用户模式下虚拟路由器的优化

尹 栋 慕德俊 戴冠中
(西北工业大学自动化学院 西安 710072)

摘 要 网络虚拟化体系实现了在同一共享底层架构之上同时运行多个虚拟网络。然而,构建多异构网络并存的、可灵活配置的虚拟网络平台具有一定的挑战性,不仅需要减少虚拟网络之间的交互,还需要提供高速传输的网络特性。对此,提出了一种传统软件虚拟路由器优化方法,实现了用户模式中有效的数据包处理方式,以支持高速、灵活的虚拟网络。采用操作系统层虚拟化过程将物理主机分成多个虚拟机,虚拟路由的数据处理过程在独立的虚拟机中运行,以确保路由处理的安全配置;同时,采用一种优化的用户模式中数据包处理机制,实现了虚拟路由器的高速传输。实验结果证明,该设计中的虚拟路由与传统用户模式中的软件路由器相比,其传输速率提高了3倍以上。

关键词 网络虚拟化,虚拟路由器,缓存共享,轮询方式,用户模式
中图法分类号 TP393.02 **文献标识码** A

Optimized User Mode Virtual Router

YIN Dong MU De-jun DAI Guan-zhong
(Automation School, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract Network virtualization provides the ability to run concurrent virtual networks over a shared substrate. However, it is challenging to design such a platform to host multiple heterogeneous and often highly customized virtual networks. Not only minimal interference among different virtual networks is desired, high-speed packet forwarding is also required. This paper presented Optimized User mode Virtual Router(OUVR), it is a virtual network platform which uses efficient user mode packet forwarding to supports high-speed and highly customizable virtual networks. With adopting lightweight OS-level virtualization to slice a physical server into virtual machines, the data plane of a virtual router runs in an isolated virtual machine so as to safe for customization. We designed a new user mode packet processing scheme for virtual routers hosted in OUVR to achieve high speed forwarding. Experiments show that an OUVR virtual router can be four times faster than conventional user mode software router.

Keywords Network virtualization, Virtual router, Shared memory, Polling scheme, User mode

1 引言

网络虚拟化体系实现了在虚拟、共享的网络资源之上应用网络新技术和服务的配置与测试。在一个网络虚拟化体系结构中,可以独立地创建、配置以及试验多种异构虚拟网络,并避免虚拟网络之间的交互。本文主要阐述了一种具有数据高速传输、高度灵活性的虚拟网络平台的设计及实现。本设计以灵活性为首要目标,将虚拟网络数据处理过程运行在操作系统用户模式的虚拟机中,确保现实网络数据处理的完全可控。为实现网络的高速传输,本设计采用包头与数据分离处理的用户模式处理机制,设计了优化用户模式虚拟路由器的方法。首先,不同于用户模式下传统的数据包处理过程,本设计采用共享内存的方法,替代了用户态和内核态之间的数据包拷贝过程。尽管此方法已经在操作系统的"零拷贝"I/O机制中提出[3],但本文首次将其用于构建灵活、高速的网络虚拟化平台。其次,建立用户模式虚拟路由器和操作系统内核独立的轮询方式,避免了系统调用的时间开销。本文第2节通过对不同软件路由器进行测试,研究用户模式下传统的包处理过程的瓶颈;第3节详细阐述了用户模式下数据有效处理机制的设计与实现;第4节通过试验对本设计进行评价;最后总结本研究工作。

2 软件路由器中数据包的传输

软件路由器既能运行在操作系统的内核模式中,也可在用户模式下进行路由处理,例如Click Router[4]所实现的路由方式。Click路由器在内核模式中运行可以达到较好的传输性能[5,6],但在用户模式中,其数据包处理速率比较缓慢。本节将对软件路由器进行性能测试实验,寻找用户模式路由器处理性能的瓶颈。

2.1 数据包传输过程

(1)内核模式的软件路由器 在Linux操作系统中,当一个数据包通过网络接口控制器(NIC)接收之后,内核模式中

到稿日期:2011-03-23 返修日期:2011-06-22 本文受西北工业大学研究生创业种子基金(Z2011049)资助。
尹 栋(1982—),男,博士生,主要研究方向为计算机网络控制、信息安全研究,E-mail:yindong@mail.nwpu.edu.cn;慕德俊(1963—),男,教授,博士生导师,主要研究方向为信息安全、网络控制;戴冠中(1937—),男,教授,博士生导师,主要研究方向为网络控制。

的软件路由器通过 DMA 方式将该包传输到内核态中的一个缓存区，并通过“Execute-In-Place”的方式处理包。在内核态路由器完成路由查询之后，数据包通过 DMA 方式发送到相应的 NIC 中输出。

(2)用户模式的软件路由器　在用户模式下，Click 路由器通过调用 PF_PACKET———一种基于 socket 的通信方式从系统的内核中接收和发送数据包。当数据包被 NIC 接收到以后，内核通过 DMA 方式将其转移至主存中，同时将数据包内容写入内核缓存区，并与 PF_PACKET socket 链接。当包接收任务被调度执行时，Click 首先进行 recvfrom()系统调用，将数据包的完整信息拷贝到用户态的缓存区中；在完成路由处理之后，Click 执行 send()系统调用，将数据包拷贝回内核缓存区，并由内核通过 NIC 输出。

对软件路由器性能进行测试，其实验平台如图 4 所示，其环境设置与第 4 节实验环境相同。通过测试，当数据包长度为最小值 64bits 时，内核模式的路由器每秒最多可传输 1050 千个包/秒(Kpps)，而用户模式汇总的路由器最多只能传输 230Kpps。

2.2　影响数据包传输的因素

通过实验，可以看到 Click 软件路由器在不同模式中的性能差异较大，其原因主要是：用户模式在数据包传输过程中需要进行两次系统调用与内核进行通信；需要在用户层和内核层之间进行两次拷贝(接收和发送)。对此，可归纳为以下两个主要方面。

(1)内存拷贝　利用 CPU 的时间戳计数工具[7]，对用户态和内核态之间进行数据拷贝所需的 CPU 处理周期进行测量。如表 1 所列，最小 64 字节数据传输时，Click 路由器需要 162 个 CPU 周期从内核态拷贝到用户态，耗时 140 个 CPU 周期将数据包传回内核态。因此，用户模式中数据处理速率远滞后于网络传输带宽。

表 1　用户态和内核态之间数据拷贝过程的时间开销

Packet Size (bytes)	64	128	256	512	1024	1500
Copy_to_user mode	162	188	239	302	442	575
Copy_from_user mode	140	157	200	259	388	507

(2)系统调用　采用相同计时工具，对用户模式下 Click 路由器中的两个主要系统调用 recvfrom()和 send()过程进行测量。实验结果如表 2 所列，在一个数据包的传输过程中，用户模式下路由器总共需要近 6400 个 CPU 时钟周期完成两次系统调用。

表 2　系统调用的时间开销

System Call	Send()	Receive()
CPU cycles	3000	3400

由此可见，用户模式路由器在传输一个数据包的过程中，需要近 6700 个 CPU 周期的额外开销。

3　优化用户模式虚拟路由器系统设计

3.1　基本思路

首先，为了避免用户层和内核之间的数据包拷贝，本文采用操作系统进程间通信方法，为每一个在主机中的虚拟路由器提供一个独立的数据包缓存区。该缓存区在虚拟路由器和操作系统内核之间共享，实现了用户模式的路由器直接处理缓存区中的数据包，不再进行用户层和内核之间的数据拷贝。其次，为减少系统调用的开销，虚拟路由器和内核之间通信采用异步方式访问共享缓存区。虚拟路由器和内核均可独立监测共享区中数据包的状态，若两者中任一方发现待处理的数据包，则立即读取数据并启动处理过程。

3.2　整体架构

优化用户模式虚拟路由器架构(OUVR)结构设计如图 1 所示，利用系统层虚拟化技术将主机资源划分为多个独立虚拟机运行于虚拟网络，每个虚拟路由器拥有一个数据处理控制器。图中描述了一台主机中运行两个虚拟路由器的情况，通过系统内核中 VR-KM 模块，从 NICs 接收数据包，并分类、发送至相应路由器。之后，VR-KM 将接收路由处理完成的数据包，经 NICs 输出。

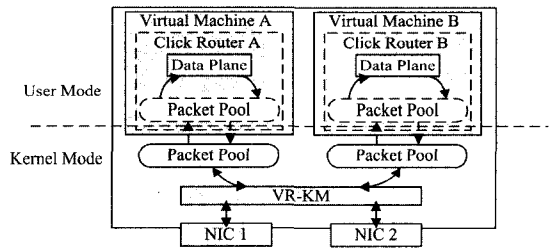


图 1　OUVR 整体设计

(1)共享缓存区设计　VR-KM 在内核模式中为每一个虚拟路由器分配一块缓冲区———数据包池(Packet Pool)。该数据包池通过操作系统文件映射函数 mmap()[9]，在用户模式中建立包池的映射，路由器通过虚拟内存地址直接访问池中的数据包。数据包池由一系列大小相等的内存块组成，每个内存块对应一个索引号，其大小足够存储一个完整的数据包(如，大于最大传输单元 MTU 的大小)。若 $Addr_{pool}$ 表示数据包池中虚拟地址的起始值，且 S_{slot} 表示内存块大小，那么虚拟路由器访问第 i 个包在池中的地址为 $Addr_{pool}+i\times S_{slot}$。

(2)异步访问模式　在数据包池中每个内存块均有一个标识来声明该包的当前状态，用户模式路由器通过轮询查看状态标识。仅当标识为“填满”时，虚拟路由器才能对内存块中的数据包进行处理。经过路由查询、包头更新等处理，将设置内存块为“完成”。此后，VR-KM 模块查询到该内存块状态，将数据包发送到 NIC，并将内存块状态复位为“空”，为新数据包提供存储空间。为保证内存操作的稳定性和安全性，虚拟路由器和 VR-KM 对同一个内存块的访问以及对其标识的操作过程必须是互斥的，因此，采用 Linux 操作系统中的原子操作设置数据包池中存储块的状态标识。

3.3　虚拟路由器设计

利用虚拟化工具，将物理主机划分成多个虚拟机，每个虚拟机支持一个虚拟网络，即一个虚拟机中运行一个路由器，虚拟路由器的数据处理过程由用户模式下的 Click 程序实现。图 2 描述了 OUVR 架构中虚拟路由器的基本架构。在此架构中，每一个虚拟路由器有对应的虚拟 NICs，每一个虚拟的 NIC 由两个环型缓存区组成，即接收环(Rxring)和发送环(Txring)，仅存储数据包池中包的索引号。其中，接收环存储来自虚拟 NIC 的数据包索引号，而发送环则保存需要输出的数据包索引号。与数据包池类似，环型缓存区也是通过 mmap 方式实现与内核态 VR-KM 的通信，因此，路由器和 VR-KM 均可直接对其进行读写操作。

如图 2 所示，虚拟路由器中数据传输过程为：从虚拟 NIC

的接收环 Rxring 中读取数据包索引号 i，然后访问数据包池 Packet Pool(内核模式中数据包池的映射)中地址为 $Addr_{pool}+i\times S_{slot}$ 的存储块，并进行路由处理过程，处理完成之后，更改包池中存储块的状态，同时通过 To Pool 模块将包索引号 i 写入 vNIC 的发送环 Txring 中。路由处理过程 Routing Process 由虚拟网络用户制定。

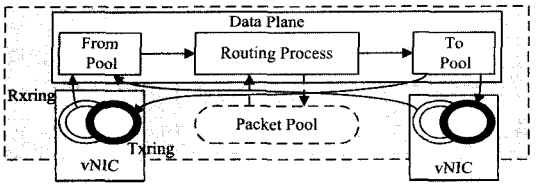


图 2　虚拟路由器结构

3.4　VR-KM 的设计

VR-KM 模块由内核模式的 Click 程序实现，其结构如图 3 所示。从网络控制器 NIC 接收到数据包之后，VR-KM 首先解析包头、分类，确定包属于哪一个虚拟网络。To Pool 模块在数据包池中寻找到标识为“空”的内存块，将数据包拷贝到该内存块，并修改该内存块为“填满”，同时将该包的索引号写入虚拟 NIC 的 Rxring 中。内核中数据包池映射在用户模式中，因此虚拟路由器可直接对该存储单元进行访问。From Pool 模块轮询 Txring 发送环，当发现数据包时，读取包索引号，并根据索引号读取相应的内存块，读取完成路由处理的数据包信息，同时将该内存块状态设置为“空”。最后，待发送的数据包经过 To Device 模块由 NIC 输出。

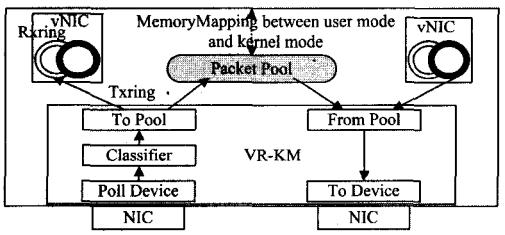


图 3　VR-KM 内部结构

4　实验结果及分析

4.1　实验平台的搭建

图 4 描述了测试环境的结构，计算机配置：Intel 2.66GHz 双核 CPU、4G 内存及 Intel PRO-1000 1Gbit NICs。运行软件路由器的操作系统为 Linux 2.6.18 内核，并安装了内核态的 OpenVZ[10] 以及 Click 程序。在实验中，采用 NetFPGA 平台产生并接收数据包，以确保发送和接收均达到物理带宽 1Gbps。在默认情况下，数据包池最多存储 128 个数据包。

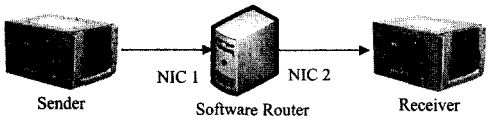
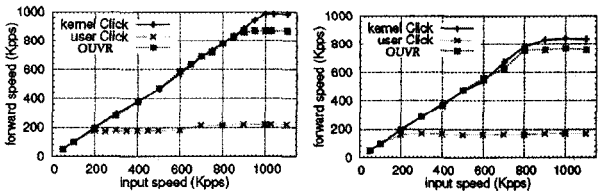


图 4　虚拟路由器测试环境

4.2　数据流测试实验

采用 UDP 数据包传输测试 OUVR 体系中虚拟路由器的运行性能，其中，UDP 包的大小最小为 64 个字节，最大为 1500 个字节。

首先，配置一个只有两条路由信息的简单 IP 路由器，一条接收数据包，另一条发送数据包。对内核模式、一般用户模式以及 OUVR 虚拟路由器进行测试，测试结果如图 5(a)所示。一般用户模式虚拟路由器的转发速率约 200Kpps，而内核模式虚拟路由器转发速率可达 1000Kpss。OUVR 虚拟路由器转发速率为 820Kpps，比一般用户模式虚拟路由器提高了近 3 倍。然后，在 OUVR 建立拥有 170K 条路由信息的 BGP 表。数据包发送端生成 64 字节的 UDP 包，且目的地址为随机 IP 地址。如图 5(b)所示，OUVR 体系性能仍和内核模式虚拟路由器接近，并远优于用户态虚拟路由器。相对于图 5(a)，OUVR 与内核虚拟路由器之间的差距减小了很多。因为，当计算任务增加(如 IP 地址查询等)时，无论是内核模式还是用户模式，所需 CPU 周期均没有太大差别，所以内核模式虚拟路由器的性能优势不明显。



(a)简单路由表　(b)复杂路由表

图 5　UDP 数据包传输性能

图 6 描述了多虚拟路由器并存时的平均性能以及系统整体传输性能。随着数量的增加，CPU 和带宽等资源被平均分配，虚拟路由器平均性能随之降低。此时，在操作系统内核模式中的虚拟交换机 VR-KM 需要更多的 CPU 资源完成虚拟路由器的数据传输，从而降低了 OUVR 的整体性能。

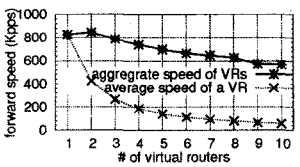
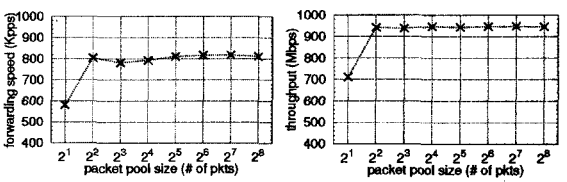


图 6　虚拟路由器数量对 UDP 数据包传输性能的影响

4.3　数据包池与传输性能的关系

在 OUVR 系统中配置两个虚拟路由器，通过改变数据包池的大小(内存块从 2 个增至 256 个)来测试数据包池空间大小对虚拟路由器传输性能的影响。在试验中，分别采用 64 个字节 DUP 包以及 TCP 包产生网络数据流，且数据包发送速率为恒定值 1200Kpps。如图 7 所示，当数据包池能容纳多于 4 个内存块的时候，包池存储空间的大小对 OUVR 体系中虚拟路由器的转发性能影响较小。



(a)UDP 包　(b)TCP 包

图 7　数据包池大小对 UDP/TCP 传输性能的影响

结束语　本文提出了一种用户模式虚拟路由器的优化方法，详细阐述了该方法的提出、实现过程，且通过实验验证了该方法的优越性。在 OUVR 系统中，路由器运行在独立的虚拟环境中，可进行安全、灵活的配置。由于采用共享存储区域以及轮询方式，OUVR 的整体性能明显优于一般用户态虚拟

路由器，并接近于软件路由器的最佳性能。

参 考 文 献

[1] Bavier A, et al. In VINI veritas: realistic and controlled network experimentation[C]// Proceedings of SIGCOMM '06. 2006

[2] Bhatia S, et al. Trellis: A platform for building flexible, fast virtual networks on commodity hardware [C] // Proceedings of ROADS 2008/CoNEXT 2008. 2008

[3] Druschel P, Peterson L L. Fbufs: a high-bandwidth cross-domain transfer facility[C]// Proceedings of SOSP '93. 1993

[4] Click Modular Router[EB/OL]. http://read.cs.ucla.edu/click/

[5] Argyraki K, et al. Can software routers scale? [C]// Proceedings of PRESTO '08. 2008

[6] Egi N, et al. Towards high performance virtual routers on commodity hardware[C]// Proceedings of CONEXT '08. 2008

[7] Intel 64 and IA-32 architectures software developer's manual volume 2B: Instruction set reference, N-Z. Sep. 2009

[8] Soltesz S, et al. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors[C]// EuroSys '07. Mar. 2007

[9] Corbet J, Rubini A, Kroah-Hartman G. Linux Device Drivers (3rd Edition)[M]. O'Reilly Media, Inc., 2005

[10] OpenVZ[EB/OL]. http://www.openvz.org/

[11] Chinni S, Hiremane R. Virtual machine device queues[M]. Intel white paper, 2007

[12] Liao Y, Yin D, Gao L. PdP: Parallelizing data plane in virtual network substrate[C]// Proceedings of SIGCOMM VISA 2009 Workshop. Aug. 2009

(上接第 32 页)

6 进一步改进

6.1 日志的可用性

本协议通过不同参与者间互相备份日志提供了相当高的日志可用性。但是，系统的故障是不可避免的，尤其是协调器节点出现故障时，会对整个系统造成较大影响。考虑到这种情况，可以参考文献[10]中的方案，为协调器节点单独建立一个容错系统，这样日志的可用性可以得到进一步提升。

6.2 拜占庭失效

本协议假设系统中的节点故障方式为故障-停止模式，并没有考虑到节点出现拜占庭故障的情形。尤其是在参与者节点间发起询问时，事务执行的效率很大程度上取决于参与者节点的回复。如果参与者节点出现拜占庭失效的情况，可能会造成事务的失败，甚至导致数据的不一致性。为了避免这种情形发生，可以对节点进行拜占庭故障检测，根据检测结果，决定参与者节点的回复信息是否可用。文献[13,15]对此进行了讨论。但是，分布式短事务系统对效率的要求较高，应尽量控制信息交换的轮数。

结束语 本算法在保持原有系统架构不变的情况下，通过参与者列表，实现了参与者间日志的互相备份，同其他协议相比，提高了事务提交效率，并提高了在网络暂时故障的情况下事务成功提交的可能性。在本文性能分析部分，将本算法与传统两阶段提交协议进行了对比分析，表明本算法在节省日志操作时间、减少信息交换方面具有更优的性能。同时，本协议能够保证较高的日志可用性，通过调整日志暂存区的大小和日志暂存时间还可以将日志可用性进一步提高。在分布式短事务系统的实际应用中，本协议具有更大的优越性。

参 考 文 献

[1] Gray J. Notes on Data Base Operating Systems[M]. Operating Systems. Heidelberg: Springer, 1978: 393-481

[2] Lampson B. Atomic Transactions[M]. Distributed Systems: Architecture and Implementation. Heidelberg: Springer, 1981: 246-265

[3] Martin L. Software Engineering: Design, Reliability and Management [M]. New York: McGraw-Hill Company, 1983: 571-574

[4] Stamos J, Cristian F. A low-cost atomic commit protocol[C]// The 9th IEEE Symp on Reliable Distributed Systems. Huntsville, AL, 1990

[5] Al-Houmaily Y J, Chrysanthis P K, Levitan S P. An argument in favor of the presumed commit protocol[C]// The 13th Int'l Conf on Data Engineering. Birmingham U K, 1997

[6] Liu M L, Abbadi A D. The Performance of Two-phase Commit Protocols in the Presence of Site Failures Distributed and Parallel Databases [M]. Hingham: Kluwer Academic Publishers, 1998: 157-182

[7] Abdallah M, Guerraoui R. One-phase commit: Does it make sense[C] // International Conference on Parallel and Distributed Systems. CA: IEEE Computer Society, 1998: 182-192

[8] Attaluri G K, Salem K. The presumed-either two-phase commit protocol[J]. IEEE Trans on Knowledge and Data Gineering, 2002(10): 1190-1196

[9] 刘云生，覃飙. 分布式实时事务提交协议[J]. 计算机研究与发展，2002，39(7)：827-832

[10] 李宏亮，金士尧，等. 短事务、强实时双机容错系统的研究[J]. 计算机学报，2003，26(2)：244-249

[11] 邱元杰，刘心松，杨峰. 一种高效的分布式并行数据库日志机制[J]. 计算机研究与发展，2004，141(11)：1942-1948

[12] 李陶深，李卫玲，马新娟. 网格环境下的事务提交机制的研究[J]. 计算机研究与发展，2006，43(suppl)：117-122

[13] Gray J. Consensus on transaction commit[J]. ACM Transactions on Database Systems, 2006, 31(1): 133-160

[14] 肖迎元，刘云生，等. 一种实时动态数据库故障恢复策略[J]. 软件学报，2007，18(10)：2516-2527

[15] Zhao Wen-bo. A byzantine fault tolerant distributed commit protocol[C] // Dependable, Autonomic and Secure Computing. USA: IEEE, 2007: 37-46

[16] Chen Jian-ying, Liu Xin-song, Li Tao, et al. Synchronized update over distributed and parallel database, Cyber-Enabled Distributed Computing and Knowledge Discovery[C] // CyberC '09. 2009

[17] Boso B, Sane S S. DTCOT-Distributed Timeout-based Transaction Commit Protocol for Mobile Database Systems[C]// International Conference and Workshop on Emerging Trends in Technology (ICWET 2010). Mumbai, India, 2010

[18] Agrawal S, Shanker U, Singh A N, et al. SPEEDITY-A Real Time Commit Protocol[J]. International Journal of Computer Applications, 2010(1-3): 74-81