**NetTLP：软硬件交互的PCIe设备开发平台**

**Yohei Kuga和Ryo Nakamura，东京大学；**

**庆应大学松叶武史；东京大学，柳岛由纪夫**

摘要

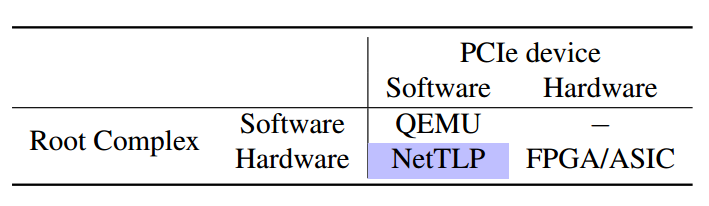
数据通信的可观测性对于通信系统的原型设计、开发和优化始终是至关重要的。然而，观察PCI-Express（PCIe）链路中的事务流仍然是一个挑战，尽管它们是新兴外围设备（如智能网卡、NVMe和加速器）的关键组件。为了在PCIe上提供实用的可观测性并有效地原型化PCIe设备，我们提出了NetTLP，一个可以与硬件根（hardware root）联合体进行交互的软件PCIe设备开发平台。

在NetTLP平台上，IP网络栈顶部的软件PCIe设备可以通过以太网链路（称为NetTLP适配器的以太网和PCIe网桥）和PCIe链路向硬件根复合体或其他设备发送和接收事务层数据包（tlp）。本文介绍了NetTLP平台及其实现：NetTLP适配器和PCIe事务层的软件实现LibTLP。此外，本文还通过三个实例展示了NetTLP的实用性：（1）观察从四个商用PCIe设备发送的tlp，（2）为硬件根复合体执行实际NIC的400loc软件以太网NIC实现，以及（3）物理内存自省。

## 1简介

PCI Express（PCIe）是一种广泛用于存储、图形、网络和加速器设备的I/O互连[16,24,25]。不限于连接外围设备，一些高性能互连采用PCIe协议[5,22,35]。此外，未来互连的规范是通过扩展PCIe协议[10,14]来设计的。PCIe的这种多功能性源于基于分组的数据通信和PCIe拓扑的灵活性。PCIe规范定义了包含PCIe拓扑的构建块：端点、交换机、网桥和根复合体。PCIe数据包通过块之间的点对点PCIe链路流动，主板制造商可以根据使用情况扩展这些块的PCIe拓扑。

表1：从软件和硬件的角度比较PCIe设备原型的平台。



与PCIe的普及相比，研究人员和软件开发人员仍然很难观察到PCIe和原型PCIe设备，尽管它们对于优化性能和开发未来的PCIe设备至关重要。观察PCIe事务很困难，因为PCIe事务被限制在硬件中。PCIe不仅仅是硬件元素之间的一个简单的fat管道；它还具有实现高性能通信的几个特性，即硬件中断、虚拟化和CPU缓存操作。利用这些特性对于有效地利用PCIe非常重要；然而，除非使用用于观察PCIe硬件的特殊捕获设备，否则无法确定PCIe链路中事务的具体行为。

除此之外，PCIe设备的原型制作还缺乏生产力。现场可编程门阵列（FPGA）是制作PCIe设备原型的主要平台[26,40,44,45,50]。然而，在FPGA上开发PCIe设备的所有部分仍然需要大量的努力，例如NetFPGA项目[52]对网络设备的巨大投入。另一种方法是采用虚拟化或模拟，例如GEM5[11,23]和RTL模拟器[4,31]。QEMU[9]是一个著名的虚拟化平台，可以从软件的角度对PCIe设备进行原型化。QEMU使研究人员和开发人员能够为新的硬件体系结构建立原型；然而，它的环境是完全软硬件化的。QEMU设备只能与模拟的根复合体通信，不能与物理根复合体和连接到根复合体的其他硬件通信。

本文的目标是弥合PCIe软件和硬件之间的差距，如表1所示。我们提出的平台名为NetTLP，提供了可与硬件根复合体交互的软硬件PCIe端点。通过使用NetTLP，研究人员和软件开发人员可以将他们的PCIe设备原型化为软件PCIe端点，并通过PCIe协议测试具有实际硬件根复合体的软件设备。这种软硬件混合平台同时提高了PCIe事务的可观察性和PCIe设备原型的生产率。

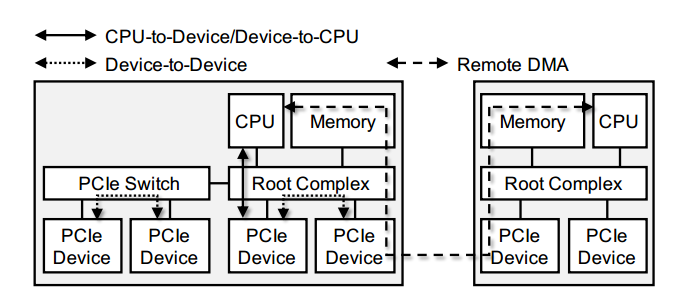
将软硬件化的PCIe端点连接到硬件根复合体的关键技术是将PCIe事务层分离为软件，并将软件事务层放在IP网络栈的顶部。我们的基于FPGA的插件称为NetTLP适配器，通过以太网和IP网络将事务层数据包（TLP）传送到远程主机。NetTLP适配器的实质是在具有LibTLP的远程主机的IP网络栈上的软件实现，它是PCIe事务层的软件实现。由适配器和库组成的NetTLP平台使IP网络栈上的软件PCIe设备能够通过NetTLP适配器与硬件根复合体交互。此外，通过以太网链路传输的tlp可以很容易地被tcpdump和Wireshark等IP网络技术观察到。

在本文中，我们描述了一个新的PCIe软件平台NetTLP。为了实现这个平台，我们从基于分组通信的角度研究了PCIe（第2章），然后描述了将软件PCIe设备与硬件根复合体连接起来并在软件中处理tlp的方法（§3），并描述了它的实现（§4）。除了微观基准测试（§5），我们还演示了NetTLP的三个用例（§6）：在TLP级别观察根复合体和商业设备的行为，在与物理根复合体交互的软件中构建以太网NIC的原型，以及使用NetTLP的物理内存自省。本文的贡献包括：

•我们提出了一种新的PCIe设备软件原型平台，软件设备可以与物理硬件（如根复合体、CPU、内存和其他PCIe设备）进行通信。此平台为具有与硬件实际交互的PCIe设备原型提供了高生产率。

•我们通过IP网络堆栈上的软硬件PCIe端点提供硬件中PCIe事务的可观察性。我们改进的tcpdump可以通过NetTLP区分以太网中封装的tlp，使我们能够以IP网络方式轻松捕获tlp。

•我们提供了根复合体和商用外围设备的PCIe事务的详细观察结果：Intel根复合体、Intel X520 10 Gbps NIC、Intel XL710 40 Gbps NIC、Intel P4500 NVMe和Samsung PM1724 NVMe。NetTLP的观察结果揭示了它们在PCIe事务上的行为差异，例如，TLP标记字段的不同使用



**图1:PCIe拓扑和三种通信模型**

•我们展示了一个不存在的带有NetTLP的以太网网卡的原型。这个原型演示了NetTLP平台的高生产率；NIC当然是在软件中实现的，但是它作为物理根复合体的实际以太网NIC来执行。

•我们展示了在NetTLP上开发内存自省方法而不使用专用设备的可能性。作为概念证明，我们实现了两个应用程序，通过NetTLP通过DMA从远程主机收集进程信息。本文描述的所有硬件和软件源代码以及捕获的数据都是公开的[1]。

## 2背景

PCIe不仅是一种互连，而且是一种基于分组的数据通信网络。与IP网络一样，PCIe具有由物理层、数据链路层和事务层组成的分层模型。数据链路层通过PCIe链路跨一个跳传送PCIe数据包，而事务层负责通过PCIe链路将tlp从PCIe端点传送到PCIe端点。PCIe互连由以下能够支持层功能的元素组成：端点、交换机、网桥和根复合体。PCIe交换机和根复合体根据内存映射I/O（MMIO）空间或请求者ID中的地址路由和转发PCIe数据包。PCIe的任何功能都位于基于分组的通信中，例如，用于硬件中断的MSI-X，都是通过对特定存储器地址的存储器写入来实现的。由于是这样一个基于分组的网络，PCIe拓扑及其通信模型是灵活的，如图1所示。

在IP网络中，我们可以很容易地原型和实现网络的任何部分，例如终端主机、交换机和路由器，并观察网络中的数据包流动；然而，尽管PCIe也是一个基于分组的网络，但PCIe无法做到这一点。PCIe最初是为计算机内部的I/O互连而设计的；因此，假设所有的PCIe元素都是在硬件中实现的。这种假设和现状给PCIe及其元件的研究和开发带来了困难。

对于研究PCIe，有两个主要平台：FPGA和QEMU。FPGA在硬件上提供可编程性，用于制作PCIe设备原型。相比之下，在FPGA上开发PCIe设备仍然需要大量的工作。即使在实现一个设备时，该设备也需要特定用途的逻辑和各种逻辑块，如PCIe内核、DMA引擎、内存控制器等。与软件库不同，此类功能块不可用。此外，我们无法观察到FPGA发送的PCIe包。使用逻辑分析仪或昂贵的专用硬件，我们只能看到部分信号。另一方面，QEMU支持在完全虚拟化环境中实现PCIe设备。然而，QEMU并不实现PCIe协议，只DMA APIs。QEMU被用作研究新的PCIe设备和讨论OS抽象和实现的平台，而不需要真正的硬件和PCIe协议。因此，QEMU-PCIe设备无法与PCIe上的实际主机和硬件交互，尽管根复合体的功能已经在不断发展。

这两种平台各有优缺点：FPGA需要大量的原型工作，缺乏可观测性，而QEMU设备无法与PCIe协议的硬件元素交互。这些缺点是因为平台只关注硬件或软件。根复合体和设备PCIe的两个主要元素是FPGA中的硬件或QEMU中的软件。

作为第三个平台，我们主张连接PCIe的软硬件元素。如果将PCIe设备移动到软件并连接到硬件根复合体，则我们可以在软件中实现高效的PCIe设备原型，并与连接到硬件根复合体的实际PCIe元素进行交互。此外，我们可以在软件PCIe设备上观察PCIe事务，而无需求助于专用硬件机制。这种关系类似于IP网络；终端主机上的IP网络堆栈是软件，而路由器和交换机是硬件。

## 3 NetTLP

为了切实地连接软件PCIe设备和硬件根复合体，我们建议将PCIe的事务层分离为软件，如图2所示。事务层负责端到端PCIe通信的基本部分：标识符，即内存地址和请求者id、路由和发出PCIe事务。软件化事务层提供了在该层之上的软件中的PCIe设备原型的高生产率，以及软件对PCIe事务的可观察性。

为了连接软硬件事务层和硬件数据链路层，NetTLP选择了一种连接PCIe链路和以太网链路的配置。因为PCIe和以太网是基于分组的网络，所以可以通过封装在以太网链路上传送tlp。一旦TLP进入以太网网络，我们就可以很容易地观察到TLP类似IP的数据包，在软件中实现事务层，并在软件IP网络堆栈上构建原型PCIe设备。

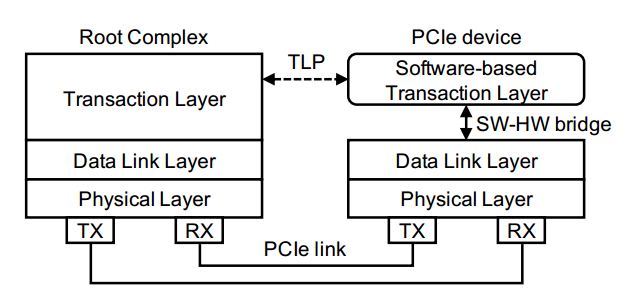


图2:PCIe的分层模型和我们将事务层分离为软件的方法。

与NetTLP一样，ExpEther[47,48]和Thunderclap[29]也可以观察和操纵TLP。ExpEther通过在以太网链路上传输TLP来扩展PCIe链路。ExpEther封装的tlp可以在ExpEther适配器和安装外围设备的硬件扩展盒之间的以太网链路上观察到。在Thunderclap中，运行在FPGA上ARM CPU上的Linux用软件处理tlp。类似地，一些智能NIC可以使用抽象的DMA API从NIC上的CPU发送和接收TLP[30,33]。

与现有技术相比，NetTLP专注于在软件中原型化新的PCIe设备。为此，使用软件操作TLP是基本功能之一。此外，设备和cpu之间的交互必须灵活设计。ExpEther没有关注这一点，因此它扩展了以太网上的PCIe链路，Thunderclap上的PCIe设备软件假装现有设备，通过它们的驱动程序暴露漏洞。在NetTLP平台中，研究人员和开发人员可以设计新的软件设备如何通过根复合体与cpu交互。更具体地说，可以在NetTLP平台上从头开始设计和实现软件设备（例如描述符环）的寄存器的使用。此功能允许设计和实现不存在的设备，并在软件中观察其交互作用（第6.2节）。

### 3.1平台概述

NetTLP通过以太网传输tlp的关键组件是一个NetTLP适配器，它是一个基于FPGA的插件，配备有连接到主机的PCIe链路和以太网链路。另一个关键组件是LibTLP，它是IP网络栈上PCIe事务层的软件库。NetTLP平台由两个主机组成，一个是具有NetTLP适配器的适配器主机，另一个是运行基于LibTLP的应用程序的设备主机，如图3所示。

NetTLP适配器负责在图2中所示的硬件数据链路层和软件事务层之间建立桥梁。NetTLP适配器在主机的PCIe链路和以太网链路之间提供TLP。当NetTLP适配器从PCIe链路接收TLP时，NetTLP适配器将每个TLP封装在以太网、IP、UDP和NetTLP报头中以进行排序和时间戳，并通过以太网链路将数据包发送到设备主机。当NetTLP适配器从以太网链路接收到UDP数据包时，NetTLP适配器将检查数据包的有效负载是否为TLP，对数据包进行解封，并将内部TLP发送到PCIe链路。因此，从适配器主机的角度来看，软件从设备主机发送的所有TLP都被识别为NetTLP适配器生成的TLP。

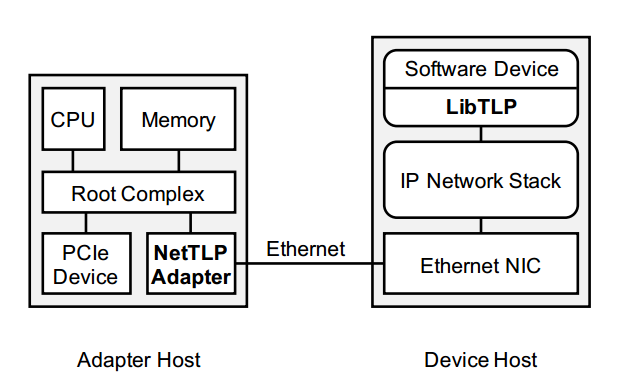


图3:NetTLP平台概述。

LibTLP在软件中实现PCIe事务层，并为应用程序提供抽象的dmaapi。设备主机上的应用程序可以通过UDP套接字向适配器主机上的NetTLP适配器发送和接收tlp。通过使用LibTLP，研究人员和软件开发人员可以在软件中实现他们自己的PCIe设备，这些设备在适配器主机上执行NetTLP适配器的实际行为。此外，分离远程主机上的软件PCIe设备和物理适配器使我们能够观察流经以太网链路的实际PCIe事务。我们可以在设备主机上通过tcpdump捕获封装的tlp，也可以通过以太网交换机上的光抽头或端口镜像来捕获以太网链路上的tlp。

尽管NetTLP适配器是单个外围设备，但NetTLP适配器可以应用于图1中组织的PCIe拓扑中的一些PCIe通信模型。当然，NetTLP适配器和软件PCIe设备可以成为CPU到设备和设备到CPU通信上的设备。将NetTLP适配器应用于设备到设备的通信，也称为对等DMA，实现了商用PCIe设备和软件PCIe设备之间的交互。第6.1节介绍了NetTLP平台和点对点DMA集成从产品设备发送的TLP。此外，NetTLP适配器可以被视为原始远程内存访问设备。设备主机上的应用程序可以通过NetTLP适配器向适配器主机上内存的任何地址发出DMA。第6.3节演示了利用NetTLP远程内存访问的内存内省方法。

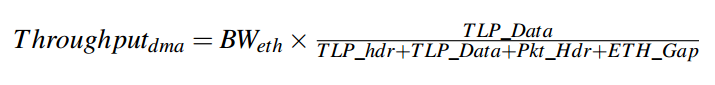
### 3.2软件中的TLP处理

在软件中处理PCIe事务具有挑战性，因为PCIe最初设计为通过硬件处理。本节介绍设计NetTLP以实现硬件和软件之间的PCIe交互的三个问题。接收突发tlp：第一个问题是LibTLP需要接收从硬件发送的突发tlp。例如，当TLP是带有32位地址字段的memoryread（MRd）TLP时，最小TLP长度为12个字节。NetTLP适配器使用以太网、IP、UDP和NetTLP报头封装tlp；因此，封装的最小数据包长度为64字节。这个长度与IP网络的最小数据包大小相同。同时，PCIe的流量控制基于信用体系[7]，只要信用保持不变，PCIe端点就可以连续发送tlp。特别是，PCIe设备通常使用TLP标签字段发送小TLP以获得高性能[21]。由于这些TLP传输间隔是连续的时钟单位，封装的TLP的吞吐量可以在具有短分组的以太网链路上瞬时为线速率。

为了通过软件接收这种突发TLP，NetTLP利用TLP标记字段将接收封装的TLP分发到以太网NIC和CPU核心的多个硬件队列中。标记字段用于区分可以独立处理的单个未过帐交易。在封装tlp时，NetTLP适配器将标签值的低位4位嵌入到UDP端口号的低位4位中。因此，到NetTLP适配器的PCIe事务通过基于标记字段的不同UDP流进行传递，并且设备主机可以通过不同的NIC队列接收流。这种技术称为基于标记的UDP端口分发，使软件端能够在与多个NIC队列相关联的多个核心上处理tlp。

完成超时：另一个问题是完成超时。根据PCIe规范，根复合体和PCIe端点支持PCIe事务层上的完成超时机制。当请求者发送内存读取请求时，请求者为每个请求设置超时时间段，完成者需要在这些时间段内发送完成。因此，NetTLP平台上的软件PCIe设备需要能够在硬件级别的超时时间内发送完成。超时时间在PCIe配置空间中配置。例如，X520 NIC的最小和最大完成超时时间分别为50微秒和50毫秒[6]。因此，构建在LibTLP上的软件PCIe设备也必须能够在这些时间段内响应内存请求。幸运的是，一般服务器机器上的Linux网络栈在毫秒级上不会太慢；因此，我们期望LibTLP能够满足需求。第5节通过一个延迟基准测试来研究这个问题。

封装开销：为了将tlp带到IP网络栈顶上的软件，NetTLP用IP报头封装tlp，尽管由于报头开销，封装会降低吞吐量。NetTLP将tlp封装成多个报头：14字节以太网、4字节FCS、20字节IP、8字节UDP和6字节NetTLP报头。NetTLP平台上通过DMA传输数据的吞吐量可以用以下公式计算：



## *BWeth是以太网链路速度，对于上述头和FCS，PktHdr为52字节，对于前导码和帧间间隙，ETH-Gap为20字节。根据这个公式，对于10gbps链路上的3DW内存写TLP，256字节TLP数据（通常的最大有效负载大小）和12字节TLP\_hdr的吞吐量大约为7.53gbps，这是10gbps以太网的理论限制。尽管根据使用情况需要吞吐量，但对于仅原型化软件PCIe设备来说，开销并不重要。*

## 4实施

在LibTLP和linuxnettlp卡上实现了该系统。本节描述NetTLP适配器、LibTLP提供的api、硬件中断和NetTLP平台的限制的详细信息。

### 4.1 NetTLP适配器

NetTLP适配器是使用Xilinx KC705 FPGA开发板实现的[51]。该板具有Xilinx Kintex 7 FPGA、以太网10 Gbps端口和PCIe Gen 2 4通道链路。我们使用该板是因为它的PCIe端点IP核支持用户定义的逻辑来处理原始TLP报头。此特性适用于设计NetTLP适配器。但是，Xilinx较新的PCIe IP核心支持

PCIe Gen 3不允许用户定义的逻辑处理原始数据

TLP标题。因此，NetTLP适配器的当前实现不支持PCIe Gen 3。

图4显示了NetTLP适配器的电路图的概述。当前的NetTLP适配器有三个用于不同角色的基址寄存器（BAR）空间。BAR0用于配置NetTLP适配器。通过BAR0的配置支持更改源和目标MAC地址以及用于封装TLP的源和目标IP地址。BAR2空间用于MSI-X表，以支持来自软件PCIe设备的硬件中断。NetTLP中MSI-X的细节在第4.3节中描述。

BAR0和BAR2的内存空间都是在FPGA上用块RAM实现的，NetTLP适配器在BAR0和BAR2之上有一个外围I/O（PIO）引擎，以完成对BARs的操作。

BAR4不同于BAR0和BAR2；BAR4空间连接到以太网PHY，而不连接到PIO引擎。从根复合体或其他设备到

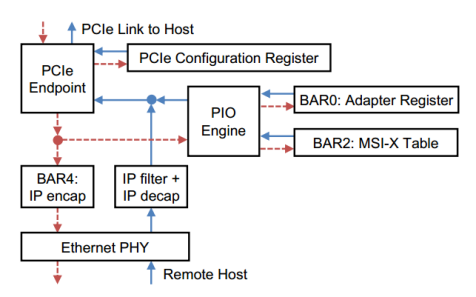


图4:NetTLP适配器的电路图。

BAR4空间封装在以太网、IP、UDP和NetTLP报头中，并通过以太网链路传输到外部主机。也就是说，设备主机上的LibTLP通过分配给BAR4的内存区域与适配器主机上的根复合体通信。

将TLP封装到BAR4时，UDP报头的源端口号和目标端口号是根据其TLP报头的tag字段生成的。这是第3.2节中描述的基于标记的UDP端口分布。在当前的实现中，UDP端口号是用0x3000+（\u标记^0x0F）生成的。因此，设备上的网卡当NetTLP适配器从设备主机接收UDP包时，IP过滤器逻辑检查IP地址是否与BAR0上配置的地址匹配。如果IP地址和端口号正确，则数据包将被解除封装，内部TLP将通过PCIe链路发送到主机。

NetTLP适配器的驱动程序取决于软件PCIe设备的类型。如果软件PCIe设备是以太网NIC，则驱动程序适用于以太网NIC；如果软件PCIe设备是NVMe SSD，则驱动程序适用于NVMe SSD。不管驱动程序类型如何，我们都实现了一个简单的驱动程序，它支持NetTLP适配器的基本功能。这个驱动程序启用NetTLP适配器，初始化MSI-X，并准备一个简单的消息传递API。设备主机上的软件PCIe设备可以获得关于NetTLP适配器的信息，即适配器的条空间的地址、PCIe总线和插槽号以及MSI-X表。NetTLP平台的用户可以通过扩展基本NetTLP驱动程序为其软件PCIe设备开发驱动程序。

### 4.2 LibTLP

LibTLP是一个实现PCIe事务层的用户空间库。在事务层实现之上，LibTLP提供了一个经过良好抽象的dmaapi和一个用于处理每种类型tlp的回调API。

图5显示了LibTLP的dmaapi。LibTLP实例包含一个套接字描述符、一个标记值和一个目标NetTLP适配器的目标地址

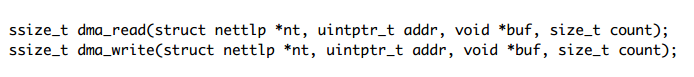


图5: The DMA API of LibTLP。

由nettlp\_init（）初始化的nettlp结构。DMA读取和DMA写入的dmaapi是通过指定nettlp结构来调用的。与read（）和write（）系统调用一样，dma\_read（）尝试将count个字节读入buf，dma\_write（）从buf写入count个字节。addr表示DMA事务的目标地址。函数的返回值是读取或写入的字节数，或者在出错时设置-1和errno。对于DMA读取，应用程序可以通过返回值和errno来通知TLP丢失或完成错误。

除了向适配器主机上的内存发出DMA的DMA API之外，LibTLP还提供了一个回调API，用于处理从适配器主机到设备主机的tlp。回调API允许应用程序注册主要TLP类型的函数：内存读取（MRd）、内存写入（MWr）、完成（Cpl）和数据完成（CplD）。当nettlp结构的套接字接收tlp时，将为tlp调用已注册的函数。通过使用这个API，设备主机上的应用程序可以通过发送相关的CplD tlp来响应从根复合体到NetTLP适配器上的BAR4的MRd tlp。

### 硬件中断

对于硬件中断，NetTLP平台支持被现代PCIe设备广泛使用的MSIX。MSI-X中断是通过从设备向指定地址发送带有特定数据的MWr TLP来调用的。中断的地址和数据存储在由设备的PCIe配置空间指定的BAR空间上的MSI-X表中。换言之，要通过MSI-X发送硬件中断，必须参考栏上的MSI-X表。

要在NetTLP平台上实现MSI-X，有两种方法：（1）将MSI-X表放置在BAR4上，设备主机上的软件PCIe设备保存MSI-X表；（2）将MSI-X表放置在PIO引擎下的其他条空间上，设备主机上的软件PCIe设备通过其他通信路径获取MSI-X表。当前的实现选择后一种方法。前一种方法不需要任何其他通信路径来从适配器主机获取MSI-X表。但是，MSI-X表由NetTLP驱动程序初始化，因此软件PCIe设备必须在设备主机上运行，然后才能将NetTLP驱动程序加载到适配器主机上。此外，软件PCIe设备0实现必须始终能够维护MSI-X表，即使它们不使用MSI-X。这些特性可能会给软件PCIe设备的开发带来不便。出于这些原因，我们将MSI-X表放在BAR2上的PIO引擎下，该引擎仅由NetTLP适配器的硬件逻辑控制。软件PCIe设备可以通过基本NetTLP驱动程序提供的简单消息传递API获取MSIX表的内容。

### 4.4 tcpdump and Wireshark

为了观察tlp，我们稍微修改了tcpdump并实现了一个Wireshark插件。在NetTLP平台中，封装的tlp通过NetTLP适配器和设备主机之间的以太网链路流动，因此可以通过IP网络的监视工具轻松地捕获tlp。改进的tcpdump可以识别封装的tlp，并在流行的tcpdump输出上显示tlp的内容。Wireshark插件还在GUI上显示tlp的内容。这些工具为研究人员和开发人员提供了一种方便的观察TLP的方法。

### 4.5限制

NetTLP平台的当前实现无法处理PCIe配置空间。PCIe配置空间管理PCIe设备的属性，如设备ID、供应商ID和条形空间的地址区域。PCIe配置空间存储在PCIe设备硬件的内存中。当主机启动或重新扫描PCIe设备时，设备使用CfgRd和CfgWr tlp与根复合体通信以设置其PCIe配置。在NetTLP适配器的当前实现中，Kintex 7 FPGA的PCIe端点IP核管理PCIe配置空间，如图4所示。IP核心不允许用户定义的逻辑通过原始TLP操作配置寄存器。因此，软件PCIe设备无法查看和更改其PCIe配置。因此，当前实现不支持需要操纵PCIe配置寄存器的功能，即改变MSI-X表和SR-IOV的结构。

## 5个微观基准

为了评估软件PCIe设备和应用程序的性能，我们在NetTLP平台上对DMAs的吞吐量和延迟进行了微基准测试。

在NetTLP平台中，PCIe事务有两个方向：（1）从LibTLP到NetTLP适配器；（2）从NetTLP适配器到LibTLP。它们分别表示从软件PCIe设备到根复合体的DMA读写，以及从根复合体到软件PCIe设备的DMA读写。在前一个方向，我们假设从软件PCIe设备发出的PCIe事务可以在不丢失数据包的情况下进行处理，因为适配器主机上的所有组件都是硬件，比10 Gbps以太网链路具有更高的带宽（NetTLP适配器的16 Gbps PCIe Gen 2 4通道链路）。相反，从根复合体到软件PCIe设备的DMA读取也不会被丢弃，因为根复合体在接收到最后一个读取请求（未发布事务）的完成之前不会发送内存读取请求。基于这个假设，我们测量了LibTLP的DMA读写吞吐量（第5.1节），以及NetTLP适配器的DMA读吞吐量（第5.2节）。

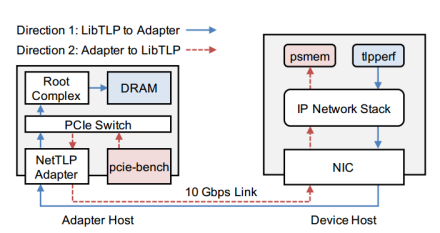
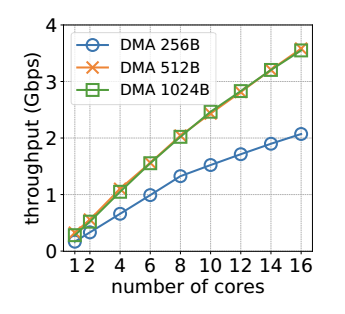
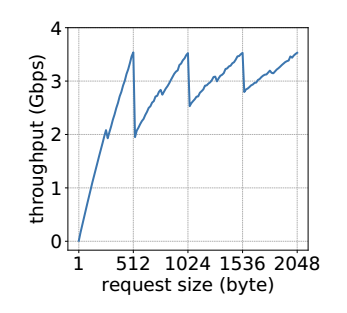


图6：基准设置。

相比之下，无法测量从根复合体到软件PCIe设备的DMA写入吞吐量。根复合体可以以NetTLP适配器的链接速度发送MWr tlp，而无需明确的确认（已发布的事务）。当前的NetTLP适配器没有机制通知到根复合体的以太网链路上的拥塞；因此，如果NetTLP适配器的10Gbps以太网链路溢出，mwrtlp将被丢弃。将溢出通知到根复合体和其他设备是未来的工作。但是，对于最近的外围设备（如以太网NIC和NVMe SSD），向PCIe设备写入内存的通常使用情况是从CPU更新设备上的寄存器，而这些并不需要显著的吞吐量。因此，我们认为当前的NetTLP适配器足以在软件中构建PCIe设备的原型。。

**图7：从LibTLP到NetTLP适配器的 图8:LibTLP到NetTLP适配器的**

**DMA读吞吐量与CPU核数的关系。 DMA读吞吐量与请求大小的关系**。

图6描述了我们用于为基准生成PCIe事务的两个方向和组件。为了生成从LibTLP到NetTLP适配器的PCIe事务，我们开发了一个基于LibTLP的基准应用程序tlpperf。用户可以使用tlpperf从设备主机通过NetTLP适配器向适配器主机上的内存发送内存读写请求。为了生成相反方向的PCIe事务，我们实现了一个基于LibTLP的伪内存设备，称为psmem，并对PCIe工作台稍作修改[34]。设备主机上的psmem假装一个与NetTLP适配器的BAR4关联的内存区域。如第4.1节所述，到NetTLP适配器的BAR4空间的tlp被传输到设备主机。当psmem接收到MWr TLP时，psmem保存数据和关联地址。当psmem接收到MRd TLP时，psmem发送CplD TLP，其中包含与请求地址相关联的正确数据。此外，为了生成对NetTLP适配器BAR4的内存请求，我们修改了NetFPGA SUME的pcie工作台实现。修改后的pcie工作台可以使用BAR4空间作为基准目标，而不是主内存。

对于微基准测试，我们为适配器和设备主机准备了两台机器。适配器主机是Intel Core i9-9820X 10核CPU和32 GB DDR4内存，带有华硕WS X299 SAGE主板。此主板有PCIe交换机。NetTLP适配器和用于pcie工作台的NetFPGASUME卡安装在同一个pcie交换机下的pcie插槽上。设备主机是一个Intel Core i97940X 12核CPU、32 GB DDR4内存和Intel X520 10 Gbps NIC和华硕PRIME X299-A主板。设备主机通过10 Gbps以太网链路连接到适配器主机上的NetTLP适配器。操作系统是Linux内核4.20.2。请注意，我们在具有12个物理核心的设备主机上启用了超线程，以便通过基于标记的UDP端口分布充分利用16个NIC队列。

在本节描述的实验中，所有的吞吐量结果都很好。吞吐量不包括TLP和封装头。此外，每次迭代中的所有内存请求都访问同一个地址。我们使用随机和顺序访问模式来测量吞吐量和延迟；但是，由于内存访问模式在任何实验中都没有差异。软件部分的处理时间相对占主导地位，并且由于内存访问模式而掩盖了性能上的差异。

### 5.1 LibTLP到NetTLP适配器

在第一个基准测试中，我们通过NetTLP适配器测量了从LibTLP到适配器主机上内存的PCIe事务的吞吐量。预计吞吐量将受到运行tlpperf的Linux内核网络堆栈性能的限制，因为适配器主机上的数据路径是完全硬件的，其链路是16 Gbps PCIe Gen 2 4通道链路和10 Gbps以太网链路。

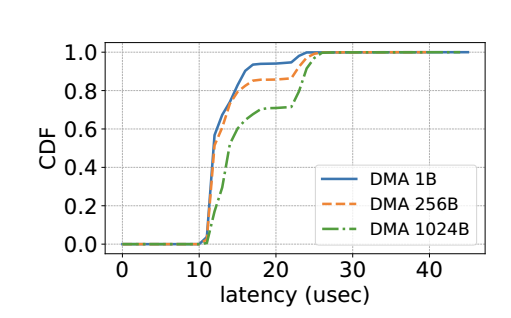
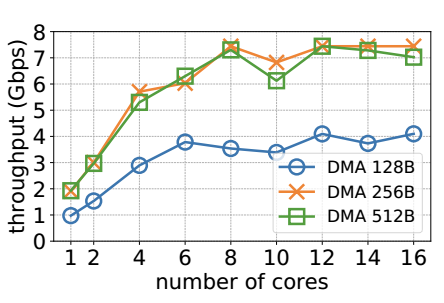
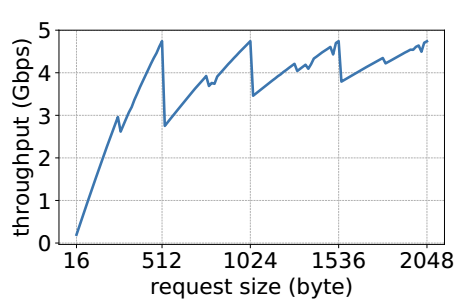
  

图9:LibTLP到NetTLP适配器的DMA 图10:LibTLP到NetTLP适配器的DMA 图11:NetTLP适配器到LibTLP

读取延迟 写入吞吐量与CPU核数的关系 的DMA读吞吐量

图7显示了由tlpperf在设备主机上发出的DMA读取的吞吐量。该键指示每个DMA读取请求的请求大小。如图所示，吞吐量随着CPU核数的增加而线性增加。结果表明，基于标签的UDP端口分发技术成功地利用了Linux设备主机上的多个队列和多个内核。另一方面，大于512字节的读请求大小对吞吐量没有贡献，因为最大读请求大小（MRR）是512字节。此方向的最大吞吐量约为3.6 Gbps，低于PCIe Gen 2 x1链路速度；但是，所需吞吐量取决于应用程序和使用情况。例如，第6节演示了不依赖于吞吐量的用例。注意，当前的LibTLP使用Linux套接字API；因此，LibTLP将通过高速网络I/O处理更多的UDP流量[28,39]。

接下来，我们测量了来自tlpperf的16核DMA读取吞吐量，同时将读取请求的大小增加了16个字节。图8中所示的结果表示锯齿状图案，这也在pcie实验文件[34]中指出。锯齿形图案是由PCIe协议的打包结构引起的。mrs是512字节；因此，当请求大小不是512的倍数时，剩余的字节将通过小容量内存读取进行传输。这种小尺寸TLP会导致吞吐量降低。小TLP的大小随着请求大小的增加而增加，因此吞吐量也会增加，直到请求大小超过下一个512的倍数。以类似的方式，最大有效负载大小（MPS）为256字节，在256字节的倍数之后，吞吐量略有下降。出现锯齿图案的结果以及pcie bench基于硬件的测量结果表明LibTLP正确地实现了pcie协议的分组。

除了吞吐量之外，我们还测量了DMA读取的延迟。PCIe规范定义了完成超时；因此，评估DMA读取延迟对于在软件中原型化PCIe设备至关重要。图9显示了tlpperf生成的10000个DMA读取的结果，其中包含1个字节、256个字节和1024个字节的读取请求。延迟随着请求大小的增加而增加；但是，99%的延迟小于27微秒，而1024字节DMA读取的最大延迟是45微秒。这些结果对应于完成超时范围A（50 us到10 ms）。因此，我们认为原型从延迟角度来看，在软件中使用硬件根复合体的PCIe设备是可行的。根据pciebench[34]，物理主机内的DMA读取延迟为400到800纳秒。因此，设备主机上的网络堆栈和tlpperf应用程序的软件处理在NetTLP平台的延迟中占主导地位。

接下来，我们测量了LibTLP的DMA写入吞吐量。与DMA读取不同，DMA写入是已发布的事务；因此，我们无法正确测量DMA写入的延迟和吞吐量。在这个实验中，tlpperf计算MWr tlp写入套接字时的吞吐量。图10显示了这个测量结果。除了DMA读取结果外，DMA写入还可以有效地使用多个核心和队列。与DMA读取相比，DMA写入以256字节DMA写入达到较高的吞吐量，因为MPS是256字节。注意，这个吞吐量可以看作是Linux网络栈的UDP套接字的传输吞吐量。

### 5.2到LibTLP的NetTLP适配器

对于第二个方向，我们通过生成从适配器主机上的PCIe工作台到设备主机上运行的psmem的PCIe事务来测量吞吐量。图11显示了这个方向上的DMA读吞吐量。结果也代表了锯齿状的图案以及相反的方向。此外，pcie工作台与软件存储设备的工作表明，NetTLP平台可以在软件。还有，最大吞吐量约为4.7英镑。我们已确认pcie工作台使用的TLP标记值从0x00到0x17；因此，基于LibTLP的psmem可以通过基于标签的UDP端口分布并行利用16个核。

表2显示了从适配器主机上的pcie工作台到设备主机上的psmem的DMA读取延迟。我们测量了每个请求大小的100000个DMA读取。如图所示，与硬件中最初的pcie工作台评估不同，请求大小没有显著差异。这是因为设备主机上接收和发送UDP包的软件处理占主导地位。但是，延迟也满足完成超时范围A。

### **表2:pcie工作台到psmem的DMA读取延迟（微秒）**

### 

**6 Use Cases6用例**

本节演示NetTLP的三个用例。我们（1）通过捕获TLP来观察商业根复合体和外围设备的具体行为,(2)实现了以太网卡作为实际网卡的理论模型,(3)演示了物理机器的内存内省。本节中的所有观察和演示都是在微观基准中使用的相同机器上进行的。

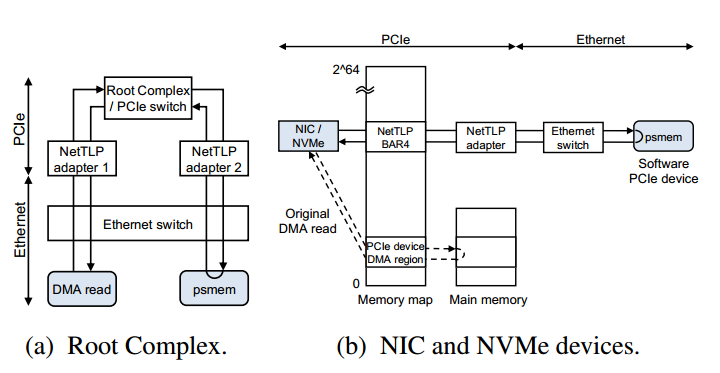
6.1捕获TLP

作为第一个演示，我们通过捕获TLP观察了一个商业根复合体、两个以太网NIC和两个NVMe SSD的PCIe事务。NetTLP适配器通过以太网链路提供tlp；因此，NetTLP使我们能够通过使用UNIX命令（即tcpdump）的功能强大的IP网络软件来分析tlp。此外，PCIe拓扑的灵活性使我们能够调整NetTLP来观察由不同元素发出和处理的各种PCIe事务。

6.1.1根复合体和PCIe交换机

第一个观察是阐明根系复合体的行为。PCIe规范不允许PCIe交换机在切换期间修改PCIe数据包。但是，当在PCIe设备之间路由PCIe包时，允许根复合体将PCIe包拆分为小的PCIe包。该规范没有描述通过根复合体在对等设备通信上进行TLP拆分的详细机制。虽然TLP拆分可能会对性能产生负面影响，但其行为依赖于每个根复杂实现，并且很难观察其行为。作为演示，我们通过比较NetTLP捕获的根复合体或PCIe交换机的实际tlp来阐明这一点。

为了捕获TLP，我们在微基准测试中使用的机器上的根复合体或PCIe交换机下准备了两个NetTLP适配器。图12a显示了这个观察的拓扑结构。在测试场景中，设备主机上的DMA读取应用程序通过适配器主机上的两个NetTLP适配器向psmem发送512字节的MRd TLP，psmem返回CplD TLP。此外，我们通过更改安装NetTLP适配器的PCIe插槽，将中间元素从PCIe交换机切换到根复合体。在这个拓扑结构中，我们在通过PCIe交换机或以太网交换机上的根复合端口镜像之前和之后捕获了tlp。



**图12：捕获TLP的两种拓扑。我们通过以太网交换机上的端口镜像捕获TLP。**

#### 图13显示了捕获的TLP。x轴表示捕获机器从镜像端口捕获TLP时的时间戳。请注意，时间戳是由NIC硬件标记的，因此精度在纳秒级。y轴表示TLP的TLP标记值。TLP被捕获两次：在PCIe交换机或根复合体之前和之后。上排和下排的图分别显示了在连接到图12a所示的NetTLP适配器1和适配器2的链路上捕获的tlp。

#### 图13a确认PCIe交换机未按预期修改TLP。DMA读取应用程序发送了一个512字节的MRd TLP，psmem返回了两个256字节的CplD TLP。相比之下，图13b显示在NetTLP适配器之间路由TLP时，根复合体将512字节MRd TLP拆分为8个具有不同TLP标记值的64字节MRd TLP。psmem返回了8个64字节的CplD tlp，根复合体从较小的CplD tlp重建了两个256字节的CplD tlp。结果，DMA读取应用程序接收到与MPS对齐的预期CplD tlp。

#### 6.1.2 Ethernet NIC and NVMe SSD6.1.2以太网NIC和NVMe SSD

接下来，我们测量并比较了商用NIC和NVMe设备产生的tlp。了解产品设备如何使用TLP将是开发高性能PCIe设备的有用指南。一般的外围设备通过DMA与CPU通信到主存储器。为了从设备捕获TLP，我们为以太网NIC使用了改进的netmap驱动程序[39]，为NVMe SSD使用了改进的UNVMe[32]，将DMA地址从主内存更改为NetTLP适配器的BAR4。因此，NetTLP能够捕获从连接到NetTLP适配器的以太网链路上的设备发送的tlp。

为了观察PCIe设备的各种行为，我们准备了不同类型和速度的设备：Intel X520和XL710 NIC，Intel P4600和Samsung PM1725a NVMe SSD。这些设备的吞吐量如下：Intel X520为10 Gbps以太网NIC，Intel XL710为40 Gbps以太网NIC，Intel P4600 NVMe设备的顺序写入速度为1575MB/s，三星PM1725的顺序写入速度为2600 MB/s。

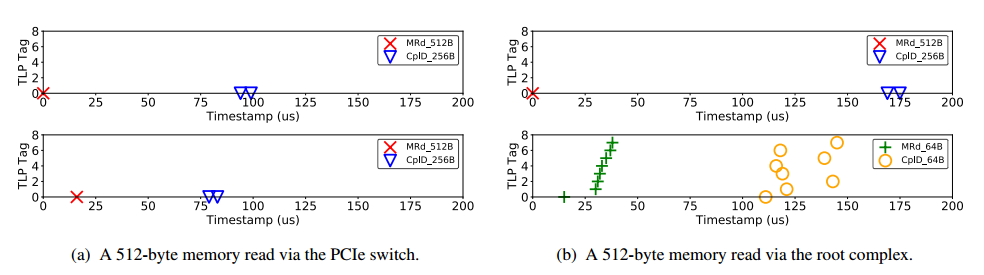
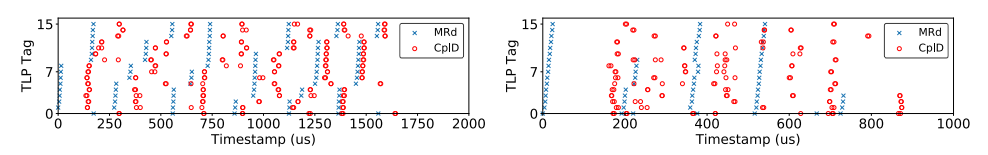
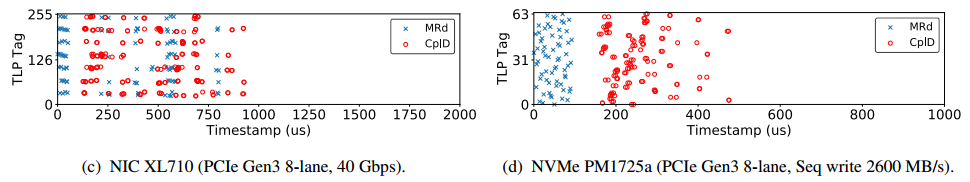


图13：通过PCIe交换机或根复合体捕获的DMA读取TLP的比较。下一行中的图形指示PCIe交换机或根复合体后面捕获的TLP。



（a） NIC X520（PCIe Gen2 8通道，10 Gbps）。（b） NVMe P4600（PCIe Gen3 4通道，顺序写入1575 MB/s）。



（c） NIC XL710(PCIe Gen3 8通道，40 Gbps)。（d） NVMe PM1725a(PCIe Gen3 8通道，顺序写入2600 MB/s)

**图14:NIC和NVMe设备的标记字段使用情况比较。**

图12b显示了这种观察的实验设置。在此设置中，NIC或NVMe和NetTLP适配器安装在同一个PCIe交换机下的PCIe插槽中。设备发送MRd tlp来发送数据包或将数据写入NVMe ssd，MRd tlp则被发送到psmem。然后，psmem返回CplD tlp，其中包含为NIC场景预先准备的以太网帧或NVMe场景的零填充数据。对于nvds-32的场景，向NIC发送了32个字节的数据包。请注意，Intel P4600的块大小为512字节，而Samsung PM1725a的块大小为4096字节；因此，我们调整了NVMe写入命令的数量，以写入32 MB的数据。为了捕获tlp，我们在NetTLP适配器和psmem运行的设备主机之间的以太网交换机上使用端口镜像，以及上一次实验。

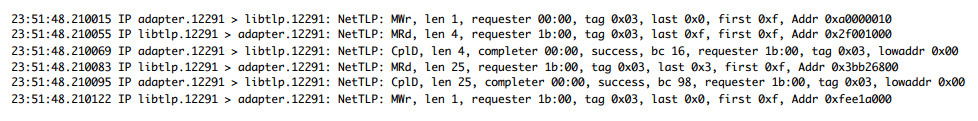
图14显示了NIC和NVMe设备捕获TLP的结果。结果表明，每个PCIe设备使用TLP标签的方式不同。X520和P4600使用0到15的标记值，PM1725a使用0到63的值，XL710使用24到249的值。随着几代PCIe的出现，PCIe链路速度得到了提高；但是MPS几乎没有得到改善。因此，这些PCIe设备通过利用TLP标签连续发送内存请求来提高数据传输吞吐量。使用的标签值的数量随着设备的期望吞吐量而增加。根据pcie工作台[34]中的延迟测量，512byte DMA读取的延迟约为580纳秒。

当不使用tag字段时，根据这个延迟计算的吞吐量约为7gbps。因此，如观察结果所示，充分利用标签字段是实现期望吞吐量的重要事项。

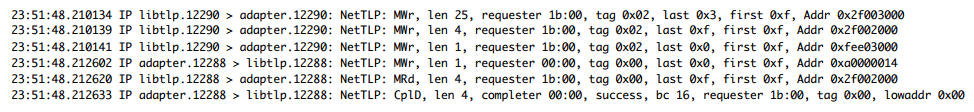
6.2 Prototyping an Ethernet NIC6.2以太网卡原型

为了确认NetTLP可以在软件中原型化PCIe设备，我们在NetTLP平台上实现了一个以太网卡作为概念证明。我们实现的目标网卡是由pcie bench[34]引入的简单网卡。最初的简单网卡是简单以太网网卡的理论模型，它没有任何性能优化，如DMA批处理，以了解PCIe交互和计算带宽。这个不存在的NIC模型是展示NetTLP生产力的一个很好的目标。

NetTLP使我们能够在软件中建立PCIe设备模型的原型，并确认这些模型是否与现有的硬件根复合体一起工作。



（a） 用于传输98字节ICMP回显数据包的TLP。



（b） 接收98字节ICMP应答包的TLP。

图15：简单NIC实现捕获的tlp的tcpdump输出。len以DWORD表示数据有效负载的长度。

在模型的实现中描述了主机和简单NIC设备之间的详细交互作用[2]。在发送端，（1）主机更新一个4字节的TX队列尾部指针，（2）设备读取主机内存上的16字节TX描述符，（3）设备读取数据包内容并发送数据包，（4）设备生成4字节中断。

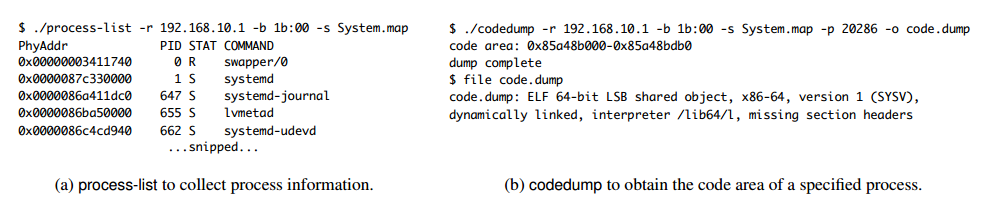
在接收端,(1)主机更新一个4字节的接收队列尾指针,(2)设备读取主机内存上的16字节RX描述符,(3)设备将接收到的数据包写入主机内存,(4)设备生成4字节RX中断。我们在NetTLP平台上的简单NIC实现在模型的PCIe交互之后，在适配器主机上执行具有物理根复合体的实际NIC。该实现包括两部分：NetTLP适配器的设备驱动程序和使用LibTLP的软件简单NIC设备实现。基于基本驱动程序的设备驱动程序将NetTLP适配器视为以太网NIC以及硬件NIC的常用驱动程序。软件简单网卡在设备主机上创建一个tap接口，并使用tap接口作为其以太网端口。传输到NetTLP适配器的以太网帧作为tlp通过PCIe链路和以太网链路传输到设备主机，以太网帧被传输到tap接口。软件简单的网卡实现是大约400行C代码，它实际上执行一个以太网网卡。

在以太网链路上可以观察到由根复合体和LibTLP生成的简单NIC的tlp。图15a显示了在通过简单NIC发送ICMP回显包时，修改的tcpdump捕获的tlp。驱动程序在NetTLP适配器（1st TLP）的BAR4上写入一个TX队列指针，简单NIC读取0x3bb26800（第二到第五个TLP）上的TX描述符和数据包内容，简单NIC将数据包发送到tap接口（6thTLP）后，生成MSI-X表指向的0xfee1a000的中断。在图15b所示的RX端，交互从将接收到的ICMP应答包写入主机内存（第一TLP）开始，因为驱动程序在接收新数据包之前将RX缓冲区告知了简单NIC。随后，简单NIC更新RX描述符（第二TLP）并生成中断(第三TLP).主机使用接收到的数据包后，驱动程序通过更新RX队列尾部指针(第4个TLP)将缓冲区发送回简单NIC，简单NIC读取RX描述符(第5个和第6个TLP).以这种方式，NetTLP允许在具有硬件根复合体的软件中实现PCIe设备。此外,通过IP网络技术可以观察到相互作用。

6.3 Physical Memory Introspection6.3物理记忆内省

NetTLP为硬件和软件之间的TLP交互提供了灵活的可编程性。这个特性提供了NetTLP对其他用例的适应性，例如内存自省。在物理[3,12,29,41,46]和虚拟[17,49]环境中，研究了在内存上监视和注入数据的方法。NetTLP还通过PCIe提供对主机内存的访问，这与以前的研究类似。然而，NetTLP适配器是远程操作主机内存的通道；因此，研究人员可以在LibTLP和IP网络堆栈之上实现他们的自省和检测方法，而无需使用专用的硬件或虚拟机监视器。如在第三个用例中，我们演示了将NetTLP应用到远程的可能性通过两个简单的应用程序进行内存内省。第一个应用程序是process list命令，类似于LibVMI[27]的一个例子。processlist在配备NetTLP适配器的Linux主机上收集进程信息。图16a显示了流程列表的示例用法。当执行进程列表时，它会找到一个表示第一个进程的task\u struct地址

从指定的系统地图适配器主机的。接下来，进程列表开始使用dma\_read（）遍历适配器主机的任务结构。



（a） 收集过程信息的过程列表 （b） 获取指定进程的代码区。

图16:NetTLP用于物理内存自省的两个示例应用程序。两个应用程序都在设备上执行并读取适配器主机的物理内存。

接下来，让我们关注一个单一的过程。codedump从适配器主机获取正在运行的进程的二进制文件。图16b显示了这个命令的示例用法。codedump使用与进程列表相同的方法查找指定进程ID的任务结构，并获得表示进程的虚拟内存的mm结构。然后，代码转储通过遍历页表将代码区的进程特定地址转换为相应的物理地址。最后，DMA从LibTLP读取的转储代码区可以看作是一个普通的二进制对象文件，甚至可以通过objdump命令重新组装。通过这些方式，研究人员和开发人员可以轻松地在NetTLP平台上实现他们的内存自省方法。

7相关工作

未来互连：一些下一代互连规范是通过扩展PCIe的功能来设计的。CCIX[13]和CXL[14]将处理器和外围设备之间的缓存一致性引入到它们的互连中。CCIX使用PCIe数据链路层并定义其事务层，CLX在PCIe数据链路层和事务层上定义CLX扩展。OpenCAPI[36]和Gen-Z[18]支持IEEE 802.3以太网和PCIe物理层。这些互连需要外设和主机芯片组的硬件扩展。尽管这种下一代互连引入了新功能，但它们仍然是基于分组的数据通信。NetTLP通过利用基于分组的通信在以太网上传送tlp。因此，我们认为只要采用分层模型和分组通信，NetTLP设计就可以应用于未来的互连。

NetTLP和RDMA的区别：与NetTLP一样，远程DMA（RDMA）协议也通过以太网和IP网络从远程主机实现DMA，以实现高速互连。RoCEv2封装了Infiniband。

带有以太网、IP和UDP报头的报头和有效载荷[8]。iWARP使用以太网、IP和TCP报头[38]。与它们的目的不同，NetTLP旨在提供PCIe事务的可观测性，因此，它采用直接将tlp封装在IP和以太网中。RDMA协议需要在RDMA适配器中将PCIe协议转换为RDMA协议。因此，它们缺乏我们通过用例演示的PCIe协议的可观察性。

软件PCIe设备的设备驱动程序：NetTLP简化了PCIe原型制作，但它没有提高设备驱动程序的生产力。开发设备驱动程序仍然需要一定的努力。为了提高NetTLP平台中设备驱动程序的效率，有两种方法：第一种方法是使用框架，根据与协议规范和设备特性相关的模板自动生成设备驱动程序。另一种方法是在用户空间中编写设备驱动程序，就像DPDK一样，同时使用一些辅助。

**8结论**

在本文中，我们提出了NetTLP，它可以开发与硬件根复合体交互的软件PCIe设备。实现该平台的关键技术是将PCIe事务层分解为软件层，然后通过以太网和IP网络传输tlp来连接软件事务层和硬件数据链路层。研究人员和开发人员可以在软件中构建自己的PCIe设备原型，并通过tcpdump等IP网络技术观察实际的tlp。本文中的用例展示了对根复合体和产品NICs和NVMe ssd的TLP级行为的观察、与硬件根复合体交互的400LoC软件以太网NIC实现以及物理内存自省。我们相信，通过用例展示的NetTLP平台上的高生产力和可观察性有助于研究和工业界当前和未来的PCIe开发。