如果 FAST 平台工作在交换机模式，UA 使用交换机控制软件的 IP 地址。

7. FAST 原型系统实现与应用

我们基于大容量 FPGA 和多核 CPU 实现了功能可扩展的可编程网络平台 iRouter。iRouter 参照 FAST 交换模型，将 FPGA 分组硬件处理流程划分为 5 级流水线，分别由 5 个独立的模块实现分组解析、关键字提取、查表、动作执行和输出调度功能，同时支持在 CPU 上的用户空间以进程模式编写各种扩展的软件模块，通过软硬件协同支持各种网络功能的扩展。

7.1 iRouter 实现

iRouter 平台基于 Altera 公司的大容量 FPGA 和 Intel 多核 CPU 实现。内部采用 FAST 交换模型，支持软硬件协同的分组交换处理。用户可通过定制并增加硬件模块和软件模块来实现分组处理功能的扩展。

7.1.1 iRouter 系统组成

(1) FPGA OS

硬件

FPGA OS 是 FPGA 中的逻辑，为用户逻辑 UM 提供运行环境。与操作系统类似，FPGA OS 一方面向 UM 屏蔽不同板卡和 FPGA 的差异，另一方面为简化 UM 的设计提供各种通用服务，如分组的接收和发送，外部存储器的访问，与 CPU 通信的高速 DMA，以及分组处理中需要的规则匹配等。

FPGA OS 还可以为简化 UM 的设计提供各种通用服务，如分组

的接收和发送，外部存储器的访问，与 CPU 通信的高速 DMA，以及分组处理中需要的规则匹配等。

（2）iRouter 支撑软件

iRouter 支撑软件包含 Linux 用户空间实现的 iRouter 库，在 Linux 内核中实现的 iRouter 内核路径控制，以及内核中的 UM 仿真软件三部分。

iRouter 库又分为 iRouter 标准库和 iRouter 定制库。iRouter 标准库对应标准的 iRouter UM 实现，为 UA 提供分组收发，规则配置，平台信息获取（如接口计数器）等功能；iRouter 定制库实现对 UM 中用户扩充模块的管理和访问。通过 Netlink 机制与内核中的 iRouter 路径控制软件通信。iRouter 内核路径控制软件主要在内核中实现多个 UA 与 UM 通信的 MUX/DMUX 功能。内核 UM 仿真软件主要实现对 FPGA 中 UM 功能的仿真，在不包含 FPGA 的标准终端和服务器的支持 UA 的运行。

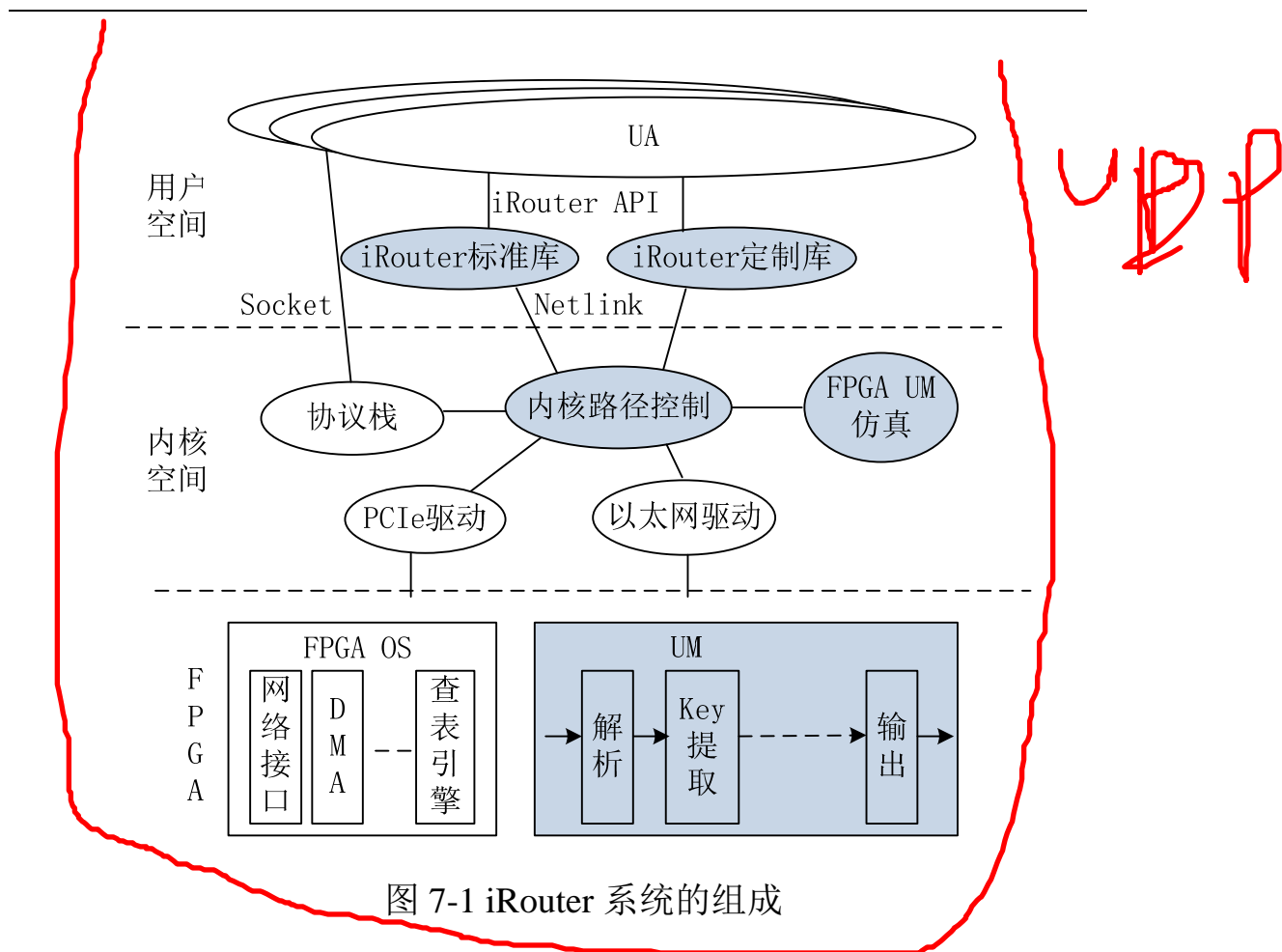


图 7-1 iRouter 系统的组成

标准 iRouter 库主要实现以下功能。一是表管理功能，负责所有控制器配置表格的维护，控制器 flowmod 命令对表操作的命令主要由核心管理软件响应和实现，并实时把新添加的流规则或者更新的流规则经过流表映射的算法，I/O 分组驱动写入到硬件 RAM 中，硬件根据表项转发，处理分组；另外该软件还提供删除流的接口，用来支持删除软件中的流规则信息以及调用接口删除硬件中对应的流规则信息；二是硬件算法实现管理功能。例如有些硬件采用 TCAM 实现带掩码的查表，而有些则采用 SBV 算法实现 288 位带掩码的查表。算法相关软件需要根据表管理软件要求，根据平台相关的算法实现特点，实现对硬件中流表的管理；三是统计管理功能，实现平台的各类

计数器的管理，将硬件实现的计数器（如流表项匹配分组数目计数器、端口接收计数器）和软件实现的计数器（流表项存活时间）进行统一的管理。例如：数据平面实现的令牌桶参数、输出调度器配置的计算等；四是内嵌的 Openflow 协议通道，支持平台与外部的 SDN 控制器建立连接，传递流表，流量统计信息和未命中报文通过 Openflow 通道上传；五是链路状态探测功能，对各端口链路状态进行实时探测，发现链路状态改变后将探测到的链路信息通过 OFP_PORT_STATUS 消息上报至控制器。

（3）用户应用 UA

用户在 linux 用户空间 编写的分组处理进程。每个处理进程对应 FAST 模型中的一个软件模块，硬件流水线通过设置分组 Metadata 中的目的模块号来指定接收该分组的软件模块。

每个 UA 在启动时向平台进行注册，获取自己的 MID 值，通过 iRouter 标准库提供的 分组接收函数 接收分组和发送分组，并向流水线配置相应的转发规则，获取 FPGA OS 中相应的状态，如接口的计数器信息，对 UM 中的硬件流水线进行管理。根据 FAST 模型，当 UA 需要将分组发回硬件流水线时，需要在分组的 Metadata 中指明接收该分组的硬件流水线模块号。

由于 UA 是用户态进程，因此也可以通过 socket 机制 与远程的终端进行通信，甚至可以通过 Libpcap 从标准的网络接口抓包，通过 libnet 向标准网络接口发送分组。UA 也可以调用标准 C 库中的函数

实现特定的处理功能，如文件处理等。

（4）用户模块 UM

FPGA OS 为用户模块 UM 提供标准的运行环境。UM 主要实现 FAST 模型中的硬件流水线功能。

7.1.2 iRouter 的实现

我们实现的 iRouter 实物如图 7-2 所示。支持 2 个万兆以太网接口和 8 个千兆以太网接口。其中 FPGA 为 Altera 公司 Arria V GT 系列，型号为 5AGTMC3D3F31I5N，拥有 58,900 个 ALM，156,000 个逻辑单元（LE）。FPGA 外部包含 1 片 18Megabit 高速同步 SRAM，以及 4 片 2GbDDR3 SDRAM，因此具有较强的片外存储能力。



图 7-2 iRouter 实物图

设备采用 Intel 双核 Atom CPU，提供 1 路 RJ45 管理串口，1 路 10/100/1000Mbps 以太网口，1 路 USB2.0 接口。32G 板载电子硬盘，用于存储操作系统。

iRouter 中的 FPGA 与 CPU 通过 PCIE2.0 总线连接，采用 SDB-DMA 机制，由 FPGA OS 实现 DMA 缓冲区的分配与回收，减

轻了 CPU 的处理负担。因此具有较高性能。

7.1.3 软硬件交互的实现

iRouter 采用 SDB-DMA 和 FAST 交换架构，其软硬件交互的原理如图 7-3 所示。以分组的上行传输为例，当 UM 中的硬件流水线需要将分组送给用户空间的 UA 时，根据 FAST 模型，该分组和元数据将会通过 UM 的 toPort 接口发出，FPGA OS 根据元数据中的标志位判断该分组需要送 CPU 处理，然后该分组将送给 FPGA OS 中的 SDB-DMA 引擎处理。

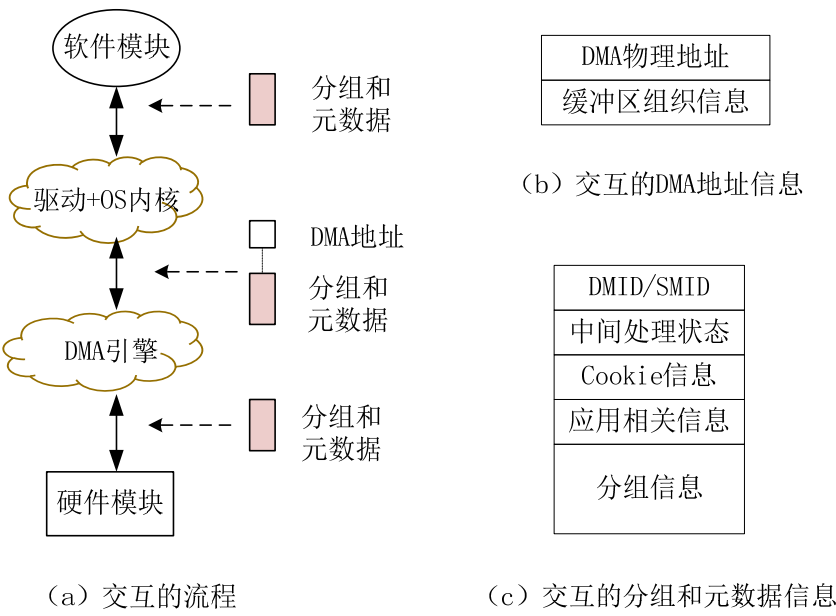


图 7-3 FAST 的软硬件信息交互原理

图 7-3 (b) 包含了为支持 SDB-DMA，在分组缓冲区中需要预先设置的信息，包括缓冲区的物理地址以及缓冲区的组织信息。缓冲区的物理地址供 FPGA 中的 SDB-DMA 使用，即使用该地址发起 PCIe

读写操作，CPU 硬件会将该地址映射到内存空间。缓冲区组织信息包含接收链表的下一个缓冲区的地址信息，该信息由 CPU 的 DMA 驱动使用，因此地址是虚拟地址，必须由 CPU 核进行虚实地址转换后使用。

图 7-3 (c) 包含了 UM 流水线与 UA 进行分组交换时的元数据信息。元数据信息直接在分组头部，主要包含通信的目的 MID 和源 MID 号，各种中间处理状态信息，如 FPGA OS 标记的分组接收时间戳，分组接收端口号，硬件流水线通过查表的到的分组的流 ID 号等。UA 还可以向 GME 的表中设置 cookie 信息，用于硬件流水线告知 UA，该分组上报 UA 的原因，如匹配的规则 ID 等。应用相关信息是与 iRouter 扩充的特定功能相关的，是对元数据的扩展，例如本文 7.3 节中，为了实现 UA 与流水线交换的分组精准发送的时间信息等。

iRouter 的可扩展分组处理流水线如图 7-4 所示，主要分为数据平面硬件模块、数据平面软件 UA 和控制平面软件三部分组成。模块是 iRouter 平台中网络功能实现的基本单元，每个模块由 8 位的模块 ID (MID) 标识，其中 0-127 代表数据平面硬件中的模块，128 代表控制平面模块 (TCP/IP 协议栈)，129-255 代表数据平面软件扩展的模块。通过定义控制不同分组处理通过的模块序列，可实现网络设备多样化的分组处理流程，通过扩充新的软硬件模块，可以支持新的分组处理功能。

每个输入接口接收的分组 (简称 P) 首先被加上 32 字节的

Metadata 信息（简称 M），M 包括分组接收的时间戳，输入接口号，输入接口的接收序列号，以及分组在流水线处理中保存的中间状态，如分组的下一处理模块号（DMID），查表匹配的流 ID 等；

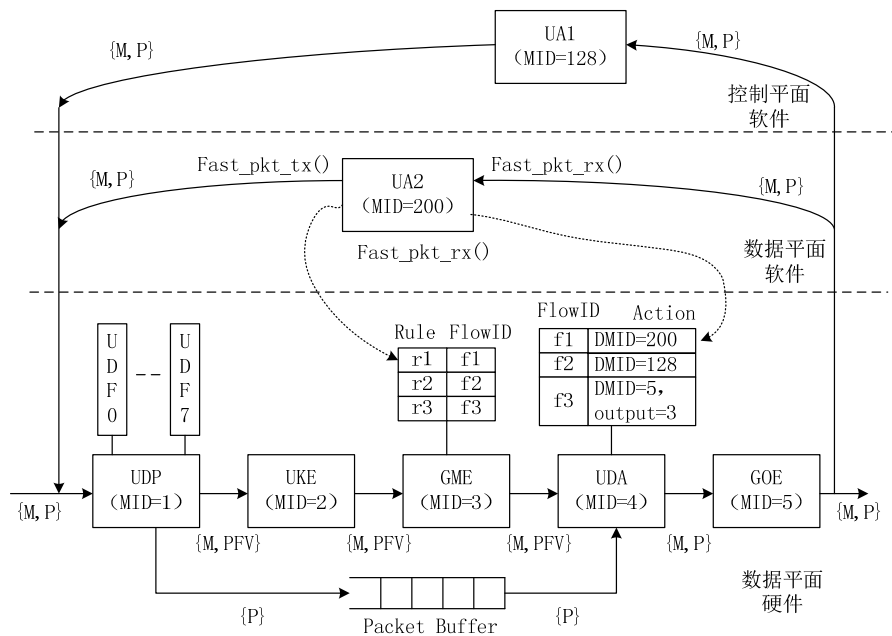


图 7-4 iRouter 的可扩展分组处理流水线

数据平面硬件由 FPGA 实现，分为 FPGA OS 和用户模块 (UM)。FPGA OS 实现了网络设备平台通用的处理功能，如分组的接收发送、CRC 校验、外部存储器接口以及与 CPU 通信的 DMA 引擎等。不论设备实现的是交换机还是网关等其他功能，FPGA OS 都保持不变，而 UM 逻辑存在差异。

数据平面 UA 软件是位于 CPU 用户空间、基于 FAST 开放数据平面接口 (ODP API) 开发的与用户功能相关的软件进程。该进程通过 `add_rule()` 函数直接配置硬件 UM 中相关表格，使数据平面硬件将满足特定规则的数据平面分组定向到本进程处理。例如，图 1 中

MID 编号为 200 的用户进程通过向 GME 流表和 UDA 的动作表配置 (r1-f1) 以及 (f1-MID=200) 的规则，将满足规则 r1 的所有分组定向到 UA 处理。UA 通过 UA_recv()函数接收分组 P 和相应的 metadata，对分组处理后，通过 UA_send()函数将分组发出。UA 在发送分组时，通过设置 metadata 中的 DMID 值，可以指定接收处理该分组的硬件流水线模块。例如将分组直接从输出接口发出时，可在 Metadata 中设置 DMID=5 以及相应的输出接口；若要硬件流水线重新解析处理该分组，只需将 DMID 设置成 1。

iRouter 数据平面模块通过将分组 metadata 中 DMID 设置为 128，即可将分组送给控制平面协议栈，由控制平面协议栈对分组进行分析后，送不同的协议模块处理（如 ARP 协议、各种路由协议），或者通过 socket 接口送本地的应用程序。

7.2 对 LISP 协议的扩展支持

7.2.1 LISP 协议工作原理

核心与边缘的分离是解决路由可扩展性的重要手段。思科公司主导的核心与边缘分离的 LISP 技术已经在 IETF 标准制定和设备实现上取得了重要突破。LISP 的基本原理是在核心边缘分离网络架构中采用路由“分治”的思想，将互联网上的网络划分为两类，包括 ISP 控制的核心网络 and 用户控制的边缘网络。核心网络路由系统可采用 PA(provider aggregateable)地址来提高路由的可聚合性，不关心边缘网络的变化。这种方式不但可以保持核心网络路由的稳定性，而且大大

减小核心路由器的路由表规模。边缘网络中可采用 **ISP** 独立的端系统地址支持网络的多宿主和动态迁移能力。边缘网络通过 **LISP** 的 **XTR** 路由器接入核心网络。

不同边缘网络中的主机通过 **LISP XTR** 路由器建立的 **LISP** 隧道通过核心网络。在 **LISP** 隧道的起点，**XTR** 需要通过查表获取目的 **XTR** 的核心网地址（**RLOC**），然后按照协议规范在原来的 **IP** 分组前面增加 **IP/UDP/LISP** 头。核心网使用外层的 **IP** 头进行核心网路由。**LISPXTR** 的工作原理以及 **LISP** 分组的封装如图 7-5 所示。

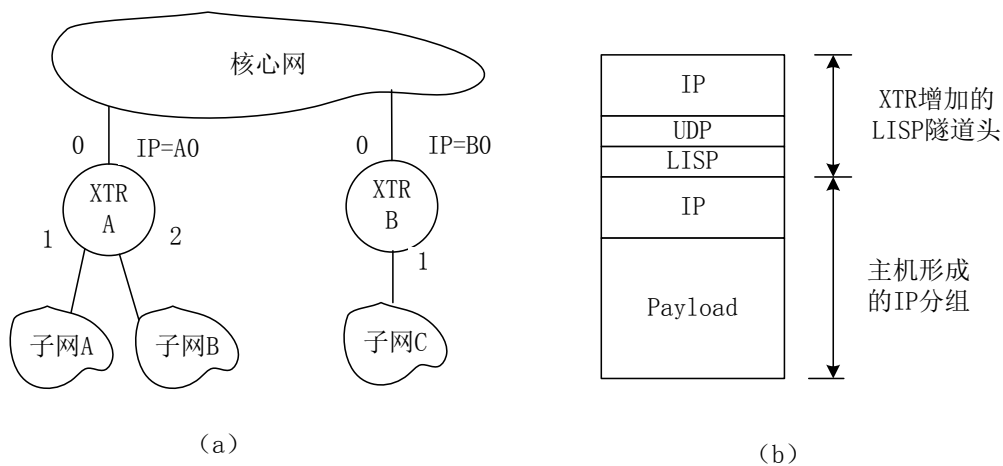


图 7-5 LISP 分组协议头封装

LISP XTR 路由器在数据平面的主要功能是在分组发往核心网时添加 **LISP** 协议头，在从核心网接收到分组时，剥离协议头。其中添加协议头的目的 **IP** 地址为对端 **XTR** 在核心网的 **IP** 地址，由控制平面的 **LISP** 映射系统获取。

一般情况下，**LISP XTR** 只有一个接口连接核心网，一个或多个

接口连接本地边缘网络，如图 7-5（a）所示。LISP XTR 在数据平面的转发操作分为从本地边缘网到核心网的上行转发操作以及从核心网到本地边缘网络的下行转发操作。这两类操作分别如表 7-1 和表 7-2 所示。

表 7-1 LISP XTR 的上行转发操作

	Upward_Forwarding(P,inport) //上行转发
1	If (P.desIP==Local_IP)
2	Send_to_Cntrol_plane(P)
3	Else
4	upfwd_info=upfwd_lookup(P.desIP)
5	If (upfwd_info.hit==1)
6	P=EncapLISP(P,upfwd_info.desRLOC);
7	tx_pkt(P,port0)
8	Else drop(P)
9	Else Drop(P)

在上行转发操作中，第 4 步是 XTR 根据分组的目的 IP 地址查找目的 XTR 的 IP 地址 upfwd_info.desRLOC。在第 6 步根据该地址在原有的 IP 分组上再增加一个 IP/UDP/LISP 协议头。

表 7-2 LISPXTR 的下行转发操作

	Downward_Forwarding(P,inport) //下行转发
1	If (P.desIP==Local_IP)
2	Send_to_Cntrol_plane(P)
3	elseif (type(P)==LISP)
4	P=DecapLISP (P);
5	downFwd_info=downfwd_lookup(P.DesIP);
6	If (downfwd_info.hit==1)
7	P=EncapEth(P, downfwd_info.desMAC);
8	tx_pkt(P, downfwd_info.output_port)
9	Else Drop(P)
10	ElseDrop(P)

在下行转发操作中,第 4 步首先解封装,去掉外层的 IP/UDP/LISP 头,然后第 5 步根据内层的 IP 地址查找邻接表(下一跳的 MAC 地址)和下一跳输出接口,第 7 步更新目的 MAC 地址后,将分组从指定的本地接口输出。

7.2.2 LISP XTR 功能的实现

由于 iRouter 数据平面本身只支持 IPv6 分组的转发,因此我们

对 iRouter 的数据平面进行扩展，分别在 UM 的流水线中插入 XTR FWD 模块以及在 Linux 用户空间设计了路径 MTU 管理模块，如图 7-6 所示。

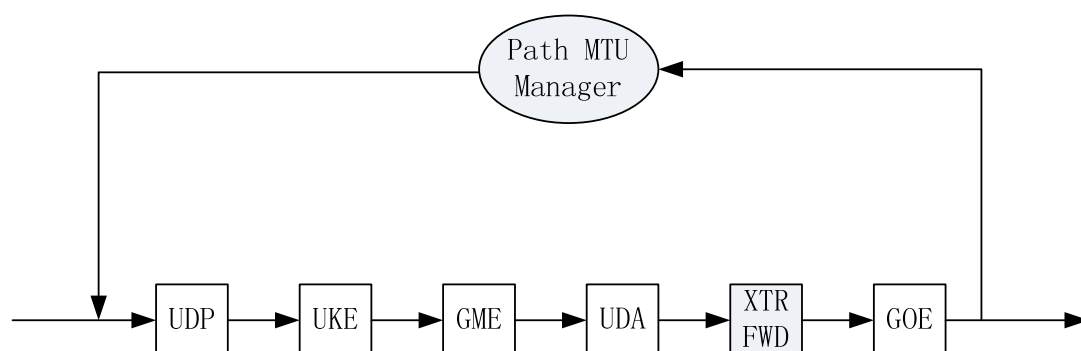


图 7-6 支持 LISP XTR 的 iRouter 数据平面扩展

(1) XTR FWD 模块

XTR FWD 模块编号为 6，插入在 UDA 模块和 GOE 模块之间。由于 UDA 模块已经从分组缓冲区中取出完整的分组，因此 XTR FWD 模块可以对分组进行上行的加头和下行的去头操作。利用分组元数据中携带的 flowID（在 GME 阶段查表获得），XTR FWD 模块实现表 7.1 中处理的 4-6 步，表 7-2 中处理的 4-5 步功能。

UDA 模块中根据分组的 flowID 查 NMI 表，若是需要 XTR FWD 处理的分组，则将分组的目的 MID 设为 6，否则设为 5。当 XTR FWD 接收到目的 MID 为 5 的分组后，不加处理，直接送到下游的 MID 为 5 的 GOE 模块。

由于 LISP 分组的加头去头全部是由流水线中得硬件模块实现，因此性能不是瓶颈。

（2）路径 MTU 管理模块（Path MTU Manager）

由于目的 LISP XTR 路由器工作在 IPv6 网络中，因此在上行的分组外面再封装 IPv6/UDP/LISP 分组头，额外增加 56 个字节，假设链接核心网的接口 0 的 MTU 为 1500 字节，那么当边缘网络到达的分组长度超过 1444 字节时，LISP 封装可能会导致分组超长。

路径 MTU 管理模块（PMM）是运行在 Linux 用户空间的 UA，模块 MID 为 200。主要功能是接收从硬件流水线中发过来的，长度超过 1444 字节的，需要封装 LISP 头的分组。根据 IPv6 链路发现机制，PMM 模块会丢弃该分组，然后向该分组的源发送 ICMP 消息，通知源将链路 MTU 设置为 1444。这样就避免了在 XTR 上进行分组的分片和重组。

我们在流水线的自定义解析器 UDP 中增加一个分组长度标记位 TLP (Too long packet)，当分组长度大于 1444 字节时，该标记位置 1，否则为 0。TLP 标记位会在分组特征向量 PFV 中携带，跟随分组传递到流线下游的每个处理模块。当 XTR FWD 模块发现到达分组需要增加 LISP 头且 TLP 标志置位时，会将该分组的目的 MID 设置为 200，直接发送给下游的 GOE 模块。GOE 模块会旁路该分组的执行，直接将分组输出到 FPGA OS，FPGA OS 会根据目的模块号 200 将该分组通过 DMA 发送给到 CPU，然后由内核中得路由控制软件将该分组

送给 LMM UA 模块处理。

PMM 在发出 ICMP 分组时，将目的 MID 设置为 5 (GOE 的 MID 号)，同时在分组的元数据中填写输出接口号。该分组虽然首先被送到硬件流水线的入口模块 UDP，但这些模块都直接旁路该分组的处理，直接送到下游模块，直至 GOE 模块，最后 GOE 模块将该 ICMP 分组从指定的输出接口发出。

7.2.3 LISP 转发功能的应用

基于对 iRouter 设备 LISP 转发功能的扩展，我们基于 LISP 隧道，在 CNGI IPv6 网络上构建了连接国防科大，清华大学，东南大学，中科院计算所，中国电信北京研究院等 5 个研究机构的 CAFASTR (Core And Edge Separating ARchitecture) 实验网，拓扑如图 7-7 所示。

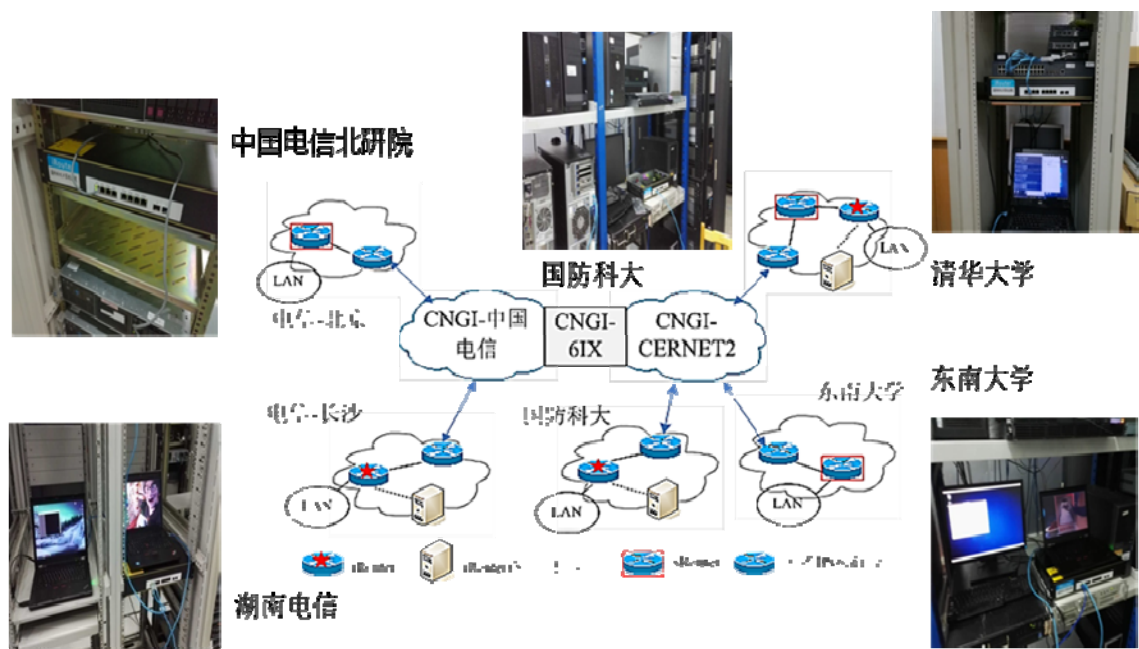


图 7-7 基于 iRouter 的实验网拓扑

实验网由统一的控制器实现对 LISP 的映射管理。由于 LISP 协议隔离了边缘实验网络 and 核心 IPv6 网络的地址空间，因此在上述研究单位的实验网络中可以部署各种新型的地址和网络转发机制。目前清华大学和国防科大基于上述实验网分别进行了新型域间路由协议的实验和假冒 IPv6 源地址识别和过滤的实验。

基于 iRouter 的 LISP 协议实现和 CAFASTR 实验网组建，说明了基于 FAST 交换模型，通过软硬件模块的扩展，可以方便的实现对新型网络协议的支持。

7.3 对假冒 IP 源地址识别的扩展支持

DDoS 是目前网络上日益增加，又难以防范的攻击行为。DDoS 通过控制多台计算机对目标发起恶意请求，让其无法提供正常服务。由于 DDoS 攻击通常使用假冒的源 IP 地址，因此对假冒源 IP 地址的识别和流量清洗是抵御 DDoS 攻击的有效手段。

7.3.1 假冒源 IP 地址识别原理

我们的假冒 IPv6 源地址识别基于清华大学提出的 CABA 编址技术，CABA 编址将拥有该 IPv6 地址的自治系统号（AS 号）嵌入 IPv6 的地址前缀，因此实现了 IPv6 地址到其位置或管理者的显式映射。CABA 编址的格式如图 7-8 所示。

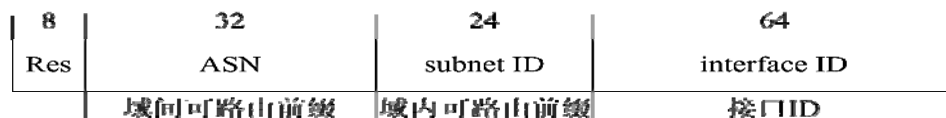


图 7-8 CABA 编址格式

我们的假冒源 IPv6 地址检测在基于 LISP 搭建的实验环境中进行。其基本原理是每个实验的边缘网络属于一个独立的 AS，内部使用 CABA 编址，因此这些 AS 中发出的分组目的 IPv6 地址中包含目的主机所在 AS 的编号，源 IPv6 地址中包含本地 AS 的编号。LISP XTR 在进行 LISP 分组转发处理时，进行假冒源 IPv6 地址检测。算法如表 7.3 所示

表 7-3 LISP XTR 进行假冒源 IPv6 地址的检测算法

1	If (XTR 上行转发分组 P)
2	If (P.sEID.asn==local_asn)
3	Forward(P);
4	Else
5	Drop(P);
6	Else if (XTR 下行转发分组 P)
7	src_asn=lookup(P.srcRLOC)
8	If (P.sEID.asn==src_asn)

9	Forward(P)
10	Else
11	Drop(P)

当 XTR 上行转发分组时，需要判断发出分组的源 IPv6 地址嵌入的 AS 号是否等于本地的 AS 号（第 2 步），如果不等，则可能本地网络内部的恶意软件在使用随机产生的 IPv6 源地址，将该分组丢弃，否则正常转发；当 XTR 下行转发时，根据分组源 IPv6 地址查找 LISP 映射表（第 7 步），获取该地址对应的 AS 号 src_asn，该编号是 LISP 映射系统中维护的，如果 src_asn 不等于分组源 IPv6 地址内部携带的 AS 号，说明该分组并非来自 src_asn 标识的自治系统，为假冒的 IPv6 源地址，因此将其丢弃（第 11 步）。

7.3.2 假冒源 IP 地址识别的实现

在 iRouter 扩展 XTR FWD 硬件模块和 PMM 模块支持 LISP XTR 转发的基础上，我们通过软硬件协同的方式，在 iRouter 上进一步扩充 FID（Faked IP Detector）硬件模块和 FIDC（Faked IP Detecting Controller）软件 UA，实现了对基于 CABA 编址的假冒源 IPv6 地址的识别，如图 7-9 所示。

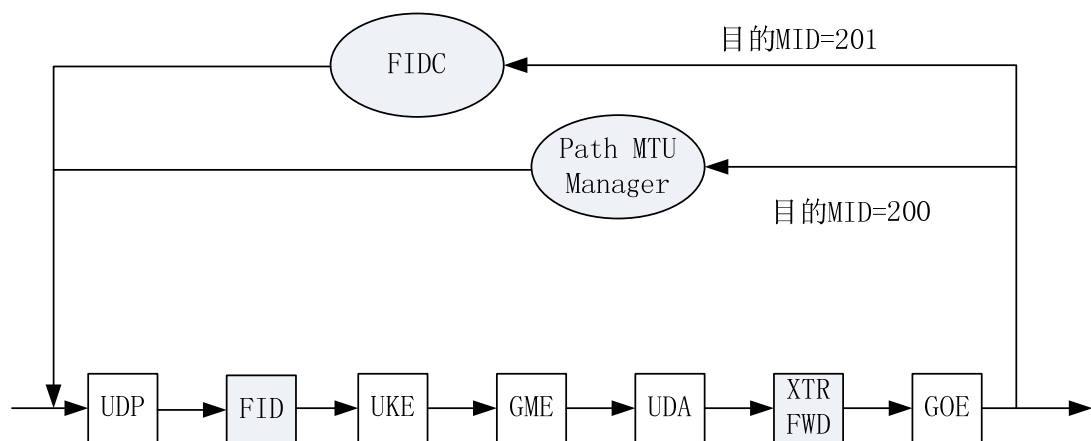


图 7-9 iRouter 支持假冒源 IP 地址检测扩展

(1) FID 模块

FID 模块 ID 为 7, 位于 UDP 和 UKE 模块之间, 主要实现对 LISP 上行和下行分组的源 IPv6 地址的合法性进行检测。由于 LISP 的下行转发检测中的查表操作 `lookup(P.srcRLOC)` 会增加 GME 模块中查找 LISP 转发表的复杂性, 因此将上述操作作为独立的 FID 模块实现。

FID 模块检测所有从 UDP 模块发出的分组, 同时不直接将判断出的假冒源 IPv6 的分组丢弃。根据配置, FID 模块对于丢弃的分组有两种处理策略。一是将该分组元数据中的目的 MID 设置为 5, 把丢弃位设置为 1, 由 GOE 模块将该分组进行丢弃。因为所有硬件流水线丢弃的分组都会发给 GOE 丢弃, 便于 GOE 模块对丢弃分组进行统一的计数; 第二种策略是将丢弃分组的目的 MID 设置为 201, 发送给软件中的 FIDC 模块进一步进行分析处理。

由于 FID 将丢弃分组目的模块号设为 5(GOE)或者 201(FIDC),

因此 FID 模块下游的处理复杂，访存开销比较大的 GME、UDA 等模块都不会处理该分组，因此 iRouter 硬件流水线不会受到假冒 IP 分组的 DDos 攻击。

（2）FIDC 模块

FIDC 模块是 Linux 用户空间编写的 UA 程序，MID 为 201。FIDC 负责对 FID 模块进行管理，同时处理和分析 FID 上报的假冒 IP 地址的分组，进行统计或详细的审计。由于软件 FIDC 的处理能力有限，因此通常将 FID 配置为采样上报模式，只有一定比例的分组，如 1%，送 FIDC，其余的送 GOE 丢弃。

iRouter 内核中的路径控制软件会根据硬件流水线上报分组的元数据中的目的 MID 编号，将分组送给 PMM 模块（MID=200）或 FIDC 模块（MID=201）。

7.3.3 假冒源 IP 地址识别的应用

我们在 CAFASTR 实验网上，我们使用 Hping3 攻击软件（一种开源的，可假冒源地址，发起多种 DDos 攻击的工具），从中科院计算所所在 CABA 边缘网络的主机 A 经 iRouter B，CNGI 核心网，iRouter A 向国防科大内部的 CABA 网络中服务器 B 发起 DDos 攻击。如图 7-10 所示。

与其他实验网不同，基于 CAFASTR 实验网采用 LISP 隧道连接各实验网，所有攻击报文都是封装在 LISP 分组格式中穿越核心网的，

因此 DDos 攻击实验不影响 CNGI IPv6 核心网的正常运行。

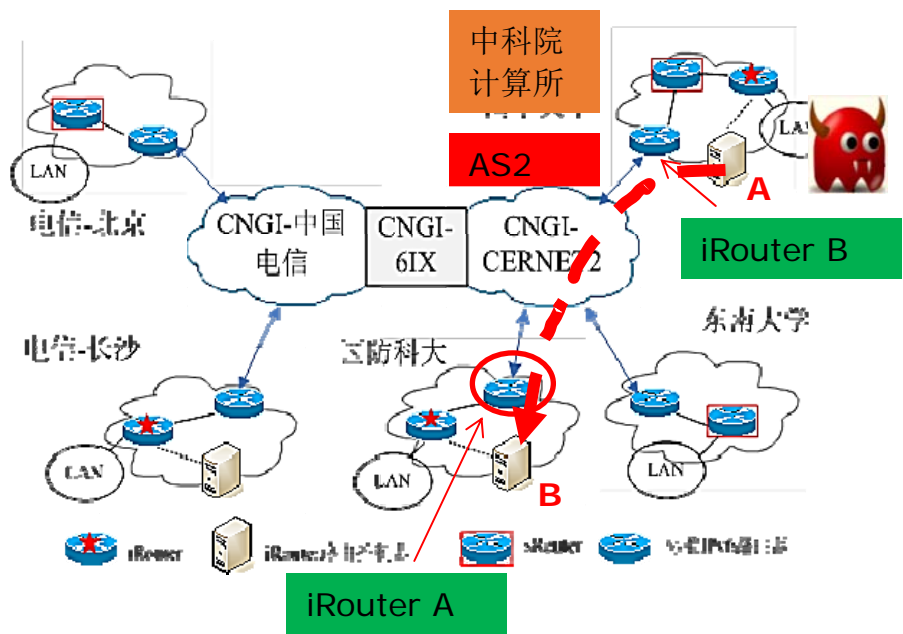


图 7-10 假冒 IPv6 源地址检测和过滤实验场景

当不启用 iRouter 的 FID/FIDC 模块时，我们可以明显看到，DDos 攻击发起时，服务器的服务能力明显下降。在分别启用 iRouter A 和 B 的 FID/FIDC 模块时，从 CAFASTR 实验网集中的管理界面可以观测到大量假冒 IPv6 源地址的分组被发现和过滤，如图 7-11 所示。

Controller

首页

全网拓扑

层叠网模式

实网模式

假冒源IP检测

DDOS流量清洗

关于

全网拓扑

iRouter信息

iRouter信息(4)

Show10entries

Search:

iRouter标识号	IP地址	站点位置	连接建立时间
01:00:11:22:33:44:55:66	/2001:250:4401:2000:0:0:100:58828	国防科大	2017-09-23 10:16:06
05:00:11:22:33:44:55:66	/2400:dd01:1034:e00:913f:9bda:7e99:574d:54996	计算所	2017-09-23 10:22:51
06:00:11:22:33:44:55:66	/2001:c68:300:40:0:0:0:81:43744	北京电信	2017-09-23 10:24:33
07:00:11:22:33:44:55:66	/2001:da8:a0:f001:0:0:0:103:51790	比威	2017-09-23 10:23:30

iRouter标识号

IP地址

站点位置

连接建立时间

Showing 1 to 4 of 4 entries

Previous

1

Next

(a) CAFASTR 实验网上 iRouter 的状态



(b) iRouter B FIDC 模块抽样获取假冒本地源 IP 地址的统计信息



(c) iRouter A FIDC 抽样获得假冒源 IPv6 地址分组的统计信息

图 7-11 iRouter 扩展支持假冒源 IP 地址分组的实验效果

上述实验表明，通过 iRouter 的软硬件模块扩展和协同处理，可以方便的实现和部署新型的网络安全防护功能。进一步说明了我们提出的 FAST 交换模型的功能扩展能力。

7.4 对精准测量服务的扩展支持

主动测量是获取网络状态的有效手段。测量分组精确的发送时间控制和接收时间戳标记是主动测量的关键。由于受到内核调度，中断处理等因素干扰，在终端上实现的主动测量很难保证测量分组发送和接收时间标记的精确性。iRouter 平台中的 FPGA 硬件可以精确的控制分组的发送时间和标记接收时间，因此可以为各种网络测量提供精准的测量服务（AMS）。

7.4.1 精准测量服务的原理

AMS 服务的基本思想是将主动测量的起点和终点由边缘的主机和服务器改为 AMS 平台，由 AMS 平台利用自身的 FPGA 实现各种测量分组的精准发送和接收时间戳的标记。由于发送和接收时间的标记都由 FPGA 实现，因此精确度可达到 ns 级。

AMS 实现的基本流程是端系统将测量需要发送的分组以及发送的时间需求（如多个分组背靠背发送，采用指定间隔的报文串（Packet train）模式发送等），通过 socket 等机制通告给 AMS 服务，AMS 服务利用 FPGA 硬件的特性，将测试分组序列按照要求向网络定时发出，记录每个分组的精准发送时间，同时接收返回的测量分组，精确

记录返回分组的时刻，然后将测量分组发出和返回的时间返回给发起测量的端节点。

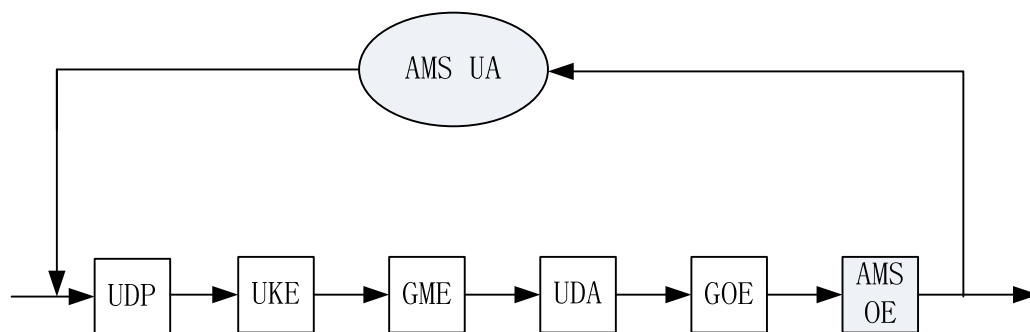
作为通用的测量服务，AMS 并不关心具体的测量算法，只是根据测量应用的指令将分组定时发送，并精确标记分组返回的接收时刻。因此 AMS 可作为基于 FPGA 转发的交换设备上的一种新的服务提供。

7.4.2 AMS 的实现

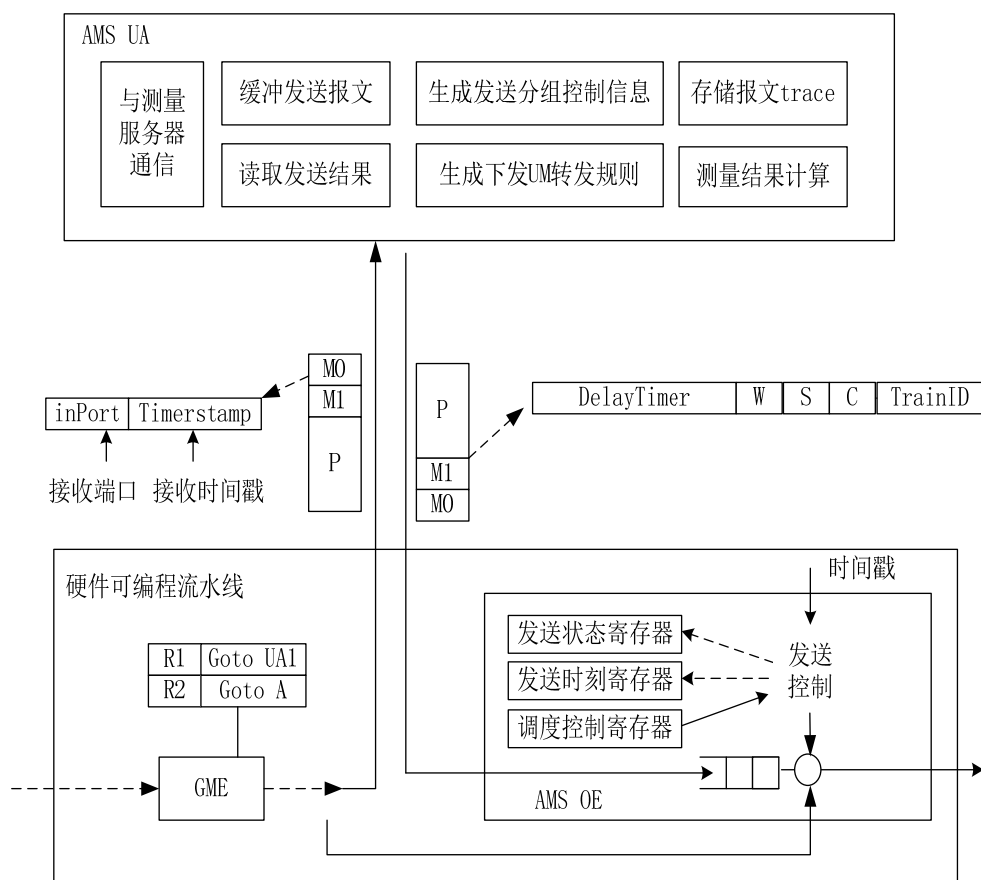
（1）iRouter 的模块扩展

我们在 iRouter 上扩展支持了 AMS 服务，主要包括硬件流水线中扩展了 AMS 输出引擎模块 AMS OE，以及在 Linux 用户空间扩展了 AMS UA，如图 7-12 所示。其中 AMS UA 负责测试分组序列的生成与返回值计算，AMS OE 模块负责分组的精确发送控制。

为不影响 iRouter 其他功能，AMS OE 连接在流水线的最末端，由于 AMS 测量分组对时间精确需求以及测量分组个数较少，所以该类分组输入调度的优先级最高，即只有在不干扰 AMS 测量分组正常发出时，才能调度输出其他分组。



(a) iRouter 通过软硬件模块扩展支持 AMS 服务



(b) AMS OE 和 AMS UA 的具体实现

图 7-12 iRouter 对 AMS 的扩展支持

(2) AMS OE 的实现

AMS OE 模块 MID 为 8, AMS OE 需要对 GOE 调度输出的所有分组进行再次调度。首先需要确保 AMS UA 发送的测量分组必须按

照指定的时刻发送, 在没有测量分组发送时, 才发送其他非测量分组。由于 iRouter CPU 软件与 FPGA OS 难以实现 ns 级的时钟同步, 因此 AMS UA 并不制定绝对的分组发送时间, 只向 AMS OE 模块指定分组序列的相对发送时间, 即发送序列中前后相邻分组的间隔。由于 AMS UA 和 AMS OE 之间需要交换测量分组的发送时间, 因此我们扩充了分组的元数据字段。扩充的内容如表 7-5 所示。

表 7-5 元数据扩充的字段

字段名称	说明
DelayTimer[15:0]	相对本序列的上一个发送分组, 需要延时发送的值, 以 8ns 为单位
S	1 表示立刻发送该分组, 0 表示延时发送该分组, 延时的值由 DelayTimer 定义
W	1 表示将发送时间写入发送时间寄存器中, 0 表示不写。
C	记录发送状态标志, 1 表示未能按照指定时间发送, 0 表示分组按时发送
TrainID[15:0]	标识不同的测试序列 ID

AMS OE 模块对于每个测试序列需要维护一个分组缓存队列 (FIFO) 和三个映射到 AMS UA 可访问的寄存器。分别为发送时刻

寄存器 (STR)、调度控制寄存器 (SCR) 以及发送状态寄存器 (SSR)。

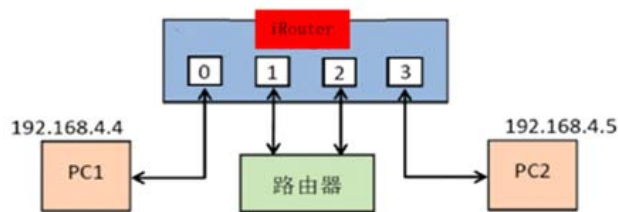
发送时刻寄存器 (STR) 保存测试序列中的一个分组发送的时刻, 在测试结束后, AMS UA 可读取该寄存器获取发送序列第一个分组的发送时刻; SCR 寄存器由每个分组元数据中的 `delaytimer` 设置, 即在上一个测试分组发送结束后, 需要等待多少时间发送本分组; SSR 寄存器记录发送本测试序列是否按照要求完成对每个分组的发送, 测试结束后, AMS UA 可读取该寄存器获取发送是否成功的状态。

(3) AMS UA 的实现

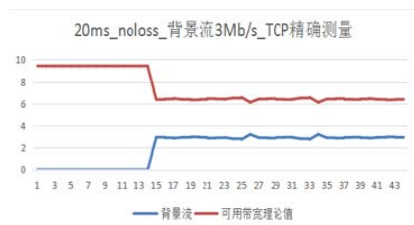
AMS UA 由用户空间的进程实现。AMS UA 进程包含多个线程, 分别实现与远端测试终端通信、分组接收 (元数据+分组数据)、存储测量分组数据以及计算测量结果等功能。若 AMS UA 需要同时发送和接收多个测量分组序列, 则采用多个独立的线程, 由每个线程实现一个测试序列的接收和发送。

7.4.3 AMS 的评估与应用

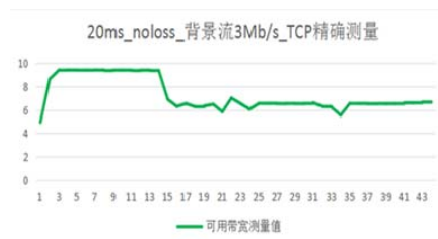
通过 AMS 服务的扩展, iRouter 平台可以支持各种精准的网络测量场景, 图 7-13 为东南大学程光教授的网络测量团队对 AMS 服务能力的初步评估结果。



(a) AMS评估测试拓扑



(b) 设置背景流后，链路可用带宽的理论值



(c) 基于AMS服务测量的链路可用带宽值

图 7-13 对 AMS 能力的评估

上述评估表明，理论值和测量值在趋势上基本吻合，在数值大小上略有上下浮动，在变化速度上，可用带宽的测量值收敛快，稳定到理论值附近大概用了 3 到 5 个测量周期，即 60 到 100 个数据包左右。目前东南大学已经对基于 iRouter 的 AMS 进行了扩展，并展开了数据中心 TCP 性能的测量研究。

8. 结束语

FAST 开源项目希望能够为国内高等院校，相关研究所和公司中从事网络技术人员提供开源的可编程开发平台，避免研究人员将大量精力花费在网络平台的研制上而不断的“重新发明轮子”。

FAST 项目中采用的软硬件协同分组处理的架构能够有效支持在网络设备上部署新的协议，新的分组处理机制和新的服务，因此能够快速实现对网络新技术的验证。

FAST 项目目前仅实现了路由和交换的基本平台，仍有大量的软硬件开发工作进行，希望更多的研究人员关注 FAST 开源项目，参与项目的开发，或为 FAST 项目的发展提出意见和建议。