分类号 TP393 学号 13060041

U D C 004.7 密级 公 开

工学硕士学位论文

基于多核 NPU 的 TCP 数据传输卸载

硕士生姓名 李杰

学 科 专 业 计算机科学与技术

研 究 方 向 网络安全

指 导 教 师 陈曙晖 研究员

国防科学技术大学研究生院

二〇一五年十一月

论文书脊

（此页只是书脊样式，学位论文不需要印刷本页。）

基于多核 NPU 的 TCP 数据传输卸载

国防科学技术大学研究生院

Multicore NPU based TCP Data Transmission Offload

Candidate：Li-jie

Advisor：Prof. Chen-shuhui

A thesis

Submitted in partial fulfillment of the requirements

for the degree of Master of Engineering

in Computer Science and Technology

Graduate School of National University of Defense Technology

Changsha, Hunan, P.R.China

November, 2015

独 创 性 声 明

本人声明所呈交的学位论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表和撰写过的研究成果，也不包含为获得国防科学技术大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文题目：基于多核 NPU 的 TCP 数据传输卸载

学位论文作者签名：李杰　　日期：2015年10月24日

学位论文版权使用授权书

本人完全了解国防科学技术大学有关保留、使用学位论文的规定。本人授权国防科学技术大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目：基于多核 NPU 的 TCP 数据传输卸载

学位论文作者签名：李杰　　日期：2015年10月24日

作者指导教师签名：陈曙晖　　日期：2015年10月24日

目 录

表 目 录

图 目 录

摘 要

TCP（Transmission Control Protocol）是互联网中的一个重要协议，在互联网中得到了广泛的应用。提升 TCP 服务的性能可以降低服务器集群的数量，降低功耗，具有很高的商业价值和环保意义。目前以太网的发展速度远高于存储器和 CPU 的发展速度，存储器访问和 CPU 处理网络协议已经成为 TCP 的性能瓶颈。网络带宽的不断增大对 CPU 造成了沉重的负担，通过优化 TCP 处理机制，可以降低主机开销，提升 TCP 性能。

传统优化 TCP 处理机制的方法中，协议处理仍由主机 CPU 执行。TOE（TCP Offload Engine）将 TCP 协议处理的功能完全卸载到网卡中执行，极大的提高端系统 TCP 性能，但其实现特别复杂，并且存在安全性和兼容性问题。LRO（Large Receive Offload）技术通过合并数据报文减少协议栈处理报文的数量，降低 CPU 开销，但其工作在网卡驱动程序层面，报文合并工作仍由主机 CPU 执行，不能很大程度上减轻 CPU 的负担。

针对 TOE 和 LRO 技术的缺点，本文提出使用多核 NPU 作为网卡卸载 TCP 乱序报文重组功能、合并报文加速 TCP 的技术，本文主要工作如下：

（1）首次提出使用多核 NPU 作为网卡，卸载 TCP 乱序报文重组功能，并将同一个 TCP 连接上的数据报文合并后交由内核协议栈处理，减少协议栈处理报文的数量和网卡产生中断的数量，提升端系统 TCP 性能的技术。

（2）设计了系统的框架结构和功能组成，并针对多核 NPU 的特点提出多接收报文描述符环、合并报文校验和计算优化、接收报文处理线程 DMA 负载均衡、主动 ACK 机制等系统优化技术。

（3）基于 XLS416 开发平台实现系统，并在 10Gbps 网络环境中测试其性能，取得 4.9Gbps 的 TCP 接收数据吞吐量。

主题词：TCP 乱序重组；TOE；LRO；多核 NPU

ABSTRACT

TCP (Transmission Control Protocol) is one of the most important network protocols and is extremely widely used. Improving TCP performance can reduce server cluster scale and power consumption, brings both commercial and environmental benefits. Nowadays, the Ethernet technology is developing much faster than storage and CPU technologies, memory access and CPU processing network stack have become the bottleneck of TCP performance on end systems. The constantly increasing network bandwidth has caused a severe burden for CPU, optimizing TCP processing mechanism can relieve CPU from this and improve end system TCP performance.

Traditional TCP acceleration techniques focus on host side optimization, protocol processing is done by host CPU. TOE (TCP Offload Engine) offloads the entire TCP protocol processing job from host CPU to NIC, and can greatly improve the end system TCP performance, but its implementation is extremely complex and it can cause security and compatibility issues. LRO (Large Receive Offload) aggregates consecutive TCP packets into a single one and decreases CPU workload by reducing the number of packets processed by the network stack, but LRO works on the NIC driver level and packets aggregation is still done by host CPU, so it cannot decrease much CPU workload.

The main work of this paper is:

(1) For the first time, we proposed the idea of using multicore NPU as NIC to accelerate TCP processing. The multicore NPU offloads TCP packets reordering, checksum calculation functions and aggregates small data packets into large but much fewer ones, thus reduces the number of packets processed by network stack and the number of interrupts generated by NIC, eventually improves TCP performance on an end system.

(2) We designed system architecture and functions, and proposed system optimization techniques such as: multiple received packet descriptor ring, checksum optimization for aggregated packets, DMA load balance for received packets processing threads, and spontaneous ACK mechanism.

(3) We implemented the system on XLS416 platform and tested its performance, experiment results show that 4.9Gbps TCP receive data throughput is achieved in a 10Gbps network environment.

Key Words：TCP packets reordering; TOE; LRO; Multicore NPU

第一章 绪论

1.1 研究背景

TCP 是互联网中的一个重要协议，它提供了端系统之间的数据可靠传输、多路并传、基本的流量控制和拥塞控制等功能，在互联网中得到了广泛的应用。虽然服务器的处理能力、PCI 总线的带宽都有很大的提高，但服务器端 TCP 服务的性能仍然是内容提供商、游戏提供商、门户网站、搜索引擎所面临的挑战。提高 TCP 服务的性能，不仅能够降低服务器集群的数量，也能降低功耗，实现更加绿色环保的 IT 服务，具有很高的商业价值和环保意义。

目前存储器和 CPU 的发展速度远落后于以太网的发展速度，千兆以太网接口已经广泛配备于端系统中，40Gbps 和 100Gbps 以太网技术也已出现。存储器访问和 CPU 处理网络协议已经成为 TCP 的性能瓶颈，不断增大的网络带宽对 CPU 造成了沉重的负担，大约需要 1GHz 的 CPU 资源处理 1Gbps 网络流量。通过优化 TCP 处理机制，可以降低主机开销，提升 TCP 性能。

传统优化 TCP 处理机制的方法主要集中在主机端，包括优化校验和计算、优化大概率事件处理、减少数据复制、减少中断数量等技术。校验和计算本身并不复杂，但计算过程需要读取所有数据，存储器访问操作使其开销变大，延时进位、循环展开等技术充分挖掘 CPU 体系结构特性，提高计算速度；利用反码加法的交换律和结合律可以在报文头部部分字段改变后，不遍历所有数据而更新校验和，减少存储器访问开销。优化大概率事件处理技术包括使用高速缓存实现 PCB（Protocol Control Block）、利用编译技术优化 cache 命中率、首部预测等，从指令级别优化 TCP 处理。数据复制涉及存储器访问操作，开销巨大，零拷贝、共享存储空间等技术可以有效减少存储器访问次数，提升 TCP 性能。由于报文到达的时间未知，网卡一般采用为异步中断的方式通知端系统进行报文接收，中断引起的上下文切换开销巨大，结合轮询、中断合并等技术可以有效减少网卡产生的中断数量，减少主机开销。处理器核心数量逐渐增多，多线程技术、多接收队列和 RSS（Receive Side Scaling）技术随之出现，TCP 连接之间呈现弱相关性，因此可以使用多线程技术并行处理多个 TCP 连接的报文，隐藏存储器访问和协议处理延时，RSS 技术将报文分类后决定接收报文的处理核心，减少线程之间的同步开销。

传统 TCP 加速技术中协议处理仍由主机 CPU 执行，随着网卡的计算能力不断增强，TCP 部分或者全部的功能被卸载到网卡中执行。TOE 卸载 TCP 协议处理的全部功能，极大地提高端系统 TCP 性能，但其实现特别复杂，需要修改内核协议栈和相关系统调用，并且存在安全性和兼容性等问题。TSO（TCP Segmentation

Offload）技术卸载 TCP 数据发送路径中的数据分段和校验和计算功能，已经发展的非常成熟，Linux 已经提供了相关的系统接口。LRO 技术通过合并接收到同一个 TCP 连接上的连续数据报文减少协议栈处理报文的数量，降低 CPU 开销，但其工作在网卡驱动程序层面，报文合并工作由主机 CPU 执行，不能很大程度上减轻 CPU 的负担。

本文针对 TOE 和 LRO 技术的缺点，考虑到多核 NPU 具有优越的报文处理能力，通过增加一级存储，达到以空间换取性能的效果，提出使用多核 NPU 作为网卡，卸载 TCP 报文乱序重组和校验和计算功能，将同一个 TCP 连接的多个报文合并为一个大报文后交由协议栈处理，通过减少协议栈处理报文的数量和网卡产生中断的数量减少主机 CPU 开销，提升端系统 TCP 性能。

1.2 TCP 处理开销

TCP 将应用层发送的数据分段，形成多个报文，并为数据的每个字节分配序列号，通过序列号机制实现报文确认、丢失和错误报文重传、乱序报文恢复顺序等功能，为应用层提供面向连接的可靠数据传输功能；通过应答报文中的窗口大小字段，TCP 控制发送方的最大发送数据量，实现简单的流量控制功能；TCP 通过 IP 地址和端口号形成 Socket，用一个 Socket 对表示一个具体连接，使得系统中的多个进程能够同时使用 TCP 通信，实现多路并传功能；通过发送数据超时和重复确认报文，TCP 检测链路的拥塞情况，在拥塞发生时减慢发送数据速率，提供拥塞控制功能。TCP 在端系统的处理开销主要包括三部分[1]：中断操作，数据复制和协议处理，下面分别进行简单介绍。

1.2.1 中断操作

由于报文到达的时间未知，端系统对报文的接收一般设计为异步中断的方式。当一个报文到达网卡时，网卡产生一个中断信号，触发驱动程序对报文进行接收：将数据从网卡存储空间 DMA 到主机内存中，构造相应的数据结构进而交由内核协议栈处理。中断会引起端系统进行上下文切换，当网络带宽较小，报文到达不频繁时，这样的设计对端系统的性能影响并不明显；然而当网络带宽增至千兆和更高，报文频繁到达时，频繁的中断操作带来的上下文切换会严重消耗 CPU 资源，降低端系统的整体性能。后来的网卡驱动程序多采用将中断与轮询结合的设计方式，以谋求一定的平衡：当一个报文到达网卡后，网卡产生中断信号，在驱动程序的中断处理函数中首先禁用网卡中断，然后轮询接收一定数量的报文，之后使能网卡中断，将接收到的报文交给协议栈处理。中断与轮询结合的设计方式可以减少网卡产生中断的数量，轮询接收报文的数量将关键影响系统的性能。中断合

并技术使网卡不再为每一个接收到的报文产生中断，而是在接收到一定数量的报文后才产生一个中断，驱动程序一次负责处理多个报文；中断合并技术可有效减少网卡产生的中断数量，减少主机 CPU 开销。

1.2.2 数据复制

TCP 为应用层提供可靠的数据传输服务，保证报文的可靠有序。数据发送路径上，应用层将数据交给内核之后，就认为数据已经被正确发送，应用程序将释放或者重复使用应用层的发送数据缓冲区，而 TCP 为了保证数据的可靠有序传输，需要在报文发送出错或超时的情况下进行重传，所以发送数据只能保持在内核空间。数据接收方面，TCP 接收到的可能是错误或者乱序的报文，TCP 需要等待发送方重传错误的报文，并将乱序的报文重新排序后，才能通知应用层接收数据，所以 TCP 接收数据也只能保持在内核空间。这样，发送过程的数据复制步骤为：CPU 将数据从用户空间复制到内核空间，驱动程序将数据从主机内存 DMA 到网卡存储空间；接收过程的数据复制步骤为：驱动程序将数据从网卡存储空间 DMA 到主机内存中，CPU 将数据从内核空间复制到用户空间。在高速网络中，频繁的数据发送和接收操作引起的 DMA 频繁启动和数据复制操作将严重影响系统的整体性能。

1.2.3 协议处理

协议处理主要分为四个部分：（1）TCP 连接管理；（2）数据传输；（3）计时器管理；（4）错误处理和拥塞控制。其开销主要包括：（1）分割发送数据为合适大小的数据块；（2）发送数据之后设定重传定时器；（3）为接收到的数据发送确认报文；（4）对接收到的数据进行校验和计算；（5）将接收到的乱序报文重新排序；（6）检测和丢弃重复报文；（7）进行流量控制和拥塞控制。

在 TCP 的具体实现中，其开销主要包括以下三方面：（1）报文头部处理和中断引起的上下文切换等报文相关开销；（2）计算校验和、数据复制等数据相关的开销；（3）内存管理等操作系统相关的开销。

1.3 本文主要工作

TCP 被广泛地应用于搜索引擎、门户网站、内容提供商等网络应用中，TCP 加速技术可以优化 TCP 协议的处理过程，提高 TCP 性能，不仅可以减少服务器集群的数量，还能减少功耗，具有很高的商业价值和环保意义。传统优化 TCP 处理的方法中，协议处理仍由主机 CPU 执行，随着网卡的计算能力不断增强，TOE（TCP

Offload Engine）的思想应运而生。TOE 将 TCP 协议处理的功能完全卸载到网卡中执行，极大的提高端系统 TCP 性能，但其实现特别复杂，并且存在安全性和兼容性问题。TSO 技术将 TCP 数据发送路径中的数据分段和校验和计算功能卸载到网卡中执行，已经发展的非常成熟。LRO 技术通过合并接收到的同一 TCP 连接上连续数据报文减少协议栈处理报文的数量，降低 CPU 开销，但其工作在网卡驱动程序层面，报文合并工作仍由主机 CPU 执行，不能很大程度上减轻 CPU 的负担。

基于此，本文提出使用多核 NPU 作为网卡卸载 TCP 乱序报文重组功能、合并数据报文加速 TCP 的技术，本文主要工作如下：

1） 深入研究 TCP 加速相关技术，分类进行阐述，主要包括优化校验和计算、减少数据复制、优化大概率事件处理、减少中断数量、多线程延时隐藏、TOE、LRO 等，并分析了各类加速技术的缺点。

2） 针对 TOE 和 LRO 技术的缺点，首次提出使用多核 NPU 作为网卡，卸载 TCP 乱序报文重组功能，并将同一个 TCP 连接上的数据报文合并后交由内核协议栈处理，减少协议栈处理报文的数量和网卡产生中断的数量，提升端系统 TCP 性能的技术。

3） 阐明了多核 NPU 良好的报文处理能力，详细说明了系统的框架结构和功能设计，并针对多核 NPU 的特点提出多接收报文描述符环、合并报文校验和计算优化、接收报文处理线程 DMA 负载均衡、主动 ACK 机制等系统优化技术。

4） 基于 XLS416 开发平台实现系统，并在 10Gbps 网络环境中测试其性能，取得 4.9Gbps 的 TCP 接收数据吞吐量。

1.4 论文结构

本文主要内容分为五章，各章安排如下：

第一章为绪论。首先说明 TCP 加速技术的研究背景，其次简单介绍 TCP 协议处理开销，最后介绍本文主要工作和以及本文结构。

第二章介绍各类 TCP 加速技术，分析各类技术的工作原理，并着重说明 TOE 和 LRO 技术的缺点。

第三章为系统结构和功能。首先简要说明 TCP/IP 接收数据的处理过程，其次说明多核 NPU 良好的报文处理能力，并针对 TOE 和 LRO 技术的缺点提出使用多核 NPU 加速 TCP 的技术，最后详细介绍系统的框架结构和功能设计，主要包括：TCP 连接管理、报文过滤、TCP 乱序报文重组、数据报文合并以及报文接收超时检测五个方面。

第四章详细介绍实现系统时使用的各项改进技术，主要包括：多接收报文描

述符环、接收报文处理线程间 DMA 负载均衡、合并报文后 IP 和 TCP 校验和计算的优化、以及主动 ACK 机制四个方面。

第五章为基于 XLS416 的系统实现。首先简单介绍 XLS416 开发平台，其次详细说明系统实现时 XLS416 的资源分配，最后对系统性能进行测试和评价。

第二章 TCP 加速技术

TCP 被广泛地应用于内容提供商、游戏提供商、搜索引擎和门户网站等各类互联网应用中，随着互联网技术的高速发展，互联网应用的性能需求逐渐表现为低主机开销、低存储开销、低延时、高带宽和高吞吐量等特点[1]。对 TCP 进行加速，优化端系统 TCP 性能不仅可以减少服务器集群的数量，节约成本，还能降低功耗，实现更加绿色环保的互联网服务，具有重要的商业价值和环保意义。TCP 加速技术一直是科研人员的研究热点，本章将对各类 TCP 加速技术进行简要介绍。

2.1 优化校验和计算

校验和用于验证数据的正确性，其计算方法为： 数据中的相邻字节组成 16 位整数，然后计算这些 16 位整数的反码和；生成校验和时，先将校验和字段清零，然后计算相关数据的 16 位整数反码和，将 16 位整数反码和的反码填充到校验和字段；检测校验和时，计算包括校验和的全部数据的 16 位整数反码和，如果结果的每一位都是 1，则检验成功，否则检验失败。

用 a，b 为表示两个字节，[a,b]表示 16 位整数 a*256+b，则计算字节序列 A，B，C，…，Y，Z 的校验和公式为：

$$[A,B]+'[C,D]+'...+'[Y,Z] \qquad (2\text{-}1)$$

其中+'表示反码加法。

校验和计算有下面三个数学特性[2]，可用于优化计算：

1. 交换律和结合律：校验和可分组计算，公式(2-2)的结果和(2-1)相同。

$$([A,B]+'[C,D]+'...+'[J,0])+'([0,K]+'...+'[Y,Z]) \qquad (2\text{-}2)$$

2. 字节序无关：公式(2-3)的计算结果与公式(2-1)相同。

$$[B,A]+'[D,C]+'...+'[Z,Y] \qquad (2\text{-}3)$$

这使得校验和的计算方法在大端和小端机器上相同。

3. 并行加：在 32 位机器中，可以每次相加四个字节，在计算结束后将高 16 位与低 16 位相加最终形成 16 位结果。

校验和计算本身并不复杂，但要遍历所有数据，存储器的访问操作使其代价变高。Borman[2]对校验和计算进行了一下四个方面的优化：

1. 延时进位：将 16 位整数的加法运算在 32 位整数中进行，这样进位被累积到高 16 位中，在主加法循环结束后处理进位。

2. 循环展开：通过增加额外指令将循环展开，可以减少循环控制开销。

3. 结合数据复制：数据复制也涉及存储器的访问，如能将校验和计算与数据复制相结合，则每字节数据只需访问一次，大幅降低存储器访问开销。

4. 头部更新优化：当头部中的某个字段被更新时，新的校验和计算可不扫描全部数据。如路由器更改 IP 头部中的 TTL 字段，假设更改前 TTL 字段值为m，更改后为m'，校验和更改前值为C，则更改后值C'的计算公式为：

$$C' = C + (-m) + m' = C + (m' - m) \qquad (2\text{-}4)$$

Mallory 和 Rijsinghani 提出了进一步优化，允许在部分数据改变后，不需要遍历所有的字节而重新计算校验和[3-4]。

上述改进虽然可以减少校验和计算的开销，但计算仍由主机 CPU 完成。Kleinpaste，Steenkiste，Zill[5]等人将报文头部和用户数据分开传输，主机 CPU 计算报文头部的校验和，之后将结果发送给硬件，由硬件计算用户数据的校验和。Henriksson，Persson，Liu[6]等人设计了可与通用处理器集成的硬件芯片，用于并行计算校验和。

2.2 减少数据复制

目前以太网的发展速度远高于存储器的发展速度，存储器访问已经成为 TCP 的性能瓶颈之一，减少数据复制可以很大程度上减少主机开销。

Jerry Chu[7]提出零拷贝技术，实现在 Solaris 系统中，通过预先将网卡的存储空间映射到主机的内核空间或用户空间，使得应用程序可以直接访问网卡的存储空间。报文到达网卡后，数据被缓存在网卡存储空间内，直到应用程序访问数据时，数据才经过总线到达 CPU。零拷贝技术需要修改相应的系统调用，同时要求网卡有较大的存储空间存放数据。

Dalton，Watson，Banks[8]等人提出内核与网卡共享存储空间的技术，由内核管理网卡的存储空间，使用 DMA 或者 Programmed IO 技术在应用程序缓冲区和网卡存储空间之间进行数据移动。这种技术不需要对应用程序进行修改，具有较高的兼容性。

Druschel，Peterson[9]等人通过应用程序与内核共享存储空间的技术实现快速缓冲，使用 DMA 在网卡存储空间、应用程序与内核共享存储空间之间进行数据移动。快速缓冲技术在用户空间和内核空间分别定义了新的 API，并要求应用程序使用新 API，兼容性较差，共享存储空间的管理需要应用程序、内核和网卡之间复杂的配合完成。

Kleinpaste，Steenkiste，Zill[5]等人通过修改协议处理，增加了报文头部和用户数据分开传输的复制语义，在协议处理过程中只传输报文头部和用户数据的索引，用户数据一直存储在用户空间或网卡存储空间，复制操作只在用户数据最终进行传输的时候进行。

Buzzard，Jacobson，Mackey[10]等人提出的 Hamlyn 框架由发送方进行存储管理，

数据发送之前，由发送方确定数据在接收方的存储位置，避免接收方缓冲区不足引起的报文丢失。

TCP 的数据发送路径功能简单，改进较为容易，网卡可以在数据传输时直接访问用户存储空间。但数据接收路径功能复杂，由于报文到达的时间和顺序未知，内核必须缓冲接收到的报文，对报文进行校验和检验、乱序报文重组后才将数据复制到用户缓冲区，通知应用程序读取数据。Rodrigues，Anderson，Culler[11]等人提出提前将接收数据放置到应用程序缓冲区的方法减少数据复制。

Culley，Garcia，Hilland[12]等人提出 RDMA（Remote Direct Memory Access）协议，在报文中包含数据在接收方的存储位置信息，接收方网卡收到报文后，直接将数据放置到应用程序缓冲区中，避免多次数据复制。RDMA 不需要操作系统管理复杂的地址映射，不会给主机增加额外的存储管理开销。RDMA 的缺点有[13]：需要考虑具体的操作系统才能获得较高的性能和灵活性；引入了很多缓冲区管理的问题，这些问题还缺乏较好的解决方案；RDMA 与应用程序、操作系统之间的接口设计非常困难；引入安全性问题。Romanow，Bailey[14]实现了位于 TCP 之上的 RDDP（Remote Direct Data Placement）框架。Erdogan，Patel[15]实现了另一种基于 IP 的 RDMA 框架。

2.3 减少中断数量

由于报文到达的时间未知，端系统对报文的接收一般设计为异步中断的方式。报文到达网卡时，网卡产生中断信号触发驱动程序对报文进行接收：将数据从网卡存储空间 DMA 到主机内存中，构造相应的数据结构进而交由协议栈处理。中断会引起操作系统的上下文切换，当网络带宽较小，报文到达不频繁时，这样的设计对端系统的性能影响并不明显，然而随着网络带宽的不断增大，报文频繁到达时，中断操作带来的上下文切换会严重消耗 CPU 资源，降低端系统的整体性能。减少中断数量对于降低主机开销，提高 TCP 性能有着重要意义。

如果以小于 MTU（Maximum Transfer Unit）的长度对数据进行分割，则会产生更多的报文，使接收方产生的中断数量增加[16]。通过增加报文中数据长度，每次以 MTU 传输数据，可以有效的减少中断数量。另外，如果接收方网卡可以过滤局域网内的某些广播报文和 UDP 报文，也可有效地减少中断数量。

Rangarajan，Bohra，Banerjee[17]等人在研究 TCP 卸载时，使用一台机器专门处理 TCP，并将异步触发的中断轮询代替，此技术的关键在于控制轮询频率，轮询频率过高将导致总线拥塞，频率过低将导致到达的报文不能及时被处理。

中断合并[1]技术可有效减少中断数量，网卡不再为每个接收到的报文产生中断，而是先将报文缓存，在缓存报文的数量到达一定阈值时才产生中断，主机在一次

中断中处理所有缓存的报文。中断合并可以有效减少中断开销，但会带来报文延时变高的问题。

增加报文中数据长度可以有效减少中断数量，它对原本的中断处理设计没有影响，但需要上层协议提供对大报文的支持。随着硬件技术的发展，网卡的计算能力不断增强，利用网卡对报文进行处理，从而减少中断数量的方法越来越得到重视。

2.4 优化大概率事件处理

同计算机体系结构设计思想一样，优化大概率事件的处理也是提高 TCP 性能的一个基本原则。PCB（Protocol Control Block）中存储 TCP 连接的状态信息，在协议处理中经常被访问，采用高速缓存实现 PCB 可以优化 TCP 协议处理[18]。

Mosberger，Peterson，Bridges[19]等人指出，通过优化编译器增加 cache 命中率，可以在不修改指令的条件下降低单条指令的执行开销，从而减少报文处理开销。程序中的每条指令执行频率不同，可通过编译器将代码中执行频率较低的指令集中在函数或者程序的尾部，将执行频率高的指令克隆到一起，减少跳转，提高指令 cache 的命中率；通过将经常调用的函数内联到代码中，可以大量减少函数调用开销，并提供更多的上下文给编译器，使编译器能更好的优化代码。

Jacobson 基于两种常见情况提出了首部预测技术[18]：（1）TCP 发送数据后，该连接上下一个接收到的报文是已发送数据的确认报文；（2）TCP 接收数据后，该连接上下一个接收到的报文是序列号连续的数据报文。实验表明，在 LAN（Local Area Network）中首部预测的命中率可达 97%，跨越 WAN（Wide Area Network）时，首部预测命中率在 83%到 90%之间。

2.5 延时隐藏

通过硬件流水技术和多线程技术，可以隐藏 TCP 协议处理的延时。

传统系统中，DMA 等待报文完全到达主机内存或者网卡存储空间后，才进行初始化，对报文进行传输。Yocum，Anderson，Chase[20]等人提出截断式 DMA 策略，流水化传输报文数据：DMA 不再等待报文的全部数据，而是在缓冲区内的数据到达一定阈值时就进行传输。

Matthias Kaiserswerth[21]提出使用并行协议处理引擎 PPE（Parallel Protocol Engine）并行处理协议，实验中测试处理 LLC（Logic Link Control）协议，得到了较明显的性能提升。但多个处理引擎之间的负载均衡开销使得协议处理性能不与流量增长呈线性关系。

Nordqvist，Liu[22]等人提出使用包含 CRC（Cyclic Redundancy Check）检验部件、XAC（Extract and Compare）部件以及加法器等数据处理加速器的 FP（Functional Pages）机制，在 SOC（System On Chip）上对数据进行流水处理。

由于多个 TCP 连接通常表现出弱相关性，所以可以采用多线程技术对多个 TCP 连接同时进行处理[23]，隐藏 TCP 协议处理过程中的存储器访问延时。

2.6 多队列和 RSS

随着主机 CPU 处理核心数量增多，现代网卡支持多报文发送和接收队列[24]，可编程控制的报文分类，以及 RSS（Receive Side Scaling）功能。

现代网卡可根据用户的配置，收到报文后提取五元组信息（源 IP 地址、目的 IP 地址、协议类型、源端口号、目的端口号），对报文进行分类，决定报文的接收队列。网卡硬件执行报文分类功能可以减少主机 CPU 开销，对于软件路由器、防火墙等应用非常有效。

RSS 技术提取报文信息后，计算哈希值，将报文发送至其连接所属 CPU 对应的接收队列中，如图 2.1 所示。根据不同的报文协议类型，RSS 提取的报文信息不同，如表 2.1 所示。

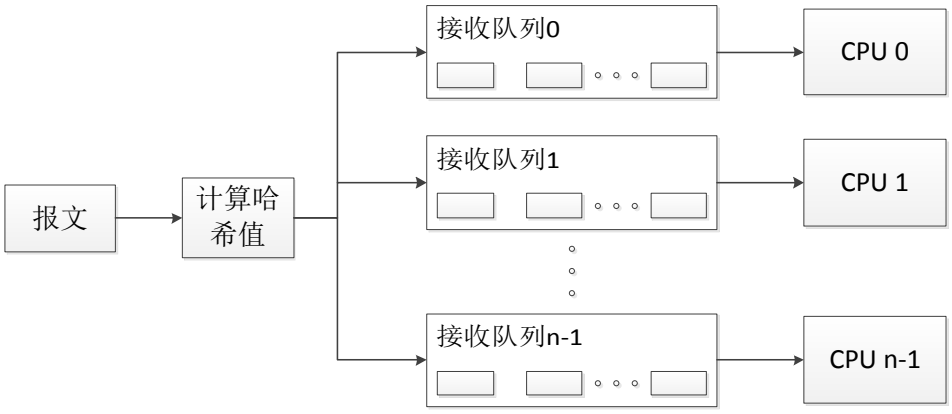

图 2.1 RSS 分发报文

表 2.1 RSS 提取报文信息表

协议类型	提取字段
TCP	源 IP 地址+目的 IP 地址+源端口号+目的端口号
UDP	源 IP 地址+目的 IP 地址+源端口号+目的端口号
其它	源 IP 地址+目的 IP 地址

2.7 用户级传输协议

TCP 工作在内核空间，数据复制和中断引起的上下文切换开销巨大，很多研究者开始设计用户级传输协议，将协议处理移到用户空间。

Mapp 和 Pope[25]实现的 A1 协议工作在用户空间，它基于数据块而非字节流进行数据传输，有选择性的进行数据重传，显示地测量 RTT，支持组播。实验结果表明，A1 相比于传统 TCP 可以获得更高的数据吞吐量。

Thekkath，Nguyen，Moyt[26]等人通过编写用户空间和内核空间的一些模块实现用户级数据传输协议。增加的模块主要有：（1）实现数据重传、校验和计算、流量控制等主要协议功能的协议库；（2）代表应用程序处理通信的注册服务器，其本质为一个特权级别进程；（3）提供数据传输功能的网络接口。他们同时指出用户级数据传输协议更容易实现和调试，并且可以利用传统 TCP 实现中减少开销的各种技术。

Edwards 和 Muir[27]将协议数据分离、缓冲区管理等复杂操作保留在内核空间，将部分协议功能移至用户空间实现了用户级的 TCP 协议。

Shivam，Wyckoff，Panda[28]等人实现了 EMP 系统，将文件描述符管理、虚拟存储空间管理和网卡初始化等功能移至用户空间，并在网卡上实现协议处理的部分功能。实验结果表明，EMP 在带宽和延时方面的性能明显优于传统 TCP。

2.8 TOE

传统优化 TCP 处理机制的方法中，协议处理仍由主机 CPU 执行。随着网卡硬件的计算能力不断增强，将 TCP 协议处理的功能完全卸载到网卡中执行，极大提高端系统 TCP 性能的思想应运而生，既 TOE[29]。在 TOE 的具体实现中，可以采用通用处理器或可编程器件 FPGA（Field Programmable Gate Array）等，也可完全采用 ASIC（Application Specific Integrated Circuit）进行设计，前者的好处在于灵活性高，而后者可以获得更高的性能[30]。

Henriksson，Nordqrist，Liu[31]等人设计了一种嵌入式协议处理器，在上面运行实时操作系统，着重于报文的快速高效接收。实验结果表明，采用 CMOS 工艺可以实现 10Gbps 数据流的线性处理。

Ang[32]使用 i960RN 当做网卡的控制处理器，试图实现 TOE。系统主机端运行实时操作系统 RTX，并优化了数据的发送和接收路径，避免数据多次复制。实验结果表明主机 CPU 使用率有所降低，但开销仍比预期高很多。Ang 指出，缓冲区管理和硬件设计是系统性能的主要瓶颈。

目前没有操作系统提供 TOE 的接口或 API（Application Programming Interface），对 TOE 的实现主要集中在 Linux 操作系统中，且需要修改内核协议栈，实现的方法大致有三种：（1）高性能 Socket[33-34]（High Performance Sockets）；（2）屏蔽

主机 TCP 协议栈[35]；（3）替换主机 TCP 协议栈[36]。

SDP（Socket Direct Protocol）是高性能 Socket 的一个实现标准，该协议增加了 AF_INET_OFFLOAD 协议族，需要修改操作系统内核，重新实现 bind，listen，connect，accept 等系统调用。

屏蔽主机 TCP 协议栈的技术卸载 TCP 连接管理功能，对一条 TCP 连接，可选择卸载或不卸载，通过对内核协议栈打补丁的方式分别处理卸载 TCP 连接和非卸载 TCP 连接。该技术还需要增加模块和重新实现传输层接口，用于管理已卸载连接的状态和发送接收数据。

替换主机 TCP 协议栈的技术在原来的 BSD socket 层和 INET socket 层之间增加 TOE socket 层，并在 INET socket 层下面由高到低分别实现 TOE INET 协议栈、TOE 网络核心层、TOE 分发器、TOE 驱动和 TOE 设备。TOE 设备和非 TOE 设备可同时存在。

TOE 产品主要有 Chelsio 10G Ethernet TOE[37]、Adaptec TOE[38]和 Alacritech SEN 2100[39]系列。Chelsio 10G Ethernet TOE 完全卸载了 TCP 协议栈，包括连接的建立和拆除功能，而在 Kim 和 Rixner 的设计中，将 TCP 连接的建立和拆除操作保留在主机端，而将数据传输卸载到硬件中[40]。

TOE 虽然可以大幅降低主机开销，却又许多缺点[13]：（1）在网卡上实现传输层功能要比实现链路层功能复杂得多；（2）TOE 系统中主机和硬件之间的接口设计特别复杂；（3）TOE 需要上层协议的支持才能将数据直接存放到存储器中；（4）TOE 需要和主机同时维护每条 TCP 连接的状态；（5）更新硬件的难度要比更新软件大；（6）TOE 设计难度大，出现错误难以定位。这些缺点，以及安全性和兼容性问题一起弱化了 TOE 的好处。

2.9 TSO 和 LRO

TSO（TCP Segmentation Offload）技术利用网卡的计算能力，将 TCP 数据发送路径上的数据分段和校验和计算功能卸载到网卡中执行[41]。TCP 数据发送路径功能较为简单，TSO 技术已经发展的非常成熟，Linux NAPI（New API）[42]网卡驱动模型已经提供相关的编程接口，只需设置相应的标志位，再编写少量代码后即可实现 TSO 功能。

传统意义上的 LRO 工作在网卡驱动程序层面，将接收到属于同一个 TCP 连接的多个连续数据报文合并为一个大报文（如图 2.2 所示），然后交给协议栈处理（如图 2.3 所示），通过减少内核协议栈处理报文的数量减轻主机 CPU 负担，进而增加 TCP 接收数据吞吐量。

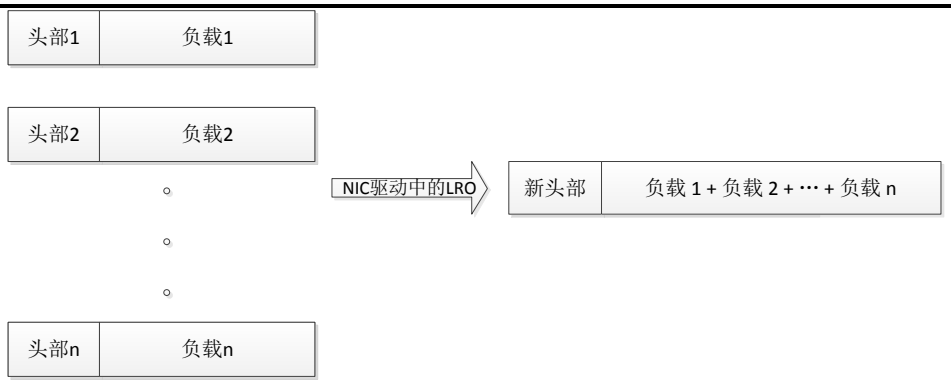

图 2.2 LRO 合并报文

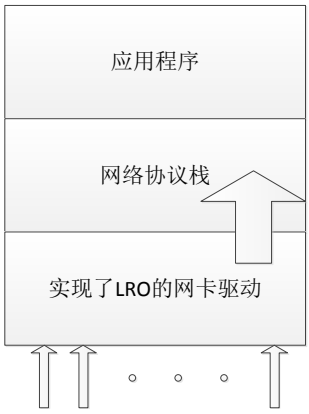

图 2.3 LRO 层级架构

LRO 最先由 Grossman[43]提出并为 Neterion Xframe-II 万兆网卡实现，该实现利用网卡特定功能，配合驱动程序完成。Neterion Xframe-II 万兆网卡提供以下特性：（1）硬件支持的多接收队列；（2）链路层、IP、TCP/UDP 校验和计算；（3）报文头部和数据分离，将报文的链路层、IP、TCP/UDP 头部和数据分离，并存储在主机提供的缓冲区内；（4）RTH（Receive Traffic Hashing）和 SPDM（Socket Pair Direct Match）流标识符计算。真正的报文合并工作由驱动程序完成，驱动程序将同一条 TCP 连接上连续的多个报文数据拼接到一起，并更新报文 IP 和 TCP 头部，当接收到不适合 LRO 的报文或者累积的报文数据长度到达一定上线时，驱动程序将已经缓存的 TCP 报文构造为一个新报文，交由内核协议栈处理。

判断报文是否适合进行 LRO 的条件如下：（1）是未分段的 IP 报文；（2）是 TCP 报文；（3）校验和正确；（4）TCP 序列号连续；（5）IP 头部无选项；（6）TCP 负载数据长度大于 0；（7）SYN、FIN、URG、PUSH 等标志均没设置；（8）TCP 头部无选项或者只有时间戳选项。只有所有条件都满足时，才判断一个报文适合进行 LRO。

新构造的报文 TCP 头部更新算法为：（1）使用第一个报文的 TCP 序列号和时间戳作为新报文的 TCP 序列号和时间戳；（2）使用最后一个报文的 ACK 和窗口

值作为新报文的 ACK 和窗口值；（3）更新 TCP 校验和。更新 IP 头部的算法为：（1）更新 IP 头部中的总长度值；（2）更新 IP 校验和。

Themann[44]将 LRO 以补丁的形式添加到 Linux 内核中，作为一种更通用的实现，使其他网卡驱动开发人员只需修改少数代码即可实现 LRO 功能。Themann 的补丁包含报文合并的核心功能，并提供了两种报文合并模式：skb 模式和 page 模式，前者针对将报文存储在套接字缓冲区（socket buffer）内的网卡驱动，后者针对将报文存储在页（page）内的网卡驱动。Gallatin 的研究指出，将 LRO 仅在软件中实现就可以提高 TCP 数据接收的性能。

Hatori[45]等人在 Xen 虚拟系统中将 LRO 分别实现在物理网络接口和虚拟网络接口上。在物理网络接口上的 LRO 实现减少了 CPU 指令数量，但没有提升 TCP 吞吐量；而在虚拟网络接口上的 LRO 实现取得了 22%的数据接收性能提升。

2.10 本章小结

TCP 被广泛地使用于门户网站、搜索引擎和内容提供商等各类商业应用中，提升 TCP 性能不仅可以减少服务器集群的数量，还可降低功耗，具有很高的商业价值和环保意义，TCP 加速技术一直是人们的研究热点。目前以太网的发展速度远高于存储器和 CPU 的发展速度，存储器访问和 CPU 处理协议已经成为 TCP 的性能瓶颈。本章简要介绍了针对这个问题的各类 TCP 加速技术，主要包括：优化校验和计算、减少数据复制、减少中断数量、优化大概率事件处理、协议处理延时隐藏、多队列和 RSS、用户级传输协议设计、TCP 卸载引擎（TOE）、TSO 和 LRO。传统的 TCP 加速技术中协议处理仍然由主机 CPU 执行，随着网卡硬件的计算能力不断增强，TOE 的思想应运而生，TOE 可大幅减少主机开销，提升 TCP 性能，但其设计和实现特别复杂，并会引入安全性和兼容性问题；LRO 的报文合并工作仍由主机 CPU 执行，不能很大程度上减少 CPU 负担。

第三章 系统结构和功能

前两章主要介绍了 TCP 协议的处理开销和各类 TCP 加速技术，并简要说明了各类加速技术的优缺点；本章首先说明传统 TCP/IP 接收数据的处理过程，介绍网卡、驱动程序和内核协议栈在数据接收过程中的具体操作；其次针对 TOE 和 LRO 技术的缺点，考虑到多核 NPU 良好的报文处理能力，提出使用多核 NPU 作为网卡，卸载 TCP 数据接收路径中乱序重组功能，将同一条 TCP 连接的多个报文合并为一个大报文后交由协议栈处理，通过减少内核协议栈处理报文的数量和网卡产生中断的数量提升端系统的 TCP 性能的技术，同时避免 TOE 技术引起的安全性和兼容性问题；最后详细说明系统的框架结构和功能设计。

3.1 TCP/IP 数据接收处理过程

本节将概括说明 TCP/IP 的数据接收处理过程，介绍从报文到达网卡开始，直至数据到达应用程序的过程中网卡、驱动程序和内核协议栈的具体操作。

TCP/IP 数据接收处理从网卡接收到以太网报文开始，总体流程如图 3.1 所示。网卡首先检验报文的 CRC 值，丢弃 CRC 错误报文；其次获取报文描述符，根据报文描述符的内容将报文头部和数据 DMA 到主机内存中；报文描述符是网卡驱动程序和网卡交换信息使用的数据结构，一般被设计为 16 字节，包含用于存放报文的缓冲区内存地址等信息。网卡驱动程序申请内存构造报文描述符，并将其以环形存放，便于网卡循环使用；驱动程序同时申请内存缓冲区，用于存放接收到的报文。网卡将报文 DMA 到主机内存后，将设置报文描述符中的相应字段，表明此报文描述符中包含有效报文，并产生中断，触发驱动程序的中断处理函数对收到的报文进行处理。

网卡驱动程序读取报文描述符，判断报文是否为 TCP 报文，若为 TCP 报文，则交给内核协议栈进行处理。协议栈首先需要判断报文属于哪条 TCP 连接：内核协议栈使用数据结构 TCB（TCP/IP Control Block）保存连接的状态信息，由于多条连接可以并存，内核协议栈需要维护多个 TCB，并使用哈希表的方法快速查找报文所属连接对应的 TCB，哈希值利用源 IP 地址、目的 IP 地址、源端口号、目的端口号计算。下一步，协议栈执行 TCP 可靠有序数据传输功能：检验 TCP 校验和，丢弃校验和错误报文和重复报文，对已经接收到的乱序报文进行重组，发送确认报文和重传数据等，内核协议栈需要更新 TCB 中的一些字段，如接收到的数据长度。最后，内核协议栈将拷贝报文负载数据到应用程序缓冲区内，并通知应用程序读取数据：协议栈首先检查应用程序是否提供了可用的数据缓冲区，若有，协议栈将拷贝数据到应用程序数据缓冲区内；若没有，则等待应用程序提供数据

缓冲区。

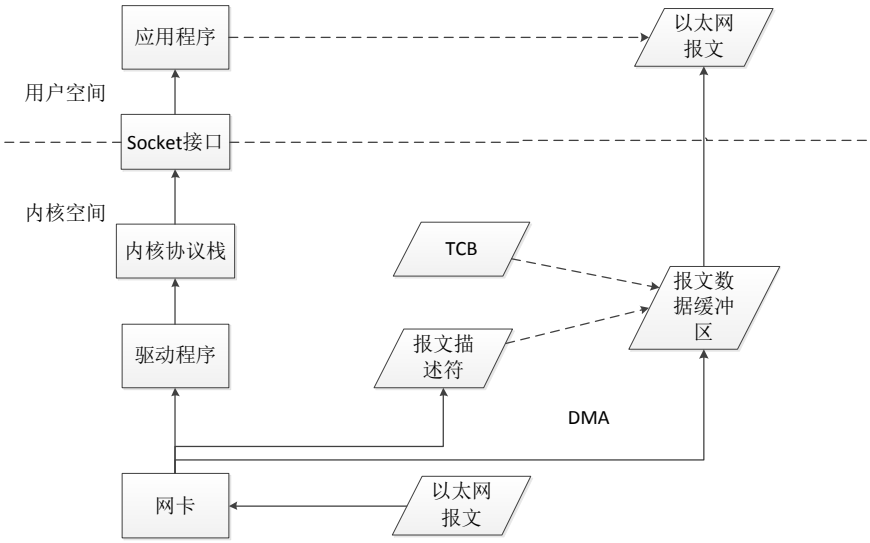

图 3.1 TCP/IP 接收数据流程图

3.2 使用多核 NPU 的必要性

上一节介绍了 TCP/IP 接收数据过程中网卡、驱动程序和内核协议栈的操作，本节先说明多核 NPU 良好的报文处理性能，然后结合 TOE 和 LRO 技术的缺点提出使用多核 NPU 作为网卡，卸载 TCP 乱序报文重组功能，合并多个报文为一个大报文对 TCP 进行加速的技术。

多核 NPU 具有以下四点特性，因而具有良好的报文处理性能：

（1）多核 NPU 具有多个处理核心，每个处理核心具有多个线程，这样总的线程数量可达两位数（通常为 16 个或 32 个），线程基于硬件而非软件，因此转换开销非常低。多条 TCP 连接之简呈现弱相关性，因此可以并行处理。多核 NPU 大量的线程可以高效地利用这点，对多条 TCP 连接同时进行处理，当一个硬件线程在等待存储器访问结果返回时，另一个线程可以被调度，进行另一个存储器访问操作而不引起很大的额外开销，这种流水线存储器访问机制可以隐藏 DRAM（Dynamic Random Access Memory）延时，增加实际数据带宽。

（2）多核 NPU 的功耗较低，可以被集成在卡上，引入额外一层存储空间。报文到达时，可以先被缓存在卡上的存储空间，多核 NPU 上运行的程序负责管理卡上存储空间和处理报文。通过增加一级存储空间，可以达到以空间换取性能的效果。

（3）多核 NPU 通常具有优越的 I/O（Input/Output）特性，报文可以较高的吞吐量从网络接口到达卡上存储空间。另外，多核 NPU 的报文分发机制可以根据程序订制，按照报文头部中的某些字段将不同的报文分发到不同的线程或处理核心

上，报文分发部件可与处理核心流水配合，具有很高的灵活性。本文提取 IP 头部中的源 IP 地址和目的 IP 地址，以及 TCP 头部中的源端口号和目的端口号，将同一个 TCP 连接上的报文都分发到同一个硬件线程中进行处理。

（4）多核 NPU 通常利用交叉网络或者高速共享 SRAM（Static Random Access Memory）作为传输媒介，实现硬件线程之间的消息传递机制，使得线程同步等操作变得简单高效。

TOE 技术将整个 TCP 协议栈的功能卸载到网卡中执行，极大地提高端系统 TCP 性能，但在网卡上实现传输层功能要比实现链路层功能复杂得多，TOE 系统中主机和硬件之间的接口设计特别复杂，更新硬件的难度远大于软件更新。TOE 的实现通常需要修改内核协议栈代码和相关系统调用，设计难度大，出现错误难以定位和修改，并且引入安全性和兼容性问题，原来的应用程序不能直接运行于使用 TOE 的系统中，而需要经过修改，使用新定义的 API。

LRO 技术工作在网卡驱动程序层面，其核心思想是将接收到的同一个 TCP 连接上的多个报文合并，构造为一个大报文，然后交给内核协议栈处理，从而减少协议栈处理报文的数量，减少主机 CPU 处理报文头部的开销，提升端系统 TCP 性能。但 LRO 技术中实际的报文合并重组工作仍由主机 CPU 而非网卡硬件完成，不能很大程度上减少主机 CPU 的负担，因此对 TCP 的性能提升不明显。

针对 TOE 和 LRO 技术的缺点，以及考虑到多核 NPU 优越的报文处理能力，本文提出使用多核 NPU 作为网卡，在不修改内核协议栈和系统调用的情况下，配合驱动程序卸载 TCP 协议的乱序报文重组、校验和计算等功能，并将同一个 TCP 连接上的多个报文合并为一个大报文后交由协议栈处理，减少网卡产生的中断数量和协议处理的报文数量，减轻主机 CPU 负担，提升端系统 TCP 性能。另外，利用多核 NPU 增加的一级存储空间，也可达到以空间换取时间的效果，获得进一步性能提升。本系统只涉及网络层级结构中的网络接口和网卡驱动层面，从内核协议栈和应用程序的角度看与普通网卡无异，不会引入新的安全性和兼容性问题。

3.3 系统框架结构

上一节说明了多核 NPU 具有良好的报文处理能力，针对 TOE 和 LRO 技术的缺点提出使用多核 NPU 对 TCP 进行加速的技术，本节将详细介绍系统的框架结构设计。

本文使用多核 NPU 作为网卡，配合驱动程序实现 TCP 数据接收路径的加速。与传统 LRO 技术不同，实际的乱序报文重组、校验和计算以及报文合并工作由多核 NPU 完成，驱动程序负责将重组后的报文提交给内核协议栈进行处理。系统在整个网络层级中的关系如图 3.2 所示，系统物理连接拓扑图如图 3.3。

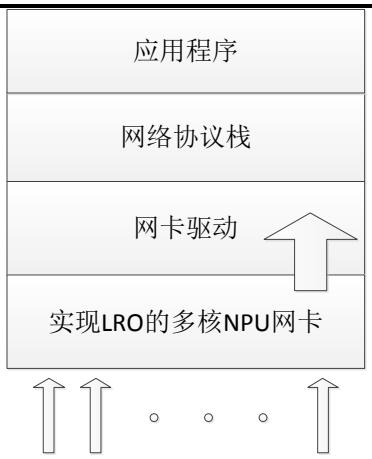

图 3.2 系统网络层级关系

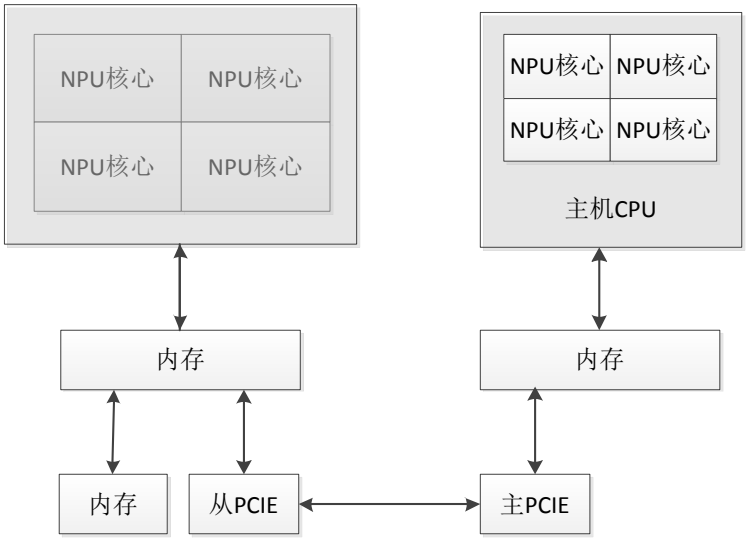

图 3.3 系统物理连接拓扑图

3.3.1 多核 NPU 功能框架

多核 NPU 的工作分为两个主要部分：报文发送和接收。

报文发送部分功能较为简单，负责为驱动程序发出的报文计算 IP 校验和以及 TCP 校验和，之后将报文通过网络接口发送。其主体流程为为一个无限循环，如图 3.4 所示。

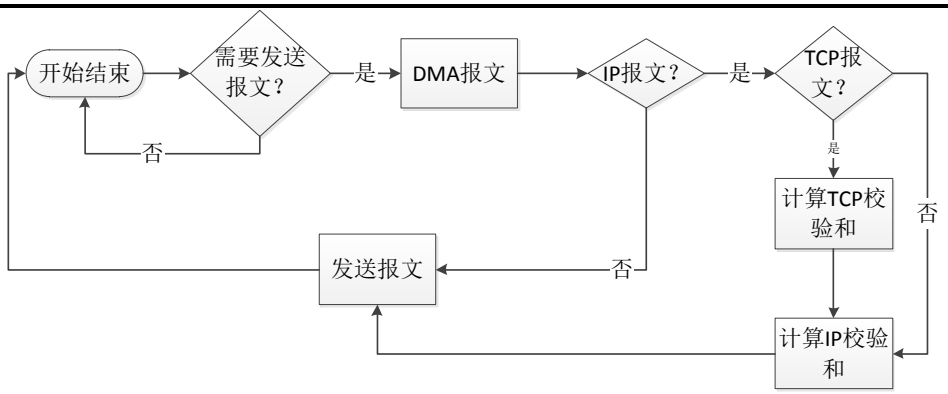

图 3.4 多核 NPU 发送报文流程图

驱动程序预先在多核 NPU 的 PCIE 共享内存中分配发送报文描述符，发送报文描述符中包含表示是否有报文需要发送的字段，报文数据在主机内存中的地址和长度信息，以及表示报文是否发送成功的字段。循环中，多核 NPU 的报文发送处理线程首先读取发送报文描述符，判断是否有报文需要发送，如果有，则根据报文描述符提供的地址和长度信息将报文数据从主机内存 DMA 到多核 NPU 的存储空间中；下一步判断报文是否为 IP 或 TCP 报文，计算 IP 校验和以及 TCP 检验和；最后将报文通过网络接口发送出去，并根据发送结果更改发送报文描述符中报文是否成功发送的字段。

多核 NPU 的报文接收功能要复杂的多，负责报文的接收、乱序重组、合并后交给驱动程序等工作，其主体流程为一个无限循环，如图 3.5 所示。

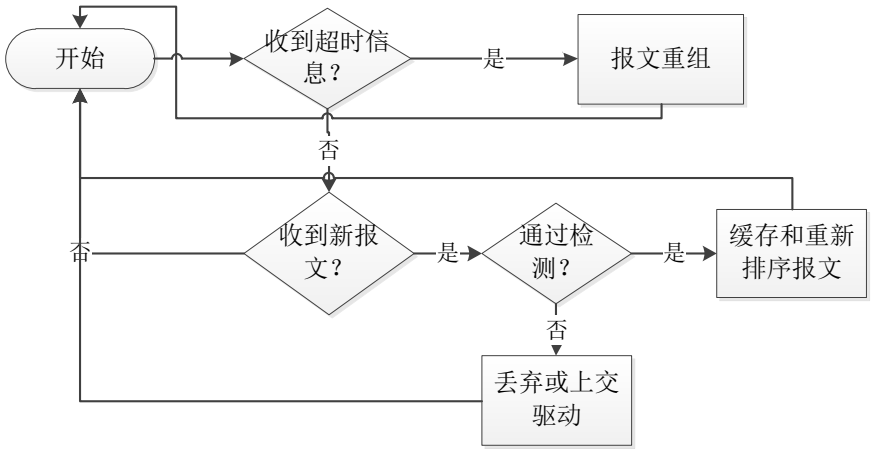

图 3.5 多核 NPU 接收报文流程图

驱动程序预先在主机上申请连续内存页，用于存放报文数据，并在多核 NPU 的 PCIE 共享内存中分配接收报文描述符，其中包含表示是否有新报文到达驱动程序的字段，以及连续内存页的起始物理地址。循环中首先判断是否收到某条连接接收报文超时的信息，若收到，则将对应连接上已缓存的报文重组，合并为大报文后提交给驱动程序；其次判断是否有新报文到达网络接口，判断新到达的报文

是否适合进行报文合并，没通过判断的报文将被丢弃（校验和错误）或直接提交给驱动程序，通过判断的报文将根据对应的连接进行缓存，并按照序列号重新排序，用于后续的报文重组。

3.3.2 网卡驱动程序功能框架

由于 TCP 乱序报文重组、报文合并工作由多核 NPU 完成，网卡驱动程序的设计比较简单，与传统设计差异不大。

报文发送方面，驱动程序的报文发送函数被内核协议栈调用时，首先获取一个预先分配的空闲发送报文描述符，根据报文的长度和内存地址等信息填充发送报文描述符的相应字段，然后进入循环，不断检查报文的发送状态是否变化，并将发送结果返回给协议栈。其流程如图 3.6 所示。在一些落后的系统中，只有 0 到 16MB 之间的物理地址能直接被用于 DMA，内核协议栈发出的报文数据可能位于高于 16MB 的内存中，这时需要将报文数据复制到低 16MB 内存中，然后进行 DMA 操作，带来额外的开销。

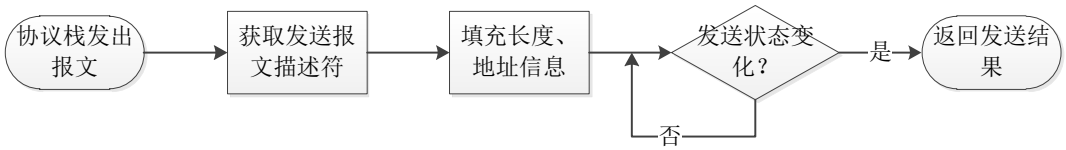

图 3.6 驱动程序发送报文流程图

数据接收方面，网卡驱动程序设计为使用 Linux 操作系统套接字缓冲区的页模式存储报文，为方便 DMA 操作，将报文存储在预先申请的连续内存页中。多核 NPU 将缓存的 TCP 报文重组后，会根据接收报文描述符中的地址信息将报文数据 DMA 到驱动程序预先申请的连续内存页中，并产生中断；驱动程序的中断函数负责为报文构造合适的数据结构，上传内核协议栈进行处理，并另外申请内存，用于以后的报文接收。其流程如图 3.7 所示。

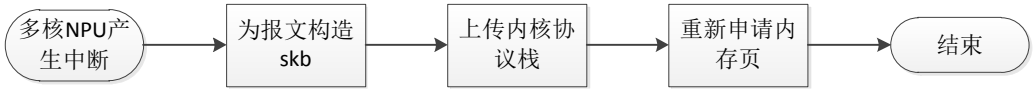

图 3.7 驱动程序接收报文流程图

3.4 系统功能设计

上一节详细介绍了系统的框架结构，并分别从报文发送和接收的角度对多核 NPU 和网卡驱动的功能框架进行了说明。本系统的主体功能为多核 NPU 在数据接收路径上对报文的处理：将接收到属于同一个 TCP 连接上的多个报文进行乱序重组、合并多个报文为一个大报文后经驱动程序交由协议栈处理。本节将从 TCP 连接管理、乱序报文重组、报文过滤、报文合并以及报文接收超时检测五个角度详

细说明系统的功能设计。

3.4.1 TCP 连接管理

本系统并不完全卸载 TCP 协议的全部功能，多核 NPU 不需要像 TOE 或内核协议栈一样维护每条 TCP 连接的全部状态信息，而只维护足够用于进行乱序报文重组和报文合并的状态信息。本节首先介绍多核 NPU 用于维护 TCP 连接的两种数据结构：连接描述符 ConnectionDescriptor 和连接表 ConnectionTable，然后介绍上述数据结构在 TCP 连接建立和拆除时对应的操作。

ConnectionDescriptor 用于描述一条 TCP 连接，包含可以唯一标识连接的四元组信息（源 IP 地址、目的 IP 地址、源端口号、目的端口号），已缓存报文的数量和长度、乱序重组后报文的最大 ACK 值，用于缓存报文的链表头部，以及表示该 ConnectionDescriptor 是否空闲的标志位。多核 NPU 为每个接收报文处理线程预先在物理地址不交叉的内存中申请大量 ConnectionDescriptor（如图 3.8 所示），并设置维护可用 ConnectionDescriptor 的空闲连接描述符队列，将预先申请的 ConnectionDescriptor 初始化后均添加到空闲连接描述符队列中。

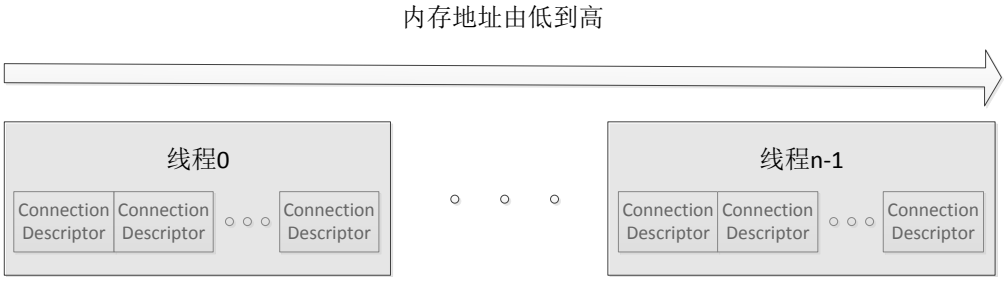

图 3.8 连接描述符的分配

多核 NPU 的每个线程可能管理多条 TCP 连接，因此需要维护多个 ConnectionDescriptor。ConnectionTable 被设计为哈希表，用于快速查找报文所属的 TCP 连接对应的 ConnectionDescriptor。系统使用多核 NPU 的报文分发机制，提取 IP 头部中的源 IP 地址和目的 IP 地址，以及 TCP 头部中的源端口号和目的端口号，将属于同一条 TCP 连接的报文分发到同一线程。线程收到报文后，根据报文四元组（源 IP 地址、目的 IP 地址、源端口号、目的端口号）信息计算哈希值，查找报文对应的 ConnectionDescriptor。多核 NPU 的每个线程维护独立的 ConnectionTable 表项，并预先在 ConnectionTable 的查找和更新操作不需要与其他线程交互，因此不会引起加锁、解锁等同步开销。ConnectionTable 使用链表结构解决碰撞问题，其结构如图 3.9 所示。

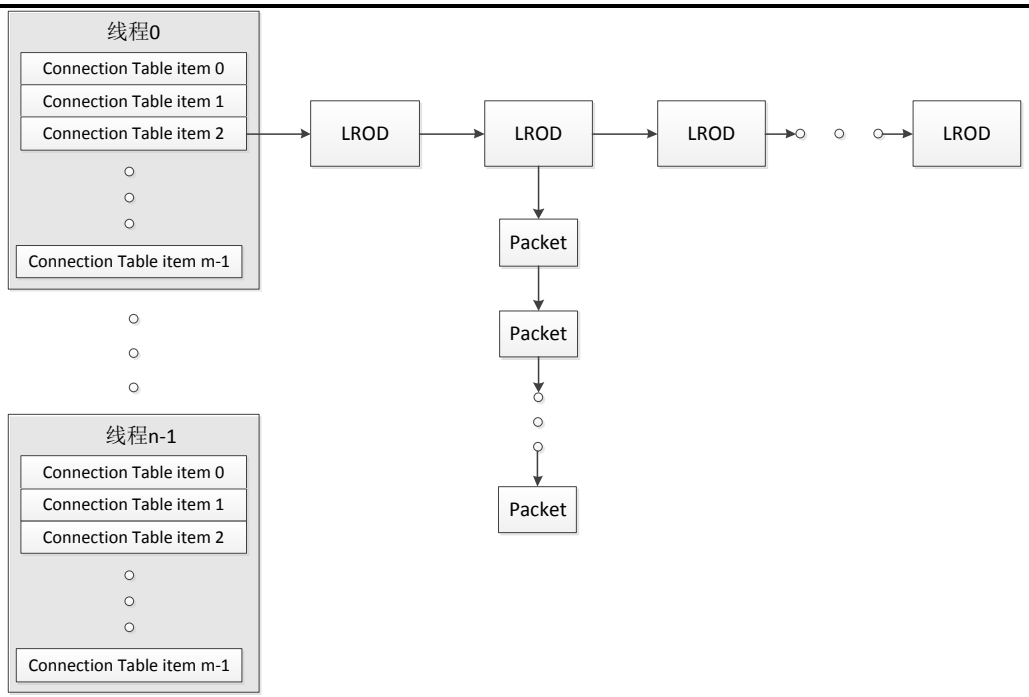

图 3.9 ConnectionTable 结构

系统监测 TCP 连接建立时的三次握手过程，多核 NPU 的接收报文处理线程收到 SYN 报文时，认为连接正在被建立，从空闲连接描述符队列中获取 ConnectionDescriptor，并提取 SYN 报文中的四元组信息（源 IP 地址，目的 IP 地址，源端口号，目的端口号）存储在 ConnectionDescriptor 中，将 ConnectionDescriptor 各字段初始化后添加到 ConnectionTable 中。

系统同时监测 FIN 报文，用于判断是否有连接被拆除。系统对多核 NPU 报文分发引擎的配置可保证将属于同一个 TCP 连接的报文分发到固定的线程上，FIN 报文也不例外。报文处理线程接收到 FIN 报文时，根据四元组信息查找 ConnectionTable，找到 FIN 报文对应的 ConnectionDescriptor，将 ConnectionDescriptor 中已缓存的报文合并后经驱动程序交由协议栈处理，最后将 ConnectionDescriptor 从 ConnectionTable 中移除并重新添加到空闲链接描述符队列中，供新 TCP 连接建立时使用。

多个 NPU 各接收报文处理线程各自维护独立的 ConnectionDescriptor，因此对 ConnectionDescriptor 的操作不需要与其他线程交互，从而避免了加锁、解锁等额外同步开销；采用预先申请大量空闲 ConnectionDescriptor、空闲链接描述符队列的机制可使 ConnectionDescriptor 被循环使用，避免连接建立和拆除时的内存申请和销毁操作，进一步提高系统性能。

3.4.2 报文过滤

多核 NPU 的接收报文处理线程在对报文进行乱序重组和合并等操作之前，首

先需要判断报文是否适合进行这些操作，过滤不适合操作的报文。没通过判断的报文将被丢弃（校验和错误）或提交给驱动程序；通过判断的报文将根据四元组信息查找对应的 ConnectionDescriptor，进而根据 TCP 序列号排序并缓存，用于后续的报文合并。

判断报文是否适合进行乱序重组和合并的条件有：（1）是 IP 报文；（2）IP 校验和正确；（3）IP 报文未分段；（4）是 TCP 报文；（5）TCP 校验和正确；（6）TCP 数据长度大于 0；（7）SYN、FIN、URG 标识未设定；（8）TCP 头部无选项或者只有时间戳选项。与 Grossman 判断报文是否适合 LRO 的不同之处在于，本文不要求报文的 TCP 序列号连续。

多核 NPU 判断报文是否适合进行乱序重组和合并操作的流程如图 3.10 所示。

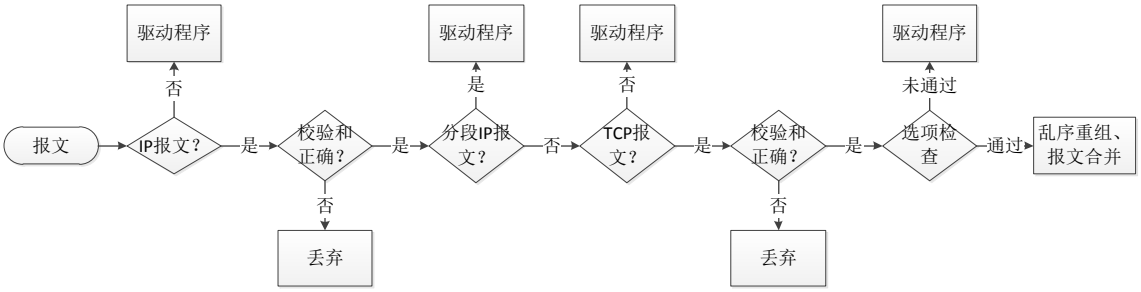

图 3.10 报文过滤流程图

3.4.3 TCP 乱序报文重组

多核 NPU 的接收报文处理线程收到 TCP 数据报文后，首先根据报文四元组信息计算哈希值，在 ConnectionTable 中查找出其所属连接对应的 ConnectionDescriptor，然后将报文缓存在 ConnectionDescriptor 中的报文链表中，同时进行乱序报文重组。缓存的 TCP 数据报文将被用于后续的报文合并操作，乱序报文重组的伪代码如表 3.1 所示。

表 3.1 TCP 乱序报文重组伪代码

```
ReorderPacket(pkt)
Input: pkt: newly arrived TCP packet.
desc <- SearchDescriptor(pkt);
if(desc->nextseq == pkt->seq){
    Insert(pkt, desc);
    UpdateNextSeq(desc);
} else if(desc->nextseq > pkt->seq){
    Drop(pkt);
} else {
    Insert(pkt, desc);
}
```

其中，Insert 函数按照 TCP 序列号为报文找到合适的位置，将报文添加到

ConnectionDescriptor 的报文缓存链表中，同时丢弃重复报文，伪代码如表 3.2 所示。

表 3.2 Insert 函数伪代码

```
Insert(pkt, desc)
Input: desc: TCP connection descriptor; pkt:TCP packet
foreach p in desc->pkt_list {
    if(p->seq == pkt->seq){
        return;
    } else if(p->seq > pkt->seq){
        insert pkt in front of p;
        return;
    }
}
insert pkt at the end of desc->pkt_list;
```

nextseq 记录当前已缓存报文中的最大连续 TCP 序列号，间接表明已缓存的连续数据长度，由 UpdateNextSeq 函数负责更新，伪代码如表 3.3 所示。

表 3.3 更新 nextseq 伪代码

```
UpdateNextSeq(desc)
Input: desc: TCP connection descriptor.
foreach pkt in desc->pkt_list {
    if(desc->nextseq == pkt->seq){
        desc->nextseq = pkt->seq + pkt->payload_len;
    }
}
```

Insert 和 UpdateNextSeq 函数将被频繁调用，因此将其内容直接编写在 ReorderPacket 内部，减少函数调用开销。乱序报文重组算法的复杂度为$O(n)$，其中n为 ConnectionDescriptor 中已缓存报文的数量。

3.4.4 数据报文合并

以下三个条件之一满足时，多核 NPU 的接收报文处理线程会将 ConnectionDescriptor 中已缓存的报文合并，之后经驱动程序交由协议栈处理：（1）ConnectionDescriptor 中已缓存的报文数据长度到达一定阈值；（2）接收报文处理线程收到 TCP 连接拆除过程中的 FIN 报文；（3）ConnectionDescriptor 对应的 TCP 连接接收数据超时。接收报文处理线程对 TCP 报文的处理如图 3.11 所示，ACK 报文中不含负载数据，无需缓存，应及时经驱动程序上传内核协议栈处理。

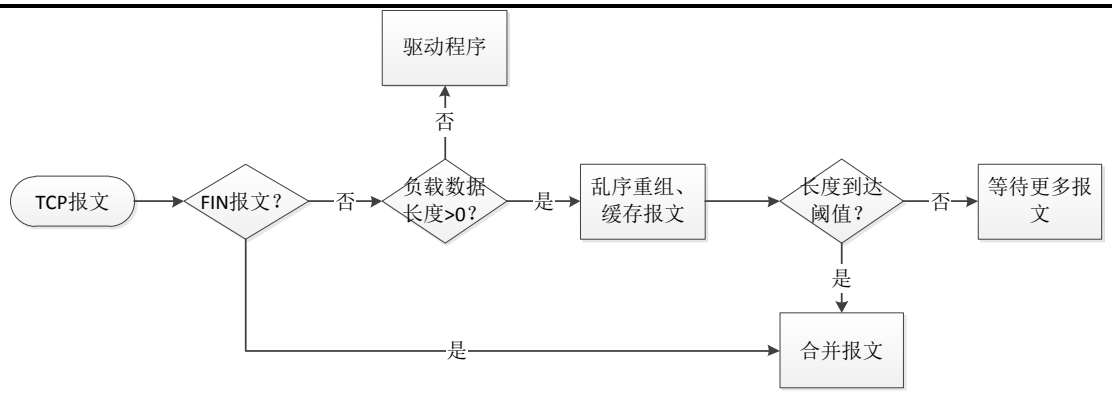

图 3.11 TCP 报文处理流程图

报文合并的主要工作分为两部分：（1）修改新报文 IP 和 TCP 头部中的相应字段；（2）将新报文数据 DMA 到主机内存，驱动程序为新报文构造 skb 数据结构，上传内核协议栈。

新报文头部具体修改方法为：（1）将 TCP 头部的序列号设定为第一个报文的序列号；（2）将 TCP 头部的 ACK 字段设定为最后一个报文的 ACK 值；（3）将 TCP 头部的窗口大小字段设定为最后一个报文的窗口大小；（4）重新计算 TCP 校验和；（5）修改 IP 头部的报文长度字段；（6）重新计算 IP 校验和。

之后，多核 NPU 获取一个由驱动程序预先在 PCIE 共享内存中分配的接收报文描述符，根据接收报文描述符中的主机内存页地址信息将报文数据 DMA 到驱动程序预先申请的内存页中，并设定接收报文描述符中的相关字段，表明有报文到达，最后产生中断，触发驱动程序的中断处理函数对报文进行接收，并重新申请连续内存页为后续报文接收使用。

3.4.5 报文接收超时检测

TCP 连接建立之后，可以长时间内不进行任何报文交互而保持连接有效，考虑以下情况：系统将合并报文的缓存数据长度阈值设定为 64KB，而发送方只需传输 8KB 数据。若系统不对连接对应的 ConnectionDescriptor 进行报文接收超时检测，发送方传输的 8KB 数据将一直被缓存在多核 NPU 中，而到达不了接收方主机；接收方内核协议栈没有收到数据，因此不会发送数据确认报文，从而引起发送方重传计时器超时，数据被重新发送；重传数据到达多核 NPU 后，被判定为重复报文而丢弃。由此可见，对 ConnectionDescriptor 进行报文接收超时检测必不可少。

接收报文处理线程在 ConnectionDescriptor 中还维护另外两个字段：（1）average_interval，表示相邻两个报文到达多核 NPU 的平均时间间隔，若 ConnectionDescriptor 在最近一次接收到报文后的 10*average_interval 时间内没有接收到报文，则判断为接收报文超时；（2）last_arrival，记录 ConnectionDescriptor

最近一次接收到报文的时刻。当报文到达多核 NPU 时，报文处理线程读取系统时间 current_time，根据报文的四元组信息查找其所属的 ConnectionDescriptor，然后更新 ConnectionDescriptor 中的 average_interval 字段：

$$average\_interval = (average\_interval + current\_time - last\_arrival)/2 \quad (3\text{-}1)$$

并将报文到达的时刻 current_time 存入 ConnectionDescriptor 的 last_arrival 字段。

系统使用多核 NPU 的一个单独线程完成所有活跃 ConnectionDescriptor 接收报文超时检测工作，该单独线程被命名为超时扫描线程。超时扫描线程的代码主体框架为一个无限循环（如图 3.12 所示），循环中每次读取一个 ConnectionDescriptor 的 last_arrival 值和系统当前时刻 current_time，如果 current_time 与 last_arrival 的差值大于 ConnectionDescriptor 的 average_interval*10，判断该连接接收报文超时；超时扫描线程提取 ConnectionDescriptor 中的四元组信息，构造接收报文超时消息发送给 ConnectionDescriptor 所属的接收报文处理线程；接收报文处理线程接收到超时消息后，根据超时消息中的四元组信息查找到对应的 ConnectionDescriptor，将其已经缓存的报文合并后经驱动程序交由内核协议栈处理。

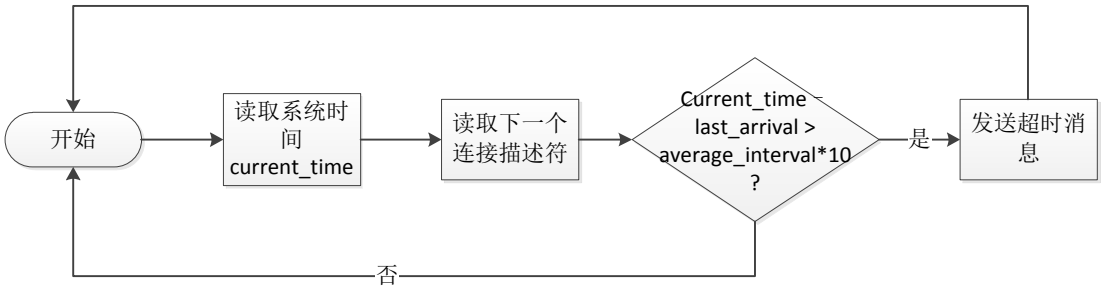

图 3.12 超时扫描线程流程图

超时扫描线程与接收报文处理线程之间利用多核 NPU 的快速消息网络（Fast Messaging Network，FMN）进行线程通信，避免了加锁、解锁等线程同步开销，简洁高效地完成了 ConnectionDescriptor 接收报文超时检测工作。

3.5 本章小结

本章本章首先简要介绍了传统 TCP/IP 接收数据的处理过程，以及网卡、驱动程序和内核协议栈在数据接收过程中的具体操作；其次详细说明了多核 NPU 良好的报文处理能力，针对 TOE 和 LRO 技术的缺点提出使用多核 NPU 作为网卡卸载 TCP 乱序报文重组功能以及执行报文合并操作，加速 TCP 的技术；之后从多核 NPU 和网卡驱动程序的角度详细介绍了系统框架结构，最后从 TCP 连接管理、乱序报文重组、报文过滤、报文合并以及报文接收超时检测等五个角度对系统功能设计进行了详细说明。

第四章 系统改进

第三章提出了使用多核 NPU 作为网卡加速 TCP 的技术，并对系统的框架结构和功能设计进行了详细说明。本章将介绍在具体实现系统时所做的改进，主要包括使用多接收报文描述符环、DMA 负载均衡、合并报文校验和计算优化以及多核 NPU 主动发送 ACK 报文等方面。

4.1 多接收报文描述符环

系统初始化时，在主机内存中申请连续的页，用于存放报文数据；同时在主机与多核 NPU 的 PCIE 共享内存中分配接收报文描述符，并将连续内存页的物理地址保存于接收报文描述符中，在多核 NPU 进行 DMA 报文数据时使用。

多核 NPU 一般拥有四个以上的处理核心，每个处理核心又拥有多个线程，总线程数量可超过 16 个。系统着重于对 TCP 数据接收路径进行优化，因此分配了多个接收报文处理线程，每个接收报文处理线程均可获取接收报文描述符，发起数据 DMA 操作；如果系统将接收报文描述符维护在一个环中，必然引起多个接收报文处理线程之间的竞争，从而引入加锁、解锁等额外同步开销。为避免此问题，系统为每个接收报文处理线程在 PCIE 共享内存的独立地址段内分配接收报文描述符，并将其维护在一个独立的报文描述符环内，如图 4.1 所示。

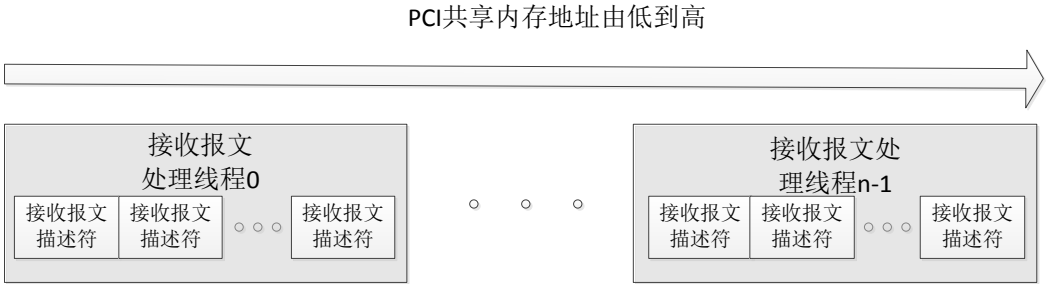

图 4.1 接收报文描述符的分配

系统为每个接收报文描述符环还额外维护一个 DMA 标志位字段，表明该报文描述符环对应的接收报文处理线程是否 DMA 数据到主机内存中，如图 4.2 所示。多核 NPU 的接收报文处理线程将报文数据 DMA 到主机内存后，设置接收报文描述符中的数据长度等字段，以及设置报文描述符环的 DMA 标志位，最后向主机发起中断；在驱动程序的中断函数中，将循环检查所有接收报文描述符环的 DMA 标志位，对 DMA 标志位设定的环进行报文接收，最后将 DMA 标志位清零。

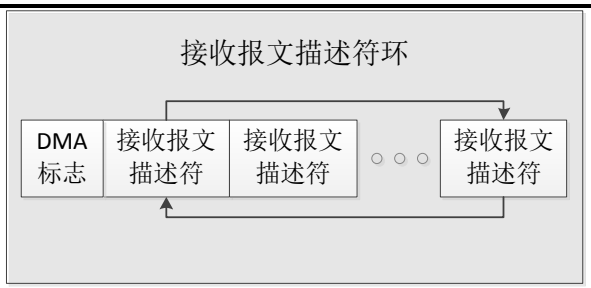

图 4.2 接收报文描述符环

采用上述设计后，驱动程序的中断处理函数将增加读取所有接收报文描述符环 DMA 标志位的操作，但接收报文处理线程数量有限（最多不过数十个），其开销与加锁、解锁等线程间同步操作的开销相比可忽略不计，因此，采用多接收报文描述符环的设计可有效地提高系统性能。

4.2 DMA 负载均衡

多核 NPU 拥有一个独立工作的 DMA 引擎，与主机内存之间 DMA 数据的流程为：（1）多核 NPU 的某线程构造包含数据源地址、数据目的地址以及数据长度的 DMA 请求消息，发送给多核 NPU 的 DMA 引擎；（2）DMA 引擎完成数据 DMA 工作；（3）DMA 引擎发送 DMA 结果（成功或失败）给发起请求的线程。消息交换时序如图 4.3 所示。

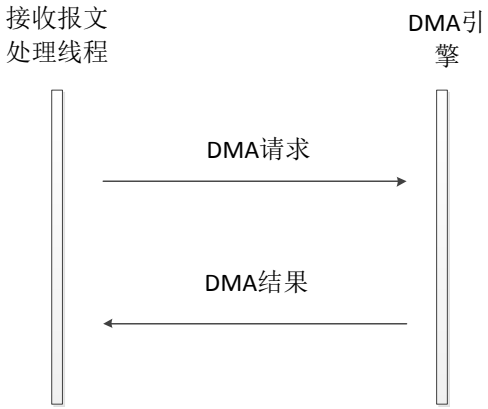

图 4.3 多核 NPU 的 DMA 消息序列

系统中设置了多个接收报文处理线程，每个接收报文处理线程在合并报文后，均可获取接收报文描述符，根据报文描述符内容发起 DMA 请求。在系统实现时，出现以下情况：数据发送方向接收方同时发起多个 TCP 连接，进行数据传输，但只有少数几个 TCP 连接能够正常建立并且传输数据，而剩余多数连接建立超时。造成这种现象的原因为：系统没有在多个接收报文处理线程之间实现 DMA 负载均衡功能，少数几个 TCP 连接建立后，其对应的接收报文处理线程将紧接着接收到数据报文，再次发起 DMA 请求，DMA 引擎将忙于处理这些请求而使多数 TCP 连

接的 SYN 报文不能到达主机，从而导致多数 TCP 连接建立超时。

为解决上述问题，系统利用超时扫描线程，为多个接收报文处理线程增加 DMA 负载均衡机制：（1）超时扫描线程为每个接收报文处理线程维护单独的 DMA 请求消息队列 DMA_MSG_Q；（2）接收报文处理线程将 DMA 请求消息发送给超时扫描线程，而非 DMA 引擎；（3）超时扫描线程接收到 DMA 请求消息后，将缓存在对应的请求消息队列中；（4）超时扫描线程采用 Round-Robin 机制进行各接收报文线程之间的 DMA 负载均衡（如图 4.4 所示），在其主体循环内，每次从一个 DMA 请求消息队列中取出一个消息，发送给 DMA 引擎；（5）超时扫描线程接收到 DMA 引擎返回的 DMA 结果消息后，将消息发送给相应的接收报文处理线程。采用负载均衡机制后，DMA 消息交换的时序如图 4.5 所示。

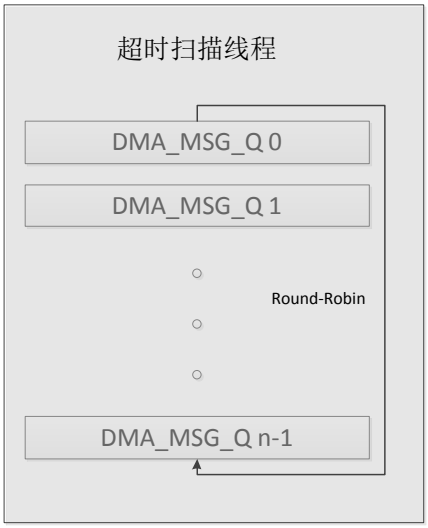

图 4.4 Round-Robin DMA 负载均衡机制

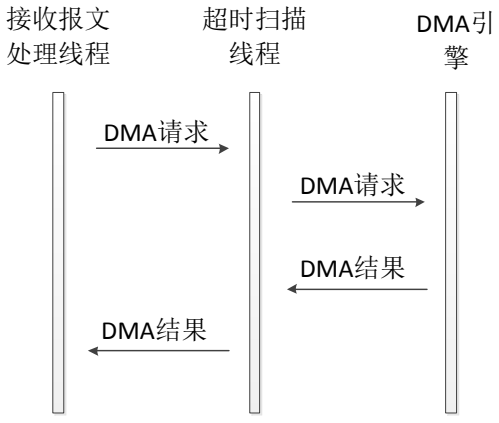

图 4.5 DMA 负载均衡消息序列

上述负载均衡机制中，每次 DMA 操作只给系统增加两次请求消息队列的访问操作（将消息加入和移出队列），两次 DMA 消息传递操作（一次超时扫描线程向 DMA 引擎发送 DMA 请求消息，一次超时扫描线程向接收报文处理线程发送 DMA

结果消息），简单高效，并且易于实现。

4.3 合并报文校验和计算优化

多核NPU的接收报文处理线程将属于同一个TCP连接的多个数据报文合并后，需要重新计算新报文中的TCP校验和以及IP校验和。目前以太网的发展速度远超存储器技术的发展速度，存储器访问已经成为端系统TCP性能的瓶颈之一；校验和的计算本身并不复杂，但其涉及所有需要校验的数据，存储器的访问操作使得校验和计算开销巨大。本文在2.1节中介绍了端系统中优化校验和计算的方法，多核NPU同样需要优化合并报文中的TCP校验和以及IP校验和的计算，提升系统性能。

4.3.1 IP校验和计算优化

IP校验和只包含IP头部的各字段，不包括TCP头部和TCP负载数据。用*length*表示IP头部长度字段，*rest*表示剩余字段，*sum*函数计算给定数据的反码和，+'表示反码加和运算符，则IP校验和*cksum*的计算公式为：

$$cksum = \sim (length + ' sum(rest)) \quad （4\text{-}1）$$

由此得

$$sum(rest) = \sim cksum + ' \sim length \quad （4\text{-}2）$$

用new_length表示合并报文后IP头部长度字段，new_cksum表示新报文的IP校验和，则有：

$$new\_cksum = \sim (new\_length + ' sum(rest)) \quad （4\text{-}3）$$

结合（4-2）得：

$$new\_cksum = \sim (new\_length + ' \sim cksum + ' \sim length) \quad （4\text{-}4）$$

多核NPU的接收报文处理线程在合并报文时采取公式（4-4）计算新报文的IP校验和，计算时只需访问IP头部的长度字段和校验和字段，避免了其他字段的访问开销。

4.3.2 TCP校验和计算优化

与IP校验和不同，TCP校验和涉及TCP头部所有字段、伪头部字段，以及TCP负载数据三部分。伪头部中包含的字段有：4字节源IP地址、4字节目的IP地址、2字节协议类型、2字节TCP长度（TCP头部长度加负载数据长度）。用psd_header表示伪头部，tcp_header表示TCP头部，tcp_data表示负载数据，则TCP校验和*cksum*的计算公式为：

$$cksum = \sim (sum(psd\_header) + 'sum(tcp\_header) + 'sum(tcp\_data)) \quad (4\text{-}5)$$

从而得

$$sum(tcp\_data) = \sim cksum + ' \sim sum(psd\_header) + 'sum(tcp\_header) \quad (4\text{-}6)$$

假设被合并的 TCP 数据报文数量为 n， psd_header'表示合并报文后的伪头部，tcp_header'表示合并报文后的 TCP 头部，tcp_data'为新报文的负载数据，则新报文的 TCP 校验和 new_cksum 为：

$$new\_cksum = \sim (sum(psd\_header') + 'sum(tcp\_header') + 'sum(tcp\_data')) \quad (4\text{-}7)$$

其中

$$sum(tcp\_data') = \sum_{i=1}^{n} sum(tcp\_data_i) \quad (4\text{-}8)$$

则有：

$$new\_cksum = \sim (sum(psd\_header') + 'sum(tcp\_header') + \sum_{i=1}^{n} sum(tcp\_data_i)) \quad (4\text{-}9)$$

合并报文后的伪头部中只有 TCP 长度字段变化，TCP 头部只有 seq、ack 以及窗口大小字段发生变化，因此 $sum(psd\_header')$ 和 $sum(tcp\_header')$ 的计算可与 IP 校验和采用相同的优化方法。

多核 NPU 的接收报文处理线程在缓存某 TCP 连接的数据报文时，根据公式（4-6）计算各报文的负载数据反码和，并将其值存储在连接对应的 ConnectionDescriptor 中；接收报文处理线程将数据报文合并后，将根据公式（4-9）计算新报文的 TCP 校验和。

采用上述方法计算合并报文的 TCP 校验和可以避免对负载数据的访问操作，大幅减少存储器的访问开销，从而提升系统性能。

4.4 主动 ACK 机制

TCP 发送数据后，将设定重传计时器，若在重传计时器超时后仍未收到数据的确认报文，则重新发送数据，并进行拥塞控制，将发送窗口设定为一个 MSS 大小。考虑以下情况：连接建立后，数据发送方向接收方传输一定数量报文，并设定重传计时器；这些报文到达多核 NPU 后被接收报文处理线程缓存在对应的 ConnectionDescriptor 上，但由于数据长度没有达到报文合并阈值，接收报文处理线程并不进行报文合并，而是等待更多数据报文；主机内核协议栈不会接收到报文，不会发出确认报文，从而使得发送方重传计时器超时。

上述问题将引起大量数据重传，严重影响发送方发送数据速率，使得 TCP 连接不能进行正常的数据交互，与系统加速 TCP 处理性能的目的背道而驰。

解决上述问题的策略之一为：将接收报文处理线程合并报文的数据长度阈值减小，设定为超时时间内接收方收到的所有报文数据长度之和，从而避免发送方重传计时器超时。但由于网络联通情况不断变化，同一个 TCP 连接的报文可能选择不同的路由路径，发送方超时时间也不断变化，多核 NPU 根据超时时间维护合并报文数据长度阈值的代价太高；另一方面，合并报文数据长度阈值减小后，多核 NPU 产生的中断数量以及主机内核协议栈处理的报文数量将增多，主机 CPU 的开销将增大，从而引起系统性能下降。

系统采用主动 ACK 机制解决上述问题：在接收报文处理线程进行 TCP 乱序报文重组并缓存报文时，由多核 NPU 主动构造 ACK 报文，并发送给发送方。ACK 报文的构造算法为：

以太网头部：（1）将源 MAC 地址设定为接收报文的目的 MAC 地址；（2）将目的 MAC 地址设定为接收报文的源 MAC 地址；（3）将以太网类型字段设定为 0x0800。

IP 头部：（1）将版本号字段设定为 4；（2）将头部长度字段设定为 5；（3）将 TOS（Type of Service）字段设定为 0；（4）将总长度字段设定为 40；（5）将标识符字段设定为 0；（6）设定 ACK 标志位和偏移量字段为 0x4000；（7）设定 TTL（Time to Live）和协议号字段为 0x4006；（8）将源 IP 地址设定为接收报文的目的 IP 地址；（9）将目的 IP 地址设定为接收报文的源 IP 地址；（10）计算 IP 校验和。

TCP 头部：（1）将源端口号设定为接收报文的目的端口号；（2）将目的端口号设定为接收报文的源端口号；（3）将序列号字段设定为接收报文的 ack 值；（4）将 ack 字段设定为接收报文所属 ConnectionDescriptor 的 nextseq；（5）将头部长度字段，保留字段以及标志位字段设定为 0x5010；（6）紧急指针字段设定为 0；（7）将窗口大小字段设定为 0xffff；（8）计算 TCP 校验和。

多核 NPU 发送报文的流程为：（1）线程根据报文数据在多核 NPU 存储空间中的地址，以及报文数据长度构造报文发送请求消息；（2）线程将报文发送请求消息发送给网络接口；（3）网络接口将报文发送后，返回报文发送结果消息（成功或失败）给相应线程。发送报文消息交换时序如图 4.6 所示。

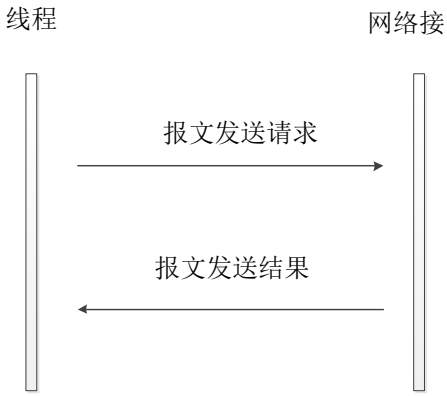

图 4.6 发送报文消息序列

接收报文处理线程需要保证 ACK 报文被成功发送，因此在 ACK 报文构造完成后，将进入一个循环，循环中首先发送报文发送请求给网络接口，之后等待网络接口返回报文发送结果，直至报文发送成功推出循环，如图 4.7 所示。

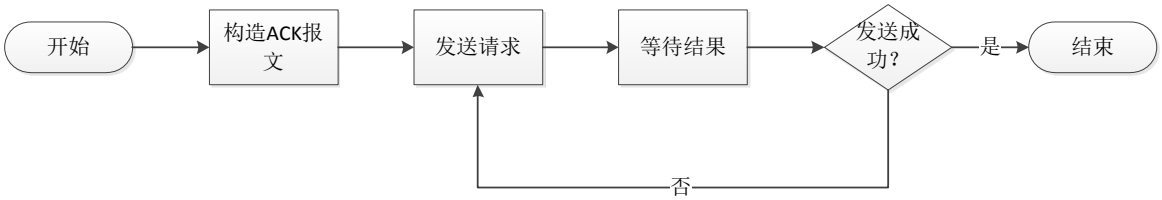

图 4.7ACK 报文发送流程图

主动 ACK 机制使接收报文处理线程为接收到的数据报文构造 ACK 报文并发送给数据发送方，解决了因发送方重传计时器超时而引起的报文重传和数据发送速率减慢的问题。此外，通过将 ACK 报文中的窗口大小设定为 0xffff，可使发送方以更高的速率发送数据，进一步提高系统性能。

4.5 本章小结

本章详细介绍了实现系统时使用的各项改进技术，主要包括：使用多接收报文描述符环，避免接收报文处理线程获取报文描述符时的同步开销；利用超时扫描线程，基于 Round-Robin 机制实现接收报文处理线程间 DMA 负载均衡功能；合并报文后，IP 校验和以及 TCP 校验和计算的优化；使用主动 ACK 机制解决因发送方重传计时器超时引起的报文重传和数据发送速率减慢的问题。

第五章　基于 XLS416 的系统实现

前两章提出了系统使用多核 NPU 作为网卡，卸载 TCP 乱序报文重组功能以及执行报文合并操作，加速 TCP 的技术，详细介绍了系统的框架结构、功能设计以及各项改进技术。系统基于 Broadcom 公司的 XLS416 多核 NPU 实现，本章首先介绍 XLS416 开发平台，其次详细说明 XLS416 资源的分配，最后对系统性能进行测试和评估。

5.1 XLS416 开发平台

本节将分别从结构和功能特性、处理核心、快速消息网络（FMN，Fast Messaging Network）以及网络加速器四个方面对 XLS416 开发平台进行介绍。

5.1.1 XLS416 结构和功能特性

XLS416 的结构如图 5.1 所示，其功能特性包括：（1）4 个基于 MIPS64 的处理核心，每个核心有 4 个线程，支持分支预测、Load/Store 操作地址自动对齐功能；（2）可扩展的网络接口，包括 8 个以太网 SGMII 接口，1 个 RGMII 接口，2 个 10Gbps XAUI 端口，每个网络接口均有硬件加速支持；（3）高性能可配置存储管理器：支持 ECC 功能的 800MHz DDR2 DRAM，4 通道 DMA；（4）高速分布式互联网络：快速消息网络 FMN 为线程和 I/O 器件之间提供了高效的信息交换功能，高速分布式存储器互联网络提供了高效的数据传输功能；（5）多项系统集成接口，包括 PCIE 接口、2 个 USB2.0 接口、PCMCIA 接口、可启动 Flash 接口、2 个 I^2C 接口、2 个 UART 接口以及 32 比特 GPIO 接口；（6）硬件网络加速：可线速处理报文的 PDE（Packet Distribution Engine），提供灵活的报文分发机制；（7）支持 2.5Gbps 网络流量的数据压缩引擎；（8）安全加速引擎 SAE（Security Acceleration Engine）：支持 2.5Gbps 的网络流量加解密，以及多种加密认证算法。

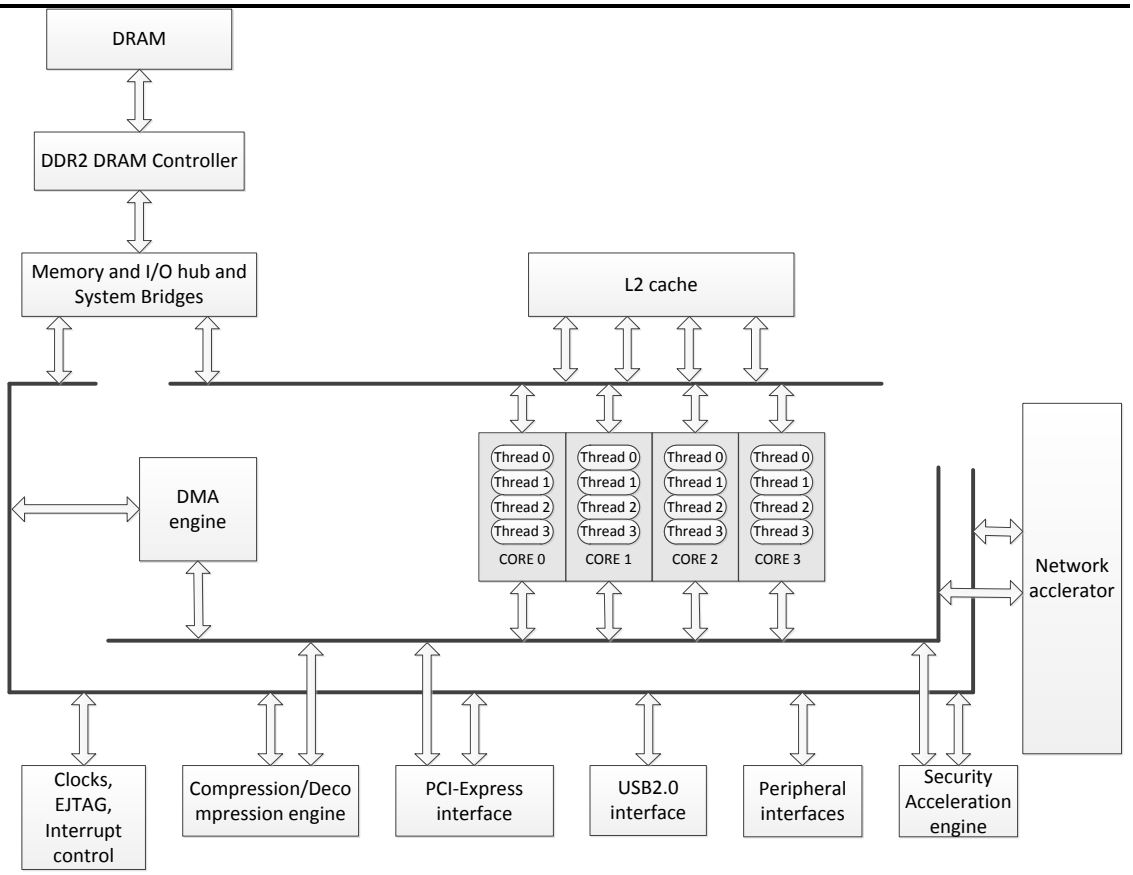

图 5.1 XLS416 系统结构

5.1.2 处理核心

XLS416 的每个处理核心拥有 4 个硬件线程，其结构如图 5.2 所示。

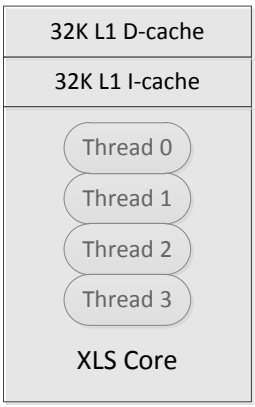

图 5.2 XLS416 处理核心结构

多线程使得每个处理核心同时开发指令级并行和线程级并行，最大化流水线的性能。处理核心中的每个线程拥有独立的寄存器组，以及中断和异常处理机制。从操作系统或应用程序的角度，一个硬件线程与物理处理器等效，因此硬件线程也被称作虚拟 CPU。

处理核心的流水线在以下事件发生时产生停顿：（1）L1/L2 缓存未命中；（2）分支预测失败；（3）TLB（Translation Lookaside Buffer）未命中和替换；（4）内存访问。上述事件发生后，处理核心的线程调度器使线程睡眠，并根据配置调度下一个线程执行，从而避免处理器周期损耗，睡眠的线程在依赖条件满足后再次变为可执行状态。

每个处理核心的线程调度器可根据具体应用的特点进行不同配置，具有很高的灵活性。每个处理核心提供以下三种线程调度模式：

（1）细粒度模式：

细粒度是处理核心默认的调度策略，如果一个线程的代码可以执行，则分配给该线程一定数量的执行周期，之后调度下一个线程，如图 5.3 所示。

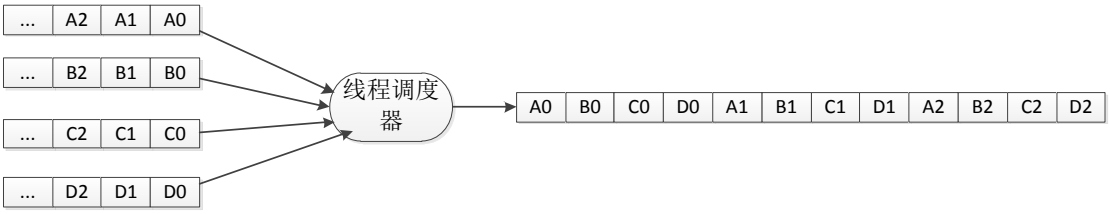

图 5.3 细粒度调度模式

如果一个线程发生停顿而不能执行，线程调度器将跳过它，调度之后的线程，如图 5.4 所示。

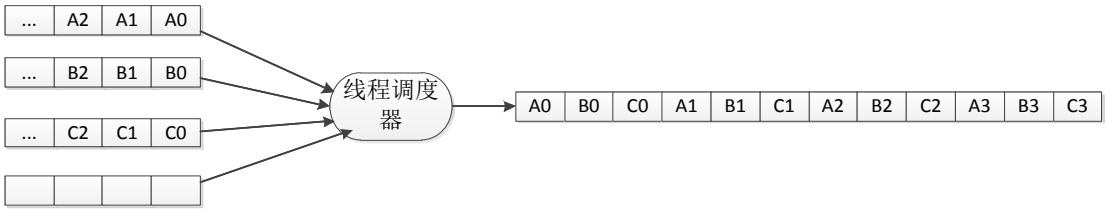

图 5.4 线程停顿时细粒度调度模式

每四个周期后，线程调度器还使用一个线性反馈移位寄存器（LFSR，Linear Feedback Shift Register）随即选择下一个调度的线程，保证在各线程在有资源竞争时进程不被停顿。

（2）粗粒度调度模式

粗粒度调度模式可为每个线程设置不同的执行周期，若一个线程不在睡眠状态，则它将被执行预设的周期数量，之后调度下一个线程。线程调度器将跳过睡眠的线程，调度执行下一个非睡眠状态的线程。若按表 5.1 的对各线程进行执行周期数量配置，则调度结果如图 5.5 所示。

表 5.1 线程执行周期数量配置

线程	D	C	B	A
执行周期数量计数器	0	50	0	200
被分配执行周期数量	1	51	1	201

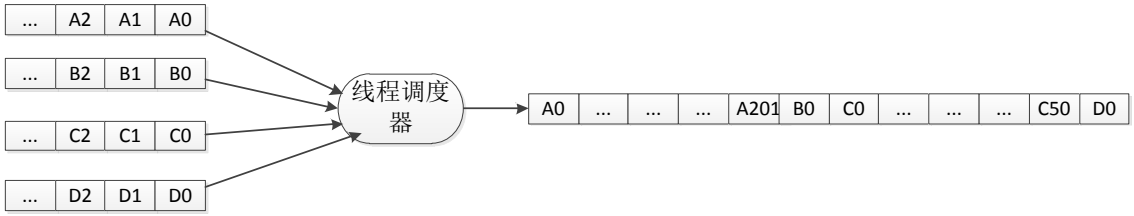

图 5.5 粗粒度调度模式

（3）优先级调度模式

优先级调度模式为每个线程维护优先级计数器，调度器按照优先级计数器的设置以不同的频率调度执行各线程。若按照表 5.2 对优先级计数器进行配置，则各线程的调度频率将为：线程 A 的调度频率不会超过每 3 个周期一次，线程 B 的调度频率不会超过每 4 个周期一次，线程 C 的调度频率不会超过每 5 个周期一次，线程 D 的调度频率不会超过每 6 个周期一次。线程调度结果如图 5.6 所示。

表 5.2 线程优先级计数器配置

线程	D	C	B	A
优先级计数器	5	4	3	2

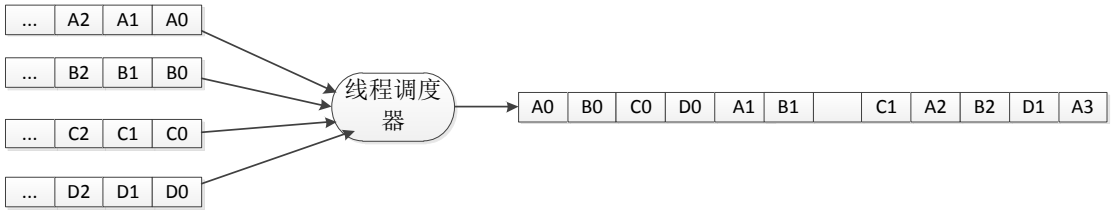

图 5.6 优先级调度模式

5.1.3 快速消息网络 FMN

FMN 将处理核心、网络加速器、DMA 引擎、安全加速引擎、压缩引擎以及 PCIE 联通，实现上述部件之间的高效消息传递功能。网络接口与应用程序之间通过 FMN 进行报文传递，经过配置后，FMN 将自动管理信息带宽而不产生额外软件开销。FMN 在进行报文信息传递时不会引起自旋锁、互斥量等同步操作，报文消息中包含报文数据的位置信息而非数据本身，报文数据通过分布式存储器网络传输。FMN 通过信用机制和 Round-Robin 机制调度各消息站，保证消息能够快速到达目的地，以及各消息站使用 FMN 的公平性。FMN 虽然被主要用于报文快速传输，但用户可以自定义消息语义，实现高效的线程间通信。

5.1.3.1 FMN 结构

FMN 是一个低延时、单向的、基于环形结构的快速消息网络，其结构如图 5.7

所示。XLS 处理器、网络接口、安全加速引擎、DMA 引擎等核心器件通过消息站连接在 FMN 上。

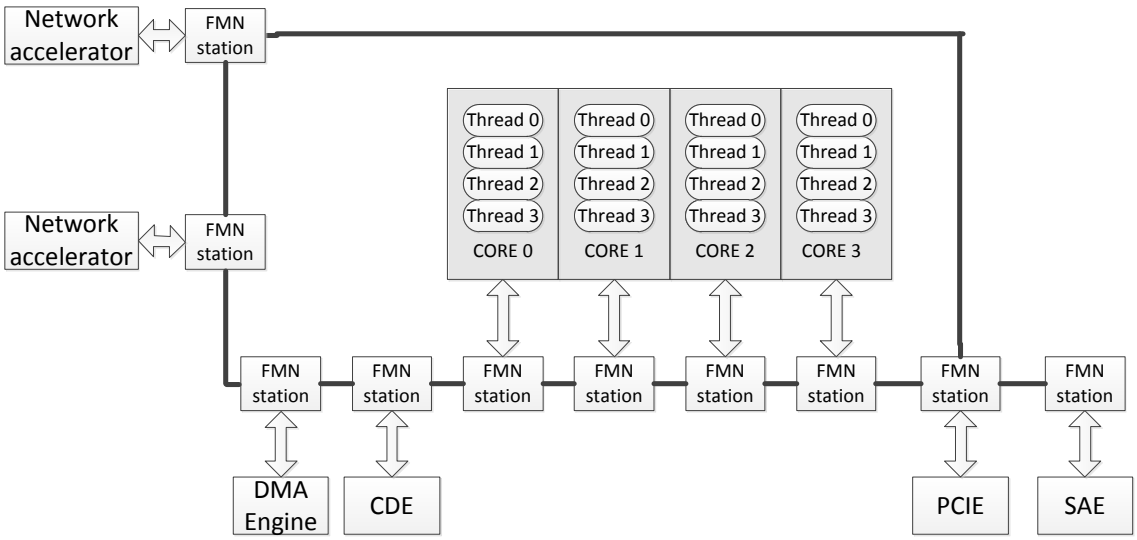

图 5.7 FMN 结构图

5.1.3.2 FMN 处理报文

报文到达被消息站接收后，将根据消息站的部件类型进行下一步动作：如果消息站为处理核心，则报文被传递给处理核心中的某个线程；如果消息站为网络接口，则报文被网络接口发送。网络加速器通过 FMN 与线程之间传递报文信息，图 5.8 为 FMN 处理接收报文的示意图。当报文通过网络接口到达 XLS416 后，其数据将被存储在内存中，同时网络加速器对报文进行解析和分类，确定向哪个线程发送报文到达消息 RxPacketDescriptor。线程所在的处理核心消息站接收到 RxPacketDescriptor 后，将通知线程新报文到达。

RxPacketDescriptor 中包含报文数据的内存地址、数据长度、报文分类等其他信息。线程通过内存地址信息读取报文数据进行处理，若处理完成后线程决定将报文转发，则用报文数据内存地址和数据长度构造 TxPacketDescriptor 消息，并通过 FMN 将消息发送给网络接口，网络接口接收到 TxPacketDescriptor 后将报文通过输出端口发送。

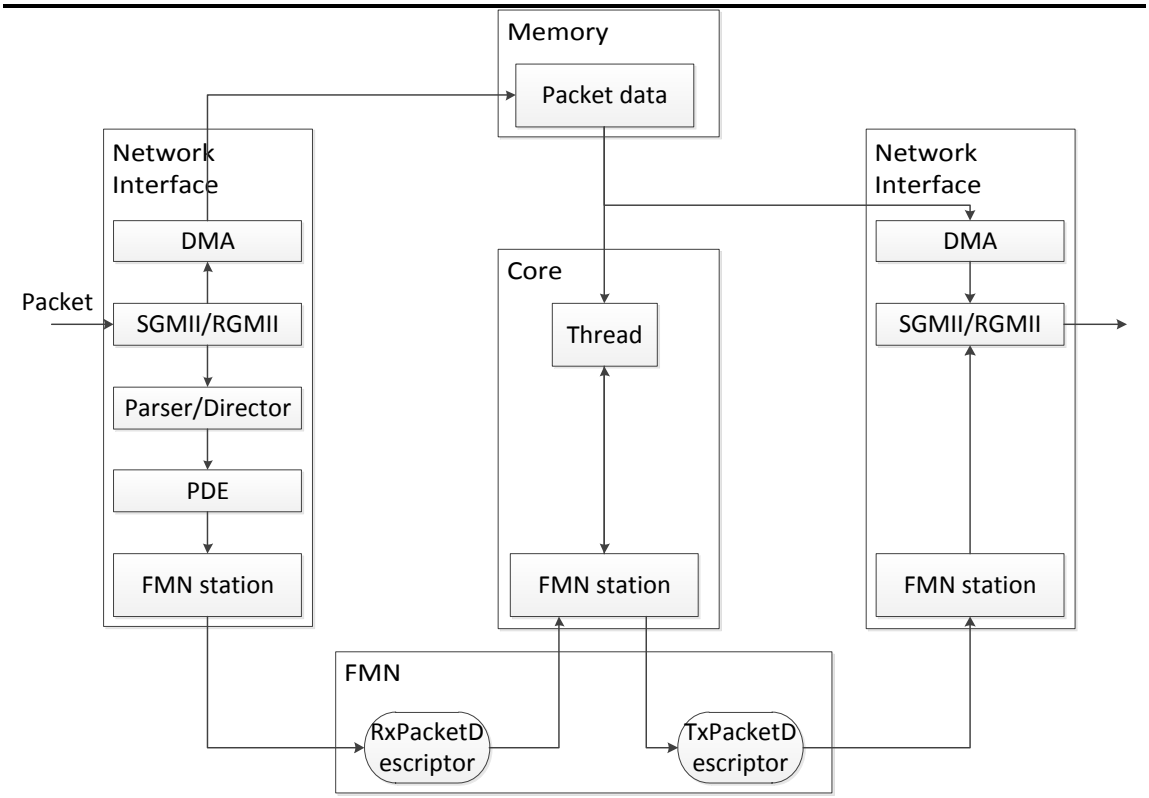

图 5.8 FMN 处理报文

5.1.3.3 基于信用机制的消息管理

软件在系统初始化时对各消息站进行带宽分配，之后消息站的带宽管理由 FMN 自动进行，无软件额外开销。FMN 的流量控制基于信用机制：每个消息站为每个消息接收桶维护一个信用计数器，当消息站要给某接收桶发送消息时，FMN 首先检查消息站的信用计数器，如果其值小于消息长度，则消息不被发送，如果其值大于等于消息长度，FMN 发送消息，并且按照消息长度减小消息站的信用计数器。接收方读取消息后，将通过特殊的带外通道向发送方消息站发送释放信用消息 FreeCreditMessage，通知消息发送方消息已经被读取，并且将发送方消息站的信用计数器增加消息长度。消息时序如图 5.9 所示。

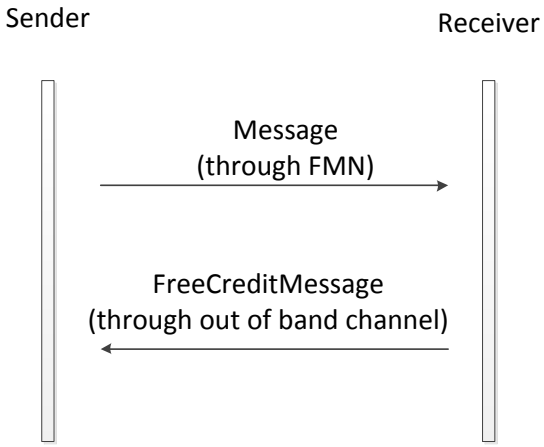

图 5.9 基于信用机制的消息时序

通过使用带外通道传输 FreeCreditMessage 不会给 FMN 带来额外开销，不会使 FMN 的消息传输吞吐量下降。

5.1.4 网络加速器

网络加速器（Network Accelerator）位于网络与 FMN 之间，具备以下功能：（1）报文语法分析；（2）报文流向控制；（3）校验和检验；（4）处理核心之间的报文分发。每个网络加速器的核心部件有：（1）DMA 引擎；（2）报文语法分析器（Parser）；（3）报文导向器（Packet Director）；（4）TCAM；（5）报文分发引擎（PDE，Packet Distribution Engine）。

5.1.4.1 Parser

报文到达网络接口后，网络加速器将报文数据存放到 XLS416 内存中，同时 Parser 根据用户的定义提取报文中的相应字段，组成 128 比特的关键字（如图 5.10），交给 Packet Director 用于分类和分发报文。

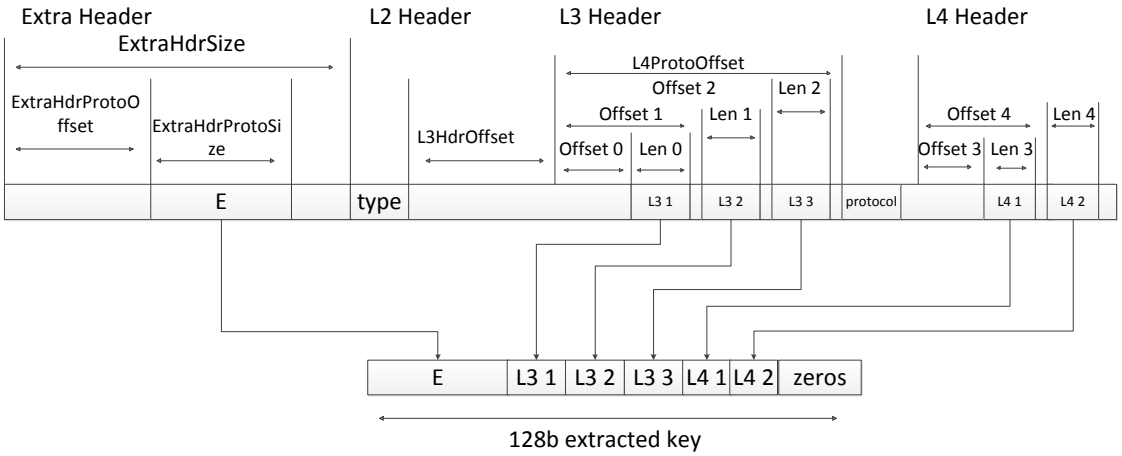

图 5.10 Parser 提取报文信息

5.1.4.2 Packet Director

Packet Director 以 Parser 生成的 128 比特报文关键字作为输入，生成 9 比特报文分类信息，作为报文分发引擎的输入。9 比特报文分类信息包括：（1）2 比特报文类别 ClassID；（2）6 比特接收桶标识符 BucketID；（3）1 比特 UseBucket 标志位；（4）1 比特转发标志位。

Packet Director 支持以下四种工作模式：

（1）TCAM：XLS416 的 TCAM 宽度为 128 比特，共四项，每一项包含 128 比特掩码，表示需要进行比较的字段。TCAM 模式下，使用 Parser 生成的 128 比特关键字与 TCAM 中每一项进行比较，若有匹配成功项，则根据匹配结果生成 9 比特报文分类信息，若无匹配成功项，则使用第二级报文分流方法。

（2）CRC7 哈希：使用 Parser 生成的 128 比特关键字作为 CRC7 哈希算法的输入，计算 9 比特分类信息。

（3）L3/L4 协议：提取报文 L3 和 L4 的相应字段，通过转换表生成 9 比特分类信息。

（4）MSB：提取 128 比特报文关键字的高 7 位，生成 9 比特报文分类信息。

5.1.4.3 报文分发引擎 PDE

PDE 以 Packet Director 生成的 9 比特报文分类信息作为输入，决定报文分发目的地址，其结构如图 5.11 所示。

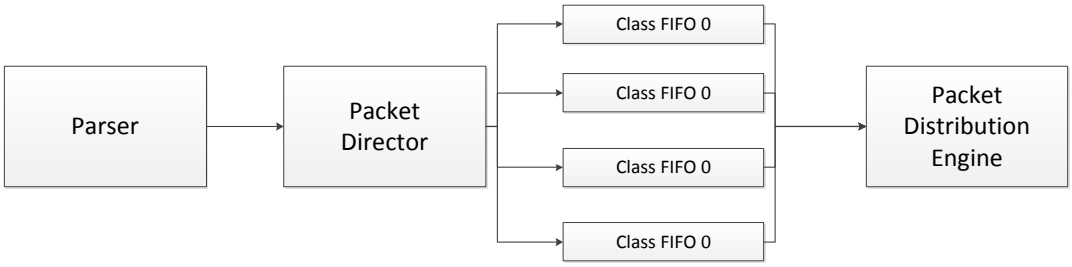

图 5.11 报文分发引擎 PDE

Packet Director 将报文分为四类，用 2 比特 ClassID 表明；1 比特 UseBucket 表示报文分发时是否使用类别信息。

若 UseBucket 为 0，PDE 将根据报文类别，采用 Round-Robin 机制在处理核心的线程之间分发报文。ClassID 表明报文的类别，每种类别用一个长度为 64 比特的掩码表示可接收本类报文的线程。

若 UseBucket 为 1，PDE 将按照 6 比特接收桶标识符 BucketID 直接将报文分发到对应的线程，而忽略报文的分类信息。

5.2 XLS416 资源分配

本文第三章和第四章详细说明了系统的框架结构和功能设计，以及实现过程中所采取的改进技术，本章将详细介绍实现系统时 XLS416 资源的分配情况。

5.2.1 PCIE 共享内存分配

XLS416 拥有 16MB 的 PCIE 共享内存，能够被主机和 XLS416 直接进行读写操作，因此 PCIE 共享内存被用于存储主机和 XLS416 需要共享的数据，包括：接收报文描述符、发送报文描述符、设备状态信息以及各类计数器，如图 5.12 所示。

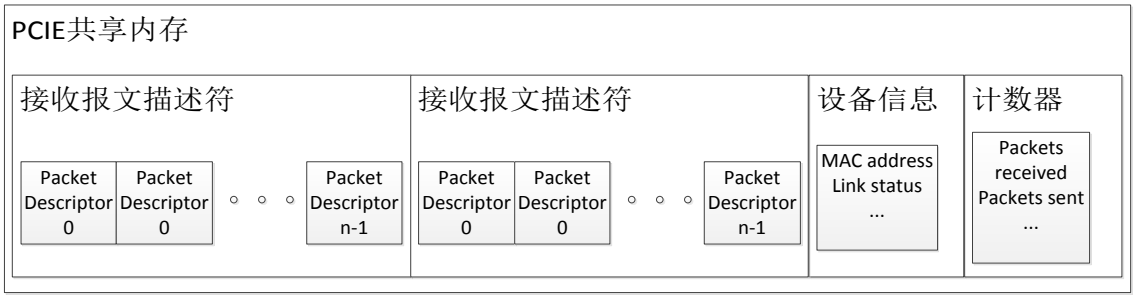

图 5.12 PCIE 共享内存分配

系统分配多条线程处理接收报文，每个接收报文处理线程均可获取接收报文描述符，向主机内存 DMA 报文数据。为消除多个接收报文处理线程获取报文描述符的同步开销，系统设置了多个接收报文描述符环。系统只使用 XLS416 的一个线程处理发送报文，因此只设置了一个发送报文描述符环。报文描述符的设计如图 5.13 所示，其中 state 表示报文描述符中是否包含需要发送或接收到的报文，pad 用于数据补齐，len 字段为报文数据长度，address 为报文数据在主机内存中的物理地址。

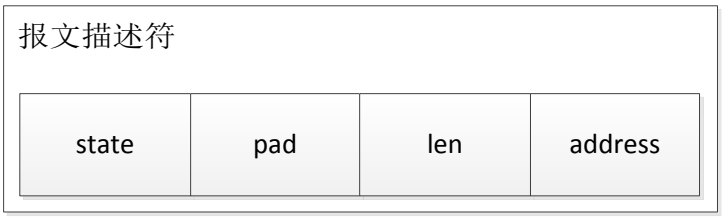

图 5.13 报文描述符

设备信息包含网络接口的 MAC 地址、接口连接状态、混合模式标志位以及使能中断标志位等信息。计数器包含网络接口发送报文数量、接收报文数量、丢弃报文数量等信息。

5.2.2 XLS416 存储空间分配

系统初始化时，设置 TLB（Translation Lookaside Buffer），建立内存物理地址和虚拟地址之间的映射，之后系统软件将使用虚拟地址内存。XLS416 的存储空间被分为三大部分：（1）系统保留，存放运行时堆栈等；（2）网络接口缓存报文；（3）存放各线程维护的数据结构。如图 5.14 所示。

图 5.14 XLS416 存储空间分配

系统中超时扫描线程在运行时直接访问接收报文处理线程分配的ConnectionDescriptor，自身不需要维护数据结构。

系统中发送报文处理线程需要预先分配发送报文数据缓冲区，以及维护包含报文信息的 PacketInfo 数据结构，如图 5.15 所示。

图 5.15 发送报文处理线程内存分配

其中 PacketInfo 包含的报文信息有：报文数据指针，数据长度，以及将各 PacketInfo 链接起来的双链表头结点。

系统分配多个接收报文处理线程，每个接收报文处理线程维护的数据结构有：（1）报文信息 PacketInfo；（2）用于快速查找报文对应 ConnectionDescriptor 的 ConnectionTable；（3）描述报文对应 TCP 连接的 ConnectionDescriptor。每个接收报文处理线程维护的数据结构分布在物理地址不重叠的内存中，如图 5.16 所示。

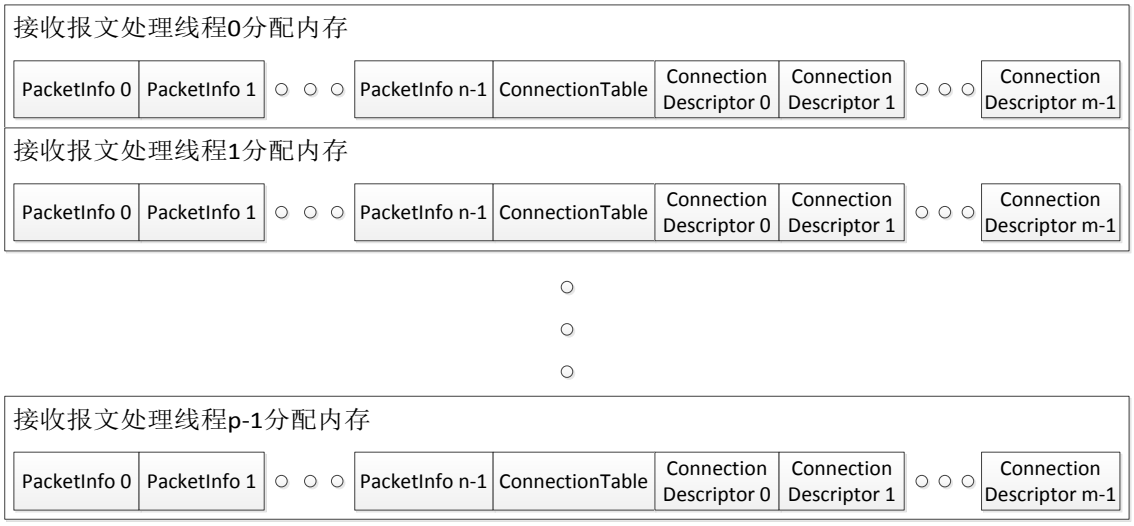

图 5.16 接收报文处理线程内存分配

系统初始化时为各线程所使用的数据结构预先分配内存，各线程运行时直接根据地址信息访问数据结构，而非在堆栈中动态申请内存。这种方式避免了大量内存管理开销，简单高效。

5.2.3 XLS416 线程分配

XLS416 拥有 4 各处理核心，每个处理核心有 4 各线程。系统将 XLS416 的 16 各线程分为四类：1 个保留，执行 XLS416 shell 程序；1 个发送报文处理线程；1 个超时扫描线程；13 个接收报文处理线程。系统着重于 TCP 数据接收路径上的优化，因此只分配 1 个发送报文处理线程，而尽量多地分配了接收报文处理线程。

XLS416 具有的硬件特性（万兆网口、多线程处理核心、DRAM、PCIE 总线以及网络加速器等），Broadcom 公司其他型号和其他厂商的多核 NPU 基本都具有，只是性能配置的不同（如万兆网口个数、处理核心个数、DRAM 性能和大小、PCIE 性能等）；另外，本文管理 TCP 连接时使用的两种数据结构 ConnectionTable，ConnectionDescriptor 以及 TCP 乱序报文重组算法可以方便的实现在其他多核 NPU 中；在其他多核 NPU 上实现本系统时，线程划分也可参考 XLS416：1 个发送报文处理线程，1 个超时扫描线程，在条件允许时分配尽量多的接收报文处理线程。

5.3 系统测试与评估

5.3.1 实验设置

系统着重优化 TCP 的数据接收路径，因此本文选取 TCP 接收数据吞吐量作为性能评价指标。为更好地反映实际应用场景，本文编写了简单的测试程序，分为服务器端和客户端；客户端和服务器端之间建立多条 TCP 连接，在所有连接建立后，客户端开始向服务器端发送数据，服务器端记录从所有 TCP 连接建立到所有数据接收完成所花费的时间，并计算数据接收吞吐量。TCP 发送方的数据发送速度将影响接收方的数据接收速度，而主动 ACK 机制可以加快发送方的数据发送速度；本文所提出的技术适合具有大量数据传输的应用场景，为了更真实地反映实际应用场景，本文不限制测试程序客户端所发出的报文长度，对客户端发出的报文顺序不做更改，使其以尽可能快的速度进行数据发送。测试环境如图 5.17 所示，客户端主机的配置为：Intel Xeon E5566 2.13GHz CPU，32GB 内存，Intel 82599 万兆网卡；服务器端主机的配置为：Intel Core i3 530 2.93GHz CPU，4GB 内存，XLS416 作为网卡；两台主机通过 10Gbps 光纤直接连接。

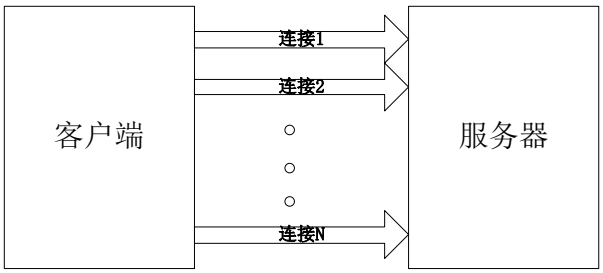

图 5.17 测试网络拓扑图

客户端采用多进程，每个进程与服务器端建立连接后，等待其他进程与服务

器建立连接，在所有连接都建立后以最快速度向服务器传输一定数量数据，所有数据发送完成后断开 TCP 连接。其流程如图 5.18 所示。

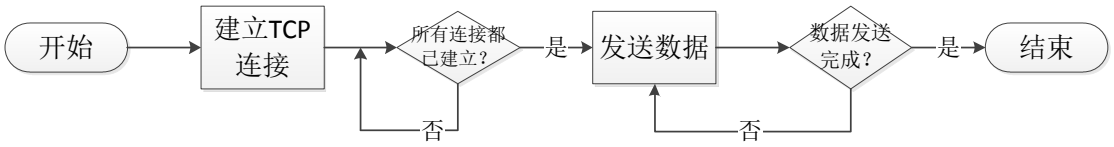

图 5.18 测试客户端流程图

服务器主进程的工作流程为：等待客户端 TCP 连接建立请求，为每个连接创建子进程用于数据接收；在所有连接建立后设置接收数据起始时刻，然后等待所有子进程结束，之后计算接收数据吞吐量，如图 5.19 所示。

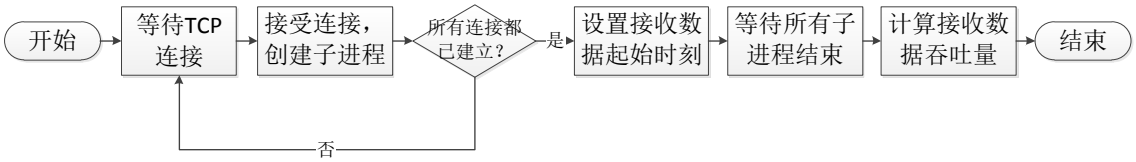

图 5.19 服务器主进程流程图

服务器子进程的工作流程为：连接建立后等待接收数据，在所有数据接收完成后设定共享内存中分配的接收数据完成时刻变量，如图 5.20 所示。

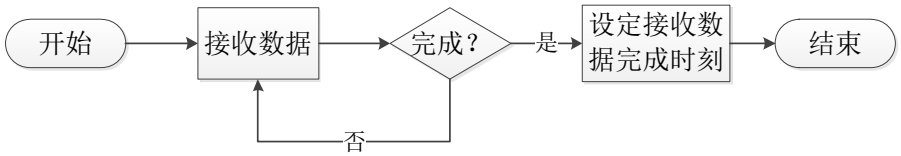

图 5.20 服务器子进程流程图

5.3.2 性能测试和评估

实验中将客户端和服务器端之间的 TCP 连接数量从 1 增加至 64，分别测试服务器端接收数据吞吐量，测试结果如图 5.21 所示。

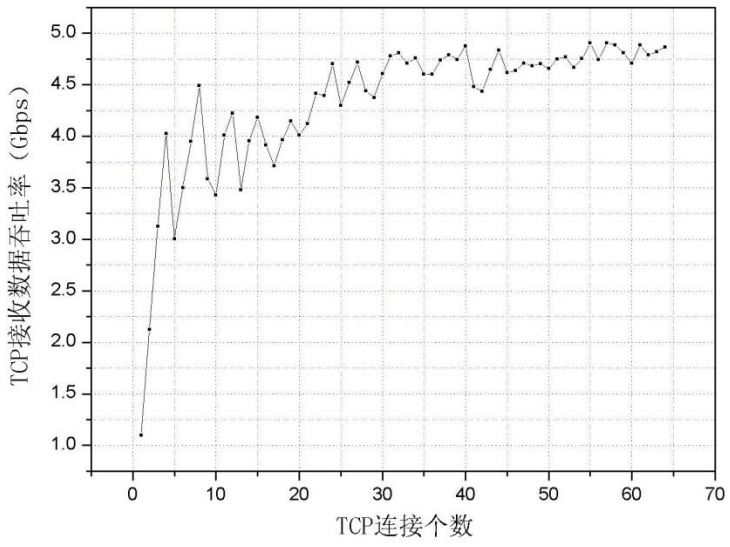

图 5.21 TCP 接收数据吞吐量测试结果

从图 5.21 中可以看出，当 TCP 连接数量较小时，TCP 接收数据吞吐量随着连接数量的增大而增大；然而随着连接数量继续增大，TCP 接收数据吞吐量增幅减小并最终稳定在 4.8Gbps 左右。这是因为，系统分配 XLS416 的接收报文处理线程数量有限（13 个），当 TCP 连接数量较小时，XLS416 不能将所有 TCP 连接的报文平均分发到 13 个接收报文处理线程，使得有些线程繁忙，有些线程空闲，XLS416 的处理能力并未达到饱和；然而当 TCP 连接增加到一定数量后，报文被更平均的分发到 13 个接收报文处理线程，所有线程都繁忙，XLS416 的处理能力也达到饱和状态，从而获得最大的 TCP 接收数据吞吐量。

本文同时实现了 XLS416 作为普通网卡的功能，在一个 TCP 连接下，不重组 TCP 报文获得 813.3Mbps 的数据接收吞吐量，而重组报文后获得 1256.5Mbps 的数据接收吞吐量。被重组报文数量越多，越能减少主机协议栈处理的报文数量和网卡产生的中断数量，越能提升 TCP 的性能。本文选择 64KB 作为重组报文的大小，为 Linux 内核协议栈的数据结构 skbuff 所能存储报文大小的上界。另外，将多个报文重组后再提交内核协议栈处理会引起反馈延迟的现象，因此本文适合具有大量数据传输的应用场景（如文件服务器等），而不适合延时敏感的应用场景（如在线游戏等）。

本文使用的 XLS416 通过 PCIE1.1×4 总线和主机连接，在简单的抓包测试中取得 6Gbps 的吞吐量，PCIE 总线的速率应当是本文 TCP 接收数据吞吐量的主要瓶颈；此外，增加多核 NPU 的线程数量也可增加系统的整体性能。本文使用 XLS416 的网络加速器硬件检验校验和，TCP 乱序报文重组算法简单高效，报文缓存和重新构造的过程中并不进行任何报文数据复制操作，所以 XLS416 的处理核心计算性能以及内存性能应当不是系统的主要性能瓶颈。

5.4 本章小结

本章首先简单介绍了实现系统使用的 XLS416 开发平台，主要包括结构和功能特性、处理核心、快速消息网络 FMN 以及网络加速器四个方面；其次详细说明了系统实现时 XLS416 资源的分配，包括 PCIE 共享内存的分配、XLS416 存储空间的分配以及 XLS416 各处理核心线程的分配；最后，本章介绍了系统 TCP 接收数据吞吐量性能测试的实验设置、测试结果以及性能瓶颈分析。

结 束 语

TCP 加速技术可以提升端系统 TCP 性能，降低服务器集群的数量的同时降低功耗，具有很高的商业价值和环保意义。传统的 TCP 加速技术致力于端系统 TCP 处理的优化，TCP 协议处理仍由主机 CPU 执行。随着网卡硬件的计算能力不断增强，将 TCP 功能的全部或部分卸载到硬件中执行的技术应运而生。TOE 技术将整个 TCP/IP 协议栈的功能卸载到硬件中执行，极大提高端系统 TCP 性能，但其实现特别复杂，并且需要修改内核协议栈代码和相关系统调用，引起安全性问题和兼容性问题。TSO 技术将 TCP 数据发送路径上的数据分段功能卸载到网卡中执行，减少主机 CPU 开销，已经发展的非常成熟。LRO 技术工作在网卡驱动程序中，将接收到属于同一个 TCP 连接的连续数据报文合并为大报文后，交由协议栈处理，通过减少 CPU 处理的报文数量减轻 CPU 负担，达到加速 TCP 的目的；但 LRO 技术的报文合并工作仍由主机 CPU 执行，不能很大程度上减轻 CPU 的负担。

针对 TOE 和 LRO 技术的缺点，考虑到多核 NPU 优越的报文处理能力，本文首次提出使用多核 NPU 作为网卡，卸载 TCP 乱序报文重组功能和校验和计算功能，并将属于同一 TCP 连接的数据报文合并为大报文后经网卡驱动程序交由协议栈处理，减少网卡产生的中断数量和内核协议栈处理的报文数量，减少主机开销；通过增加一级存储空间，还能达到以空间换取性能的效果，实现优化 TCP 性能的目的。

针对多核 NPU 的特点，本文详细设计了系统的框架结构和功能组成，提出多接收报文描述符环、合并报文校验和计算优化、接收报文处理线程 DMA 负载均衡、以及主动 ACK 机制等系统优化技术，并在 XLS416 开发平台中实现和测试了系统，在 10Gbps 的网络环境中，系统取得了 4.9Gbps 的 TCP 接收数据吞吐量。

下一步工作将考虑使用其他配置的多核 NPU 对 TCP 的数据接收功能进行卸载，选取更多的指标（如 CPU 使用率、中断数量）对系统性能进行评价，进一步分析系统性能瓶颈并提高数据接收吞吐量。此外，将考虑利用多核 NPU 优越的报文处理性能，配合主机网络协议栈替换的方法，在多核 NPU 上实现 TOE，进一步提高端系统 TCP 的性能。

致　谢

硕士毕业论文即将完成，不禁想起这两年来的过往，心中泛起对家人师友的感激之情。

衷心感谢我的导师陈曙晖老师。陈老师担任网络所副所长职务，工作认真负责，一丝不苟，经常在周末还能看到他在办公室加班的身影，让我钦佩又自惭；于百忙之中，陈老师依旧保持很高的学术水平和热情，并且对我的毕设选题、研究方法、论文撰写等方面给予了大量帮助，让我能顺利完成本毕业论文；陈老师在生活上也给予了我很多帮助，教导我遵守作为军人的基本行为准则，让我获益匪浅。

感谢戎腾网络公司的陈建华、李翔、童江鹏等人。李翔为我提供了毕设实验所用的电脑主机、XLS416、光模块、光纤等器材；陈建华为我安排了合适的机位，并告知我各服务器的登录方式，让我迅速熟悉了公司环境；童江鹏为我提供了很多技术上的支持，包括 XLS416 的编程基础、驱动程序框架的设计等，为我耐心解答各种细节问题，让我少走了很多弯路。

感谢陈思齐、程冕、徐成成、李静、李鹏飞等师兄师姐，两年来他们在生活学习上给予了我很多帮助。

感谢五队的兄弟姐妹们，与你们一起的时光将永远难忘。感谢邱远强、赵宇翔、程力等科大直上同学，让我迅速熟悉科大和军队的基本情况；感谢运动会齐心协力项目中一起奋争夺荣誉的兄弟们；感谢班里同学们的积极配合，让我能够高效优质的完成班长工作；感谢 664 教研室的老师和同学们，在学习和各类行政事务上给了我很多帮助；感谢五队胡浩政委和宋浩队长，为我们提供了良好的学习生活环境；感谢好友刘松，让我学到了很多处事的方法；感谢交大一起来科大的兄弟姐妹们，每次与你们相聚总是如此欢乐和温暖，感谢交大。

最后，感谢我的家人，你们是我永远的归宿。感谢父母的养育之恩，三春之晖，寸草难报；感谢姐姐们，拥有你们是我最幸福和骄傲的事情；感谢姐夫，帮常年在外的我照顾父母和家人；感谢家中的小朋友们，每次拥你们入怀，看到你们的笑脸时，世界都安静了。

参考文献

[1] Chase J S, Gallatin A J, Yocum K G. End system optimizations for high-speed TCP[J]. IEEE Communications Magazine, 2001, 39(4):68-74.

[2] Braden R, Borman D, Partridge C. Computing the Internet Checksum[J]. Acm Computer Communication Review, 1989, 19(19):86-94.

[3] Mallory T, Kullberg A. Incremental updating of the Internet checksum[J]. 1990.

[4] Rijsinghani A. Computation of the internet checksum via incremental update[J]. 1994.

[5] Kleinpaste K, Steenkiste P, Zill B. Software support for outboard buffering and checksumming[C]//ACM SIGCOMM Computer Communication Review. ACM, 1995, 25(4): 87-98.

[6] Henriksson T, Persson N, Liu D. VLSI implementation of Internet checksum calculation for 10 gigabit Ethernet[J]. Proceedings of Design and Diganostics of Electronics, Cricuits and Systems, 2002: 114-121.

[7] Chu H J. Zero-copy TCP in Solaris[C]//Proceedings of the 1996 annual conference on USENIX Annual Technical Conference. Usenix Association, 1996: 21-21.

[8] Dalton C, Watson G, Banks D, et al. Afterburner (network-independent card for protocols)[J]. Network, IEEE, 1993, 7(4): 36-43.

[9] Druschel P, Peterson L L. Fbufs: A high-bandwidth cross-domain transfer facility[C]//ACM SIGOPS Operating Systems Review. ACM, 1994, 27(5): 189-202.

[10] Buzzard G, Jacobson D, Mackey M, et al. An implementation of the Hamlyn sender-managed interface architecture[J]. ACM SIGOPS Operating Systems Review, 1996, 30(si): 245-259.

[11] Rodrigues S H, Anderson T E, Culler D E. High-performance local area communication with fast sockets[C]//Proceedings of the annual conference on USENIX Annual Technical Conference. 1997: 20-20.

[12] Recio R, Culley P, Garcia D, et al. An RDMA protocol specification[R]. IETF Internet-draft draft-ietf-rddp-rdmap-03. txt (work in progress), 2005.

[13] Mogul J C. TCP Offload Is a Dumb Idea Whose Time Has Come[C]//HotOS. 2003: 25-30.

[14] Romanow A, Bailey S. An Overview of RDMA over IP[C]//Proceedings of the First International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet 2003). 2003.

[15] Erdogan O, Patel P K. Design and implementation of RDMA as a best-efforts service and providing reliability over it[R]. Technical Report, 2003. http://www. stanford. edu/~ priyank9/projects, 2003.

[16] Sol á-Sloan J M. UDP, TCP, and IP Fragmentation Analysis and Its Importance in TOE Devices[J]. 2003, 2003.

[17] Rangarajan M, Bohra A, Banerjee K, et al. TCP servers: Offloading TCP processing in internet servers. design, implementation and performance[J]. Computer Science Department, Rutgers University, 2002.

[18] Stevens W R, Wright G. TCP/IP illustrated: the implementation, vol. 2[J]. 1994.

[19] Mosberger D, Peterson L L, Bridges P G, et al. Analysis of techniques to improve protocol processing latency[J]. ACM SIGCOMM Computer Communication Review, 1996, 26(4): 73-84.

[20] Yocum K G, Anderson D C, Chase J S, et al. Balancing DMA Lantency and Bandwidth in a High-Speed Network Adapter[J]. 1997.

[21] Kaiserwerth M. The parallel protocol engine[J]. Networking, IEEE/ACM Transactions on, 1993, 1(6): 650-663.

[22] Nordqvist U, Liu D K. A comparative study of protocol processors[J]. Proc. of CCSSE, 2002.

[23] Minturn D, Regnier G, Krueger J, et al. Addressing TCP/IP Processing Challenges Using the IA and IXP Processors[J]. Intel Technology Journal, 2003, 7(4).

[24] Dong Y, Xu D, Zhang Y, et al. Optimizing network I/O virtualization with efficient interrupt coalescing and virtual receive side scaling[C]//Cluster Computing (CLUSTER), 2011 IEEE International Conference on. IEEE, 2011: 26-34.

[25] Mapp G, Pope S, Hopper A. The design and implementation of a high-speed user-space transport protocol[C]//Global Telecommunications Conference, 1997. GLOBECOM'97., IEEE. IEEE, 1997, 3: 1958-1962.

[26] Thekkath C A, Nguyen T D, Moy E, et al. Implementing network protocols at user level[J]. IEEE/ACM Transactions on Networking (TON), 1993, 1(5): 554-565.

[27] Edwards A, Muir S. Experiences implementing a high performance TCP in user-space[M]. ACM, 1995.

[28] Shivam P, Wyckoff P, Panda D. EMP: zero-copy OS-bypass NIC-driven gigabit ethernet message passing[C]//Supercomputing, ACM/IEEE 2001 Conference. IEEE, 2001: 49-49.

[29] Yeh E, Chao H, Mannem V, et al. Introduction to TCP/IP offload engine (TOE)[J]. 10 Gigabit Ethernet Alliance (10GEA), 2002.

[30] Altman E, Avrachenkov K, Barakat C. TCP in presence of bursty losses[J]. Performance evaluation, 2000, 42(2): 129-147.

[31] Henriksson T, Nordqvist U, Liu D. Embedded protocol processor for fast and efficient packet reception[C]//Computer Design: VLSI in Computers and Processors, 2002. Proceedings. 2002 IEEE International Conference on. IEEE, 2002: 414-419.

[32] Ang B S. An evaluation of an attempt at offloading TCP/IP protocol processing onto an i960RN-based iNIC[J]. Computer Systems and Technology Laboratory, HP Laboratories, 2001.

[33] Shah H V, Pu C, Madukkarumukumana R S. High performance sockets and

RPC over virtual interface (VI) architecture[M]//Network-Based Parallel Computing. Communication, Architecture, and Applications. Springer Berlin Heidelberg, 1999: 91-107.

[34] Kim J S, Kim K, Jung S I. SOVIA: a user-level sockets layer over Virtual Interface Architecture[C]//cluster. IEEE, 2001: 399.

[35] Feng W, Balaji P, Baron C, et al. Performance characterization of a 10-Gigabit Ethernet TOE[C]//High Performance Interconnects, 2005. Proceedings. 13th Symposium on. IEEE, 2005: 58-63.

[36] Advantages of a tcp/ip offload ASIC[EB/OL].[2015-06-10]. http://www.snsuk.info/news_full.php?id=14466.

[37] Ethernet storage whitepapers[EB/OL].[2015-06-10]. http://www.snia.org/forums/esf/resources/whitepapers.

[38] Chelsio Accelerates Adoption of Unified Wire Networking with 10GbE iSCSI Initiator Adapter and 10G iSCSI-FC Gateway[EB/OL].[2015-06-10]. http://www.chelsio.com/chelsio-accelerates-adoption-of-unified-wire-networking-with-10gbe-iscsi-initiator-adapter-and-10g-iscsi-fc-gateway/.

[39] Hardware documentation[EB/OL].[2015-06-10]. http://www.alacritech.com/support/legacy-accelerator/hardware-documentation/.

[40] Kim H, Rixner S. TCP offload through connection handoff[C]//ACM SIGOPS Operating Systems Review. ACM, 2006, 40(4): 279-290.

[41] Connery G W, Sherer W P, Jaszewski G, et al. Offload of TCP segmentation to a smart adapter: U.S. Patent 5,937,169[P]. 1999-8-10.

[42] Padioleau Y, Lawall J L, Muller G. Understanding collateral evolution in Linux device drivers[C]//ACM SIGOPS Operating Systems Review. ACM, 2006, 40(4): 59-71.

[43] Grossman L. Large receive offload implementation in neterion 10GbE Ethernet driver[C]//Linux Symposium. 2005: 195.

[44] Theman J B. lro: Generic Large Receive Offload for TCP traffic[J]. 2007.

[45] Hatori T, Oi H. Implementation and analysis of large receive offload in a virtualized system[J]. Proceedings of the Virtualization Performance: Analysis, Characterization, and Tools (VPACT'08), 2008.

作者在学期间取得的学术成果

[1] 李杰，陈曙晖.《基于多核 NPU 的 TCP 数据接收卸载》. 计算机工程与科学, 已录用.