OpenNetVM：一个面向高性能网络服务链的平台

张伟[*] 刘月[*] 张文辉[*] 内尔沙[*] 菲尔·洛普雷亚托[*] Todeschi[†]
K.K.拉马克里什南[†] 蒂莫西木材[*]
[*]乔治华盛顿大学 [‡]INP ENSEEIHT [†]加州大学河滨分校

摘要

正如软件定义网络(SDN)的研究和产品开发随着多个开源SDN平台的发布而大大加快一样，我们相信网络功能虚拟化(NFV)研究可以看到类似的增长，开发一个实现高性能NFV实现的灵活高效平台。现有的NFV研究原型提供的性能、灵活性或隔离性不足。此外，英特尔的DPDK等高性能I/O平台缺乏更高层次的抽象。我们提出了OpenNetVM，一个高效的数据包处理框架，它极大地简化了网络功能的开发，以及它们的管理和优化。OpenNetVM基于NetVM体系结构，在轻量级的Docker容器中运行网络功能。OpenNetVM平台管理器提供了负载平衡、灵活的流管理和服务名称抽象。OpenNetVM使用DPDK进行高性能的I/O，并通过动态创建的服务链有效地路由数据包。我们的评估在跨两个NF副本的负载平衡时达到68Gbps，在穿越5个nf链时实现40Gbps，实现了在生产网络中部署软件服务的潜力。

1. 介绍

网络功能虚拟化(NFV)承诺使大量的网络内软件功能在虚拟化环境中有效运行。网络和数据中心运营商都设想了可以作为虚拟中点框部署的新功能，消除了过去基于硬件的方法的成本和不灵活性。这些功能范围从轻量级的软件交换机，高每-

2016年8月22日至26日，巴西佛罗里亚诺波利斯

ⓒc2016ACM。ISBN978-1-4503-4424-08年16月16日。..$15.00
Doi：http://dx.doi.org/10.1145/2940147.2940155

对复杂入侵检测系统(IDS)的一致性代理引擎。提供商和从业人员试图将这些功能部署为包含多个网络功能(NFs)的复杂服务链的一部分。

最近，一些高性能的I/O库，如网图、DPDK和PF_RING，允许开发人员和研究人员构建高效的NF原型。这些库通常通过避免内核的网络堆栈并允许从用户空间应用程序直接访问包数据，从而实现10gbps或更高的包处理速率。虽然这是加速单个应用程序有好处，但这些库并不帮助NFs的组成或管理。E.g.、SoftNIC[2]和DPDK管道[5]只支持固定的、预定义的服务链，它们不能动态地引导数据包到NFs。此外，由于像DPDK这样的库假设完全控制NIC端口并将它们专用于单个进程，因此不可能在同一服务器上运行几个加速函数，除非它们经过精心设计以相互共存。因此，当构建网络功能的低级工具越来越可用时，我们缺乏一个平台来提供所需的高级抽象，以将它们组合成服务链，控制它们之间的数据包流，并管理它们的资源。

从这些需求中，我们推导出三个设计原则指导我们的架构：

- NF管理框架必须具有轻量级和高效，同时提供灵活的数据包转向。
- 管理框架和nf都必须控制如何通过服务链管理数据包。NFs必须被隔离，快速部署，并易于由不同的供
- 应商开发。

OpenNetVM通过提供一个灵活和高性能的NFV框架来支持一个“智能”数据平面，从而包含了这些原则。我们的工作是基于为我们的NetVM平台[3]开发的体系结构，但扩展到支持灵活的管理功能、更轻的NFs以及改进的部署和互操作性。

基于容器的NFs：网络函数作为Docker容器中的标准用户空间进程运行，使得它们比虚拟机更轻，但仍然允许一个通用的linux开发环境。这大大简化了多个供应商的NF部署，因为容器是独立的和模块化的。容器可以是

迅速按需开始，并产生最小的开销。

NF和流管理：OpenNetVM的管理框架跟踪正在运行哪些NFs，并提供服务类抽象(如防火墙、IDS等)。要映射到特定的实例。流表在服务链中的NFs之间引导数据包；这可以由SDN控制器或其他NFs动态配置。通过在管理器和NFs内同时提供控制功能，OpenNetVM减少了与SDN控制器的通信，提高了灵活性和效率。

高效的I/O：使用DPDK消除了内核开销，允许从用户空间轮询模式驱动程序对DMA的数据包进行零拷贝访问。我们将其扩展到支持使用公共安全域内的每个Docker容器可访问的共享内存的多个Docker服务链中的零拷贝I/O。

可伸缩性和优化：NFs可以很容易地复制以实现可扩展性，并且NF管理器将自动跨副本加载平衡数据包，以最大限度地提高性能。该框架通过缓存流表查找等技术进行了仔细的优化，以避免管理层中的瓶颈。

虽然其中一些技术在过去已经被探索过，但我们的贡献在于将它们结合成一个高效和易于使用的平台，可供社区使用。[1]在本文中，我们描述了OpenNetVM体系结构，讨论了保证其性能所需的优化和高效的数据结构，并评估了其与现有方法相比的有效性。

2. 背景及相关工作

NFV通过软件设备，而不是硬件设备[4]来提供网络服务。在硬件中运行网络功能的经典方法，如路由器和防火墙，需要为不同的网络功能提供专门的设备，并且每个设备都需要单独部署。NFV使我们能够在现成的商品服务器上运行NFs，从而简化了来自不同供应商的服务部署。NFV承诺将极大地提高可以部署和修改服务的灵活性，同时降低成本。

最近，一些高性能数据平面平台被提出和开发了这些数据平面平台来支持网络应用。DPDK[5]绕过了传统的基于中断的内核网络堆栈数据包处理中固有的开销，并允许应用程序直接从nic访问数据。PF_Ring[1]通过利用允许用户空间和内核空间访问的环缓冲区来实现线速数据包捕获。Netmap[9]预分配了数据包缓冲区，通过大批量使用减少了系统调用时间，并在用户空间和内核空间之间使用共享内存缓冲区实现了零拷贝。所有这些平台都专注于快速将数据包从网卡传输到用户空间应用程序。

NFV框架可以建立在这些I/O平台之上。ClickOS[7]使用网图[9]和VALE开关[10]可以有效地在运行点击软件路由器元素的轻量级虚拟机之间移动数据包。他们专注于通过解决Xen网络I/O管道中的瓶颈和使用ClickOS来提高其网络性能，ClickOS是一种重量轻、为网络包处理定制的快速启动的迷你操作系统。这允许密集和快速地部署nf，我们通过使用容器来实现这一点。ClickOS还支持可以共享cpu的中断驱动的NFs；OpenNetVM将核心奉献给NFs，它可以消耗更多的资源，但提供更高的性能，如我们的评估所示。此外，ClickOS为NFs提供了一个相当有限的开发环境，因为它们必须在Click框架的规范中设计，并且不在标准的Linux环境中运行。该框架也是为函数的静态服务链而设计的，并且不提供nf对分组如何在虚拟机之间路由的灵活控制。

E2[8]和OpenNetVM一样，都是使用DPDK构建的。E2是a端到端编排的框架，包括位置、资源和元数据管理，以及服务链。然而，E2具有固定的服务链——阻碍了新服务的动态实例化、灵活的数据包路由以及来自竞争供应商的NFs的部署。

虽然这些项目说明了在商品服务器上以线率处理数据包的潜力，但它们缺乏完整NFV平台所需的灵活性和隔离。OpenNetVM试图通过添加更高层次的抽象来填补这一差距，如服务链、动态NF实例化、灵活的流控制、负载平衡和命名，同时仍然保持底层DPDK平台的高效率。

Docker和LXC等容器技术最近作为服务器虚拟化平台的替代方案而广受欢迎。容器使用内核提供的名称空间和资源隔离来将进程或进程集封装在一个轻量级胶囊中。由于容器在同一主机操作系统中作为进程运行，因此它们之间的内存共享(对于OpenNetVM的零拷贝I/O非常重要)，因为可以使用现有的内存映射机制，因此将大大简化。容器也比虚拟机消耗更少的资源，因为它们不包括自己的操作系统，而是依赖于主机的内核。虽然这阻止容器运行完全不同的操作系统(例如，Linux主机上的Windows容器)，但容器仍然可以封装与主机完全不同的库和进程(例如，使用自定义版本的libc运行RedHat发行版的容器可以在Ubuntu主机上运行)。

不像虚拟机是由磁盘定义的虚拟机图像的大小可能是g，容器是由要安装的包和文件的配置列表定义的。这大大简化了容器的部署和整理，因为它们的配置文件可以很容易地复制、修改和版本化。OpenNetVM使用Docker容器，这意味着NFs可以在DockerHub图像存储库上轻松地共享。使用容器来运行网络函数提供了许多好处：通过dif-

[1] 来源 编码 和 NSF 云实验室 图像 在http://sdnfv.github.io/

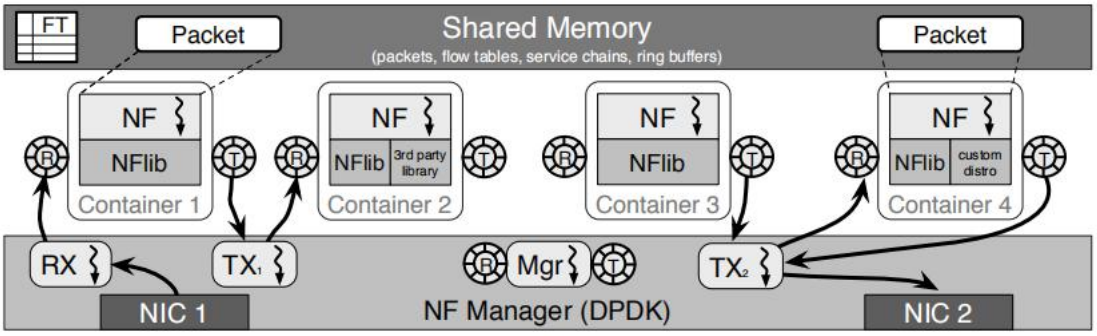

图1：NF管理器创建一个共享内存区域来存储数据包和元数据，如流表和服务链列表。数据包通过RX和TX线程在NFs之间移动，这些线程将数据包描述符复制到NF的接收(R)并传输(T)环缓冲区中。NFs在封装所有依赖项的孤立容器中运行。

狂热的供应商可以在保持隔离的同时轻松地共存，每个容器都可以包含自己的库和依赖项，并且可以精确地分配资源。

3. OpenNetVM架构

如图1所示，OpenNetVM分为NF管理器，该管理器与NIC接口，维护流表，促进NFS之间的通信；NFLib，在NF中提供API与管理器交互的API；以及使用该API的用户编写的NFI。NF管理器通常在主机系统上运行，尽管如果需要，它可以在容器内运行。NFs与NFlib和任何所需的第三方库一起编译，作为Docker容器运行。

3.1 NF经理

NF管理器通过管理NFs和路由数据包来维护网络状态。该管理器由三种类型的组件组成：一个配置共享内存并跟踪活动NFs的管理器线程，从NIC读取数据包的RX线程，以及在不同NFs之间移动数据包的TX线程。

内存管理：当OpenNetVM首次启动时，管理器线程使用DPDK库在大型页面中分配内存池，以存储传入的数据包。这些池被划分为所有可用的CPU套接字，因此数据包将只能由保存它们的内存dimm的本地套接字中的核心进行处理。当NFs启动时，它们会在配置文件中找到共享的内存地址，然后使用与管理器相同的基本虚拟地址来映射内存区域。这确保了包含数据存储位置的虚拟地址的数据包描述符可以被管理器和nf正确地解释，而不进行任何翻译。与我们之前在NetVM[3]中采用的方法相比，这大大简化了内存共享。在NetVM[3]中，需要仔细翻译地址，以允许基于KVM的NFs通过模拟的PCI设备进行访问。

NF管理：在执行其初始化例程后，管理器线程开始通过轮询共享内存区域中的消息队列来检查新的NFs。新的NFs使用此消息通道通知管理器，以便将其添加到活动NFs的注册表中；同样，当NFs关闭时，它们会通知管理器，以便它可以清理任何相关的数据结构。消息队列（作为环缓冲区实现）在整个OpenNetVM架构中使用，因为它们允许高效的、异步的通信。除了管理器的消息环外，每个NF在共享内存中还创建了两个描述符环缓冲区（接收和传输）。

数据包流：管理器的RX和TX线程处理NFs和NIC端口之间的数据包路由。数据包最初基于数据包的接收端缩放(RSS)散列到达NIC端口上的队列中。DPDK轮询模式驱动程序确保包数据直接将DMA发送到共享内存池中，并且将包描述符（即地址和元数据）复制到与固定在内存池同一套接字上的RX线程相关联的队列中。RX线程将从其队列中删除一批数据包，然后单独检查它们，以决定如何将它们路由到目标NF的接收环。TX线程执行类似的功能，但它们不是从NIC队列读取，而是从NFs的传输环缓冲区读取，然后将数据包发送出NIC端口或发送到不同的NF接收环。数据包可以使用流程表或默认服务链进行指导。

OpenNetVM使用TX线程来提供一个抽象NFs之间的一层。在其他系统中，如Click[6]和E2[8]，可以将NFs的服务链一起编译成一个进程；因此，在它们的情况下，在服务链之间移动只是一个新的过程调用。虽然这种方法可以更有效，但它限制了灵活性，因为服务链顺序通常是硬编码的，不能由管理实体进行动态调整，也不能因为资源可用性限制或工作负载需求而动态移动NF。相比之下，OpenNetVM的TX线程可以根据最后一个NF请求的操作或流表查找来确定如何路由数据包。TX线程还提供了跨NF副本的负载平衡功能，而不需要NFs知道当前正在运行的其他容器。

3.2 网络功能

OpenNetVM网络函数使用我们的NFLibAPI与管理器交互。当NF进程启动时（在本机中或在容器中），它会调用一个向管理器注册其服务类型的初始化函数。NFlib运行函数，然后轮询其接收环，以寻找来自NIC或其他NFs的数据包。对于接收到的每个数据包，NFlib将执行由NF开发人员提供的回调函数来处理该数据包。NFlib处理与共享内存环的所有交互，以及用于传输的批处理数据包。我们目前的版本只支持具有专用CPU核心的“轮询”NFs，尽管我们正在研究对可以共享核心的中断驱动NFs的支持。

当NF运行其回调来处理传入的包时，它会得到一个包中包含一些元数据的地址的描述符。NF将执行所期望的功能，如路由、深度包检查(DPI)、入侵预防等。然后，NF可以指示随后应该对数据包执行什么操作：在流程表中查找，发送到另一个NF，发送到网卡，或删除。默认情况下，NF将发出查找流表操作，从而根据第4节中所述的流表规则确定下一步。另一方面，如果没有使用流表，或者发生了一些异常，NF可以指定要发送到的NF服务类型、要发送的网卡端口，或者可以简单地要求删除数据包。在所有情况下，数据包描述符（包含所请求的操作）都将被放置在NF的传输队列中，这将由管理器的一个TX线程来处理。然后，TX线程就可以执行或否决该操作。

4. 流量管理

OpenNetVM通过多种方式促进了数据包通过nf的服务链的移动。首先，它提供了一个服务类型抽象，以便新的NFs宣布它们提供的函数类型，允许管理器将数据包引导到所需的函数类型，并跨类型副本的负载平衡。OpenNetVM将服务链定义为操作列表，通常是向不同服务类型的一系列发送操作。最后，流控制器提供了一种方便和灵活的方式来匹配单个数据包流到负责它们的服务链。这些规则既可以由NFs提供，也可以通过SDN控制器提供。

4.1 服务类型

当NF启动时，它会向管理器声明其“服务ID”。服务ID是表示一种NF类型的数字标识符。可以同时运行许多具有相同服务ID的NF，但每个都将被分配一个唯一的实例ID。NFs总是由其服务ID处理。基于服务的寻址意味着一个NF不需要知道正在使用的其他实例id（因为这可能会随着时间的推移而改变）。相反，NF只需将其数据包发送到所需的服务，然后管理器决定使用哪个实例。另外，如果一个NF意外退出，那么数据包流将继续不中断，因为管理器将数据包路由到相同类型和服务ID的其他nf。基于服务ID而不是实例将数据包寻址到nf中，还允许管理器轻松地加载平衡数据包。当管理器执行服务ID查找时，它将确定相关实例ID的列表。然后，管理器将包的对称RSS散列(来自由包的IP协议、源/目标地址和端口组成的元组)调制所需类型的可用NFs的数量，以选择将包路由到哪个实例。这种方法确保，只要NFs没有启动和停止，相同（双向）流中的数据包总是被路由到它们通过的每个服务类型的相同实例，因为RSS散列不改变。

4.2 服务链

虽然OpenNetVM试图提供数据包如何通过数据平面的重要控制，但网络管理员通常将提供为每个服务链定义的一组默认规则。服务链是流必须遵循的路径上的操作列表。例如，系统可能希望所有数据包传输到服务ID3，然后是5，然后在网卡上发送出端口3。NF可以设置这个默认的服务链，然后服务3和服务5中的NFs只需使用流表操作才能使数据包继续通过服务链。如果需要（并且管理器的配置允许），如果NFs希望特定的替代服务包不遵循默认路由，那么他们仍然可以将数据包发送给特定的替代服务包。

4.3 流量总监和流程表

NF管理器中的OpenNetVM的流控制器组件包含一个流表，它将数据包流映射到服务链。与不支持服务链的英特尔的流控制器等硬件技术相比，在管理层中这样做在如何管理数据包方面提供了更大的灵活性。当TX线程必须在服务链中移动数据包时，首先需要查找要使用哪个链（如果给定流不存在默认服务链）。流控制器还向NFs公开一个API，以便它们可以动态地分配映射到每个流的服务链。

OpenNetVM包括一个SDNNF，它将通过OpenFlow协议联系SDN控制器，以确定每个流所需的服务链。然后，SDNNF使用流控制器API来配置管理器的流表。这种方法使管理器尽可能简单；它只是强制执行流表规则，同时允许其他应用程序提供策略来设置它们。

当在数据平面的关键路径上执行流表查找可能会很昂贵——我们在5.2节中的结果显示，在每个查找上散列包头的朴素实现降低了50%以上的吞吐量。因为类似流表的数据结构，即散希表从

OpenNetVM是许多nf常见的数据流，它提供了一个基于DPDK布谷鸟哈希实现的包查找优化的流表库。

NF管理器的流控制器使用库创建到服务链的表映射流，但是其他NF，如IDS可能使用表来存储关于每个流的状态。OpenNetVM没有重新计算每次查找的数据包散列，而是重新使用NIC在硬件中计算的RSS散列，以简化NF管理器内和NFs内流表的数据包查找。这种优化可以提供一个显著的性能改进，特别是对于

包含使用散列表存储流状态的多个Nf的服务链。

5. 评价

在本节中，我们将根据服务链性能、流表开销和多端口可扩展性来评估OpenNetVM。

我们的实验设置由两类服务器组成。为了流量生成(通过Pktgen-DPDK)和需要单个网卡端口的实验，我们使用HP服务器的In-tel-tenXeOUX5650@2.67GHz（6核）、64GB内存和Intel82599ES10GNI卡。我们还使用了一台戴尔机器，它带有IntelXeonCPUE5-2697v3@2.60GHz（14核）、164GB内存和4个双端口网卡[2]。HP服务器运行Ubuntu14.04.3（内核3.19.0），Dell运行Ubuntu12.04.5（内核3.2.0）和Dockerv1.9.1。

5.1 服务链性能

我们首先评估了OpenNetVM作为进程和Docker容器运行的服务链性能，并与使用Xen虚拟机的ClickOS进行了比较。为了避免NIC处理造成的开销差异，我们让链中的第一个NF创建一组数据包，然后在单个主机上反复发送，每个NF只需简单地将数据包转发给下一个数据包。图2显示了64字节数据包的吞吐量如何随着我们调整链的长度而变化。使用ClickOS，我们无法启动超过三个NFs，但看到了与ClickOS论文相当的趋势。OpenNetVM的管理器每个NF运行一个TX线程，每个线程都有一个核心；当使用ClickOS时，我们分配相同数量的内核。

OpenNetVM在使用流程或容器时的性能差异可以忽略不计；为简单起见，我们剩下的实验是使用流程而不是容器。OpenNetVM比ClickOS具有更高的性能，因为它基于DPDK的轮询模式驱动程序（而不是中断），并且NFs之间的包传输成本更低，因为它避免了内核开销。即使有6个NF链，OpenNetVM的吞吐量也只下降了4%，而ClickOS的3个NFs链也下降了39%。

5.2 流量总监头顶

接下来，我们考虑一个更灵活的情况，即真实数据包通过流表路由，而不是NF中的硬编码规则路由。图3测量了Open-的吞吐量

戴尔机器有三台英特尔10G双端口X520nic卡
第四个是Intel82599EB10G双端口网卡。我们的卡都不能满足64字节包的20gbps，我们认为这是一个硬件限制。

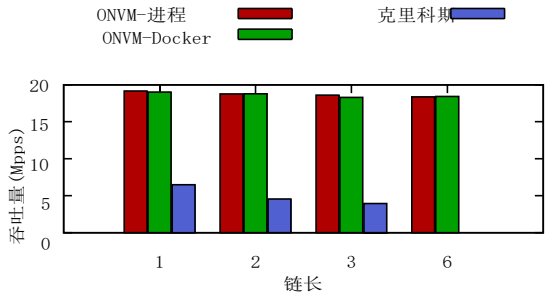

图2：OpenNetVM通过避免e，即使运行通过长服务链，也能实现高吞吐量

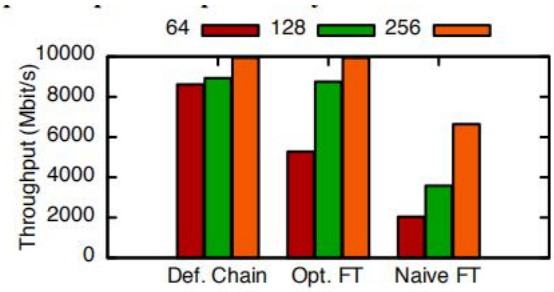

图3：流表查找增加了RX和TX线程的开销，但是我们的优化消除了这种影响，除了64字节的数据包。

当流量通过10Gbps链路从数据包生成器到达单个NF时，NetVM；管理器使用1RX线程和1TX线程。在默认链的情况下，RX线程接收数据包，并使用全局定义的默认链将它们发送到NF，而不进行任何流表查找；即使发送64字节的数据包，这也能达到接近10Gbps的速率。如果RX线程必须使用我们优化的流控制器执行流表查找，那么由于RX线程成为瓶颈，吞吐量就会下降；这可以通过使用多个RX线程来减轻，并且对于合理大小的数据包，可以很容易地满足10gbps的速率。但是，如果使用了一个朴素的流表实现，它必须散列每个数据包的头，吞吐量将进一步下降，即使是256字节的数据包也无法满足行率。

为了精确地测量延迟和避免其他的比较
没有干扰（例如，网卡读写，网络拥塞），我们使用本地生成的数据包测试每个流控制器方法的延迟。我们有一个NF创建一组数据包并将它们发送到TX线程，该线程使用默认服务链或流表查找将数据包发送回NF。使用默认服务链（4.4us）和我们的优化流表查找（6.9us）的延迟是相似的。然而，朴素的流表查找需要
与默认服务链相比，延迟增加了5.68倍。

5.3 多端口可伸缩性

为了评估OpenNentVM在更现实的环境中的规模，我们在Dell服务器上使用8个端口，并从直接连接的HP服务器向其发送数据包。流量生成器回放HTTP数据包的PCAP跟踪：44.3%

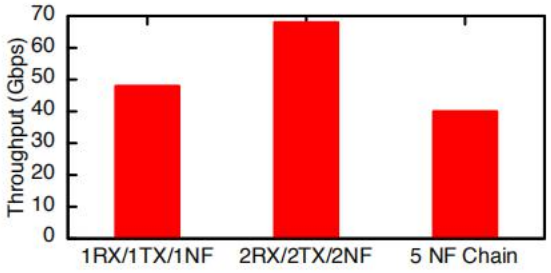

图4：OpenNetVM仅使用6个核就在实际流量上实现了近70gbps。

是64字节，18.5%是65-1023字节，37.2%是10241518字节。数据包被发送到默认NF，默认NF立即将数据包转发回网卡端口。

当配置一个RX线程、一个TX线程和一个NF时，OpenNetVM可达到48Gbps。如果我们启动第二个NF并将另一个添加到RX和TX线程到管理器中，OpenNetVM会自动加载跨NFs的平衡流，以达到68Gbps。这说明了使用OpenNetVM作为软件路由器平台的潜力，甚至是在数据中心网络的深处。

最后，我们重新配置系统，创建一个由5个NFs组成的线性默认服务链，其中包含2个RX和5个TX线程。这个设置的吞吐量为40gbps。请注意，包含5个NFs的链使管理器上的负载增加了5倍——TX线程总共以每秒3400万个数据包的速度引导数据包。这种配置消耗了我们服务器上的所有核心，但我们希望通过额外的核心，可以复制整个服务链，以进一步提高吞吐量。

5.4 柔性夹克转向

最后，我们演示了使用OpenNetVM管理包的不同方法：SDN控制器可以提供流规则，管理器可以将包重定向到负载平衡，NFs可以基于包数据转移流。系统从默认服务链开始，如果出现流控制器失败，该服务链将所有新数据包发送到SDNNF。SDNNF会为每个到达的新流联系我们的SDN控制器。在这个实验中，控制器返回所有流（发送到IDS，发送端口1），但这可以动态调整。一旦规则安装到流控制器中，该流中的后续数据包将直接发送到假的IDSNF，而不是发送到SDNNF。

当IDSNF上的工作负载增加时，我们将使用相同的服务ID启动一个其他副本。在Docker容器中启动此NF从容器启动到NF接收其第一个包平均需要0.526秒（与本机进程的0.524秒的差异不显著）。这比KVM虚拟机要快得多，后者在我们的NetVM平台上初始化至少需要12秒。一旦NF被初始化，管理器将自动加载跨两个副本的平衡数据包。

随后，我们模拟了IDSNF检测到可疑流量的情况。当这种情况发生时，IDS会根据它们的通信条件从原始服务链中转移一些数据包帐篷。IDSNF能够直接引导数据包（受管理器施加的任何约束），而无需与SDN控制器交互。OpenNetVM支持这种类型的智能数据平面行为，以减少SDN控制器上的负载，并增加可以重定向数据包的敏捷性。

6. 结论

OpenNetVM是一个可伸缩和高效的包处理框架，它支持通过服务链动态指导包。与以前使用有限的编程范式或限制性运行时环境的方法不同，OpenNetVM在Docker容器中部署了网络功能，促进了不同服务提供商开发nf的过程，同时最大限度地减少了内存消耗和启动时间。OpenNetVM利用DPDK进行高性能的I/O，并且仅使用6个CPU内核就实现了68Gbps的吞吐量。这为复杂的基于软件的服务在网络和数据中心内部深入运行提供了可能性。我们已经发布了OpenNetVM，供社区使用，在NSFCloudLab平台上提供了源代码和实验模板。我们相信OpenNetVM将为NFV/SDN实验和商业原型开发提供理想的基础。

感谢：这项工作得到了部分由NSF资助了CNS-1422362和CNS-1522546。

参考文献

[1] L.德里和其他人。改进被动数据包捕获：超越设备轮询。在《理智的论文集》中，2004卷，第85-93页。荷兰，阿姆斯特丹，2004年。

[2] S.Han， A. 熊猫，帕尔卡尔， D. 韩和拉特纳萨米。一种用来增强硬件的软件技术。技术报告，技术报告UCB/EECS-2015-155，加州大学伯克利分校EECS系，2015年。

[3] J.Hwang，K.拉马克里什南，和T。木材。*NetVM：使用商品平台上的虚拟化的高性能和灵活的网络。在网络系统设计与实施研讨会上，NSDI14,4月14日。*2014.

[4] E.T.S.研究所。网络功能的虚拟化。在SDN和开放流，2012年的世界大会上。

[5] 英特尔。数据平面开发工具包。

[6] E.科勒。单击模块化路由器。*博士论文，2000。*

[7] J.马丁斯，M。艾哈迈德，C。Raiciu，V.奥尔塔努，M。本田，
R.Bifulco和F。Huici。单击k操作系统和网络功能虚拟化的艺术。在第11届USENIX网络系统设计与实现研讨会(NSDI14)，459-473页，西雅图，4月。2014.USENIX协会。

[8] S.帕尔卡，C。兰，S。韩，K。琼，A。熊猫，S。拉特纳萨米，
L.Rizzo和S.Shenker。E2：一个针对NFV应用程序的框架。在第25届操作系统原理研讨会论文集中，SOSP‘15页，第121-136页，纽约，纽约，美国，2015年。ACM。

[9] L.Rizzo。网图：一种新的快速数据包I/O框架。在USENIX年度技术大会，101-112页，伯克利，2012页。美国尼克斯。

[10] L.Rizzo和G.莱蒂耶里。*VALE，一个针对虚拟机的交换机以太网。在第八届新兴网络实验与技术国际会议论文集上，附件12，第61-72页，纽约，纽约，美国，2012年。*ACM。