专题：SDN/NFV

NFV 数据平面的网络性能优化技术

黄昱恺，耿金坤，令瑞林，李丹

（清华大学，北京 100084）

摘　要：近年来，NFV 因为其部署灵活、成本低廉等优良特性，受到了学术界与工业界的广泛关注。然而，性能问题在 NFV 发展过程中变得愈加明显，逐渐成为制约 NFV 技术进步的主要瓶颈。针对影响 NFV 数据平面网络性能的主要因素进行了归纳概述，分析了 NFV 的性能瓶颈所在，并在此基础上进一步归纳了相应的优化方法。最后，简要介绍了当前比较成熟的两种系统架构，能够显著提升 NFV 数据平面的网络性能。

关键词：NFV；性能优化；系统架构

中图分类号：TP393　　**文献标识码：**A

doi: 10.11959/j.issn.1000−0801. 2017088

Performance optimization solutions for NFV data plane

HUANG Yukai, GENG Jinkun, LING Ruilin, LI Dan

Tsinghua University, Beijing 100084, China

Abstract: In recent years, network function virtualization (NFV) is arousing concern widely from both academia and industry due to its flexible deployment and low cost. However, the performance bottleneck becomes increasingly distinctive and hinders the progress of NFV technology. The major factors were studied which could seriously affect the performance of NFV data plane. Firstly, an analysis was conducted on the causes for various kinds of performance overheads. Then, the optimization solutions were summarized to mitigate these overheads. Finally, a brief introduction was made on two types of architecture, which could significantly improve the performance of NFV data plane.

Key words: network function virtualization, performance optimization, system architecture

1　引言

在过去的很长一段时间里，电信运营商每新增一项网络服务都需要采购大量的专用设备，另一方面，设备提供商也需要投入大量的人员进行设计研发。从开始规划到最后的部署，不仅周期漫长，而且造价昂贵。考虑到这些因素，运营商开始寻求成本低廉、部署灵活的新型解决方案[1]。网络功能虚拟化（network function virtualization, NFV）正是在这种需求背景下产生的。2012 年 10 月，ETSI（European Telecommunications Standards Institute，欧洲电信标准化协会）发布 NFV 白皮书，其最终目标是通过软件定义网络功能，实现网络功能的灵活部署。网络功能虚拟化可以实现软硬件的解耦，运营商只需购置通用硬件设备，甚至可以采用云计算的采购模式部署新型的网络

收稿日期：2017−01−13；修回日期：2017−03−27

服务。更重要的是，由于网络功能软件化，网络功能可以轻松地迁移与扩展。然而，NFV 同样面临着诸多挑战[2]，性能问题成为制约 NFV 技术发展的主要瓶颈，大量工作围绕如何提高 NFV 数据平面的网络性能而展开，力图通过 NFV 数据平面的优化以实现接近硬件设备的性能。

本文首先总结了影响NFV数据平面网络性能的主要因素，包括网卡中断、内存复制、系统调用、锁的争夺、上下文切换、文件系统、缓存未命中等开销，并针对这些开销归纳总结了主流的优化方法，最后介绍了两种可供参考的 NFV 架构设计，并简要分析了两种架构的优缺点。

2 影响 NFV 网络性能的因素

NFV 应用的性能很大程度上取决于网络 I/O 的性能，而数据分组从到达网卡到应用处理需要经历多个阶段。数据分组处理的流程如图 1 所示。

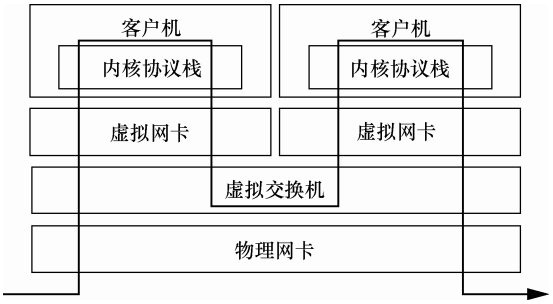


图 1　数据分组处理的流程

图 1 展示了在传统 NFV 解决方案中数据分组处理的流程，当数据分组到达网卡后，通过 DMA（direct memory access）复制到宿主机的内存空间并触发中断，宿主机协议栈处理完数据分组后交由虚拟交换机处理，随后数据由虚拟交换机递交至上层客户机，经由虚拟网卡，客户机内核协议栈最后到达提供网络功能的应用程序。显然，上述过程存在着诸多开销，具体来讲可以分为以下几类。

2.1 网卡中断开销

轮询与中断是操作系统与外设进行 I/O 通信的两种主要方式。一般情况下，网络中数据分组的到来具有不可预测性[3]。若采用轮询模式对数据分组进行持续监听，会造成很高的CPU 占用率，进而影响系统其他进程的运行效率，故主流操作系统都采用中断来处理网络的请求。中断处理方式在低速网络 I/O 场景下非常有效[4]，然而随着高速网络接口等技术的迅速发展，10 Gbit/s、40 Gbit/s 甚至 100 Gbit/s 的网络端口已经出现。随着网络 I/O 速率的不断上升，网卡面对大量高速数据分组引发频繁的中断，中断引起的上下文切换开销将变得不可忽视，造成较高的时延[5]，并引起吞吐量下降。

2.2 内存复制开销

提供服务的应用进程运行在用户空间，而通常情况下，操作系统首先将到来的数据分组读取到内核空间，为了使上层应用能够对接收到的数据进行相应处理，需要将内核将数据从内核空间复制到用户空间；而对于应用进程产生的数据，也需要由用户空间复制到内核空间，最后由网卡发出。操作系统在处理数据过程中往往需要进行多次复制操作，严重影响了高速网络服务的性能[6,7]。

2.3 锁开销

当多个线程或进程需要对某一共享资源进行操作时，往往需要通过锁机制来保证数据的一致性和同步性，而加锁带来的开销会显著降低数据处理的性能[3]。具体来说，锁机制造成的开销主要有两方面：一方面，线程在为共享资源上锁或者去锁的过程中通常需要耗时几十纳秒[8]；另一方面，在竞争锁的过程中，等待线程在阻塞过程中无法进行有效的数据处理和计算，从而降低了整个系统的并发性能[9]。无论是宿主机还是客户机的内核协议栈中，都存在大量的共享资源，这也制约着整个系统的可扩展性。

2.4 上下文切换开销

NFV 的扩展需要多核并行化的支持，然而在

该场景下，数据平面需要进行资源的分配调度，调度过程中涉及多种类型的上下文切换，在网卡中断、系统调用、进程调度与跨核资源访问等上下文切换过程中，操作系统均需要保存当前状态[10-14]，而这一类的切换开销往往相当昂贵，严重影响系统性能。

2.5　文件系统管理开销

socket（套接字）作为网络通信过程中的基本操作单位，在 NFV 数据平面中同样扮演重要的角色。本质上说，socket 也是一种文件抽象。Linux 为了实现统一文件管理，通过虚拟文件系统（virtual file system，VFS）为套接字绑定了一系列对应的数据结构如 inode、dentry 等。通常情况下，这些重量级的数据结构对于套接字本身的功能来说是不必要的[15]。而在 NFV 的应用场景下，往往需要对 socket 进行频繁的分配和释放，因而操作系统在管理这些数据结构的过程中会引起较大的性能开销[10]。

2.6　缓存未命中开销

缓存是一种能够有效提高系统性能的方式，然而由于设计的不合理造成频繁的缓存未命中，则会严重削弱 NFV 数据平面的性能。以 Intel XEON 5500 为例，在 L3 缓存命中与缓存未命中条件下的数据操作耗时相差数倍[16]。如果在系统设计中没有考虑到这一点，存在频繁的跨核调用，由此带来的缓存未命中会造成严重的数据读写时延，从而降低系统整体性能[3]。

3 NFV 数据平面性能优化技术

综合第 2 节中的分析可以看到，NFV 数据平面操作过程中的开销类型复杂多样且彼此之间相互关联。针对第 2 节中的性能开销因素，目前主要的优化方法可归纳见表 1。

表 1　性能开销及优化方法

开销类型	优化方法
网卡中断	轮询取代中断
内存复制	零复制
系统调用	用户态协议栈、批量处理
锁	资源本地化
上下文切换	高效虚拟化技术
文件系统管理	轻量级套接字
缓存未命中	资源本地化

3.1　轮询取代中断

作为 I/O 通信的另一种方式，轮询不存在中断所固有的开销。以网卡接收分组为例，在轮询模式下，系统会在初始化时屏蔽收发分组中断，并使用一个线程或进程不断检测收取分组描述符里的收取分组成功标志是否被网卡置位，以此来判断是否有数据分组，整个收取过程没有发生上下文切换，因此也就避免了相应的开销。在高速网络 I/O 下，轮询带来的性能提升是非常显著的。根据 PLVison 公司的相关实验，基于中断方式的网络吞吐速率与基于轮询处理方式的网络吞吐速率相差近 20 倍。

然而需要注意的是，轮询并不一定总是优于中断。确切地说，只有当 I/O 速率接近 CPU 速率时，中断的开销变得不可忽略，轮询模式的优势才能体现；相反，如果数据吞吐率很低，中断能有更好的 CPU 利用率，此时不宜采用中断模式。基于以上分析，针对网络流量抖动较大的场景，可以选用中断与轮询的混合模式，即：在流量小时使用中断模式，当遇到大流量时切换为轮询模式。目前 Linux 内核（NAPI）与 DPDK（data plane development kit，数据平面开发工作集）都支持这种混合中断轮询模式。

3.2　零复制技术

零复制技术主要用以避免 CPU 将数据从一个内存区域复制到另一个内存区域带来的开销。在 NFV 数据平面操作的场景下，零复制指的是除网卡将数据 DMA 复制进内存外（非 CPU 参与），从数据分组接收到应用程序处理数据分组，整个过程中不存在数据复制。零复制技术对于高速网络而言是十分必要的。在高速网络环境下，网络

I/O 性能接近甚至超过 CPU 的处理能力，过于频繁的数据复制几乎会耗尽 CPU 所有的计算资源，从而使得 CPU 成为性能瓶颈。同时，数据从内核空间进入应用程序所处的用户空间，该过程中引发的上下文切换开销同样不可忽略。DPDK、Netmap、PF-ring 等高性能数据分组处理框架都运用了零复制技术，可以实现在通用平台下高效的网络处理。

3.3 高效虚拟化技术

目前在NFV领域常用的高效虚拟化技术大致可以归为以下两类。

（1）基于硬件的虚拟化技术

I/O 透传与 SR-IOV 是两种经典的虚拟化技术。I/O 透传，指的是将物理网卡直接分配给客户机使用。这种由硬件支持的技术可以达到接近宿主机的性能。不过由于 PCIe 设备有限，PCI-SIG 提出并制定了一套虚拟化规范——SR-IOV，即单根 I/O 虚拟化。通过 SR-IOV，可以使一块物理网卡提供多个虚拟功能（VF），而每个 VF 都可以直接分配给客户机使用。通过SR-IOV解决了PCIe设备有限的问题，不过与透传一样，SR-IOV 无法做到实时迁移，并且支持的 VF 仍有上限。

（2）半虚拟化技术

半虚拟化（paravirtualization），区别于完全虚拟化，无需对硬件做完全的模拟，而是通过客户机的前端驱动与宿主机的后端驱动一同配合完成通信，而客户机操作系统能够感知自己处在虚拟化环境，故称半虚拟化。由于半虚拟化拥有前后端驱动，不会造成 VM-exit，所以半虚拟化拥有更高的性能。主流虚拟化平台 KVM 就使用了 Virtio 作为半虚拟化的驱动，半虚拟化比起 SR-IOV 的优势在于支持热迁移，并且可以与主流 vSwitch 对接。相比之下 SR-IOV 只支持二层连接，因此半虚拟化技术具有更好的灵活性。

3.4 网络协议栈优化

由于现有网络功能如代理、IDS、负载均衡等都依赖于协议栈对到来数据分组进行拆封解析，短时间内构建完全脱离协议栈的网络功能是不现实的，而协议栈的处理性能在小分组、高并发的场景下并不理想[15]。因此，众多工作将重点落在了优化协议栈上。如前所述，内核协议栈除了中断开销和内存复制开销之外，还有锁、系统调用、文件系统、跨核等一系列开销，这些开销均严重制约了协议栈的可扩展性，使得协议栈成为性能瓶颈。

减少锁开销和跨核开销，目前的主要解决方案是将共享的数据结构分区与连接本地化，即设法将同一个流的所有数据分组交由同一个CPU核处理，从而避免了不同核对于同一资源的竞争，进而也就避免了加锁的开销。同时，在同一个 CPU 核处理同一条流也可以提升缓存的命中率，避免跨核的开销。Affinity-Accept、Megapipe 以及 Linux 的 So_Reuseport 均实现了被动连接的本地化，通过拆分监听散列表，实现了从 3 次握手到连接建立均在同一个核上进行。Fastsocket[15]则更进一步，通过控制主动连接的源端口，并配合网卡的 Flow Director 功能，实现了主动连接的本地化。

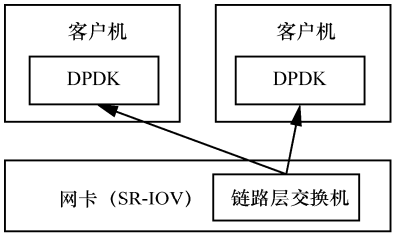
减少系统调用开销，目前的解决方案可以分为两种：一种方案是直接将协议栈移动到应用层实现，避免昂贵的上下文切换开销，如 mTCP、lwIP 等方案；另一种方案则是采用批处理的思想，批量调用系统调用以平摊开销，MegaPipe、FlexSC、VOS 均使用批量系统调用来减小开销。

减少文件系统开销，目前的解决方案也可分为两种：一种方案为自定义轻量级的 socket，一些优化方案如 mTCP、lwIP 等，选择直接绕过 VFS，在用户态重新定义实现 socket 结构体，另外一些优化方案如 MegaPipe 等，虽然在内核实现，但也通过自定义 API 避免了原有 VFS 的文件操作；另一种方案则是继承 VFS 的 socket 实现，但是简化掉 inode 与 dentry 的初始化与销毁过程，抛弃其中的锁。这是因为对于socket而言，inode与dentry是完全无用的。

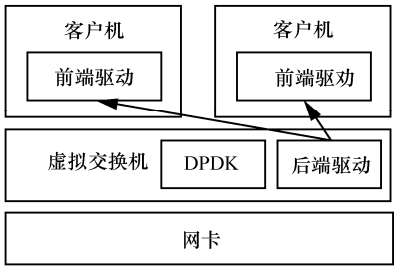
此类方案的代表性工作是 Fastsocket。相比前一种方案，该方案的优点在于能够完全兼容传统 socket，降低 NFV 应用的移植难度。

4 NFV 性能综合提升方案

综合以上分析可以看出，NFV 数据平面中的性能开销类型多样，且相互关联。因此，现有的比较成熟的解决方案往往需要结合多种优化技术，从而实现性能的显著提升。针对 NFV 的特殊场景，存在两种高性能的网络数据分组 I/O 体系架构。NFV 数据平面架构设计如图 2 所示。



(a) 架构一



(b) 架构二

图 2　NFV 数据平面架构设计

如图 2(a)所示为第一种架构，使用 SR-IOV 或者 PCIe 穿透技术直接将物理设备或者虚拟功能分配给客户机，在客户机内部署成熟的协议栈优化方案（如 Fastsocket），或者基于 DPDK 重构一个新型的应用层协议栈。这种架构的优点在于实现简单，几乎不需要修改现有网络功能的程序就能实现整体性能的提升；然而缺点也十分明显：由于 SR-IOV 的本身缺陷，使得业务链缺乏灵活性，不同虚拟机之间只能通过二层路由进行通信，且不支持热迁移，这与 NFV 的思想是相悖的。因此该方案更适合业务链简单、功能单一并希望尽量不修改应用程序的场景。

如图 2(b)所示为第二种架构，客户机使用半虚拟化网卡，并与宿主机里的 vSwitch 相连，由 vSwitch 控制业务链，而 vSwitch 和半虚拟化的前后端均可使用 DPDK 优化，可以采用轮询取代中断、零复制等技术实现性能的大幅提升。NetVM[3] 是采用此架构实现的典型代表方案。在 NetVM 的解决方案中，通过在客户机与宿主机之间进行内存共享，从而实现了全程零复制。具体地，NetVM 在宿主机里使用 DPDK 轮询收取分组并放入共享的大页内存，在客户机内使用前端驱动映射地址空间并模拟 PCI 设备。客户机内的应用程序可以通过 NetVM 提供的接口直接访问共享内存的数据分组，并指定数据分组接下来的流向。NetVM 通过这一整套机制，消除了宿主机与客户机、客户机与客户机交互过程中的内存复制开销，故可以提供极高的性能。尽管第二种架构的部署灵活、性能优秀，但是架构实现复杂，应用程序往往需要根据半虚拟化前端驱动做出修改，甚至需要直接处理裸分组，应用的代价远高于第一种方案。

5　结束语

本文分析并总结了影响 NFV 数据平面处理性能的主要因素，具体包括网卡中断开销、内存分组开销、锁开销、上下文切换开销、文件系统管理开销以及缓存未命中开销等。在充分分析以上开销成因的基础上，进一步归纳总结了相应的性能优化方案。最后介绍了两个可供参考的系统架构，并简要分析了两种架构的优点与缺点。未来 NFV 性能优化将主要侧重两个方面。一方面，随着容器技术的日益成熟，基于容器的轻量级 NFV 优化方案将被广泛关注，并逐渐取代基于虚拟机的传统 NFV 方案，从而对系统资源实现更加高效灵活的利用；另一方面，通用性的 NFV 应用平台将成为主要研究方向，今后工作应重点考虑如何兼顾高性能与通用性，简化 NFV 应用的移植过程，

实现应用与平台的无缝对接。

参考文献：

[1] 李晨, 段晓东, 陈炜, 等. SDN 和 NFV 的思考与实践[J]. 电信科学, 2014, 30(8): 23-27.
LI C, DUAN X D, CHEN W, et al. Thoughts and practices about SDN and NFV [J]. Telecommunications Science, 2014, 30(8): 23-27.

[2] 赵慧玲, 史凡. SDN/NFV 的发展与挑战[J]. 电信科学, 2014, 30(8): 13-18.
ZHAO H L,SHI F. Development and challenge of SDN/NFV[J]. Telecommunications Science, 2014, 30(8): 13-18.

[3] HWANG J, RAMAKRISHNAN K K, WOOD T. NetVM: high performance and flexible networking using virtualization on commodity platforms[J]. IEEE Transactions on Network and Service Management, 2015, 12(1): 34-47.

[4] FIGURE E. Eliminating receive livelock in an interrupt-driven kernel[J]. ACM Transactions on Computer Systems, 1997, 15(3): 217-252.

[5] YANG J, MINTURN D B, HADY F. When poll is better than interrupt[C]//FAST, February 14−17, 2012, San Jose, CA, USA. New York: ACM Press, 2012: 3.

[6] WU W, CRAWFORD M, BOWDEN M. The performance analysis of Linux networking-packet receiving[J]. Computer Communications, 2007, 30(5): 1044-1057.

[7] KOH Y, PU C, BHATIA S, et al. Efficient packet processing in user-level Oses: a study of UML[C]//The 31th IEEE Conference on Local Computer Networks, November 14−16, 2006, Sydney, Australia. New Jersey: IEEE Press, 2006: 63-70.

[8] DEAN J. Designs, lessons and advice from building large distributed systems[J]. Keynote from LADIS, 2009(1).

[9] DINIZ P C, RINARD M C. Lock coarsening: eliminating lock overhead in automatically parallelized object-based programs[J]. Journal of Parallel and Distributed Computing, 1998, 49(2): 218-244.

[10] BOYD-WICKIZER S, CLEMENTS A T, MAO Y, et al. An analysis of Linux scalability to many cores[C]//Usenix Symposium on Operating Systems Design & Implementation, October 4−6, 2010, Vancouver, BC, Canada. New York: ACM Press, 2010: 86-93.

[11] KAMRUZZAMAN M, SWANSON S, TULLSEN D M. Inter-core prefetching for multicore processors using migrating helper threads[J]. ACM SIGPLAN Notices, 2011, 46(3): 393-404.

[12] KAZEMPOUR V, KAMALI A, FEDOROVA A. AASH: an asymmetry-aware scheduler for hypervisors[J]. ACM SIGPLAN Notices, 2010, 45(7): 85-96.

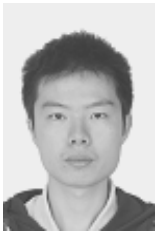
[13] VAHDAT A, YOCUM K, WALSH K, et al. Scalability and accuracy in a large-scale network emulator[J]. ACM SIGOPS Operating Systems Review, 2002, 36(SI): 271-284.

[14] EMMERICH P, RAUMER D, WOHLFART F, et al. Performance characteristics of virtual switching[C]// 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet), October 8−10, 2014, Luxembourg. New Jersey: IEEE Press, 2014: 120-125.

[15] LIN X, CHEN Y, LI X, et al. Scalable kernel TCP design and implementation for short-lived connections[C]//International Conference on Architectural Support for Programming Languages & Operating Systems, April 2−6, 2016, Atlanta, Georgia, USA. New York: ACM Press, 2016: 339-352.

[16] LEVINTHAL D. Performance analysis guide for intel core i7 processor and intel XEON 5500 processors[J]. Intel Performance Analysis Guide, 2009(30): 18.

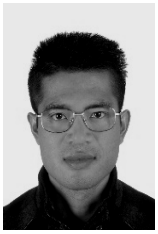
[作者简介]



黄昱恺（1992−），男，清华大学硕士生，主要研究方向为软件定义网络、网络功能虚拟化、高性能网络协议栈设计与实现。

耿金坤（1994−），男，清华大学硕士生，主要研究方向为高性能网络协议栈的设计与实现、数据中心带宽资源分配方案的设计与实现。

令瑞林（1992−），男，清华大学硕士生，主要研究方向为数据中心网络、网络安全与网络性能优化技术等。



李丹（1981−），男，清华大学计算机系特别研究员、博士生导师，主要从事网络体系结构、网络协议与网络系统的研究工作。担任国家“973”计划（青年科学家专题）项目“软件定义的云数据中心网络基础理论与关键技术”首席科学家，获得 2014 年清华大学“学术新人奖”，获得 2015 年国家优秀青年科学基金资助。主持或参加国家科研项目 10 多项，曾担任国际学术期刊 IEEE Transactions on Computer 编委（2013−2015 年）。《电信科学》杂志编委。发表论文 50 多篇，论文被引用 2 000 多次，获得国内外专利 20 多项。