NetVM：使用商品平台上的虚拟化的高性能和灵活的网络

黄浩，乔治华盛顿大学；罗马克里什南，罗格斯大学；
蒂莫西·伍德，乔治华盛顿大学

https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/hwang

本文被收录在第11届USENIX网络系统设计与实现研讨会的论文集(NSDI ‘14)中。

2014年4月2日至4日，美国华盛顿州西雅图市

ISBN978-1-931971-

开放获取第11届USENIX网络系统设计与实现研讨会论文集(NSDI ‘14)由USENIX赞助

NetVM：使用商品平台上的虚拟化的高性能和灵活的网络

黄金浩† K.K.拉马克里什南* 蒂莫西木材†

†乔治华盛顿大学 *WINLAB，罗格斯大学

摘要

NetVM通过使高带宽网络功能能够以接近线的速度运行，同时利用低成本商品服务器的灵活性和定制化，为网络带来了虚拟化。NetVM允许可自定义的数据平面处理功能，如防火墙、代理和路由器嵌入到虚拟机中，补充了软件定义网络的控制平面功能。NetVM使动态扩展、部署和重新编程网络功能变得很容易。这比现有的专门构建的，有时是专有的硬件提供了更多的灵活性，同时仍然允许复杂的策略和完整的数据包检查来确定后续的处理。它比现有的软件路由器平台的吞吐量要高得多。

NetVM建立在KVM平台和IntelDPDK库之上。我们详细介绍了我们已经解决的许多挑战，例如通过共享的大型页面增加对高速虚拟机间通信的支持，以及增强CPU调度器，以防止核心间通信和上下文切换造成的开销。NetVM允许将数据以真正的零拷贝传输到虚拟机，用于信任边界内虚拟机之间的数据包处理和消息传递。我们的评估显示了NetVM如何从多个流水线虚拟机中组成复杂的网络功能，并且仍然可以获得高达10Gbps的吞吐量，与现有的使用SR-IOV进行虚拟化网络的技术相比，提高了250%以上。

1 介绍

虚拟化通过允许更大的灵活性、更容易的部署和改进的资源多路复用，已经彻底改变了数据中心服务器的管理方式。随着网络功能虚拟化的发展，以及软件定义网络(SDN)的使用，通信网络中也开始发生类似的变化。虽然将网络功能迁移到更基于软件的基础设施的过程很可能从更注重“控制平面”的边缘平台开始，但通过使用通用的现成硬件和系统所获得的灵活性和成本效益将使其他网络功能的迁移具有吸引力。一个主要的障碍是这种虚拟化平台的可实现的性能和可伸缩性与专门构建的（通常是专有的）网络相比-使用基于自定义asic的硬件或中端盒。中间盒是典型的硬件软件包在特殊设备上的年龄，通常成本很高。相比之下，基于虚拟机(vm)的高吞吐量平台将允许网络功能以低成本在网络节点上动态部署。此外，向虚拟机的转变将使企业在现有云平台上运行网络服务，为网络功能带来多路复用和经济的规模效益。一旦数据可以以所有数据包大小的行率移动到虚拟机之间，我们就接近长期愿景，数据中心和网络驻留“框”之间的线开始模糊：软件和网络基础设施都可以以相同的方式开发、管理和部署。

网络虚拟化标准和SDN在网络中提供更大的可配置性方面取得了进展[1-4]。SDN通过允许软件管理网络控制平面来提高灵活性，而性能关键的数据平面仍然使用专有的网络硬件实现。SDN允许在数据转发方面有新的灵活性，但对控制平面的关注阻止了依赖于数据平面的许多类型的网络功能的动态管理，例如包有效负载中携带的信息。

这限制了可以“虚拟化”成软件的网络功能的类型，使得网络继续依赖于基于专门构建的硬件的相对昂贵的网络设备。

网络接口卡(NICs)的最新进展允许使用英特尔的数据平面开发工具包(DPDK)[5]等技术进行高吞吐量、低延迟的数据包处理。该软件框架允许终端主机应用程序直接从网卡接收数据，从而消除了传统中断驱动的操作系统级数据包处理中固有的开销。不幸的是，DPDK框架在支持虚拟化方面的选项集有些有限，而且它本身并不能支持网络和数据中心管理员所希望的灵活、高性能的功能类型。

为了改善这种情况，我们开发了NetVM，这是一个使用商用硬件以线速(10gbps)运行复杂网络功能的平台。NetVM利用了DPDK的高吞吐量数据包处理能力，并增加了一些抽象，使其能够灵活地创建网络内服务，

锁紧，负载平衡。由于这些“虚拟凸起”可以检查完整的数据包数据，因此与使用现有的基于SDN的控制器操作硬件交换机的框架相比，可以支持更广泛的数据包处理功能。因此，NetVM进行了以下创新：

1. 一种基于虚拟化的平台，用于灵活的网络服务部署，可以满足定制硬件的性能，特别是那些涉及复杂数据包处理的硬件。
2. 一个共享内存框架，真正利用DPDK库向虚拟机和虚拟机之间提供零拷贝交付。
3. 基于管理程序的交换机，可以以依赖状态（例如，智能负载平衡）和/或数据依赖方式（例如，通过深度包检查）动态调整流的目的地。
4. 一种支持高速虚拟机间通信的体系结构，使复杂的网络服务能够跨多个虚拟机传播。
5. 限制仅对受信任虚拟机的数据包数据访问的安全域。

我们已经使用KVM和DPDK平台实现了NetVM，所有上述创新都建立在DPDK之上。我们的研究结果显示，NetVM如何从多个流水线虚拟机中组成复杂的网络功能，并且仍然获得10Gbps的线路传输率，与现有的基于SR-IOV的技术相比，提高了250%以上。我们相信，NetVM将在具有额外的nic和处理核心的机器上扩展到更高的吞吐量。

2 背景和动机

本节介绍在虚拟化商品服务器上提供灵活网络服务的背景。

2.1 高速COTS网络

与传统的网络硬件相比，基于软件路由器、SDN和虚拟机管理程序的交换技术一直试图降低部署成本，并增加灵活性。然而，这些方法受到了大宗商品服务器可实现的性能的阻碍[6-8]。这些对吞吐量和延迟的限制阻碍了软件路由器无法取代自定义设计的硬件[9-11]。

有两个主要的挑战阻止了商业现成的(COTS)服务器能够以在线速度处理网络流。首先，网络数据包到达不可预测的时间，因此中断通常用于通知操作系统数据已经准备好处理。然而，中断处理可能是昂贵的，因为现代超量量处理器使用

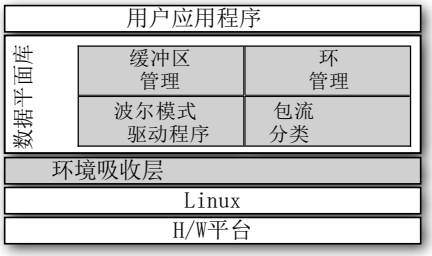

图1：DPDK通过Linux运行的运行时环境。

长管道、无序和投机性执行，以及多级内存系统，所有这些都往往会增加周期[12,13]中中断所支付的惩罚。当分组接收速率进一步增加时，在这样的系统[14]中所实现的（接收）吞吐量可以显著下降。其次，现有的操作系统通常会将传入的数据包读入内核空间，然后将数据复制到对其中感兴趣的应用程序的用户空间。在虚拟化设置中，这些额外的副本可能会产生更大的开销，其中可能需要在管理程序和来宾操作系统之间复制额外的时间。这两个开销来源限制了在商品服务器上运行网络服务的能力，特别是那些使用虚拟化[15,16]的服务器。

IntelDPDK平台试图通过允许用户空间应用程序直接轮询NIC以获取数据来减少这些开销。该模型使用Linux的大页面来预先分配大的内存区域，然后允许应用程序将DMA数据直接分配到这些页面中。. 图1显示了在应用程序层中运行的DPDK体系结构。轮询模式驱动程序允许应用程序直接访问网卡，而不涉及内核处理，而缓冲区和环管理系统类似于内核中通常使用的持有sk缓冲区的内存管理系统。虽然DPDK支持高吞吐量用户空间应用程序，但它还没有提供构建和互连接复杂网络服务的完整框架。此外，DPDK提供往返于VM的直接DMA的直通模式的性能可能明显低于本机IO[1]。例如，DPDK支持单根I/O虚拟化(SR-IOV[2])，允许多个虚拟机访问网卡，但数据包“交换”（即解复用或负载平衡）只能基于L2地址执行。如图所示ure2(a)，当使用SR-IOV时，会打开数据包

[1]在桑迪桥之前，性能接近本地性能的一半，但使用下一代常春藤桥处理器，由于IOTLB(I/O翻译查找缓冲区)超级页面支持[17]，性能得到了改善。但尚未公布任何性能结果。

[2]SR-IOV使得从逻辑上划分一个NIC成为可能，并向每个虚拟机公开一个单独的基于PCI的NIC，称为“虚拟函数”[18]。

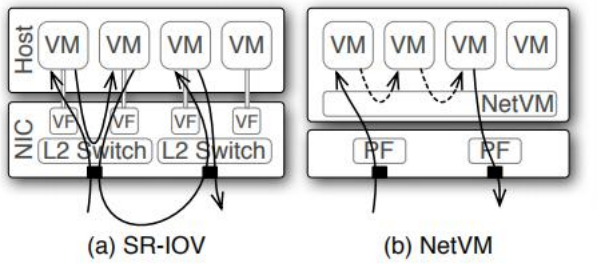

图2：DPDK使用SR-IOV使用每个端口交换，而NetVM在管理程序和共享内存数据包传输中提供了一个全局交换机（虚线）。

以网卡中的每个端口为基础，这意味着如果在共享端口上的虚拟机之间转发数据包，则需要第二个数据副本。更糟糕的是，数据包必须离开主机，并通过外部交换机返回，才能传输到连接到另一个端口的虚拟功能的虚拟机。其他虚拟机交换平台也会出现类似的开销，例如，OpenvSwitch[19]和VMware的v网络分布式交换机[20]。我们试图通过在NetVM中提供一种灵活的交换能力来克服这一限制，如图2(b)所示。这提高了虚拟机之间的通信性能，这在部署链状服务时起着重要作用。

英特尔最近发布了DPDK和OpenvSwitch[21]的集成，以减少SR-IOV交换的限制。但是，DPDKvSwitch仍然需要在管理程序和虚拟机的内存之间复制数据包，并且不支持直接链接的虚拟机通信。NetVM的增强超越了DPDKvSwitch，它提供了一个灵活的状态或数据依赖交换、高效的VM通信和安全域来隔离VM组。

2.2 灵活的网络服务

虽然像DPDK这样的平台允许更快的处理，但它们仍然提供了灵活性的限制，特别是对于虚拟环境。DPDK+SR-IOV支持的基于NIC的交换机不仅很昂贵，而且很有限，因为NIC只具有对第2层标头的可见性。使用当前的技术，每个具有不同目标MAC的数据包都可以传送到不同的目标虚拟机。但是，在网络驻留框中(例如充当防火墙的中框、代理，或者即使COTS平台充当路由器)，传入数据包的目标MAC是相同的。虽然NIC设计的进步可以减少这些限制，但基于硬件的解决方案将永远无法与基于软件的方法的灵活性相匹配。

通过让管理程序执行初始包交换，NetVM可以支持更复杂和动态的功能。例如，每个应用程序都是这样的

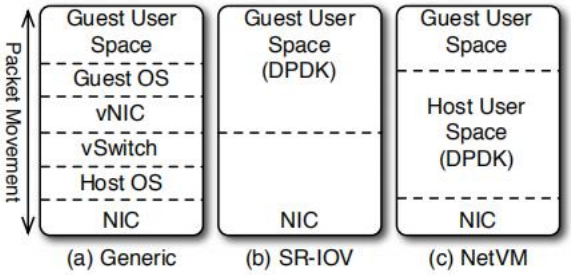

图3：数据包交付的体系结构差异在虚拟化平台中。

支持不同的功能可能存在在单独的虚拟机中，可能需要利用流分类来基于浅层（基于头）或深度（数据库）数据包分析等机制正确地路由数据包。同时，NetVM的交换机可以使用依赖于状态的信息，如VM负载级别、一天中的时间或动态配置的策略来控制交换机算法。基于这些规则交付数据包在当前的平台上是不可行的。

2.3 基于虚拟机的网络网络提供商通过结合中点盒和网络硬件来构建整体的网络功能，这些硬件通常是由不同的供应商构建的。虽然NetVM可以在软件中实现快速的数据包处理，但虚拟化的使用将允许这组不同的服务彼此“玩得很好”——虚拟化使封装一块软件及其操作系统依赖性变得很简单，与在一个裸金属服务器上运行多个进程相比，大大简化了部署。在虚拟机中运行这些服务还可以允许用户控制的网络功能部署到新的环境中，如云计算平台，在这些环境中，虚拟机是规范的，不同网络服务之间的隔离将是至关重要的。

虚拟化的整合和资源管理的好处也是众所周知的。与硬件中点盒不同，虚拟机可以在需要的时间和地点按需要实例化。这允许NetVM为一个服务器提供多个相关的网络功能，或者在需要新服务的地方动态生成虚拟机。与在裸金属上运行的网络软件相比，为每个服务使用虚拟机简化了资源分配，并提高了性能隔离。这些特性对于通常具有严格的性能要求的网络服务至关重要。

3 系统设计

图3比较了两种现有的、通常实现的网络虚拟化技术和NetVM。在第一种情况下，表示传统的虚拟化plat-

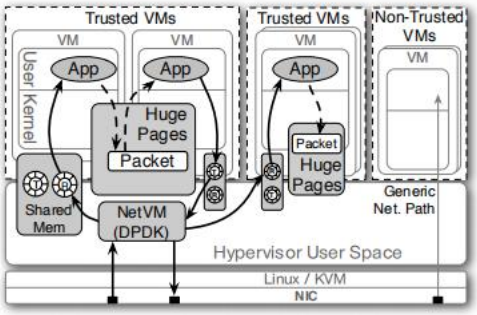

图4：NetVM只需要一个简单的描述符来通过共享内存（实心箭头）进行复制，然后让VM直接访问存储在大页面中的数据包数据（虚线箭头）。

表单、数据包到达网卡并被复制到管理程序中。然后，虚拟交换机执行L2（或基于完整的5元组数据包头的更复杂的功能）切换，以确定哪个虚拟机是该数据包的接收者，并通知相应的虚拟网卡。然后，将包含该数据包的内存页面复制或授予来宾操作系统，最后将该数据复制到用户空间应用程序中。毫不奇怪，这个过程涉及到大量的开销，阻碍了线路速度的吞吐量。

在第二种情况下(图3(b))，SR-IOV用于执行网卡本身的L2切换，数据可以直接复制到适当虚拟机的用户空间中。虽然这将数据移动最小化，但它确实以包路由到虚拟机的有限灵活性为代价，因为网卡必须配置静态映射，而MAC地址以外的包头信息不能用于路由。

NetVM的体系结构如图3(c)所示。它不依赖于SR-IOV，而是允许管理程序中的用户空间应用程序分析数据包并决定如何转发它们。但是，我们没有使用共享内存机制来将数据复制到Guest，而不是直接允许Guest用户空间应用程序读取它所需的数据包数据。这提供了灵活的切换和高性能。

3.1 零拷贝包交付

网络提供商正在越来越多地部署由路由器、代理、视频转换编码器等组成的复杂服务，NetVM可以将其整合到单个主机上。为了支持这些组件之间的快速通信，NetVM使用了两个通信通道来快速移动数据，如图4所示。第一个是一个小的共享存储区域（在管理程序和每个虚拟机之间共享），用于传输数据包描述符。第二个是与一组受信任的虚拟机共享的巨大页面区域，它允许链式应用程序直接读取或写数据包数据。通过“授予”机制共享内存共享是常见的用于在管理程序和客户之间传输页面控制；通过将其扩展到所有受信任的客户虚拟机可访问的内存区域，NetVM可以实现有效处理跨多个虚拟机的流。

NetVMCore作为一个启用DPDK的用户应用程序运行，使用DMA投票NIC将读取包直接读到巨大的页面区域。它根据诸如包头、可能的内容和/或VM负载统计信息等信息来决定将每个包发送到哪里。NetVM将数据包的描述符插入到环缓冲区中，该缓冲区设置在单个目标VM和管理器之间。每个VM由一个“角色号”标识，每个网络功能的表示，由VM管理器分配。该描述符包括一个mbuf位置(相当于Linux内核中的一个skbuff)和用于数据包接收的巨大页面偏移量。当传输或转发数据包时，描述符还指定操作（通过NIC、丢弃或转发到另一个VM）和角色号（即转发时的目标VM角色号）。虽然这个描述符数据必须在管理程序和来宾之间复制，但它允许来宾应用程序直接访问存储在共享的大页面中的数据包数据。

在客户应用程序（通常实现某种形式的网络功能，如路由器或防火墙）分析包后，它可以要求NetVM将包转发到不同的VM或通过网络传输它。转发只是重复上述过程——NetVM将描述符复制到不同虚拟机的环缓冲区中，以便再次处理；包数据保持在巨大的页面区域，不需要复制（尽管如果需要，它可以由来宾应用程序独立修改）。

3.2 无锁设计

共享内存通常使用锁进行管理，但锁通过序列化数据访问和增加通信开销，不可避免地会降低性能。这对于高速网络来说尤其成问题：为了保持全10的吞吐量独立于分组大小的Gbps，包必须在67.2ns[22]内处理，然而有争议的锁的上下文切换需要微秒[23,24]的顺序，甚至无争议的锁操作可能需要数十纳秒[25]。因此，单个上下文切换可能会导致系统落后，从而可能导致数十个数据包被丢弃。

我们通过使用具有服务的专用核心的并行队列来避免这些问题。当使用具有多个队列和接收侧缩放(RSS)功能的NIC时[1]，NIC接收包-

[1]现代nic支持RSS，这是一种网络驱动程序技术，允许数据包接收处理跨多个处理器或核心[26]进行负载平衡。

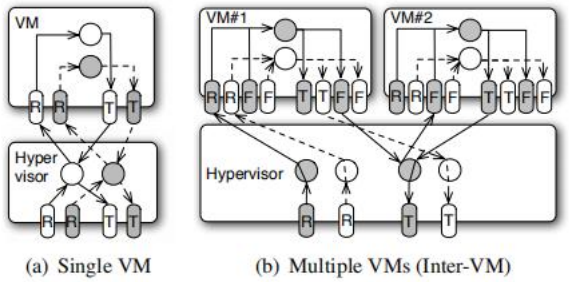

图5：无锁和感知队列/线程管理（R=接收队列、T=传输队列和F=转发队列）。

并基于可配置的(通常是n个元组)散列[27]将它们放到几个流队列中的一个中。NetVM只允许两个线程来操作此共享的循环队列——由管理程序中的一个核心运行的（生产者）DPDK线程和对数据包执行处理的客户VM中的（消费者）线程。只有一个生产者和一个消费者，因此不需要同步，因为它们都不会同时读取或写入到同一区域。

我们的方法通过为每个队列指定核心，从而消除了锁定的开销。这仍然允许可伸缩性，因为我们可以简单地创建额外的队列（每个队列由一对线程/核心管理）。这与NIC对RSS的支持一起工作，因为传入的流可以跨可用队列自动进行负载平衡。请注意，管理大型页面区域也不需要同步，因为只有一个应用程序能够控制包含数据包地址的描述符。

图5(a)描述了虚拟机中的两个线程如何在不相互中断的情况下传递数据包。管理程序中的每个核心（标记为圆）从NIC接收数据包，并将描述符添加到其自己队列的尾部。客户操作系统还有两个专用核心，每个核心都从队列头读取，执行处理，然后将数据包添加到传输队列中。管理程序从这些队列的尾部读取描述符，并导致网卡传输相关的数据包。这种线程/队列分离保证了一次只有单个实体可以访问数据。

3.3 NUMA-Aware设计

多处理器系统表现出NUMA特性，其中内存存取时间取决于相对于处理器的内存位置。应该避免在不同的套接字上使用内核可以访问映射到同一缓存行的内存，因为这将导致两个内核之间的昂贵的缓存无效信息来回打乒乓球。因此，忽略现代服务器的NUMA方面可能会导致net-等延迟敏感任务的性能显著下降

工作处理[28,29]。

从数量上说，在3GHzIntelXeon5500处理器上命中的最后一次级别缓存(L3)最多需要40个周期，但错过的惩罚最多可达201个周期[30]。因此，如果NetVM中的两个独立的套接字最终处理存储在附近内存位置的数据，性能下降可能会高达5倍，因为缓存行最终将不断失效。

幸运的是，NetVM可以通过以numa感知的方式仔细分配和使用大型页面来避免这个问题。当请求一个大型页面区域时，内存区域被统一划分为所有套接字，因此每个套接字从本地dimm到套接字分配总（总大页面大小/套接字数量）字节。在管理程序中，NetVM然后创建与套接字相同数量的接收/传输线程，每个线程只用于处理该套接字本地的大型页面中的数据。来宾虚拟机内部的线程会以类似的方式创建并固定到适当的套接字上。这确保了当数据包由主机或来宾处理时，它始终留在本地内存库中，并且缓存行永远不需要在套接字之间传递。

图5说明了如何管理两个套接字（灰色和白色）。也就是说，由灰色线程处理的数据包永远不会移动到白色线程，从而确保快速内存访问并防止缓存一致性开销。这还显示了NetVM如何跨多个核心处理包——处理来自NIC的DMAed数据的初始工作由管理程序中的核心执行，然后来宾中的核心执行包处理。在多虚拟机部署中，复杂的网络功能是通过连接在一起构建的，管道扩展到管理机中的另外一对核心，该核心可以将数据包转发到下一个虚拟机中的核心。我们的评估表明，只要有额外的内核来执行处理（在我们的测试台中多达三个独立的虚拟机），就可以扩展这个管道。

3.4 巨大的页面虚拟地址映射虽然每个巨大的页面代表一个大的连续内存区域，但整个巨大的页面区域分布在物理内存中，这是因为第3.3节中描述的对套接字分配，而且如果页面大小大于默认的总大小，默认单位大小可以在/Ã/meminfo下找到。这就带来了一个问题，因为管理程序中的地址空间布局是来宾不知道的，但是来宾必须能够根据描述符中的地址在共享的大页面区域中找到数据包。因此，网卡放置数据包的地址只对管理程序有意义；该地址必须被翻译，以便客人将

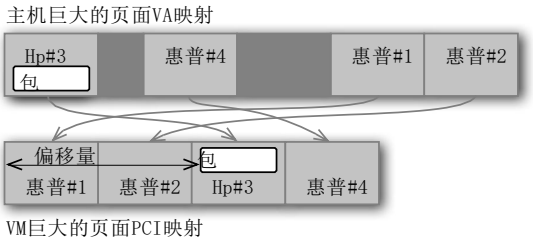

图6：遍布主机内存中的大型页面必须在虚拟机中连续对齐。NetVM必须能够将新数据包的地址从主机的虚拟地址空间快速转换为VM地址空间内的偏移量。

能够在共享内存区域中访问它。此外，必须尽快查找这些地址，以便执行行速数据包处理。

NetVM通过将巨大的页面映射到相邻区域的客户体中，克服了第一个挑战，如图6所示。NetVM使用模拟的PCI设备将这些大型页面公开给来宾虚拟机。客户虚拟机运行一个驱动程序，轮询设备并将其内存映射到用户空间，如第4.3节所述。实际上，这将在所有受信任的来宾虚拟机和管理程序中共享整个巨大的页面区域。任何其他不受信任的虚拟机都会通过管理程序使用常规的网络接口，这意味着它们无法看到从NetVM接收到的数据包。

即使巨大的页面在客人的内存空间中作为一个连续的区域出现，计算数据包存储的位置也很简单。*当NetVMDMAs将一个包进入巨大的页面区域时，它会收到一个在保护符的虚拟地址空间中有一个地址的描述符，这对必须处理包的来宾应用程序来说毫无意义*。虽然可以扫描已分配的大页面列表来确定数据包存储在哪里，但这种处理对于高速数据包速率来说实在太贵了，因为每个数据包都需要经过这个过程。为了解决这个问题，NetVM只使用了位操作和预先计算的查找表；我们的实验表明，与朴素查找相比，这在最坏情况下提高了10%（8个大页面)和15%(16个大页面）。

当收到一个数据包时，我们需要知道它属于哪个大页面。首先，我们建立了一个索引映射，将一个数据包地址转换为一个巨大的页面索引。该索引取自其地址的上8位st位至38th位)。前30位是相应的大页面中的偏移量，以及其余的位(38位的左边th位)可以忽略。我们表示这个函数为
IDMAP(h)=(h>>30)&0xFF，其中h为内存地址。然后将这个值用作数组HMAP[i]中的索引，以确定巨大的页数。

要获取地址库（即，的起始地址

在有序和对齐的大页面中，我们需要建立一个累积的地址库。如果所有的大页面都有相同的大小，我们就不需要这个地址库，只要相乘就足够了，但是由于可能有不同的大页面大小，我们需要跟踪一个累积的地址库。函数HIGH(i)
保留每个大的页面索引i的起始地址。
最后，使用LOW(a)=a&0x3FFFFFFFOFFSET(p)=HIGH(HMAP[IDMAP(p)])LOW(p)从数据包地址的最后30位中获取剩余地址.
返回中连续的大页面的地址偏移量
模拟PCI。

3.5 受信任和不受信任的vm

安全性是虚拟化云平台中的一个关键问题。由于NetVM的目标是提供零拷贝包传输，同时也具有灵活地管理协作vm之间的流，因此它与多个来宾虚拟机共享在管理程序中分配的大型页面。恶意虚拟机可以猜测数据包在这个共享区域中的位置，以窃听或操作其他虚拟机的流量。因此，必须在信任虚拟机和非信任虚拟机之间有明确的分离。NetVM提供了一个组分离，以实现必要的安全保证。当创建虚拟机时，它将被分配给一个信任组，信任组决定它将能够访问什么内存范围（以及哪些数据包）。

虽然我们当前的实现只支持受信任或不受信任的虚拟机，但可以进一步细分它。在将数据包数据映射到一个巨大的页面之前，DPDK的分类引擎可以对数据包进行浅层分析，并决定将其复制到哪个巨大的页面内存池。例如，这将允许面向一个云客户的流量流由一个信任组处理，而不同客户的流量流由同一主机上的第二个NetVM信任组处理。通过这种方式，NetVM不仅提高了网络功能虚拟化的灵活性，而且在共享主机上复用资源时还具有更大的安全性。

图4显示了受信任的虚拟机组和非受信任的虚拟机之间的分离。每个受信任的VM组都有自己的内存区域，每个虚拟机都有一个与NetVM通信的环缓冲区。相比之下，不受信任的虚拟机只能使用通用的网络路径，如图3(a)或(b)中的网络路径。

4 实施细节

NetVM的实现包括NetVM核心引擎(运行在管理程序中的DPDK应用程序)、NetVM管理器、模拟PCI设备的驱动程序、对KVM的CPU分配策略的修改和NetLib(我们用于构建网络内功能的库-

VM的用户空间中的ity)。我们的实现是建立在QEMU1.5.0(包括KVM)和DPDK1.4.1之上的。

KVM和QEMU允许常规Linux主机运行一个或多个虚拟机。我们的功能分为客户虚拟机中的代码和在主机操作系统的用户空间中运行的代码。我们在本讨论中交替使用术语主机操作系统和管理程序。

4.1 NetVM管理器

NetVM管理器在管理程序中运行，并提供一个通信通道，以便QEMU可以向NetVM核心引擎传递关于虚拟机的创建和破坏的信息，以及它们的信任级别。当NetVM管理器启动时，它将创建一个服务器套接字来与QEMU通信。每当QEMU启动一个新的虚拟机时，它都会连接到套接字，要求NetVM核心初始化新的虚拟机的数据结构和共享内存区域。该连接使用套接字类型的字符开发实现，在虚拟机配置中具有“字字套接字、路径=<路径>、id=<id>”。这是在VM和运行在KVM主机上的应用程序之间创建通信通道的一种常见方法，而不是依赖于基于管理程序管理员的消息传递[31]。

NetVM管理器还负责存储确定VM信任组的配置信息(即，哪些虚拟机应该能够连接到NetVMCore)和交换规则。这些规则被传递给实现这些策略的NetVM核心引擎。

4.2 NetVM核心引擎

NetVM核心引擎是一个在管理程序中运行的DPDK用户空间应用程序。NetVM核心是通过用户设置进行初始化的，如处理器核心映射、网卡端口设置和队列的配置。这些设置决定为接收和传输数据包创建了多少个队列，以及为这些任务为每个虚拟机分配了哪些核心。然后NetVMCore分配巨大页面区域并初始化网卡，这样在轮询时将DMA数据包输入到该区域。

NetVM核心引擎有两个角色：第一个角色是接收数据包并按照指定的策略交付/切换到虚拟机（使用零拷贝），另一个角色是与NetVM管理器通信，以同步有关新虚拟机的信息。主控制回路首先将NIC和DMA包轮询到突发（批处理）中的大页面，然后对于每个包，NetVM决定通知哪个VM。NetVM不是复制一个包，而是创建一个包含巨大页面地址的小包描述符，并将其放入私有共享环缓冲区(VM和NetVMCore之间共享)。实际的数据包数据可以通过共享的方式被虚拟机访问

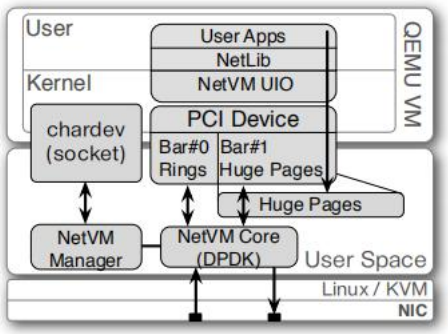

图7：NetVM的体系结构跨越客户系统和主机系统；使用模拟的PCI设备在它们之间共享内存。

内存，可通过下面描述的模拟PCI设备访问。

4.3 考虑的PCI

QEMU和KVM不直接允许在管理程序和虚拟机之间共享内存。为了克服这一限制，我们使用了一个模拟的PCI设备，它允许虚拟机映射设备的内存——由于该设备是用软件编写的，因此该内存可以被重定向到管理程序拥有的任何内存位置。NetVM需要两个独立的内存区域：一个私有共享内存(其地址存储在设备的BAR#0寄存器中)和一个巨大的页面共享内存(BAR#1)。私有共享内存被用作环形缓冲区，以传递用户应用程序（VM虚拟机管理程序）和数据包描述符（双向）的状态。每个虚拟机都有这个单独的私有共享内存。巨大的页面区域虽然在管理程序中不连续，但必须使用内存区域添加子区域函数映射为一个连续的块。我们在第3.4节的前面说明了这些大型页面是如何映射到虚拟地址的。在我们当前的实现中，所有vm机都访问相同的共享大页面区域，尽管这可以像3.5中讨论的那样放松。

在一个希望使用NetVM的高速IO的客户虚拟机中，我们运行一个前端驱动程序，使用Linux的用户空间I/O框架(UIO)访问这个模拟的PCI设备。UIO是在Linux2.6.23中引入的，它允许设备驱动程序几乎完全在用户空间中编写。该驱动程序将来自PCI设备的两个内存区域映射到客户的内存中，允许NetVM用户应用程序，如路由器或防火墙，直接与传入的数据包数据一起工作。

4.4 NetLib和用户应用程序应用程序开发人员不需要了解任何关于DPDK或NetVM的基于PCI设备的通信通道。相反，我们的NetLib框架提供了PCI和用户应用程序之间的接口。用户应用程序只需要提供一个包含

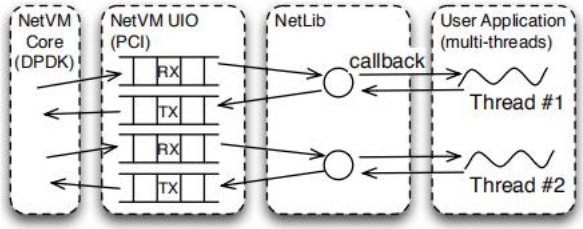

图8：NetLib提供了PCI设备之间的桥和用户应用程序。

配置设置，如内核数和回调函数。回调函数的工作原理类似于linux内核[32]中的NetFilter，这是一个用于包过滤和操作的流行框架。在接收到数据包时调用回调函数。用户应用程序可以读取和写入数据包，并决定下一步要做什么。操作包括丢弃、发送到NIC和转发到另一个虚拟机。如第4.1节所述，用户应用程序知道其他虚拟机的角色编号。因此，当将数据包转发到另一个VM时，用户应用程序可以指定角色号，而不是网络地址。这种抽象提供了一种实现虚拟机之间通信通道的简单方法。

图8说明了一个数据包流。当从管理程序接收到包时，NetLib中的一个线程获取它，并使用包数据调回用户应用程序。然后用户应用程序处理数据包（读或/和写），并通过操作返回。NetLib将该操作放在数据包描述符中，并将其发送到传输队列。NetLib通过为每个用户线程提供自己的输入和输出队列对来支持多线程。线程之间没有数据交换，因为NetLib提供了像NetVM那样的无锁模型。

5 评价

NetVM支持虚拟机和虚拟机之间的高速数据包交付，并提供灵活性来管理驻留在NetVM平台上不同虚拟机中的功能组件之间的流量。在本节中，我们将通过以下目标来评估NetVM：

- 展示了NetVM为典型应用程序提供高速数据包传输的能力，如：第三层转发、用户空间软件路由器和防火墙（5.2），
- 显示使用NetVM作为中框功能的额外延迟是最小的（5.3），
- 根据任务段（5.4）分析CPU时间，以及
- 演示NetVM灵活管理虚拟机之间流量的能力（5.5）。

在我们的实验设置中，我们使用了两个XeonCPUX5650@2.67GHz(2x6核)服务器——一个用于被测试的系统，另一个用于充当流量

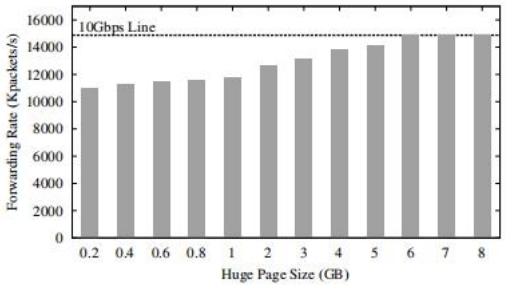

图9：巨大的页面大小可以降低高达26%的吞吐量（64字节的数据包）。NetVM需要6GB才能实现线路速率速度。

生成器——每个器都有一个Intel82599EB10G双端口NIC（有一个端口用于我们的性能实验）和48GB内存。我们使用8GB的大页面，因为图9显示至少需要6GB来实现完整的行率(我们在Intel的性能报告中看到了将8GB设置为默认的大页面大小)。主机操作系统是RedHat6.2（内核2.6.32），来宾操作系统是Ubuntu12.10（内核3.5）。使用了DPDK-1.4.1和QEMU-1.5.0。我们使用来自温德河的PktGen来生成流量[33]。另外提到的基本核心分配包括2个核接收，4个核传输/转发，每个虚拟机2个核。

我们还比较了NetVM与普遍使用的高性能SR-IOV系统。SRIOV允许NIC被逻辑地划分为“虚拟函数”，每个虚拟函数都可以映射到不同的虚拟机。我们衡量和比较这些体系结构提供的性能和灵活性。

5.1 应用程序

L3转发器[34]：我们使用一个简单的第3层路由器。转发函数在流分类阶段使用散列映射。哈希与流表结合使用，将每个输入包映射到运行时的流。哈希查找键由一个5个元组表示。从标识的流表条目中读出输入包的输出接口的ID。应用程序使用的流集被静态配置并在初始化时加载到哈希中(这个简单的第3层路由器类似于DPDK库中提供的示例L3转发器)。

点击用户空间路由器[10]：我们还使用点击，一个更高级的用户空间路由器工具包来衡量通过将现有路由器实现“插入”虚拟机，将其作为“容器”来实现的性能。单击支持每个执行简单计算的元素的组成，但一起可以提供更高级的功能，如IP路由。我们稍微修改了点击，添加了使用接收和lib的元素来加快网络IO。我们的变化总共大约有1000个

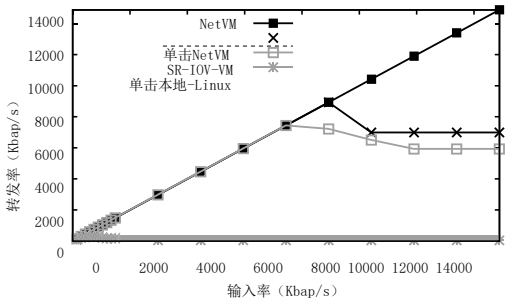

图10：转发速率作为NetVM输入速率的函数，使用NetVM、SR-IOV(VM中的DPDK)和本机Linux单击-NetVM路由器（64字节数据包）单击。

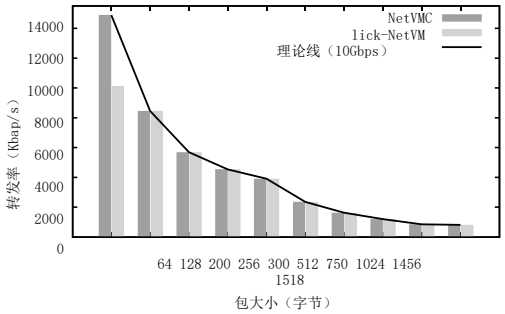

图11：NetVM提供了一个不管数据包大小如何的线率速度。由于巨大的应用程序开销，ClickNetVM以64字节的数据包大小达到6.8Gbps。

代码行。我们使用LinuxIO和我们的Netlib零拷贝版本来测试一个标准版本的点击。

防火墙[35]：防火墙根据安全策略控制网络流量流。我们使用Netlib来构建防火墙的基本特性——数据包过滤器。带有包过滤器的防火墙在网络层第3层操作。这提供了基于包中的多个信息的网络访问控制，包括通常的5元组：包的源IP地址和目标IP地址、网络或传输协议ID、源和目标端口；此外，其决策规则还将考虑包遍历的接口及其方向（入站或出站）。

5.2 高速包递送

包转发性能：NetVM的目标是提供线率吞吐量，尽管运行在虚拟化平台上。为了证明NetVM确实可以实现这一点，我们展示了L3包转发率与输入流量率。对于前导大小为8字节的10G以太网接口的标称64字节IP包的最小帧间间隙12字节的理论值为14,880,952个包。

图10显示了三种情况下的输入速率和数据包/秒的转发速率：NetVM的简单L3转发器，使用NetVM(点击-NetVM单击路由器)，以及使用本机Linux单击路由器(单击-NativeLinux)。NetVM实现了全行速率，而Click-NetVM的最大速率约为6gbps。这是因为Click添加了调度元素的管理费用（由我们随后在表1中显示的延迟分析确认）。请注意，增加输入速率会导致转发速率的轻微下降（由于最终被丢弃的数据包处理的浪费），或者以这个最大速率趋于稳定。我们相信Click-netvm的性能可以通过增加多线程支持或使用更快的处理器来进一步提高，但SR-IOV不能通过这种方式实现更好的性能。毫不奇怪，点击-NativeLinux的性能非常差(最大327Mbps)，说明了提供了简单的巨大改进

由零拷贝IO。[10]。

使用SR-IOV，虚拟机有两个与其相关联的虚拟函数，并使用两个核使用两个端口运行DPDK。SR-IOV的最大吞吐量为5Gbps。我们已经观察到，增加虚拟函数或核的数量并不能提高最大的吞吐量。我们推测，这一限制来自于硬件开关的速度限制。

图11现在显示了当数据包大小变化时的转发速率。由于NetVM没有由于数据包大小的增加而产生进一步的额外费用(数据由DMA交付)，因此它很容易实现完整的行率。此外，Click-NetVM还可以为128字节和更大的包大小的全行率。

虚拟机间包交付：NetVM的目标是通过组成虚拟机链来构建复杂的网络功能。为了评估流水线化的虚拟机处理元素如何影响吞吐量，我们测量了在改变数据包必须通过的虚拟机数量时所实现的吞吐量。我们将NetVM与一组SR-IOV虚拟机进行了比较，这是最先进的虚拟化网络。

图12显示，NetVM为一个虚拟机实现了显著更高的基吞吐量，并且它能够维持最多三个虚拟机链的几乎线率。在此之后，我们的12核系统没有足够的核来用于每个虚拟机，因此开始出现处理瓶颈(例如，4个虚拟机总共需要14个核：每个处理器2个核用于NUMA意识接收数据包，4个核用于虚拟机之间传输/转发，每个虚拟机2个核用于应用程序级处理)。我们相信，更强大的系统应该能够很容易地使用我们的体系结构来支持更长的链。

对于更现实的场景，我们考虑一个链，其中40%的传入流量只由第一个虚拟机（L2交换机）处理，而其余60%通过防火墙虚拟机发送，然后由L3交换机虚拟机（例如，负载平衡器）。在这种情况下，我们的测试机有足够的CPU容量来实现线路率

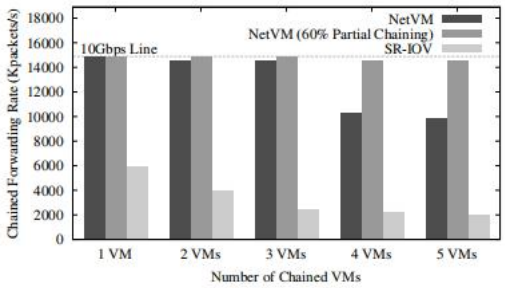

图12：当虚拟机在不同CPU核心(最多3个VM)时，使用NetVM的虚拟机间通信可以实现线率速度。

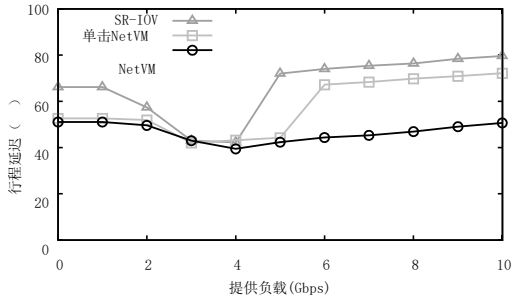

图13：L3转发的平均往返延迟。

这三个虚拟机链，如果在链的末端添加额外的L3开关虚拟机，就只能看到一个很小的减少。相比之下，SR-IOV的性能受到IOTLB缓存缺失的负面影响，以及在vm之间移动的高数据复制成本的影响。输入/输出内存管理单元(IOMMUs)使用IOTLB来加快地址分辨率，但每个IOTLB都会显著增加DMA延迟，DMA密集型包处理[36,37]的性能下降。

5.3 延迟

虽然保持行速率吞吐量对于网络内服务至关重要，但对于尽量减少处理元素添加的延迟也很重要。我们通过测量每个平台上L3转发的平均往返延迟来量化这一点。通过循环回通过平台发送的64字节数据包，在流量生成器上进行测量。我们在传输的数据包上包含了一个时间戳。图13显示了这三种情况下的往返延迟：NetVM、Click-NetVM和使用相同的L3转发功能的SR-IOV。Cleck-NetVM和SR-IOV的延迟增加，特别是在较高的负载时，当过载下有额外的数据包处理延迟时。我们推测，在非常低的输入速率下，没有一个系统能够充分利用批处理dma和内核之间的管道，这解释了所有方法最初略差的性能。在提供的负载超过5Gbps后，SR-IOV和单击无法跟上，导致大部分数据包被丢弃。在本实验中，队列长度相对较小，防止了延迟的显著上升。SR-IOV的下降率在10Gbps时上升到60%，而NetVM下降为零。

5.4 CPU时间分解

表1分解了通过NetVM转发数据包的CPU成本。成本从至强的循环计数器[38]转换为纳秒。每次测量都是超过10秒的测试的平均值。这些测量比真实值要大，因为使用Xeon循环计数器具有显著的开销(所实现的吞吐量从10Gbps下降到8.7Gbps)。

由NetVM的CPU执行的大多数任务都包含在表中。“nic” 管理程序“测量DPDK从网卡的接收DMA环读取数据包所需的时间。然后NetVM决定将哪个虚拟机发送给数据包，并在虚拟机的接收环中放置一个小的数据包描述符(“管理程序管理符”Vm”)。这两个操作都是由一个核心执行的。“vm APP”是NetVM需要从环形缓冲区中获取数据包并将其传递到用户应用程序的时间；然后应用程序花费“APP(L3转发)”时间；转发应用程序(NetVM或Click)将数据包发送回VM(“APP” 而NetVM将其放入虚拟机的传输环缓冲区(“VM” 高管”)。最后，管理程序花费“管理程序” 网卡”发送时间到网卡传输DMA环的数据包。

Core#列演示了数据包描述符如何通过不同的核心通过管道处理不同的任务。如第3.3节所述，数据包处理被限制在相同的套接字上，以防止NUMA开销。在这种情况下，只有“APP（L3转发）”会读写数据包内容。

5.5 灵活性

NetVM允许灵活的交换功能，这也可以帮助提高性能。虽然IntelSRIOV只能根据L2地址交换数据包，而NetVM可以引导流量（每个数据包或每个流）到一个spe-

核心#	任务	时间（ns/包）	
		简单	单击
0	NIC→管理器	27.8	27.8
0	管理器→VM	16.7	16.7
1	vm→应用程序	1.8	29.4
1	APP（L3转发）	37.7	41.5
1	APP→vm	1.8	129.0
1	VM→管理器	1.8	1.8
2	高管→NIC	0.6	0.6
总数		88.3	246.8

表1：NetLib的简单L3路由器的CPU时间成本分解并单击CPU路由器。

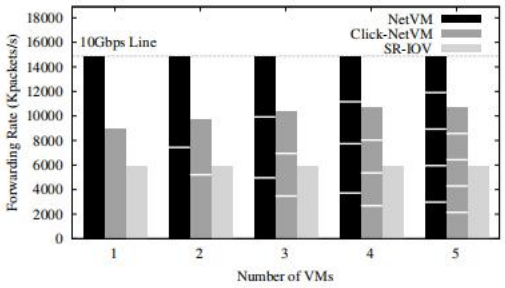

图14：依赖状态（或依赖数据）的负载平衡能够灵活地转向流量。该图显示了一个均匀分布的负载平衡。

面对性能下降，这取决于系统负载（例如，使用包描述符环的占用情况作为指示）、浅包检查（头检查）或深度包检查（头+有效负载检查）。图14说明了负载均衡基于队列分组的负载时，队列数最小的转发率。堆叠的条形图显示了每个虚拟机接收到的流量和总数。NetVM能够在虚拟机之间均衡平衡负载。Click-NetVM显示了使用多个虚拟机的显著性能改进（高达20%），因为额外的核心能够负载平衡更昂贵的应用程序级处理。SR-IOV系统根本无法以这种方式使用多个虚拟机，因为来自数据包生成器的MAC地址都是相同的。向单个SR-IOV虚拟机添加更多的核心也不能提高性能。我们相信这将在网络中是一个现实的场景（不仅仅是在我们的测试台），因为中框或路由器传入包的MAC地址在所有包中可能是相同的。

我们还观察到NetVM的浅数据包检查与基于协议类型的负荷平衡相同的性能图；深度数据包检查开销将取决于分析数据包时所需的计算量。随着部署了许多不同的网络功能，更多具有SDN功能的动态工作负载将留给未来的工作。

6 讨论

我们已经证明了NetVM的零拷贝数据包交付框架可以有效地为通过虚拟化网络平台移动的网络流量带来高性能。在这里，我们将讨论相关的问题、局限性和未来的发展方向。

扩展到下一代机器：在这项工作中，我们使用了支持英特尔DPDK的第一个CPU版本(Nehalem架构)。来自英特尔的下一代处理器，桑迪桥和常春藤桥处理器有显著的额外硬件能力（即核心），所以我们预计这将允许两者更大的总吞吐量(通过并行连接到多个NIC端口)，以及更深的虚拟机链。商业媒体和供应商声明的报告表明，本机Linux（即非虚拟化）的核心数量几乎有线性改善。由于NetVM消除了其他虚拟IO技术，如SR-IOV的开销，我们也希望通过添加更多的核心和nic来看到同样的线性改进。

使用NetVM构建边缘路由器：我们认识到NetVM作为网络元素的能力，比如ISP上下文中的边缘路由器，依赖于拥有大量的接口，尽管速度较低。虽然COTS平台可能有有限数量的网卡，但每个网卡都为10gbps，但低成本的第2层（以太网）交换机和NetVM的明智组合可能是当前边缘路由器平台（通常成本高的边缘路由器平台的替代品）。由于在边缘路由器平台上所需的特性和能力(在策略和QoS方面)通常更为复杂，因此ASIC实现的成本往往会急剧上升。这正是最近处理器与NetVM架构相结合的额外处理能力是一个极具吸引力的替代方案。对低成本的L2交换机的使用提供了必要的多路复用/多路复用，以补充NetVM吸收复杂功能的能力，这可能与这些功能的动态组成相结合。

开放vSwitch和SDN集成：SDN使控制平面管理具有更大的灵活性。然而，交换机和路由器的硬件实现的限制常常阻止SDN规则不基于简单的包头信息。开放的vSwitch实现了更大的网络自动化和可重构性，但由于需要复制数据，其性能受到了限制。我们在NetVM的目标是建立一个基础平台，可以提供更大的灵活性，同时提供高速数据移动。我们的目标是将开放的vSwitch功能集成到我们的NetVM管理器中。这样，使用OpenFlow来自SDN控制器的输入就可以用来指导NetVM的管理和切换行为。NetVM在解复用方面的灵活性可以适应更复杂的规则集，这可能允许SDN控制原语进化。

其他管理程序：我们的实现使用KVM，但我们相信NetVM体系结构可以应用于其他虚拟化平台。例如，类似的设置可以应用于Xen；NetVM核心将在Domain-0中运行，并且Xen的授予表功能将用于直接共享用于存储数据包数据的内存区域。然而，Xen对大型页面的有限支持将必须得到加强。

7 相关工作

多核和多处理器系统的引入导致了基于软件的路由器能力的重大进步。RouteBricks项目试图通过利用CPU和服务器级[39]的并行性来提高软件路由器的速度。同样，Kime人。al.[11]演示了批处理I/O和CPU操作如何提高多核系统上的路由性能。包而不是使用常规的CPU核心，显色器[28]利用通用图形处理单元(GPGPU)的能力来加速包处理。另一方面，超交换机[40]使用了一种考虑CPU缓存本地性的低开销机制，特别是在NUMA系统中。所有这些方法都表明，阻止Click[10]等软件路由器执行行速率处理的内存访问时间瓶颈正在开始改变。然而，这些现有的方法都不支持在虚拟环境中部署网络服务，我们认为这一要求对于低成本的COTS平台取代有目的地的硬件并提供自动化、灵活的网络功能管理至关重要。

由于希望在软件中实现网络功能，从而在COTS硬件上实现灵活性和降低成本，最近形成了具体的形状，许多网络运营商和供应商开始在各种行业论坛上合作。特别是，由欧洲电信标准研究所(ETSI)牵头的关于网络功能虚拟化(NFV)的工作最近概述了[41,42]的概念。虽然NFV在降低设备成本和功耗、提高灵活性、减少部署时间和在一个平台上启用多个应用程序（而不是网络中有多个特定的网络设备）方面的好处是明显的，但实现高性能仍然存在突出的问题。为了实现完全能力的NFV，需要高速数据包传递和低延迟。NetVM为实现这一点提供了基本的基础平台。

在虚拟化环境中提高I/O速度长期以来一直是一个挑战。桑托斯等人。通过优化Xen的驱动程序域模型来降低千兆以太网网[43]的执行成本来缩小性能差距。vBallen动态和自适应地将中断从抢占的vCPU迁移到正在运行的vCPU，从而避免中断处理延迟，以提高SMP-VMs[44]的I/O性能。VTurbo通过将该任务卸载到一个称为turbo核心的指定核心来加速虚拟机的I/O处理，该核心的运行时间段比生产虚拟机[45]共享的核心要小得多。VPE通过使用专用的CPU核心[46]，提高了I/O设备虚拟化的性能。

然而，这些都不能实现以10gbps或更高速度运行的网络链路的全线率数据包转发（和处理）。虽然我们的平台基于DPDK，但其他方法，如网图[47]也为用户空间I/O提供高速网卡。

研究人员已经研究了大宗商品服务器上的中点框虚拟化。Split/Merge[48]描述了一种新的抽象（Split/合并）和一种系统（自由流），它使有状态的虚拟中框具有透明、平衡的弹性，从而能够动态迁移流。XOMB[6]提供了基于大宗商品服务器和操作系统的灵活、可编程和增量可扩展的中端框，以实现高可伸缩性和动态流管理。CoMb[8]解决了在中框部署中利用整合的好处所带来的关键资源管理和实现挑战。这些系统提供了灵活的网络管理，是对NetVM的高速数据包转发和处理能力的补充。

8 结论

我们描述了一个高速网络数据包处理平台NetVM，它由使用虚拟化的商用服务器构建。通过利用Intel的DPDK库，NetVM在管理程序的控制下提供了灵活的交通转向能力，克服了现有流行的SR-IOV硬件交换技术的性能限制。NetVM提供了平台上链网络功能的能力，提供了包含多种功能的灵活、高性能的网络元素。同时，NetVM允许将虚拟机分组到多个信任域中，允许一个服务器安全地多路复用，用于竞争用户的网络功能。

我们已经演示了如何解决NetVM的设计和实现挑战。我们的评估显示，NetVM在转发功能和跨多个虚拟机的功能方面优于当前基于SR-IOV的系统，无论是在高吞吐量还是减少数据包处理延迟方面。NetVM在包交换/多路复用方面提供了更大的灵活性，包括支持状态相关的负载平衡。NetVM表明，多核处理器和NIC硬件的最新进展已经将瓶颈从基于软件的网络处理转移开来，即使对于通常具有更大IO开销的虚拟平台也是如此。

确认

我们感谢我们的牧羊人杨松公园和审稿人帮助改进这篇论文。这项研究得到了国家科学基金会CNS-1253575的部分支持。

参考文献

[1] 余明兰、何西和瑞苗。软件定义的流量测量。在第十届USENIX网络系统设计和实现会议的论文集上，nsdi‘13,29-42页，加州伯克利，美国，2013年。USENIX协会。

[2] 克里斯托弗·孟山都，约书亚·赖希，内特·福斯特，詹妮弗·雷克斯福德和大卫·沃克。组成由软件定义的网络。*在第十届USENIX网络系统设计和实现会议的论文集上，nsdi‘13,1-14页，美国加州伯克利，2013年。USENIX协会。*

[3] 艾哈迈德、周文轩、马修·凯撒和P。布赖顿戈弗雷。验证流：实时验证网络范围内的不变量。*在关于软件定义网络中的热门话题的第一个研讨会的论文集中，HotSDN‘12，第49-54页，纽约，纽约，美国，2012。*ACM。

[4] 本普法夫，贾斯汀佩蒂特，特蒙科波宁，基思阿米顿，马丁卡萨多，和斯科特申克。将网络扩展到虚拟化层。*在第8届ACM网络热门主题研讨会上(HotNets-VIII)。纽约市（2009年10月）。*

[5] 英特尔公司。英特尔数据平面开发工具包：入门指南。2013.

[6] 詹姆斯W。安德森，瑞安布劳德，里希卡普尔，乔治波特，和阿明瓦赫达特。Xomb：可扩展的具有大宗商品服务器的开放中点框。*在第八届ACM/IEEE网络和通信系统架构研讨会的论文集中，ANCS‘12，第49-60页，纽约，纽约，美国，2012年。*ACM。

[7] 亚当格林哈尔，费利佩胡奇，米克尔霍尔特，帕帕迪米里奥，马克汉德利和洛朗马西。流程处理与商品网络硬件的兴起。*Sigcomm组合。评论。牧师。*，39（2）：2009年3月20-26日。

[8] 赛卡尔、艾吉、拉特纳萨米、赖特、石光宇。设计和实现一个统一的中点框架构。在第9届USENIX网络系统设计与实现会议论文集上，NSDI‘12,24-24页，加州伯克利，美国，2012年。USENIX协会。

[9] 拉菲尔·博拉和罗伯托·布鲁斯奇。基于pc的软件路由器：高性能和应用程序服务支持。在明天可扩展服务可编程路由器ACM研讨会论文集中，2008年，第27-32页，纽约，纽约，美国，2008页。ACM。

[10] 埃迪科勒。单击模块化路由器。博士论文，2000。

[11] 金俊、徐谷、张成泽、朴智园和苏文。点击模块化路由器的处理能力。在《亚太地区系统研讨会论文集》中，APSYS‘12，第14：1-14：6页，纽约，纽约，美国，2012年。ACM。

[12] 君士坦丁诺斯，多罗利斯，布拉德，塞耶和帕拉姆瓦兰，拉马纳坦。 髋关节：针对网络接口的混合中断轮询。*ACM操作系统评论，2001年35：50-60。*

[13] 杨吉友，明特恩和弗兰克哈迪。当投票比中断更好时。在第10届USENIX文件和存储技术会议论文集上，FAST‘12，第3-3页，伯克利，美国，2012。USENIX协会。

[14] 杰弗里C.莫格尔和K.K.拉马克里什南。在中断驱动内核中消除接收生命。*ACM计算机系统学报，1997年15：217-252。*

[15] 吴文姬，马特克劳福德和马克鲍登。linux网络数据包接收的性能分析。*组合。评论。*，30（5）：1044-1057,2007年3月。

[16] 小云，卡尔顿普，萨班巴蒂亚和查尔斯康塞尔。在用户级操作系统中的高效数据包处理：对uml的研究。在2006年第31届IEEE本地计算机网络会议记录(LCN06)，2006。

[17] 英特尔公司。英特尔虚拟化技术的定向输入/o。2007.

[18] 英特尔公司。英特尔数据平面开发工具包：程序员指南。2013.

[19] 打开vSwitch。http://www.openvswitch.org.

[20] VMWare白皮书。虚拟网络分布式交换机。*2013.*

[21] 英特尔开源技术中心。https://01.org/packet-processing.

[22] 英特尔公司。英特尔数据平面开发工具包：入门指南。2013.

[23] 弗朗西斯m.大卫，杰弗里C.卡莱尔，和罗伊H。坎贝尔。*臂式平台上的linux的上下文交换开销。在2007年实验计算机科学研讨会的论文集上，ExpCS‘07，纽约，纽约，美国，2007。*ACM。

[24] 李传鹏、陈丁、沈凯恩。*量化上下文切换的成本。在2007年实验计算机科学研讨会的论文集上，ExpCS‘07，纽约，纽约，美国，2007。*ACM。

[25] 杰夫迪安。来自构建大型分布式系统的设计、经验教训和建议。LADISKeypont，2009。

[26] 斯里哈里马基尼尼，拉维伊耶尔，帕塔萨兰甘，唐纳德纽维尔，李赵，拉梅什伊利卡尔和杰德普摩西。交流电源的接收侧合并

加速tcp/ip处理。在第13届高性能计算国际会议的论文集上，HiPC ‘06，第289-300页，柏林，海德堡，2006年。施普林格公司。

[27] 风河白皮书。高性能多核网络软件设计选项。2013.

[28] 韩三津、张成、庆秀园、苏文。软件包着色器：一个gpu加速的软件路由器。在ACM2010签名通信会议记录中，签名通信第10页，195-206页，纽约，纽约，美国，2010。ACM。

[29] 李一南、潘迪斯、穆勒、拉曼和罗曼。numa感知算法：数据洗牌的情况。*2013年创新数据系统研究两年期会议(CIDR)。*

[30] 大卫·列文塔尔。英特尔核心i7处理器和英特尔xeon5500的性能分析指南。2013.

[31] A. 卡梅隆麦克唐内尔。针对虚拟机的共享内存优化。*博士论文。*

[32] 生锈的罗素和哈拉尔德·韦尔特。Linux净 过滤器 黑 客howto.*http://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO.html.*

[33] 风河技术报告。风力河流应用加速发动机。2013.

[34] 英特尔公司。英特尔数据平面开发工具包：示例应用程序用户指南。2013.

[35] 凯伦斯卡芬和保罗霍夫曼。关于防火墙和防火墙策略的指南。*美国国家标准与技术研究所，2009年。*

[36] 阿米特，穆里本耶胡达，和本阿米亚苏尔。*减轻iotlb瓶颈的策略。在2010年计算机架构国际会议论文集上，ISCA ‘10页，256-274页，柏林，海德堡，2012年。*施普林格-滞后。

[37] 穆里本-耶胡达、吉米塞尼迪斯、米卡尔奥斯特罗斯基、卡尔里斯特、亚历克西斯布鲁默和伦德特范多恩。*安全性价格：评价iommu性能。在2007年Linux研讨会论文集，2007。*

[38] 英特尔公司。Intel64和ia-32架构软件开发手册。2013.

[39] 米海多布雷斯库、诺伯特埃吉、卡特琳娜、梁春、凯文福尔、吉安卢卡伊纳卡内、艾伦尼斯、马齐亚马尼什和西尔维亚拉特纳萨米。路由砖：利用并行性来扩展软件路由器。*在ACMSIGOPS第22届操作系统原理研讨会的论文集中，SOSP ‘09，第1528页，纽约，纽约，美国，2009年。*ACM。

[40] 库马尔公羊，艾伦L。考克斯，梅胡尔·查达，和斯科特·里克斯纳。超开关：可伸缩软开关软件的虚拟交换机架构。*USENIX年度技术会议(USENIXATC)，2013年。*

[41] SDN和开放流世界大会的介绍性白皮书。网络功能的虚拟化。*http://portal.etsi.org/NFV/NFV白色Paper.pdf，2012。*

[42] 弗兰克 越。 网络 函数 虚拟化 - 一切 旧的 是 新的 再次。*http://www.f5.com/pdf/white-papers/service-provider-nfv-white-paper.pdf，2013.*

[43] 桑托斯、特纳吉夫、贾纳基拉曼和普拉特。缩小i/o虚拟化的软件和硬件技术之间的差距。在USENIX2008年度技术会议年会，ATC ‘08，第29-42页，伯克利，美国，2008年。USENIX协会。

[44] 程路伟和王赵丽。我们将中断负载平衡，以提高smp虚拟机的i/o性能。*在第三届ACM云计算研讨会论文集上，SoCC ‘12页，第2：1-2：14，纽约，纽约，美国，2012。*ACM。

[45] 徐丛、萨韩游戏、许慧、无伴奏合唱、徐东岩。Vturbo：使用指定的涡轮切片核心加速虚拟机i/o处理。*USENIX年度技术大会，2013年。*

[46] 刘九星和恶霸阿里。虚拟化轮询引擎(vpe)：使用专用的cpu核心来加速i/o虚拟化。在第23届超级计算国际会议论文集，ICS09,225-234页，纽约，美国，2009年。ACM。

[47] 路易吉·里佐。一种新的快速数据包输入/输出的框架。在USENIX年度技术大会，101-112页，伯克利，2012页。美国尼克斯。

[48] 拉贾戈帕兰，丹威廉姆斯，哈尼贾姆约姆，和安德鲁沃菲尔德。拆分/合并：系统支持虚拟中框中的弹性执行。在第10届USENIX网络系统设计与实现会议论文集中，nsdi ‘13页，227-240页，加州伯克利，美国，2013年。USENIX协会。