

基于多核处理器的无锁零拷贝数据包转发框架

肖月振, 华 蓓

(中国科学技术大学计算机科学与技术学院, 合肥 230026)

摘 要: 为突破高速软件路由器转发路径中存储访问方面的性能瓶颈, 分析 PacketShader 和 Netmap 2 个软件路由器的转发结构和存在的问题, 设计一个基于多核处理器的零拷贝包转发框架 MapRouter。采用零拷贝技术去除包转发路径中的包拷贝, 并借助并发无锁队列设计一个适合多端口间数据包零拷贝转发的包缓冲区管理方案。将高度优化的包 I/O 驱动、包缓冲区回收机制、无锁队列实现等一系列优化措施相结合, 以提高转发速度。实验结果表明, 对于不包括 IP 路由表查找的最小转发, MapRouter 在模拟的两端口路由器上可以达到 10 Gb/s 的转发速度, 与 PacketShader 和 Netmap 相比, 其转发速度更高、CPU 利用率更低。

关键词: 多核处理器; 数据包转发; 零拷贝; 包缓冲区管理; 并发无锁队列

Lock-free and Zero-copy Framework for Packet Forwarding Based on Multi-core Processor

XIAO Yue-zhen, HUA Bei

(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China)

【Abstract】 To work out performance bottleneck relating to memory accessing in forwarding path of high-speed software router, after introducing the forwarding frameworks of two software routers——PacketShader and Netmap, and analyzing their problems, this paper presents MapRouter, a zero-copy forwarding framework based on multi-core processors. MapRouter eliminates packet copying using zero-copy technology, and solves the problem of packet buffer management among multiple ports based on currently lock-free First In First Out(FIFO). By exploiting a series of optimization techniques including highly-optimized packet I/O driver, efficient packet buffer recycling mechanism, and high-efficient lock-free FIFO queue implementation, MapRouter achieves 10 Gb/s minimal forwarding(without IP address lookup) throughput on a two-port software router, which is higher than that of PacketShader and Netmap, meanwhile it has much lower CPU utilization ratio.

【Key words】 multi-core processor; packet forwarding; zero-copy; packet buffer management; concurrent lock-free queue

DOI: 10.3969/j.issn.1000-3428.2013.12.008

1 概述

当前网络速度已经普遍达到 10 Gb/s, 主干网速度达到 40 Gb/s, 这对网络路由器产生了巨大的压力。目前高端路由器主要依靠硬件, 如 ASIC(Application Specific Integrated Circuit)获得高速度, 但 ASIC 价格昂贵, 且灵活性差。以通用硬件平台和开源软件 Linux 为基础的软件路由器具有开放的软硬件架构和良好的可编程性, 但性能较低。随着通用多核处理器的出现及迅速普及, 利用多核服务器构建高性能软件路由器已成为一个研究热点, PacketShader^[1]和 Netmap^[2-3]是该领域的典型工作。

已有研究表明, 未经优化的 Linux 内核的网络转发性能很不理想。性能瓶颈主要来自 3 个方面: 收发包操作, 包缓冲区管理(包缓冲区动态分配和释放)和包拷贝, 因此, 优化 I/O(Input/Output)驱动、预分配包缓冲区和缓冲区循环

利用、零拷贝被认为是提高性能的有效方法^[4]。PacketShader 通过预分配包缓冲区和优化 I/O 驱动消除了动态缓冲区管理和数据包收发的性能瓶颈, 但其转发过程仍然需要 3 次包拷贝。Netmap 提出一种支持零拷贝的 IP 转发框架, 通过在收、发端口之间交换包缓冲区来消除包拷贝。但文献[2]及公开的代码均未涉及多个端口之间的转发情形, 也未说明这种情况下如何有效管理包缓冲区, 而包缓冲区管理是实现零拷贝转发的核心。为了解决 Netmap 零拷贝转发包缓存管理的问题, 本文设计并实现了一个高速的零拷贝转发框架 MapRouter。

2 相关工作

PacketShader 的高速 I/O 驱动(IO-Engine)使用包缓冲区预分配来代替动态缓冲区分配。初始化时为每个接收和发送队列分配一定数量的包缓冲区, 组织在接收环(Rx Ring)

作者简介: 肖月振(1985—), 男, 硕士研究生, 主研方向: 多核计算; 华 蓓, 教授、博士生导师

收稿日期: 2012-11-02 **修回日期:** 2012-12-28 **E-mail:** xyuez@mail.ustc.edu.cn

和发送环(Tx Ring)中供网卡 DMA(Direct Memory Access)使用, 同时在内核中各申请一块内存 HugeBuf 映射给用户(图 1 中用空心箭头表示的 mmap), 驱动收发包操作就是在 Rx/Tx Ring 和 Rx/Tx HugeBuf 之间拷贝包。基于 IO-Engine 的转发过程涉及 3 次拷贝(图 1 中用实心箭头表示)。内核态收包程序把包从 Rx Ring 拷贝到 Rx HugeBuf, 转发程序把包从 Rx HugeBuf 拷贝到目的端口的 Tx HugeBuf, 发送程序把包从 Tx HugeBuf 拷贝到 Tx Ring。

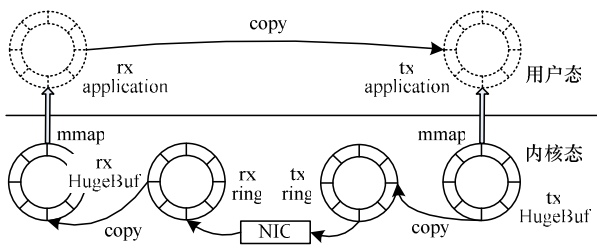


图 1 基于 IO-Engine 的转发

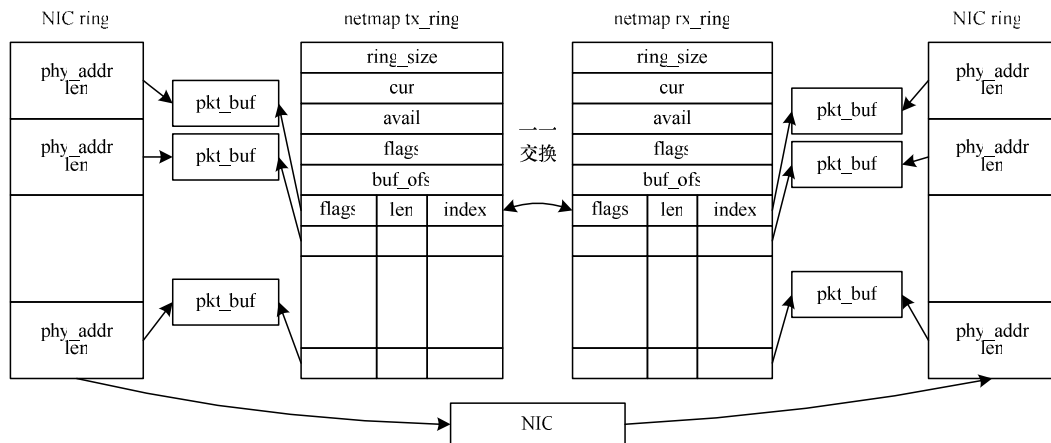


图 2 Netmap 的转发模型

当输入端口的转发程序确定了数据包的输出端口后, 转发程序从目的端口的包缓冲区队列取一个空闲缓冲区, 与输入端口中装有该数据包的缓冲区交换, 实现零拷贝转发。但文献[2]及公开的代码均只给出了数据包从一个端口固定发往另一个端口的简单情形, 没有分析以下问题并给出解决方案: (1)队头阻塞: 当输出端口没有空闲缓冲区时, 转发过程停顿, 即使后续数据包发往其他端口也不能进行。(2)netmap ring 的并发访问: 当多个输入端口向同一个输出端口发送时, 访问输出端口的 netmap ring 将产生冲突, 而基于互斥锁的同步机制在高速网络中性能很差。

3 支持无锁零拷贝转发的包缓冲区管理

本文给出一种支持无锁零拷贝转发的包缓冲区管理方案 MapRouter 以解决 Netmap 的上述问题。零拷贝转发是指在数据包转发过程中不涉及数据包体在内存中的拷贝。

MapRouter 的包分发结构如图 3 所示。每一个输入端口与每一个输出端口之间建立单独的包分发队列, 以解决队

IO-Engine 为网卡和用户(或驱动)分配不同的缓存区域, 并使用拷贝来进行数据同步。这种隔离网卡硬件和驱动的方法简化了网卡缓冲区的维护, 但拷贝会带来性能下降。实验发现, 当包长大于 256 Byte 时, 拷贝会导致至少 40% 的性能下降^[5]。

Netmap 预分配一定数量的包缓冲区供所有端口使用, 并将它们映射到用户空间以避免拷贝。每个包缓冲区由一个全局 ID 进行索引, 根据该 ID 可计算出包缓冲区的虚拟地址和物理地址。

每个端口的接收或发送队列使用 netmap ring 和 NIC ring 2 个数据结构来管理自己的包缓冲区(pkt_buf)队列, 如图 2 所示。其中, netmap ring 由用户程序和内核使用, 存放队列中 pkt_buf 的索引; NIC ring 由内核和网卡使用, 存放队列中 pkt_buf 的物理地址和包长。用户程序通过 netmap ring 来访问 pkt_buf, 驱动程序(内核)负责同步 netmap ring 和 NIC ring 中的信息。

头阻塞问题, 具体细节将在 4.2 节阐述。队列中传递的是包描述信息(包缓冲区序号和包长), 而不是完整的数据包。由于每一对输入-输出端口拥有单独的包分发队列, 因此该方案可以同时避免多个输入端口访问同一个输出端口的冲突。采用这样的包分发结构后, 每个输出端口都有多个输入队列, 每个输入队列对应一个输入端口。输出端口通过执行某种调度策略来确定发送顺序, 如轮询或公平队列等, 如图 4 所示。

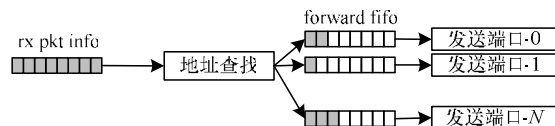


图 3 包分发队列

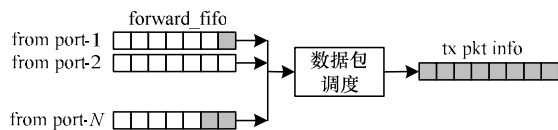


图 4 目的端口轮询多个队列发包

数据包在输入端口被分配了包缓冲区, 输出端口在将数据包发送后应将包缓冲区归还输入端口, 以实现包缓冲区的循环利用。为避免多个输出端口将释放的包缓冲区写入同一个输入端口的空闲缓冲区队列产生冲突, 为每一对输出-输入端口分配单独的回收队列, 如图5所示。回收队列中传递的是包缓冲区序号。目的端口把已经发送出去的包的缓冲区序号写入与输入端口对应的回收队列(recycle-fifo)。输入端口需要空闲包缓冲区时, 首先从空闲包缓冲区序号环(fb-idx-ring)中取, 如果 fb-idx-ring 中没有足够的空闲包缓冲区, 则会启动回收过程, 轮询所有属于当前接收端的 recycle-fifo, 从每个队列中回收批量数目的空闲包缓冲区。回收过程采用惰性回收和批量回收机制, 能够有效减少访问 recycle-fifo 的频率, 提高 recycle-fifo 的性能。

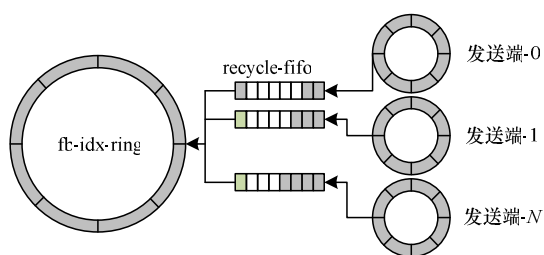


图5 包缓冲区回收

4 MapRouter 的实现

零拷贝包转发框架 MapRouter 包括3个部分: 优化的 I/O 驱动, 包转发和包缓冲区回收, 如图6所示。

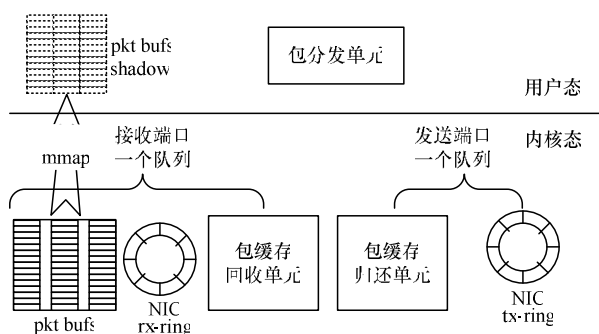


图6 MapRouter 的架构示意图

包分发单元主要完成包的零拷贝分发, 分发单元以外的部分是 MapRouter 的 I/O 驱动, 包缓存归还和回收功能单元主要完成包缓冲区的回收和循环利用。

4.1 优化的包 I/O 驱动——MapIO

本文设计的包 I/O 驱动(MapIO)是在 IO-Engine-0.2 版^[6]的基础上修改而成的, 能够支持网卡的多队列操作, 且不同的队列可以绑定到不同的 CPU 核上处理。为实现零拷贝收发包, MapIO 增加了包缓冲区回收结构, 如图6所示。同时, 由于不需要进行数据包拷贝, 发送端不再需要单独的包缓存, MapIO 去除了发送端的包缓冲区分配。

MapIO 在初始化时预分配网卡的接收/发送队列(NIC rx/tx-ring)、每个接收队列的包缓冲区池(pkt-bufs)以及包缓

冲区回收所需的数据结构, 系统运行过程中不再进行资源分配。包缓冲区用来缓存网卡从网络接收的包, 并通过内存映射(mmap)技术映射到用户空间, 消除数据包在内核和用户空间之间的拷贝。NIC rx/tx-ring 存储网卡 DMA 引擎读/写包缓冲区需要的包描述符和状态等信息, 是网卡和 CPU 的通信接口。由于网卡 DMA 需要物理地址连续的内存空间, 而向内核一次申请很大(大于 2 MB)的物理地址连续空间成功率很低, MapRouter 分几次申请长度较小的物理地址连续空间(图6中非连续的 pkt bufs 区域), 然后使用内存映射技术映射到用户空间中虚拟地址连续的一块区域(pkt-bufs-shadow)。

MapIO 的收发包过程由用户程序控制, 如图7所示。用户程序通过系统调用主动轮询 NIC rx-ring, 系统批量返回包描述信息。包描述信息(pkt info)如图7所示, 由长度(len)和包缓冲区序号(pkt-buf-idx)组成, 其中, pkt-buf-idx 由端口序号(ifidx)、队列序号(qidx)和包缓冲区偏移序号(bidx)组成, 分别表示包缓冲区所属的接收端口、端口上的接收队列及在队列中的偏移量。为避免轮询引起的活锁, MapIO 采用中断辅助轮询的方式。当 NIC rx-ring 中没有足够数量的包时, 收包过程进入等待队列自我挂起, 并开启中断模式; 当有数据包到达时, 网卡产生中断, 唤醒收包过程。该收包机制与 Linux-NAPI^[7](New API)机制非常类似, 不仅适用于高速网络, 同样也适用于中低速网络。

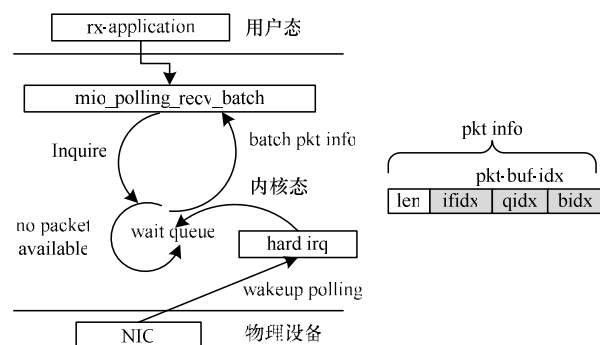


图7 MapIO 的收包过程以及包描述信息格式

MapIO 的发包过程也采用批量发送机制, 从分发队列一次取一批包描述信息。发包驱动首先检查当前发送队列上可用的描述符数目, 如果当前可用的描述符(descriptor)数目大于要发送的包数, 则发送所有的包; 反之, 发送允许数目的包; 发送结束时返回发送的包数。发包的具体操作是: 首先对用户传递的包描述信息中的 len 和 pkt-buf-idx 进行合法性检查, 有效避免了用户传送非法的包描述信息对系统带来的安全隐患。然后根据 pkt-buf-idx 计算包的物理地址, 并使用其初始化 NIC tx-ring 上的包描述符。

收发包过程涉及2类主要数据结果: NIC rx/tx-ring 和 pkt-bufs/pkt-bufs-shadow。而内核态的收发包驱动只访问 NIC rx/tx-ring, 收包驱动根据 NIC rx-ring 中3个硬件回写的包状态信息构建 pkt-info 或根据空闲包缓冲区的 pkt-buf-

idx 来计算包缓冲区的物理地址,并用其初始化 NIC rx-ring 上的描述符,而发包驱动根据 pkt-info,计算 pkt-buf 的物理地址,并用其初始化 NIC tx-ring 中的描述符。内核态的驱动程序只根据 pkt-buf-idx 计算 pkt-buf 的物理地址,不访问 pkt-buf,而用户态的包处理函数需要访问 pkt-buf,因此内核态和用户态不存在共同访问的很大空间的存储区,不会引起页表的频繁修改,从而避免了由于页表频繁修改引起的 TLB(Translation Look-aside Buffers)校验对系统性能带来的影响。

系统调用也是包处理过程中一个很大的开销。为了分摊系统调用的开销,MapIO 采用批处理技术,一次系统调用处理一个批量的包。MapIO 在其他地方也大量使用了批处理技术,包括批量传输和回写 DMA 描述符、批量读/写网卡寄存器、批量映射包缓冲区到网卡 DMA 引擎、批量探测 NIC rx-ring。批量探测 NIC rx-ring 是指一次探测 NIC rx-ring 中一个 cache 行中的包描述符,其方法是探测当前 cache 行内的最后一个描述符(如图 8 中的 desc-i+3 和 desc-i+7),如果最后一个描述符可用,则收取当前 cache 行内描述符指向的所有包,否则收包过程返回。当 NIC rx-ring 中的数据包较少时,批量探测可以减少访问 NIC rx-ring 产生的 cache 抖动(CPU 和网卡 DMA 共享 NIC rx-ring)。由于批量探测会增加包的处理延迟,因此应选取合适的批量大小(推荐 4 或 8)。

desc-i+0	desc-i+1	desc-i+2	desc-i+3	CACHE: line-j
desc-i+4	desc-i+5	desc-i+6	desc-i+7	

图 8 批量探测 NIC rx-ring

MapIO 支持网卡的多队列操作,不同队列上的操作可以绑定到不同 CPU 核上运行,并利用网卡的 RSS(Receive-Side Scaling)^[8]功能实现包的分离和分流。如果收包端开启多队列模式,由于网卡 RSS 硬件自身的问题可能会导致丢包。虽然单队列模式不存在上述收包端的问题,但该模式下发包达不到线速性能,所以在收包端使用单队列模式,而在发包端使用多队列模式。一个端口上的收包和发包操作绑定到不同的 CPU 核上运行,并且不同的端口绑定到不同的 CPU 核上,使得相同端口上的不同任务被分配到不同的 CPU 核上运行,而不同端口上任务也被分配到不同的 CPU 核上运行。

4.2 包缓冲区管理的实现

MapRouter 包缓冲区管理分为 2 个部分:驱动层面(内核态)的包缓冲区管理和应用层面(用户态)的包缓冲区管理,如图 6 所示。

驱动层面的包缓冲区管理主要完成包缓冲区的回收和循环利用,使用的数据结构是 map-rx/tx-ring、fb-idx-ring 和 recycle-fifo(图 9);应用层面的包缓冲区管理完成包的零拷贝分发,使用数据结构 forward-fifo(图 3)。

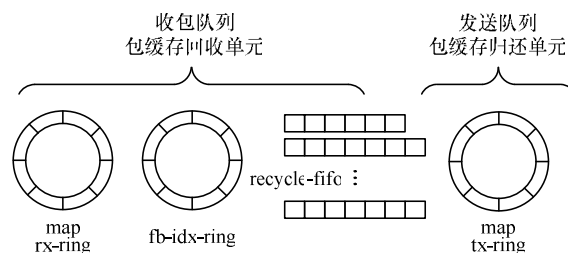


图 9 归还和回收的数据结构

由于用户空间程序只能使用包缓冲区序号,不能访问 NIC rx/tx-ring 中的信息,设计了 2 个与 NIC rx/tx-ring 对应的数据结构 map rx/tx-ring,用于存储与包缓冲区物理地址对应的包缓冲区序号。fb-idx-ring(空闲包缓冲区队列)存储从 recycle-fifo(回收队列)中回收的包缓冲区序号。由于 forward-fifo(图 3)和 recycle-fifo 对 MapRouter 的性能影响较大,采用 Lee P^[9]描述的并发无锁队列(Concurrent Lock Free First In First Out, CLF-FIFO)来实现。Lee P 中描述的 CLF-FIFO 的性能与其自身大小无关,因此把 forward-fifo 和 recycle-fifo 的大小设置成与 pkt-buf 中包缓冲数目相同,并且每一个收包端的包缓冲只在属于自己的 forward-fifo 和 recycle-fifo 上循环,从而消除了包缓冲分发和回收过程中的对头堵塞问题。

输入端口的收包程序从 NIC rx-ring 的描述符中获得包长信息,从对应的 map-rx-ring 的相应位置获得包缓冲区序号,组成包描述信息,批量返回给用户。用户态转发程序根据包缓冲区序号计算包的虚拟地址,就可以访问到数据包,在确定输出端口后将包描述信息写入与输出端口绑定的分发队列。输出端口的发包程序从分发队列中取出一批包描述信息,根据包缓冲区序号计算包的物理地址,写入 NIC tx-ring,同时将包缓冲区序号写入 map-tx-ring。

当发包过程发现 NIC tx-ring 中的空闲描述符单元不够时,启动包缓冲区归还过程,如图 9 所示。归还程序检查 NIC tx-ring 中已经完成发送的包,从对应的 map-tx-ring 中获得包缓冲区序号,根据其中的 ifidx 和 qidx 找到对应的回收队列,将包缓冲区序号写入回收队列。当收包程序需要空闲的包缓冲区时,从 fb-idx-ring 中获取包缓冲区序号,计算物理地址,写入 NIC rx-ring,并将包缓冲区序号写入 map rx-ring。如果 fb-idx-ring 中没有足够的包缓冲区,则启动回收过程,轮询与之关联的所有回收队列,从每个队列中回收批量数量的包缓冲区序号,写入 fb-idx-ring,如图 5 所示。

从上述的描述中可以看出,一个端口上的收包和分发操作被分配到单独的核上运行,而发包操作被分配到另外的核上运行,执行收包与分发过程的处理器核与执行发包过程的处理器核间通过 CLF-FIFO 进行通信,这种结构是一种二级流水线^[10]模型,不同的端口上运行的是独立的流水线,即系统运行的是多个并行执行的二级流水线,如图 10 所示。

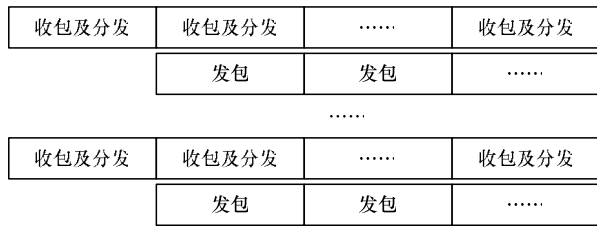


图 10 系统中的流水线模型

5 实验结果与分析

本文的实验平台为一台 Dell Power Edge R710 服务器, 内含 2 个 4 核 Intel® Xeon® E5620 处理器, 每个处理器自带容量为 4 GB、频率为 1 066 MHz 的内存。本文实验只使用了一个 4 核处理器。服务器中插一块双端口 Intel 10 Gb/s 网卡。操作系统为 Linux-2.6.39.4 x86_64 内核, 编译器为 GCC 4.5.1, 编译时使用-O2 选项。

实验比较基于 IO-Engine、Netmap 和 MapRouter 的最小转发(不包括路由查找)的性能, 同时比较它们的 CPU 利用率。实验中使用的 IO-Engine 代码为文献[7]上公布的 io-engine-0.2, Netmap 是 2012 年 6 月 8 日^[11]公布的 Linux 版本的代码。实验模拟一个双端口路由器, 数据包从一个端口输入, 并可从本端口或另一端输出(使用随机策略选择输出端口)。输入操作绑定一个处理器核, 输出操作绑定另一个处理器核。

数据包踪迹文件由本文开发的包生成器生成, 其中, 数据包的目的 IP 地址随机生成, 包长和发包速率可由发包器控制。对于任何包长, 发包器都可以提供 10 Gb/s 的发包速率。

5.1 转发性能比较

图 11 为 3 个系统在不丢包的情况下, 对于不同包长能够达到的最大转发速率(单位 Mpps), 不同包长在 10 Gb/s 链路上对应的最大包速也在图中标出(标记为理论值)。

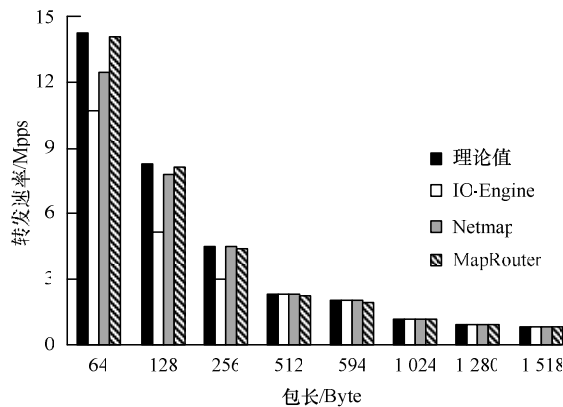


图 11 3 种平台的转发性能

对于图 11 中给出的各种包长, MapRouter 都可以实现线速转发, 而基于 IO-Engine 的转发系统在包长大于等于 512 Byte 时能够达到线速(由于包拷贝的缘故), Netmap 在

包长大于等于 256 Byte 时可以达到线速。

需要说明的是, 本文实验使用的 netmap 代码为笔者公布的 Linux 版本的代码, 且使用了推荐的参数设置, 测试结果比基于 FreeBSD 的性能^[7]有一定程度的下降。经与文献[7]作者沟通得知, 该作者没有测试 Linux 下的系统性能, 目前尚不清楚这 2 个平台上的性能差异是什么原因。

5.2 CPU 利用率比较

表 1 为这 3 个系统在到达各自的最大转发速率时的 CPU 利用率。可以看到, 在包长大于等于 128 Byte 时, MapRouter 的 CPU 利用率明显低于其他 2 个系统。

表 1 最大转发速率下的 CPU 利用率

包长/Byte	IO-Engine/(%)	Netmap/(%)	MapRouter/(%)
64	99.7	99.7	99.6
128	99.6	99.3	50.6
256	99.6	99.0	27.6
512	99.9	97.3	13.9
594	99.8	96.0	12.6
1 024	99.7	94.3	7.6
1 280	99.7	91.7	6.0
1 514	99.6	91.0	5.0

6 结束语

本文分析了基于 IO-Engine 和 Netmap 的转发模型存在的问题, 以解决零拷贝转发带来的包缓冲区管理问题, 结合高度优化的 I/O 驱动实现了一个基于多核平台的零拷贝包转发架构 MapRouter。在模拟双端口(端口速率 10 Gb/s)的路由器上, 对于不包括 IP 路由表查找的最小转发, MapRouter 可以达到 10 Gb/s 的转发速度, 且在包长较大的情况下 CPU 利用率很低。

由于 MapRouter 旁路了 Linux 内核协议栈, 其收发过程必须使用系统特定的 API(Application Program Interface)接口, 因此与基于传统协议栈的应用程序不兼容。在下一步的工作中, 将实现基于 MapIO 的 Libpcap^[12]接口, 这样基于 Libpcap 的应用程序就可以直接运行在 MapIO 上, 并获得性能提升。

参考文献

- [1] Han S J, Jang K, Park K, et al. PacketShader: A GPU-accelerated Software Router[C]//Proc. of ACM SIGCOMM'10. New York, USA: [s. n.], 2010.
- [2] Rizzo L, Matteo L. Netmap: Memory Mapped Access to Network Devices[C]//Proc of ACM SIGCOMM'11. New York, USA: [s. n.], 2011.
- [3] Rizzo L, Lettieri G. VALE, a Switched Ethernet for Virtual Machines[EB/OL]. (2010-06-08). <http://info.iet.unipi.it/~luigi/netmap/>.

(下转第 53 页)