计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(12):1−9 [doi: 10.15888/j.cnki.csa.008245]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

网卡虚拟化综述①

刘　强[1], 淡　唯[1], 蒋金虎[2], 张为华[2]

[1](复旦大学 软件学院, 上海 200438)
[2](复旦大学 上海市数据科学重点实验室, 上海 200438)
通讯作者: 张为华, E-mail: zhangweihua@fudan.edu.cn

摘　要: 近年来, 越来越多的应用或微服务部署到云端. 虚拟网络是云端部署运行的基本保障. 为了构建面向虚拟机和容器等虚拟实例的虚拟网络, 网卡虚拟化在物理网卡的基础上, 构建虚拟网卡和虚拟网桥等设备, 并对各虚拟设备进行配置和管理. 本文从虚拟网卡和虚拟网桥出发, 调研了网卡虚拟化中目前流行的虚拟技术, 并将这些技术进行了分类和比较, 最后就网卡虚拟化的现状及未来进行了总结和展望.

关键词: 网卡虚拟化; 云计算; 虚拟网卡; 虚拟网桥

引用格式: 刘强,淡唯,蒋金虎,张为华.网卡虚拟化综述.计算机系统应用,2021,30(12):1–9. http://www.c-s-a.org.cn/1003-3254/8245.html

Survey on Virtualization of Network Interface Card

LIU Qiang[1], DAN Wei[1], JIANG Jin-Hu[2], ZHANG Wei-Hua[2]

[1](Software School, Fudan University, Shanghai 200438, China)
[2](Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 200438, China)

Abstract: In recent years, more and more applications or micro services have been deployed to the cloud. The virtual network is the basic guarantee in a cloud environment. Virtual Network Interface Cards (NICs) and virtual network bridges are built based on physical NICs in virtualization of NIC to construct the virtual network for virtual instances such as virtual machines and containers, and these virtual network devices are uniformly configured and managed. From the perspective of the virtual NIC and the virtual network bridge, the current popular virtualization technologies of NICs are sarveyed, classified, and compared in this study. Finally, the current situation and future development in NIC virtualization are analyzed.

Key words: virtualization of Network Interface Card (NIC); cloud computing; virtual network interface card; virtual network bridge

1　引言

随着互联网时代和大数据时代的到来, 应用软件的数量和用户规模不断扩展, 传统的服务器资源管理难以适应不断增长和动态变化的应用需求. 云计算通过对传统服务器资源进行虚拟化, 在一台服务器上构建多个虚拟实例, 各个应用可以部署到相互隔离的虚拟实例中, 具有高效利用、弹性使用和按需分配等特点. 目前, 云计算技术在业界已经得到了广泛应用, 例如, 亚马逊、谷歌和阿里巴巴等国内外公司都向外提供了云计算服务[1,2].

云计算中的虚拟化技术, 主要将 CPU、内存、I/O 设备和网络等物理资源进行虚拟化, 为上层的虚拟实例提供对应的虚拟资源. 目前, 主要有虚拟机和容器两种形式的虚拟实例. 虚拟机主要使用硬件级别的虚拟

① 基金项目: 国家自然科学基金 (61672160)
Foundation item: National Natural Science Foundation of China (61672160)
收稿时间: 2021-03-06; 修改时间: 2021-03-31; 采用时间: 2021-04-26

化技术, 通过虚拟机监视器模拟出不同的虚拟机硬件环境. 常见的虚拟机监视器有如 Xen[3] 和 QEMU/KVM[4–6] 等. 容器是轻量级的操作系统级别的虚拟化, 通过命名空间和资源控制提供隔离的运行时环境, 如 Docker[7] 和 LXD[8] 等.

网络是互联网和大数据时代极为重要的基础设施. 在传统数据中心, 网络架构可以分为接入层、汇聚层和集中层等[9]. 网络虚拟化在物理网络的基础上, 抽象出多个隔离的虚拟网络, 以供不同的用户使用, 提高物理网络资源的利用率. 对应的, 网络虚拟化包括接入层虚拟化、集中层网络虚拟化等, 这些网络虚拟化工作通常依托于服务器外部的各层物理交换机.

网卡虚拟化, 主要面向虚拟机或容器等虚拟实例[10–12]. 在云环境中, 容器或虚拟机等各虚拟实例成为网络通信的端点, 因此, 需要借助网卡虚拟化技术. 网卡虚拟化, 构建虚拟网卡和虚拟网桥等虚拟网络设备, 并对它们进行统一配置与管理, 使得容器和虚拟机等虚拟实例能够连接到外部物理网络, 从而构建各虚拟实例之间的虚拟网络[10–12]. 通常, 各个虚拟机或容器使用虚拟网卡设备进行网络 I/O, 网络数据包流经虚拟网桥按照对应的管理规则完成处理和转发, 从而实现虚拟实例之间的网络通信. 虚拟网桥通过功能扩展, 具备了物理交换机和路由器等设备的网络功能.

在云环境中, 网卡虚拟化除了追求网络的高性能以外, 还应关注网络的隔离性、管理性和安全性等. 虚拟网卡和虚拟网桥是构建虚拟网络的重要设备, 针对不同的虚拟化方式, 这些虚拟网络设备的性能和隔离性等也存在差异. 目前, 流行的云计算系统 Kubernetes[13] 和 OpenStack[14] 等, 使用的网络虚拟化方案包含了多种网卡虚拟化技术, 以适应于不同的业务场景.

本文首先讨论了网卡虚拟化所需面临的挑战, 阐述了网卡虚拟化的需求和目标. 然后分别讨论了网卡虚拟化中的核心虚拟设备: 虚拟网卡和虚拟网桥. 在虚拟网卡中, 调研了主要的网卡虚拟化技术, 其中包含硬件虚拟化和软件虚拟化; 在虚拟网桥中, 先从软件实现层次的角度介绍了 3 种不同的虚拟网桥, 然后重点介绍了虚拟网桥中对于数据包的不同处理方式, 如隧道网络、路由转发和 NAT (Network Address Translation) 等. 然后, 以 OpenStack 和 Kubernetes 为例, 对它们使用的网卡虚拟化技术进行了分析. 最后, 本文对网卡虚拟化技术进行了总结和展望.

2 网卡虚拟化挑战

网络将各个计算实例连接起来, 以形成协作的整体. 在当今的大数据和互联网时代, 网络是各分布式系统底层的核心基础设施. 虚拟网络是云计算的网络基础, 也是云计算得以成功应用的保障.无论是云平台对容器或虚拟机等虚拟实例的管理, 还是虚拟实例之间通信以及虚拟实例向外提供服务等, 都需要借助网卡虚拟化技术.

网卡虚拟化在云环境下必不可少, 其核心是为各虚拟实例构建虚拟网卡和虚拟网桥等虚拟网络设备, 将各虚拟实例的数据传输和网络管理连接到外部的物理网络. 关于云环境中网卡虚拟化的目标: 首先, 虚拟网卡和虚拟网桥的数据传输性能是重要的目标; 其次, 高管理性也是网卡虚拟化追求的目标, 例如网络隔离性和虚拟实例可移植性等; 最后, 根据特定场景, 网卡虚拟化还需满足其他目标, 例如网络通信安全等.

要满足上述目标, 在网卡虚拟化的过程中无可避免存在一些挑战: 其一, 网卡虚拟化的目标之间存在权衡, 例如, 隔离性导致虚拟化开销必然存在, 与性能追求相矛盾; 其二, 网卡虚拟化面临可扩展性挑战. 相较于传统的服务器数量, 云端的虚拟实例数量呈指数级上升, 这对虚拟网络的高性能和高可管理性等目标形成了挑战; 最后, 云环境具有极高的多样性和变化性, 如何确保虚拟网络满足不同用户的需求, 适应动态变化, 也是网卡虚拟化面临的挑战.

针对上述目标和挑战, 先后涌现了许多不同的网卡虚拟化技术, 它们的基本架构均如图 1 所示: 首先, 网卡虚拟化需要完成虚拟网卡的模拟并分配给各虚拟实例; 其次, 在多个虚拟实例存在时, 需要构造虚拟网桥并通过虚拟网桥的端口连接不同的虚拟网卡或物理网卡. 虚拟实例的数据包经虚拟网卡进入虚拟网桥后, 虚拟网桥对数据包进行统一处理和转发, 从而确保各虚拟实例之间能够建立网络连接并传输数据.

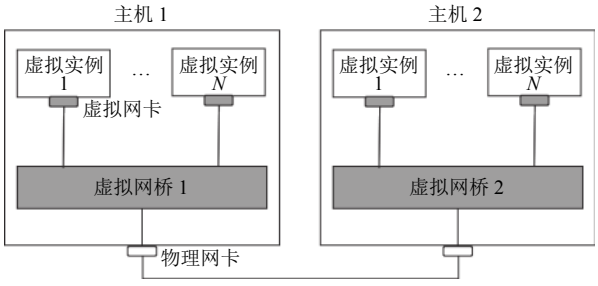

图 1 网卡虚拟化的基本架构

3 虚拟网卡

在虚拟机或容器中, 由于虚拟实例的隔离性, 应用无法直接使用主机的物理网卡, 这时需要借助网卡虚拟化技术. 网卡虚拟化属于 I/O 虚拟化的范畴. 对于网卡设备的虚拟化, 按实现方式可以分为基于硬件的虚拟化方法, 如设备直达和 SR-IOV 等, 基于软件的虚拟化方法, 如全虚拟化、半虚拟化和其他虚拟化等[15].

3.1 基于硬件的虚拟化

基于硬件的网卡虚拟化方法让虚拟机或容器直接使用物理网卡, 不用经过软件虚拟层. 具体的硬件虚拟化方法有设备直达和 SR-IOV:

(1) 设备直达: 在虚拟机中, 物理网卡可以通过 PCI 透传技术分配给虚拟机, 供虚拟机独占使用. PCI 透传技术需要 VT-d[16] 或 IOMMU[17] 等硬件支持, 其作用是将 PCI 设备地址空间映射到客户虚拟机中, 因此, 客户虚拟机和主机一样, 在安装物理网卡驱动后便可直接使用该网卡. PCI 透传技术确保单个虚拟机独占网卡设备, 实现了网络隔离以及与原生网卡相当的性能, 但不利于网卡设备被多个虚拟机共享使用, 无法应用于多个虚拟机用户的场景中. 此外, 虚拟机独占会导致宿主机无法对物理网卡进行管控, 而且虚拟机在迁移过程中, 需要重新配置对应的物理网卡接口.

(2) SR-IOV: SR-IOV (Single Root I/O Virtualization) 在设备直达的基础上, 解决了多个虚拟实例无法共享物理网卡的问题. SR-IOV 从一个物理网卡设备虚拟出多个接口, 各虚拟实例可以分别使用各个接口[18,19]. SR-IOV 能够虚拟出具有完整设备功能的 PF (Physical Function) 以及轻量级功能的 VF (Virtual Function). 其中, PF 可以完全配置或控制整个网卡设备, 而 VF 仅能控制自身的配置资源, 其 PCI 内存空间与其他 VF 分开. 通常, 一个具备 SR-IOV 功能的物理网卡, 都可提供一个 PF 及多个 VF, 其中 VF 的数量与具体网卡的硬件资源相关. 对于虚拟机, 各个 VF 接口可以通过 PCI 透传的方式分配给客户机直接使用. 对于容器, 各 VF 接口也可以分配到不同的容器网络命名空间[20]. 如图 2 所示, 当虚拟实例的数据包进入物理网卡的 VF 接口后, 物理网卡内部的硬件桥接模块将数据包进行转发, 从相应的 PF 或 VF 接口发出.

在设备直达的基础上, SR-IOV 既满足了虚拟实例对物理网卡设备的共享使用, 也实现了网卡接口的隔离. 此外, 由于 SR-IOV 的虚拟化工作均由硬件完成, 其网络性能与物理网卡相当. 但是, 各虚拟实例的数据包仍然直达硬件网卡设备, 网卡接口配置与物理网络耦合, 因此 SR-IOV 缺乏可移植性. 最后, 物理网卡的硬件资源限制了 VF 数量, 因此, 仅使用 SR-IOV 难以适应大规模虚拟实例的场景.

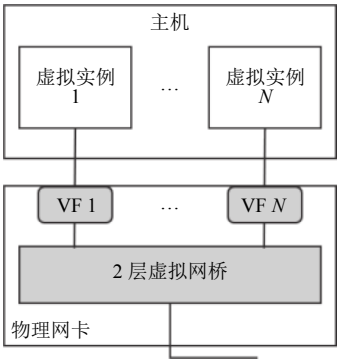

图 2 SR-IOV 架构

3.2 基于软件的虚拟化

基于硬件的虚拟化技术, 如 SR-IOV 等, 需要硬件支持, 同时, 受限于硬件资源, 难以应用于大规模虚拟实例集群. 基于软件的网卡虚拟化, 在实现上更加灵活. 按照是否完全模拟网卡设备功能和硬件接口, 网卡虚拟化可以分为全虚拟化和半虚拟化. 此外, 还有针对容器的虚拟网卡, 如 veth pair[21].

3.2.1 全虚拟化

由虚拟机监视器完整模拟网卡设备. 虚拟机监视器完整实现了物理网卡设备的功能和接口, 包括网卡每条 I/O 请求的处理逻辑. 客户机虚拟网卡在执行任何 I/O 指令时, 均会陷入到虚拟机监视器. 后者将 I/O 指令按照既定的处理逻辑, 通过软件模拟或交给硬件进行处理. 在操作完成后, 虚拟机监视器会将结果反馈给虚拟机. 全虚拟化架构如图 3 所示, 虚拟机不用修改自身操作系统, 直接使用现有的物理网卡驱动就可以执行网络 I/O. 全虚拟化网卡设备有 QEMU e1000[22] 等.

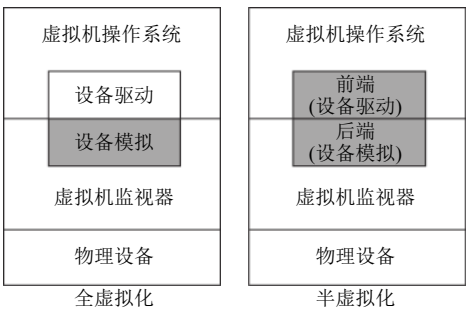

图 3 全虚拟化和半虚拟化

全虚拟化尽管对虚拟机中的原生网络应用实现了无缝支持, 但是需要监视器模拟物理网卡的所有功能和接口, 因此, 全虚拟化适合硬件功能简单和普遍使用的网卡类型. 此外, 全虚拟化每条 I/O 请求的处理路径过长, 上下文切换和数据拷贝次数过多, 导致网络 I/O 的性能下降, 也消耗了更多的 CPU 资源.

3.2.2 半虚拟化

全虚拟化网卡实现比较复杂, 而且存在过多的切换和拷贝开销, 因此, 出现了更通用且更高性能的半虚拟化网卡. 半虚拟化技术并未完整按照原生硬件模拟设备的功能和硬件接口, 而是采用自定义功能和接口的方式. 如图 3 所示, 半虚拟化通过前后端框架的协作来完成网络 I/O 功能的模拟. 其中, 后端在位于虚拟机监视器层, 作为一个可定制的 I/O 设备; 前端位于客户机中, 视为后端设备的特有驱动. 前后端通过共享队列、特征协商和事件通知等机制为客户机提供高性能与定制化的 I/O 功能.

在半虚拟化技术中, 前后端通过控制命令进行特征协商, 确定网卡接口具备的功能集合以及初始化共享队列. 完成初始化后, 虚拟机使用虚拟网卡发送数据时, 通常由其前端驱动将数据写入共享队列, 然后触发 I/O 指令陷入到虚拟机监视器. 后端收到事件通知, 从对应的共享队列中读取数据, 然后由虚拟网桥或物理网卡进行传输. 反之亦然, 虚拟机要接收数据时, 由后端将数据写入共享队列, 通过监视器向虚拟机注入中断, 虚拟机前端从共享队列读取数据并传输给应用.

半虚拟化技术简化了网卡设备的虚拟化过程, 通过共享队列机制, 减少了虚拟机切换和数据拷贝的次数, 提高了网络 I/O 的性能. 此外, 特征位协商机制让半虚拟化的兼容性更强, 易于形成适应各虚拟机监视器平台和各虚拟设备的通用协议标准. 但是, 半虚拟化技术需要前后端搭配, 在使用特定的半虚拟化设备时, 虚拟机无法使用原生驱动, 需要重新部署与后端对应的前端驱动.

目前, virtio 已经成为最主要的半虚拟化标准[23]. 按照 virtio 的半虚拟化标准出现了多种半虚拟化网卡, 如 virtio-net[24]、vhost-net[25] 和 vhost-user-net[26]. 它们的后端模块实现层次不同, 因此, 在数据的传输或同步上存在差异: virtio-net 后端实现在监视器内部, 共享队列位于虚拟机进程用户空间, 数据在传输后仍需拷贝和切换到内核进一步处理; vhost-net 后端实现在内核空间, 共享队列的数据可交由内核其他模块如网桥或协议栈等直接处理, 无需额外拷贝和切换开销; vhost-user-net 中后端从虚拟机进程解耦, 位于另一进程的用户空间, 借助 DPDK[27] 等处理库, 后端收发数据可以绕过主机内核, 通过轮询也避免了虚拟机陷入和注入中断等开销.

3.2.3 其他虚拟化

容器的网络隔离方式与虚拟机不同, 其基于内核的网络命名空间实现. 因此, 容器采用的虚拟网卡无需具备独立的设备形态, 仅需对网卡接口完成模拟. 对于不同的网络命名空间, 虚拟网卡接口是隔离的. 但是, 容器的网络 I/O 仍需要数据包跨网络命名空间进出, 例如, 不同容器间通信或容器与外界通信. 为了解决这一问题, 出现了 veth pair. 它是一对特殊的虚拟网卡接口, 网卡接口分别位于不同的网络命名空间, 数据包可以从一端无障碍地流向对端网卡接口.

4 虚拟网桥

在多个虚拟实例的场景下, 需要使用网桥将各虚拟实例的网卡进行桥接, 以便进行统一的数据处理和转发. 除了基于硬件的网卡虚拟化, 如 SR-IOV, 虚拟网桥需要由软件模拟. 其中, 虚拟网桥的实现层次以及对数据包的处理方式, 会影响虚拟网络的性能和管理.

4.1 虚拟网桥实现

虚拟网桥的主要功能是对来自虚拟网卡数据包进行统一的处理和转发. 虚拟网桥具备不同的端口, 用以连接多个的虚拟网卡. 如图 4 所示, 根据网桥功能的实现层次, 虚拟网桥通常可以分为以下 3 种:

(1) 内核态网桥: 如图 4 所示, 虚拟网桥功能完全在内核态实现, 如 Linux Bridge[28]. 通常, 各个虚拟网卡在桥接虚拟网桥时, 会注册该虚拟网桥的处理函数. 虚拟网卡的数据包在内核协议栈处理后, 会通过对应的函数进行转发处理. 当数据包从端口进入网桥后, 网桥会自动学习对应数据包的 MAC 地址, 和网桥端口建立对应关系, 形成数据包的转发表. 当数据包从网桥转发时, 根据目的 MAC 地址, 从转发表中寻找对应的端口号, 然后数据包按端口号转发. 若未找到对应端口, 数据包从虚拟网桥向各端口广播.

(2) 用户态网桥: 虚拟网桥的功能完全在用户态实现, 如 Snabb Switch[29]. 用户态网桥通常与用户态网络协议栈配合使用. 此时, 用户空间应用程序的数据不需

要拷贝到内核, 无需切换上下文, 由用户协议栈进行处理. 如图 4 所示, 数据包由用户态网桥进行转发时, 在经过处理后, 可以通过物理网卡的用户态驱动进行收发.

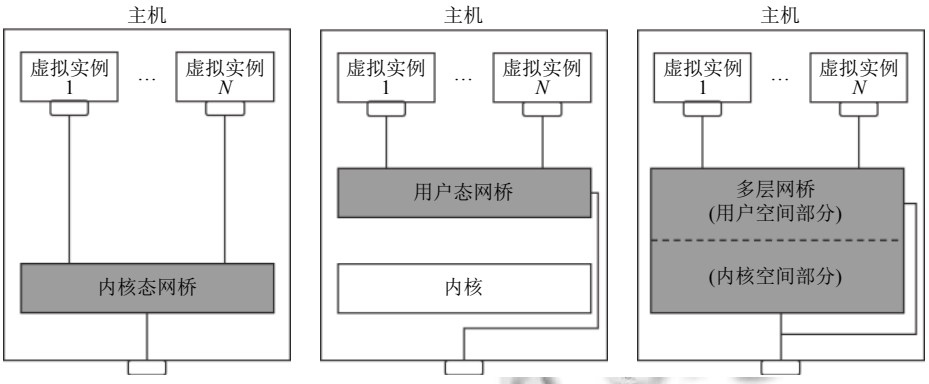

图 4 虚拟网桥的不同实现层次

(3) 多层网桥: 虚拟网桥的功能由内核模块和用户空间守护进程共同完成. 通常, 多层网桥的应用场景和功能比前两者都要强大, 更容易适应复杂的虚拟化网络场景. 目前, 流行的多层网桥有 Open VSwitch (OVS)[30,31]. 通过对 OpenFlow[32] 协议的支持, OVS 可以无缝对接物理交换机等网络设备, 设置复杂的数据转发或处理机制, 完成高效统一的网络管理. 在大规模的复杂网络场景中, 其流表规则通常由 OpenFlow 控制中心下发到各主机的用户空间守护进程, 数据包仅在第一次通过网桥时, 才切换到用户空间进行流表匹配, 之后, 匹配的规则结果会以缓存的方式存在于内核中, 用于处理后续的数据包. 基于网络数据流的局部性, OVS 高效地完成了功能复杂的数据包处理.

总之, 虚拟网桥在不同的层级实现, 其性能和管理能力各不相同. 对于通常的网卡接口, 其数据包的协议栈处理在内核空间进行, 因此, 内核态网桥和多层网桥在此场景下性能更优, 无需在用户态和内核态之间重复拷贝数据包. 但是, 内核态网桥受限于内核空间, 无法完成复杂的数据包转发逻辑, 因此, 多层网桥通过用户空间扩展具备了更高的管理能力. 而用户态网桥则通常与用户态协议栈搭配, 常用于网络功能虚拟化场景, 在用户空间直接处理网络数据包[33].

4.2 数据包处理

在虚拟网桥中, 除了基本的数据包转发功能, 往往还需要对数据包进行处理, 才能在底层物理网络之上构建虚拟网络. 虚拟网桥可以组合不同的功能模块来实现各种数据包处理方式. 常见的数据包处理方式有隧道网络、路由转发和 NAT 等.

4.2.1 隧道网络

底层网络 (通常是物理网络) 充当上层虚拟网络的通信隧道. 虚拟实例的网络应用数据经过内部协议栈封装成数据包, 再经虚拟网卡到达虚拟网桥, 隧道网络将该数据包视为底层网络的有效数据, 按底层网络协议进行再次封装和转发. 数据包传输到终端主机后, 会进行解封处理并经虚拟网桥转发到对应的网卡接口. 此时, 虚拟网卡接收到的仍然是虚拟网络的数据包, 再按协议栈解封即可获得应用传输的数据.

隧道网络的关键在于, 依靠何种协议对虚拟网络的数据包进行封装. 在隧道网络方法中, 封装虚拟网络的数据包时可以选择不同网络层次的封装协议. 例如, 针对 2 层进行封装的 VXLAN 协议[34], 针对 3 层进行封装的 GRE 协议[35] 等. 目前, Weave[36] 和 Flannel[37] 等网络虚拟化方案都采用了隧道网络的数据处理方式, 可应用于 Kubernetes 和 OpenStack 等云计算系统.

隧道网络不用修改物理网络的现有配置, 具有可移植的特点; 同时, 在封装协议中加入标识位, 能够实现多租户隔离, 解决网络规模的可扩展性问题等. 但是, 无论是何种隧道网络, 都需要对原始的虚拟网络数据包进行封装, 导致物理网络中的有效数据负荷降低. 其次, 除数据拷贝开销以外, 封装和解封的过程也存在开销, 在消耗 CPU 的同时也引入了更多网络延迟.

4.2.2 路由转发

路由转发的处理方式, 通过修改主机的路由表, 添加物理网络 IP 与虚拟网络 IP 的路由规则, 从而实现虚

拟网络数据包到物理网络的转发. 各虚拟实例的数据包经过虚拟网卡到达虚拟网桥后, 根据配置的路由转发规则, 可以在 3 层网络进行转发并传输到终端主机, 同样地, 数据包在终端主机同样根据路由规则转发给对应虚拟实例的网卡接口.

路由转发的关键在于配置正确的路由规则. 路由规则直接影响虚拟数据包能否成功转发以及高效送达. 每台主机服务器可以看作一个软件的路由交换机, 由服务器 CPU 按照主机路由表规则进行网络数据包的转发. 当虚拟实例规模较大, 需要借助 BGP 等路由发现协议, 将单个主机的路由信息传播给集群内的其他主机点, 从而实现不同主机之间路由的聚合, 确保各虚拟实例能够通过聚合后的路由规则进行跨主机通信.

路由转发无需像隧道网络进行额外的数据包封装或解封操作, 因此, CPU 占用较低, 网络延迟更低. 但是, 路由转发容易受限于虚拟实例的规模: 大规模集群中, 路由规则的增长容易超过物理资源的承载能力[38]. 另外, 路由转发时, 虚拟实例使用的 IP 地址与主机网络是耦合的, 仅依靠路由转发无法完成多租户隔离等功能[39]. 目前, 流行的路由转发方案有 Calico[40] 等.

4.2.3 NAT

NAT, 即网络地址转换[41]. NAT 技术构建虚拟网络的 IP 和端口与物理网络地址之间的映射关系, 当虚拟网络数据包到达虚拟网桥后, 按 NAT 规则将虚拟数据包的虚拟地址转换成对应的物理地址, 然后经物理网络发送到终端主机. 终端主机同样可以根据 NAT 完成对等转换, 实现虚拟实例跨主机互通. NAT 技术和上述的路由转发不同, 不需要交换各个节点的路由信息, 仅需维护本地 NAT 规则. 在 Linux 系统中, 用户可以通过 iptables 配置 NAT 规则, 虚拟网络数据包经过主机内核协议栈时, 将完成网络地址转换.

NAT 技术无需复杂的协议, 就可以完成虚拟实例之间的跨主机通信. 但是, NAT 在地址转换操作时存在一定的性能损耗. 此外, 物理网络地址资源是有限的, 仅靠 NAT 技术无法构建大规模的虚拟网络. 同时, 在动态变化的虚拟实例场景下, 如何让 NAT 规则能迅速适应, 避免物理地址冲突, 也是一大难题. 总之, NAT 技术往往与其他虚拟化技术配合使用, 更多地应用于各虚拟实例与外部网络的通信场景中, 以实现服务负载均衡和网络防火墙等功能.

5 案例分析

OpenStack 和 Kubernetes 是目前流行的云计算系统, 分别管理虚拟机和编排容器. 两者均需要借助网卡虚拟化技术来构建虚拟机或容器之间的虚拟网络.

5.1 OpenStack

OpenStack 是面向虚拟机的云计算管理平台, 用以提供弹性使用的公有云或私有云服务. OpenStack 集群, 通常由不同的服务器节点组成, 例如计算节点、控制节点和网络节点等. 其中, 用户使用的虚拟机实例部署于各节点, 各虚拟实例使用 OpenStack 构建的虚拟网络.

OpenStack 的网络虚拟化由 neutron 模块[42] 实现, 该模块可以配置不同网络层级的代理, 从而为各虚拟机实例构建不同的网络服务, 例如 L2 层的代理有 OVS 或 Linux Bridge 等. 目前, OpenStack 中一个流行的网卡虚拟化配置是 OVS 和 VXLAN. 其网络架构可以简化为如图 5 所示: 在同一节点中, 各虚拟机实例通过 OVS 集成, 彼此间的通信可以通过内部的 VLAN 网络进行; 在不同节点间, 各虚拟机通过 OVS 隧道网桥提供的 VXLAN 功能连接其他计算节点的虚拟实例, 或者经过网络节点的虚拟路由与其他网络相连.

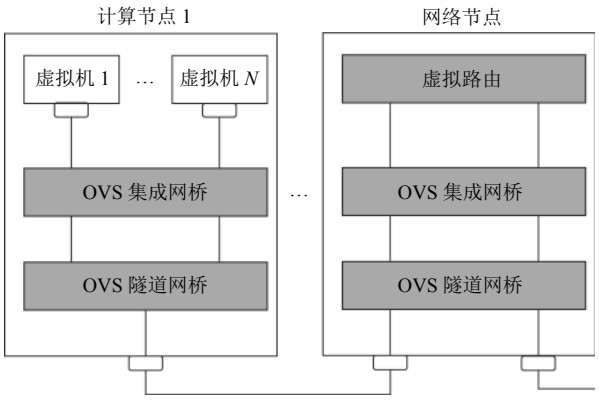

图 5 OpenStack 网络架构

在图 5 中, OpenStack 中各虚拟机的跨节点通信流程是: 当虚拟机发送数据包后, OVS 集成网桥会将数据包中的 VLAN 标记替换为隧道 ID, 其中隧道 ID 用以区分各租户及其网络的不同组合; 当 OVS 隧道网桥接收来自 OVS 集成网桥的数据包后, OVS 集成网在对数据包进行 VXLAN 封装时会打上 VNI 标签. 然后, 数据包通过物理网络流到其他节点. 虚拟机接收数据包的过程, 则与上述相反.

5.2 Kubernetes

Kubernetes 是目前最流行的容器编排系统, 用以管理云环境中各个容器应用. 在 Kubernetes 中, 各应用或微服务被部署于 Pod 中, 其中 Pod 是由一个或多个共享命名空间的容器组成. Pod 是 Kubernetes 中的容器调度的最小单位. Kubernetes 中各 Pod 使用虚拟网络.

Kubernetes 将构建 Pod 网络的功能抽离出来, 由 CNI (Container Network Interface) 插件[43] 完成, 通过 CNI 插件, Kubernetes 可方便地使用不同的第三方网络虚拟化方案, 例如 Weave、Flannel 或 Calico 等. 其中, Flannel 是 Kubernetes 中流行的隧道网络虚拟化方案, 其网络架构如图 6 所示: 在单台服务器中, 各 Pod 会通过 veth pair 桥接到同一虚拟网桥, 如 CNI 0 网桥. 同一服务器 Pod 的 veth 会被配置同一子网的 IP 地址, 如节点 1 中各 Pod 的 veth 均被分配子网 10.24.2.0/24 的 IP 地址. 当各 Pod 之间跨节点通信时, 主机的 Flannel 模块 (如 Flannel.1) 默认会基于 VXLAN 协议给各 Pod 的数据包进行封装, 然后数据包经物理网卡收发.

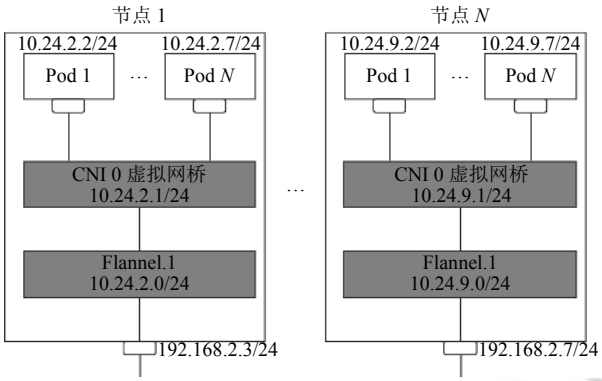

图 6 Kubernetes 网络架构

6 总结和展望

网卡虚拟化, 基于物理网卡, 通过虚拟网卡和虚拟网桥等虚拟设备构建虚拟机或容器等虚拟实例之间的虚拟网络. 针对虚拟网卡, 既有基于硬件的虚拟化技术 SR-IOV, 也有基于软件的全虚拟化或半虚拟化方式; 针对虚拟网桥, 在实现上可以分为内核态、用户态以及多层网桥; 关于虚拟网桥对数据包的处理, 有隧道网络、路由转发和 NAT 等不同方式. 随着云计算的流行与底层虚拟化技术的发展, 网卡的虚拟化技术仍然需要不断地前进. 综上, 本文对网卡虚拟化技术的研究进行了以下展望:

(1) 混合虚拟环境: 随着云计算的发展, 为了满足不同的业务场景, 出现了多种云环境, 例如不同运营商、不同云平台系统以及不同虚拟实例形式等. 在混合的云环境下, 如何进行统一的网卡虚拟化, 实现各虚拟实例网络的统一管理, 是需解决的问题. 例如, 实现 OpenStack 和 Kubernetes 两者虚拟网络的统一管理[44].

(2) 特殊网络: 在普通网卡的虚拟化过程中, 网络性能终将受限于普通以太网卡的硬件处理能力. 因此, 能否对新型的高性能网卡进行虚拟化, 如具备 RDMA (远程直接内存访问) 功能的网卡. RDMA 的网络原理和机制与 TCP/IP 网络存在显著差异. 这类特殊网卡在云环境中如何虚拟化, 也是值得研究的方向[45,46].

(3) 安全性: 在云环境中, 同一服务器上面不同用户的虚拟实例共享底层物理资源, 因此, 虚拟网络在安全方面存在更多的挑战. 在虚拟网络中, 同样需要确保用户的网络安全, 例如, 网络通信安全, 数量流量隐私保护等. 云计算的发展让安全性问题难以得到稳定地解决. 这些安全和隐私问题在未来仍需重点思考[47,48].

参考文献

1 AWS. https://aws.amazon.com/cn/. [2021-02-27].
2 阿里云. https://cn.aliyun.com/. [2021-02-27].
3 Xen Project. https://xenproject.org/. [2021-02-27].
4 QEMU. https://www.qemu.org/. [2021-02-27].
5 Wang ZG, Liu R, Chen YF, *et al*. COREMU: A scalable and portable parallel full-system emulator. Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming. New York: Association for Computing Machinery, 2011. 213–222. [doi: 10.1145/1941553.1941583]
6 Jiang J, Dong RC, Zhou ZJ, *et al*. More with less deriving more translation rules with less training data for DBTs using parameterization. Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture. Athens: IEEE, 2020. 415–426. [doi: 10.1109/MICRO50266.2020.00043]
7 Docker. https://www.docker.com/. [2021-02-27].
8 LXD. https://linuxcontainers.org/. [2021-02-27].
9 Bari MF, Boutaba R, Esteves R, *et al*. Data center network virtualization: A survey. IEEE Communications Surveys & Tutorials, 2013, 15(2): 909–928.
10 Shea R, Liu JC. Network interface virtualization: Challenges and solutions. IEEE Network, 2012, 26(5): 28–34. [doi: 10.1109/MNET.2012.6308072]
11 Jain R, Paul S. Network virtualization and software defined

networking for cloud computing: A survey. IEEE Communications Magazine, 2013, 51(11): 24–31. [doi: 10.1109/MCOM.2013.6658648]
12 Wang AJ, Iyer M, Dutta R, *et al*. Network virtualization: Technologies, perspectives, and frontiers. Journal of Lightwave Technology, 2013, 31(4): 523–537. [doi: 10.1109/JLT.2012.2213796]
13 Verma A, Pedrosa L, Korupolu M, *et al*. Large-scale cluster management at Google with Borg. Proceedings of the 10th European Conference on Computer Systems. New York: Association for Computing Machinery, 2015. 1–17.
14 OpenStack. https://www.openstack.org/. [2021-02-27].
15 Zhang BB, Wang XL, Lai RF, *et al*. A survey on I/O virtualization and optimization. Proceedings of the 5th Annual ChinaGrid Conference. Guangzhou: IEEE, 2010. 117–123. [doi: 10.1109/ChinaGrid.2010.54]
16 Intel. Intel® virtualization technology for directed I/O (VT-d). https://www.intel.com/content/dam/develop/external/us/en/documents/vt-directed-io-spec.pdf. (2020-10-06).
17 AMD. AMD I/O virtualization technology (IOMMU) specification. https://www.amd.com/zh-hans/support/tech-docs/amd-io-virtualization-technology-iommu-specification. [2021-02-15].
18 Single root I/O virtualization. http://pcisig.com/specifications/iov/single_root/. [2021-01-13].
19 Dong YZ, Yang XW, Li JH, *et al*. High performance network virtualization with SR-IOV. Journal of Parallel and Distributed Computing, 2012, 72(11): 1471–1480. [doi: 10.1016/j.jpdc.2012.01.020]
20 采用 Linux* Containers 的单根输入/输出虚拟化 (SR-IOV). https://software.intel.com/content/www/cn/zh/develop/articles/single-root-inputoutput-virtualization-sr-iov-with-linux-containers.html. (2015-11-10)[2021-02-04].
21 Veth Pair. https://man7.org/linux/man-pages/man4/veth.4.html. [2021-02-08].
22 QEMU e1000. https://github.com/qemu/qemu/blob/master/hw/net/e1000.c. [2021-01-08].
23 Russell R. Virtio: Towards a de-facto standard for virtual I/O devices. ACM SIGOPS Operating Systems Review, 2008, 42(5): 95–103. [doi: 10.1145/1400097.1400108]
24 Virtio. https://www.linux-kvm.org/page/Virtio. (2016-10-12).
25 VhostNet. https://www.linux-kvm.org/page/UsingVhost. (2011-03-08).
26 EMU. Features/VirtioVhostUser. https://wiki.qemu.org/Features/VirtioVhostUser. (2018-02-06).
27 DPDK. Developer quick start guide learn how to get involved with DPDK. https://www.dpdk.org/. [2021-03-02].
28 Linux bridge. https://wiki.archlinux.org/index.php/Network_bridge. [2021-01-21].
29 Paolino M, Nikolaev N, Fanguede J, *et al*. SnabbSwitch user space virtual switch benchmark and performance optimization for NFV. Proceedings of 2015 IEEE Conference on Network Function Virtualization and Software Defined Network. San Francisco: IEEE, 2015. 86–92. [doi: 10.1109/NFV-SDN.2015.7387411]
30 Pfaff B, Pettit J, Koponen T, *et al*. The design and implementation of Open VSwitch. Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation. Oakland: USENIX, 2015. 117–130.
31 Koponen T, Amidon K, Balland P, *et al*. Network virtualization in multi-tenant datacenters. Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation. Seattle: USENIX, 2014. 203–216.
32 McKeown N, Anderson T, Balakrishnan H, *et al*. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69–74. [doi: 10.1145/1355734.1355746]
33 王进文, 张晓丽, 李琦, 等. 网络功能虚拟化技术研究进展. 计算机学报, 2019, 42(2): 415–436. [doi: 10.11897/SP.J.1016.2019.00415]
34 Virtual eXtensible Local Area Network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks. https://tools.ietf.org/html/rfc7348. (2014-08-03).
35 Generic Routing Encapsulation (GRE). https://tools.ietf.org/html/rfc2784. (2000-03-21).
36 Weave work. https://www.weave.works/oss/net/. [2021-02-18].
37 Flannel. https://github.com/flannel-io/flannel. [2021-02-26].
38 Suo K, Zhao Y, Chen W, *et al*. An analysis and empirical study of container networks. Proceedings of the IEEE INFOCOM 2018-IEEE Conference on Computer Communications. Honolulu: IEEE, 2018. 189–197. [doi: 10.1109/INFOCOM.2018.8485865]
39 Zhuo DY, Zhang KY, Zhu YB, *et al*. Slim: OS kernel support for a low-overhead container overlay network. Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation. Boston: USENIX Association, 2019. 331–344. [doi: 10.5555/3323234.3323263]
40 Calico. What is project calico? https://www.projectcalico.org/. [2021-02-26].
41 Zhang LX. A retrospective view of network address

translation. IEEE Network, 2008, 22(5): 8–12. [doi: 10.1109/MNET.2008.4626226]

42 Openstack. https://docs.openstack.org/neutron/pike/. (2020-07-28).

43 Kubernetes CNI. https://kubernetes.io/zh/docs/concepts/extelnd-kubernetes/compute-storage-net/network-plugins/. [2021-01-24].

44 Kuryr-kubernetes. hattps://github.com/openstack/kuryr-kubernetes. [2021-03-03].

45 Kim D, Yu TL, Liu HH, *et al*. Freeflow: Software-based virtual RDMA networking for containerized clouds. Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation. Boston: USENIX Association, 2019. 113–125. [doi: 10.5555/3323234.3323245]

46 Pfefferle J, Stuedi P, Trivedi A, *et al*. A hybrid I/O virtualization framework for RDMA-capable network interfaces. Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. New York: Association for Computing Machinery, 2015. 17–30. [doi: 10.1145/2731186.2731200]

47 Thimmaraju K, Hermak S, Rétvári G, *et al*. MTS: Bringing multi-tenancy to virtual networking. Proceedings of 2019 USENIX Annual Technical Conference. Renton: USENIX Association, 2019. 521–536.

48 Cui J, Zhang XY, Zhong H, *et al*. Extensible conditional privacy protection authentication scheme for secure vehicular networks in a multi-cloud environment. IEEE Transactions on Information Forensics and Security, 2019, 15: 1654–1667. [doi: 10.1109/TIFS.2019.2946933]