云生态 | 专刊

Cloud Ecosystem

第8期
Vol.08

打造中国最优质的云生态媒体

InfoQ

100G + 线速LB的设计与实现

国内云服务商UCloud进一步迭代了负载均衡产品Vortex，成功地提升了单机性能。

基于Kubernetes打造容器云

国内第一个公有云计算平台，SAE从2009年已经走过了6个年头，积累了近百万开发者。

基于Calico的容器网络实践

来自宜信的洪强宁分享了“基于Calico打造容器云网络的实践”。

廖春毅：再造一个WinTel联盟

机缘巧合之下笔者与Pica8创始人之一的廖春毅有了一次交流，于是有了本文内容。

案例实践

美团云的网络架构演进之路

青云SDN/NFV2.0架构剖析

YY游戏云平台2.0网络设计分析

云生态专刊 08

本期主编　魏　星

流程编辑　丁晓昀

发行人　　霍泰稳

联系我们

提供反馈　feedback@cn.infoq.com

商务合作　sales@cn.infoq.com

内容合作　editors@cn.infoq.com

ArchSummit
全球架构师峰会2016

2016

2016年7月15日-16日

深圳站

扫描二维码了解大会

腾讯云

腾讯"云+未来"峰会

7月5日-7月6日 · 深圳

立即报名

卷首语

SDN，因开源开放而绽放

SDN（软件定义网络）发源于美国 IT 技术创业者的摇篮斯坦福大学，之后学术界与产业界积极跟进，涌现了众多优秀的创业公司并引发了大笔的融资并购事件。SDN 让大家的关注视角从硬件转向软件，更多转向应用服务的创新。同时亦在软件定义数据中心、网络自动化运维、流量工程等方向发挥重要价值。从 OpenFlow 到 P4、从 SDN 1.0 到 SDN 2.0，SDN 的思想和切入点在逐步进行改进和升级，这也预示着产业链秉承创新与落地并举，让行业发展更接地气！

或许由于开源文化的独特魅力，我们看到了开源技术对推动技术创新起着重要的作用。SDN 数据和控制平面相分离、开放可编程的特性也恰巧迎合了开源思想。传统设备制造商及老牌网络巨头在“大教堂”和“集市”间进行着博弈，并且感受到来自初创公司和大型互联网公司加大对 SDN 的投入所带来的双重压力。期间涌现出了像 NOX/POX、Floodlight、OpenContrail、OpenDaylight、

ONOS、Open vSwitch 等众多优秀的开源项目。在此进程中，我们亦非常欣喜地看到开源所扮演的角色变得愈发重要，它使得通信、网络这些原本非常传统、垂直、封闭的行业碰撞出了很多创新的思路和开放的产品。从商业角度来看，SDN 的开放思想让老牌网络巨头承受着“被革命”的重压，预示着行业内新一轮的资源分配模式和商业模式的变迁。但从技术方向观之，SDN 的开放思想加快了国内网络领域开源文化氛围的积累，迸发出了更多新鲜的血液和思维。诸如在 OpenDaylight、ONOS 等核心项目中，中国代码贡献者比重也在逐渐增多，缩短了中国与国外网络技术领域研究能力的差距。

如果从 OpenFlow 算起的话，SDN 的演进行程已奔跑了 10 个年头。无论是以 Google B4 为代表的数据中心流量调度解决方案还是 VMware NSX 网络虚拟化产品、无论是数据中心互联（DCI）还是广域网流量调度（SDWAN）亦或是软件定义光网络，SDN 正在逐步拓展其应用场景。伴随着从起初的概念纠结到后续落地案例的逐渐丰富，我们也看到大家对待 SDN 的态度愈发走向理性。软件定义所形成的低门槛，使得更多的力量参与到 SDN 运动中。从芯片到交换机、从控制器到网络应用、从数据中心到广域网，每个角色都在探索自己在 SDN 生态圈中的位置。高校、科研院所、设备厂商、运营商、服务提供商，上述不同的角色在整个 SDN 生态中都在贡献着自己的力量，其中也不乏优秀的中国初创公司。我们愈发感觉到这种开放的思想让生态圈中的每个角色都发挥着各自的优势，整个行业也逐渐从厂商的一揽子计划向多方合作的共赢生态进行转变。

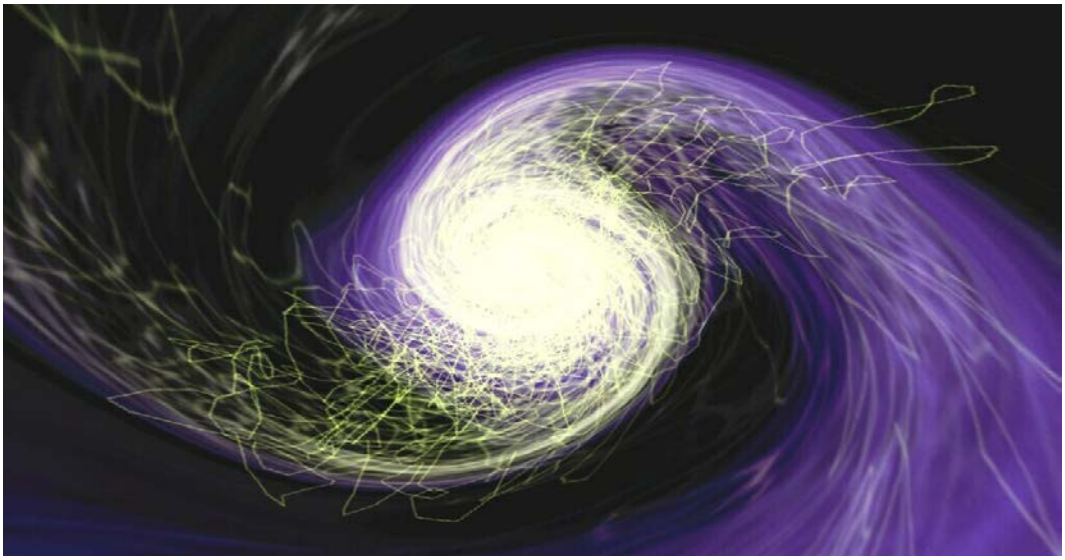
SDN 所代表的开放网络精神驱动着此生态中各项技术的高效前进，也促进了各个环节协作模式的创新，进而通过打造一个开源开放的生态系统推动整个行业的发展与绽放。

魏亮

江苏省未来网络创新研究院团队总监、SDNLAB 联合创始人

作者 俞圆圆

从 Maglev 到 Vortex 揭秘 100G + 线速负载均衡的设计



【编者按】Maglev 是谷歌为自己的数据中心研发的解决方案，并于 2008 开始用于生产环境。在第十三届网络系统设计与实现 USENIX 研讨会（NSDI '16）上， 来自谷歌、加州大学洛杉矶分校、SpaceX 公司的工程师们分享了这一商用服务器负载均衡器 Maglev 的详细信息。Maglev 安装后不需要预热 5 秒内就能应付每秒 100 万次请求令人惊叹不已。在谷歌的性能基准测试中，Maglev 实例运行在一个 8 核 CPU 下，网络吞吐率上限为 12M PPS（数据包每秒），如果 Maglev 使用 Linux 内核网络堆栈则速度会小于 4M PPS。无独有偶，国内云服务商 UCloud 进一步迭代了负载均衡产品——Vortex，成功地提升了单机性能。在技术实现上，UCloud Vortex 与 Google Maglev 颇为相似。以一台普通性价比的 x86 1U 服务器为例，Vortex 可以实现吞吐量达 14M PPS(10G, 64 字节线速），新建连接 200k CPS 以上，并发连接数达到 3000 万、10G 线速的转发。在本文中，UCloud 网络研发团

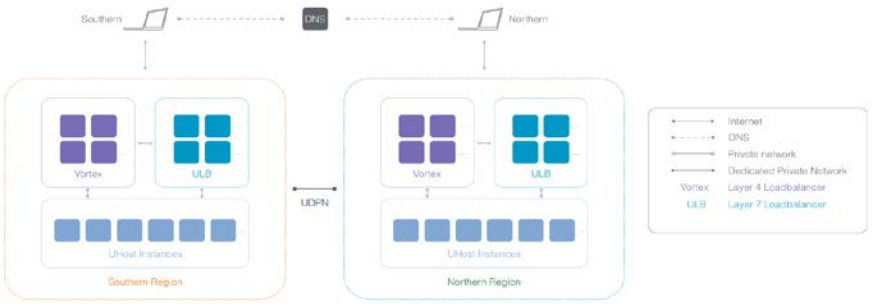


图 1：多层次负载均衡

队分享 UCloud Vortex 的具体实现。

什么是负载均衡

一台服务器的处理能力，主要受限于服务器自身的可扩展硬件能力。所以，在需要处理大量用户请求的时候，通常都会引入负载均衡器，将多台普通服务器组成一个系统，来完成高并发的请求处理任务。

最早的负载均衡技术是通过 DNS 来实现的，将多台服务器配置为相同的域名，使不同客户端在进行域名解析时，从这一组服务器中的请求随机分发到不同的服务器地址，从而达到负载均衡的目的。

但在使用 DNS 均衡负载时，由于 DNS 数据刷新的延迟问题，无法确保用户请求的完全均衡。而且，一旦其中某台服务器出现故障，即使修改了 DNS 配置，仍然需要等到新的配置生效后，故障服务器才不会被用户访问到。目前，DNS 负载均衡仍在大量使用，但多用于实现“多地就近接入”的应用场景。

1996 年之后，出现了新的网络负载均衡技术。通过设置虚拟服务地址（IP），将位于同一地域（Region）的多台服务器虚拟成一个高性能、高可用的应用服务池；再根据应用指定的方式，将来自客户端的网络请求分发到服务器池中。网络负载均衡会检查服务器池中后端服务器的健康状态，自动隔离异常状态的后端服务器，从而解决了单台后端服务器的单点问题，同时提高了应用的整体服务能力。

网络负载均衡主要有硬件与软件两种实现方式。主流负载均衡解决方案中，硬件厂商以 F5 为代表，软件主要为 Nginx 与 LVS。但是，无论硬件或软件实现，都逃不出基于四层交互

图 2：F5 12250v

技术的“报文转发”或基于七层协议的“请求代理”这两种方式。四层的转发模式通常性能会更好，但七层的代理模式可以根据更多的信息做到更智能地分发流量。一般大规模应用中，这两种方式会同时存在。

为什么要研发 Vortex

在研发 UCloud Vortex 之前，我们一直在思考是不是在重造轮子。这要求我们站在 2016 年这个时间点上去分析现状，深入思考各种负载均衡实现的优、劣势。负载均衡大致可分为以 F5、Netscaler 为代表的硬件负载均衡和以 LVS 为代表的软件负载均衡。

不妨，我们先以 F5 为代表来看硬件负载均衡的优劣势。

F5 的硬件负载均衡产品又分单机和集群系列。12250v 是单机中的高端版本，能支撑每秒 150 万新建连接数，8000 万并发连接数，84G 数据吞吐量。从 F5 的 datasheet 中，我们推算出并发连接数受限于内存，高端的 12250v 和次一级的 11050 内存相差 4 倍，并发连接数也是 4 比 1 的关系，后者大概为 2400 万；根据 UCloud 自身云平台的运营经验，150 万新建连接在特定大压力场景下是非常危险的，很有可能被击穿；而 84G 的吞吐量，一般来说是够用的，数据中心南北向的流量有限。

集群系列中 VIPRION 4800 阵列是旗舰产品，每个阵列支持最多 8 个 Blade，每个 Blade 提供每秒 290 万新建连接数，1.8 亿并发连接数以及 140G 数据吞吐量。按线性比例计算，一个顶配的 VIPRION 4800 阵列能满足绝大多数海量互联网应用的业务需求了。其中，需要指出的是单片 Blade 都是用了普通 X86 架构，2 块英特尔

模式	优势	劣势
NAT	o 后端RS不需任何特殊配置	o LVS必须作为网关 o VIP和RIP必须处于不同网段 o 都是客户端源IP信息
DR	o 回程报文不经过LVS，入/出不对称流量性能优化明显 o 同名保留源IP地址	o LVS和RS必须在同一个子网，必须二层可达 o RS无法做到零配置，需要配置VIP，同时要对ARP做特殊处理
FULLNAT	o 后端RS不需任何特殊配置 o 物理网络仅要求三层可达	o 丢失源IP信息（TOA方式对RS改动过大） o 维护及其复杂：FULLNAT无法进入Linux内核主线

图 3：NAT、DR 和 FULL NAT 模式优缺点对比

12 核 CPU 配 256G 内存，这个转变使其支撑量产生了飞越，也进一步证实并发连接数与内存呈相关性。

从技术角度来说，我们认为硬件负载均衡最终的路线是回到 X86 服务器架构，通过横向扩展来提升性能。这里软硬件的区分已经不再存在，因为如果 F5 能做到，具备深入研发能力的技术公司如 Google、Facebook 也能逼近甚至超越。

从商业角度来说，硬件负载均衡产品过于昂贵，高端产品动辄几十万甚至数百万的价格对于用户是几乎不可承受的负担。在文章末尾，我们提供了一份根据网上数据整理后的比较内容，主要来源为 Google 搜索：F5 Networks - Price List - January 11th, 2014 - Amended for the WSCA-NASPO JP14001 Request for Proposal，供大家参考。

从使用角度来说，硬件负载均衡是黑盒，有 BUG 需要联系厂商等待解决，时间不可控、新特性迭代缓慢且需资深人员维护升级，也是变相增加昂贵的人力成本。

再来看（开源）软件负载均衡代表 LVS

LVS 作为目前互联网时代最知名的负载均衡软件，在性能与成本方面结合地很好，阿里云的 SLB 产品也通过 LVS 实现，因此也继承了 LVS 的优点和缺点。LVS 最常用的有 NAT、DR 以及新的 FULL NAT 模式。以下是各模式详细的优缺点比较：

我们认为 LVS 的每种模式都有较大的缺点，但这并不是最为致命的。最为致命的是 LVS 本质上是一个工作于 Linux 内核层的负载均衡器，它的上限取决于 Linux 内核的网络性能，但 Linux 内核的网络路径过长导致了

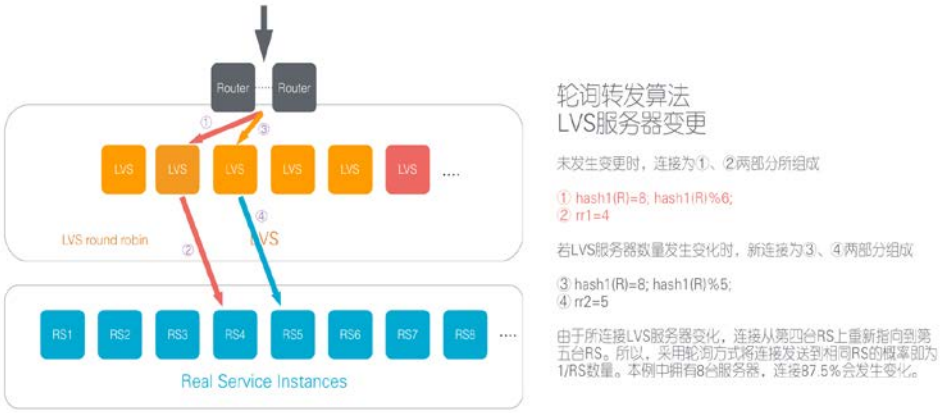


图 4：负载均衡服务器变化场景

大量开销，使得 LVS 单机性能较低。因此，Google 于 2016 年 3 月最新公布的负载均衡 Maglev 实现完全绕过了 Linux 内核 (Kernel Bypass)，也就是说 Google 已经采用了与 LVS 不同的技术思路。

LVS 在运维层面还要考虑容灾的问题，大多使用 Keepalived 完成主备模式的容灾。有 3 个主要缺点：

- 主备模式利用率低；
- 不能横向平行扩展；
- VRRP协议存在脑裂(Split Brain)的风险。Split Brain从逻辑角度来说是无解的，除非更换架构。

也有少数公司通过 ECMP 等价路由协议搭配 LVS 来避免 Keepalived 的缺陷。综合来看，ECMP 有几个问题：

需要了解动态路由协议，LVS 和交换机均需要复杂配置；

交换机的 Hash 算法一般比较简单，增加删除节点会造成 Hash 重分布，可能导致当前 TCP 连接全部中断；

部分交换机（华为 6810）的 ECMP 在处理分片包时会有 BUG。

这些问题均在生产环境下，需要使用者有资深的运维专家、网络专家确保运营结果。

理性的剖析下，我们发现没有任何的负载均衡实现在价格、性能、部署难度、运维人力成本各方面能达到最优，所以 Vortex 必然不是在重造轮子而是在推动这个领域的革新：Vortex 必须不能像 LVS 那样被 Linux 内核限制住性能，Vortex 也不能像 F5 那么的昂贵。

Vortex 负载均衡器的设计理念

用户使用负载均衡器最重要的需求是“High Availability” 和

图 5：后端服务器变化的场景

“Scalability”，Vortex 的架构设计重心就是满足用户需求，提供极致的“可靠性”和“可收缩性”，而在这两者之间我们又把“可靠性”放在更重要的位置。

1.Vortex 的 High Availability 实现

四层负载均衡器的主要功能是将收到的数据包转发给不同的后端服务器，但必须保证将五元组相同的数据包发送到同一台后端服务器，否则后端服务器将无法正确处理该数据包。以常见的 HTTP 连接为例，如果报文没有被发送到同一台后端服务器，操作系统的 TCP 协议栈无法找到对应的 TCP 连接或者是验证 TCP 序列号错误将会无声无息的丢弃报文，发送端不会得到任何的通知。如果应用层没有超时机制的话，服务将会长期不可用。

Vortex 的可靠性设计面临的最大问题就是如何在任何情况下避免该情况发生。Vortex 通过 ECMP 集群和一致性 Hash 来实现极致程度的可靠性。

首先，我们来考察一下负载均衡服务器变化场景。这种场景下，可能由于负载均衡服务器故障被动触发，也可能由于运维需要主动增加或者减少负载均衡服务器。此时交换机会通过动态路由协议检测负载均衡服务器集群的变化，但除思科的某些型号外大多数交换机都采用简单的取模算法，导致大多数数据包被发送到不同的负载均衡服务器。

Vortex 服务器的一致性 Hash 算法能够保证即使是不同的 Vortex 服务器收到了数据包，仍然能够将该数据包转发到同一台后端服务器，从而保证客户应用对此类变化无感知，业务不受任何影响。

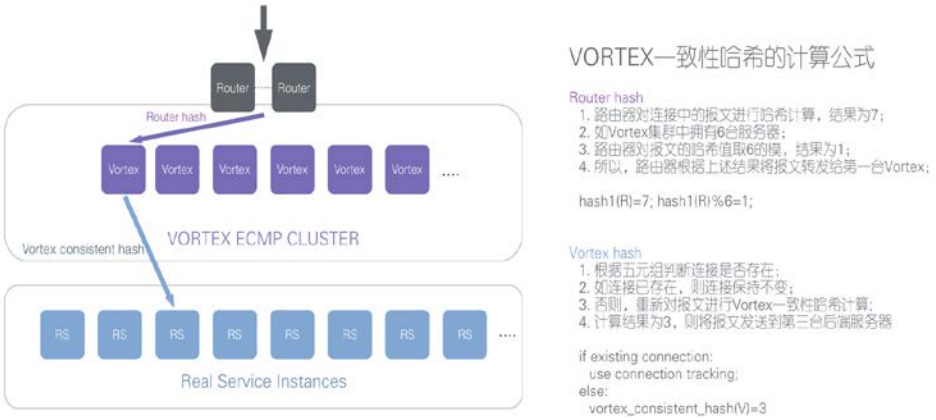
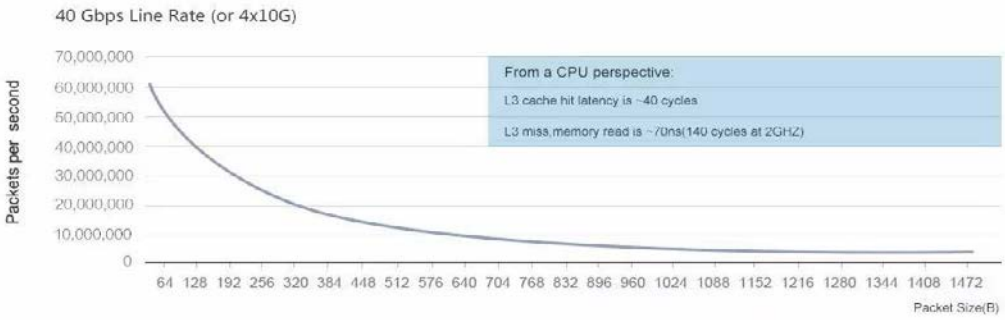


图 6：Vortex 一致性 Hash 算法的计算公式

图 7：负载均衡服务器和后端服务器集群同时变化的场景

这种场景下，如果负载均衡器是 LVS 且采用 RR(Round Robin) 算法的话，该数据包会被送到错误的后端服务器，且上层应用无法得到任何通知。如果 LVS 配置了 SH(Source Hash) 算法的话，该数据包会被送到正确的后端服务器，上层应用对此类变化无感知，业务不受任何影响；如果负载均衡器是 Nginx 的话，该数据包会被 TCP 协议栈无声无息地丢弃，上层应用不会得到任何通知。

其次，来考察后端服务器变化的场景。这种场景下，可能由于后端服务器故障由健康检查机制检查出来，也可能由于运维需要主动增加或者减少后端服务器。此时，Vortex 服务器会通过连接追踪机制保证当前活动连接的数据包被送往之前选择的服务器，而所有新建连接则会在变化后的服务器集群中进行负载分担。



Network Infrastructure Packet Sizes

Packet Size	64 bytes
40G Packets/second	59.5 Million each way
Packet arrival rate	16.8 ns
2 GHz Clock cycles	33 cycles

Typical Server Packet Sizes

Packet Size	1024 bytes
40G Packets/second	4.8 Million each way
Packet arrival rate	208.8 ns
2 GHz Clock cycles	417 cycles

图 8：DPDK 性能数据图

同时，Vortex 一致性 Hash 算法能保证大部分新建连接与后端服务器的映射关系保持不变，只有最少数量的映射关系发生变化，从而最大限度地减小了对客户端到端的应用层面的影响。这种场景下，如果负载均衡器是 LVS 且 SH 算法的话，大部分新建连接与后端服务器的映射关系会发生变化。某些应用，例如缓存服务器，如果发生映射关系的突变，将造成大量的 cache miss，从而需要从数据源重新读取内容，由此导致性能的大幅下降。而 Nginx 在该场景下如果配置了一致性 Hash 的话可以达到和 Vortex 一样的效果。

最后，让我们来看一下负载均衡服务器和后端服务器集群同时变化的场景。在这种场景下，Vortex 能够保证大多数活动连接不受影响，少数活动连接被送往错误的后端服务器且上层应用不会得到任何通知。并且大多数新建连接与后端服务器的映射关系保持不变，只有最少数量的映射关系发生变化。

如果负载均衡器是 LVS 且 SH 算法的话几乎所有活动连接都会被送往错误的后端服务器且上层应用不会得到任何通知。大多数新建连接与后端服务器的映射关系同样也会发生变化。如果是 Nginx 的话因为交换机将数据包送往不同的 Nginx 服务器，几乎所有数据包会被无声无息的丢弃，上层应用不会得到任何通知。

2.Vortex 的 Scalability 实现

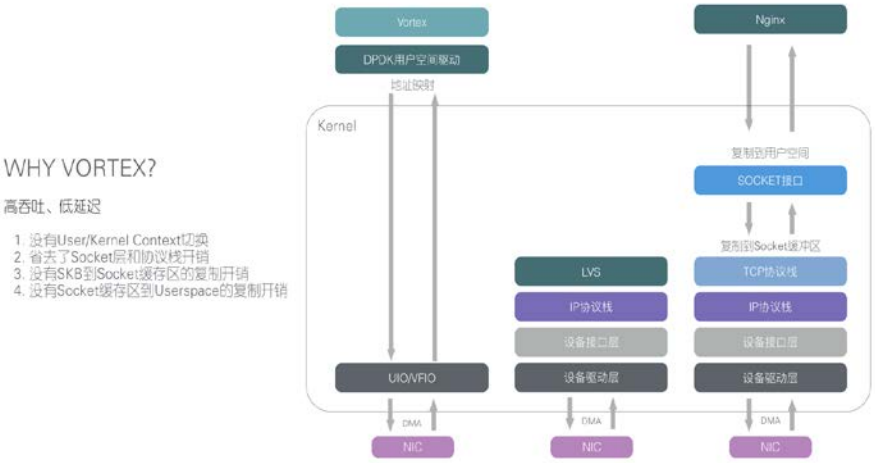


图 9：LVS、Google Maglev 和 UCloud Vortex 实现差异

2.1基于ECMP集群的Scaling Out设计

Vortex 采用动态路由的方式通过路由器 ECMP(Equal-cost multi-path routing) 来实现 Vortex 集群的负载均衡。一般路由机支持至少 16 或 32 路 ECMP 集群，特殊的 SDN 交换机支持多达 256 路 ECMP 集群。而一致性 Hash 的使用是的 ECMP 集群的变化对上层应用基本无感知，用户业务不受影响。

2.2基于DPDK的Scaling Up设计

虽然 ECMP 提供了良好的 Scaling Out 的能力，但是考虑到网络设备的价格仍然希望单机性能够尽可能的强。例如，转发能力最好是能够达到 10G 甚至 40G 的线速，同时能够支持尽可能高的每秒新建连接数。Vortex 利用 DPDK 提供的高性能用户空间 (user space) 网卡驱动、高效无锁数据结构成功的将单机性能提升到转发 14M PPS(10G,64 字节线速)，新建连接 200K CPS 以上。

内核不是解决方案，而是问题所在！

从图 8 可以看到随着高速网络的发展 64 字节小包线速的要求越来越高，对 10G 网络来说平均 67ns，对 40G 网络来说只有 17ns。而于此同时 CPU、内存的速度提升却没有那么多。以 2G 主频的 CPU 为例，每次命中 L3 缓存的读取操作需要约 40 个 CPU 周期，而一旦没有命中缓存从主存读取则需要 140 个 CPU 周期。为了达到线速就必须采用多核并发处理、批量数据包处理的方式来摊销单个数据包处理所需要的 CPU 周期数。此外，即使采用

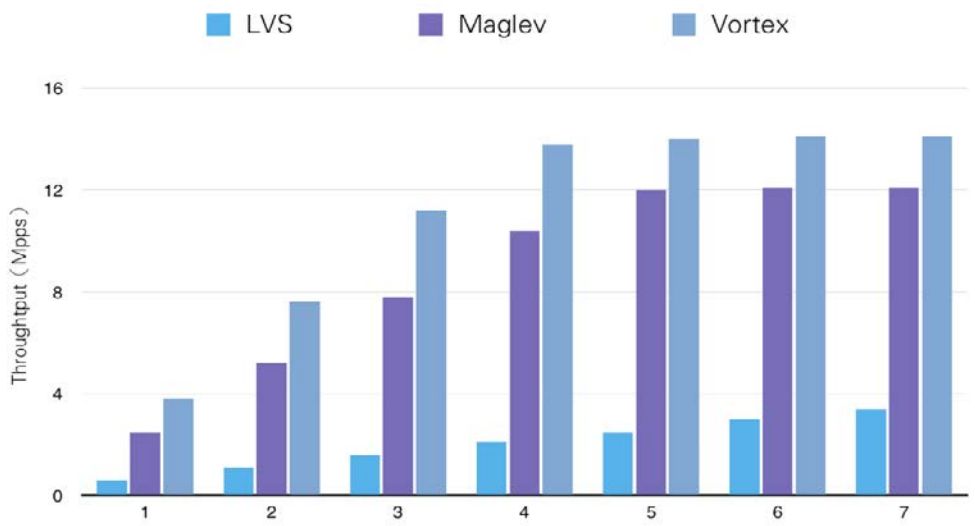


图 10：单机性能横向测试比较

上述方式，如果没有得到充分的优化，发生多次 cache miss 或者是 memory copy 都无法达到 10G 线速的目标。

像 Nginx 这样的代理模式，转发程序因为需要 TCP 协议栈的处理以及至少一次内存复制性能要远低于 LVS。而 LVS 又因为通用 Kernel 的限制，会考虑节省内存等设计策略，而不会向 Vortex 一样在一开始就特别注重转发性能。例如 LVS 缺省的连接追踪 Hash 表大小为 4K，而 Vortex 直接使用了 50% 以上的内存作为连接追踪表。

图 9 简明地比较了三者在实现上的差异：

Vortex 通过 DPDK 提供函数库充分利用 CPU 和网卡的能力从而达到了单机 10G 线速的转发性能。

- 用户空间驱动，完全Zero-Copy；
- 采用批处理摊销单个报文处理的成本；
- 充分利用硬件特性；
  - Intel DataDirect I/O Technology (Intel DDIO)
  - NUMA
  - Huge Pages，Cache Alignment，Memory channel use

Vortex 直接采用多队列 10G 网卡，通过 RSS 直接将网卡队列和 CPU Core 绑定，消除线程的上下文切换带来的开销。Vortex 线程间采用高并发无锁的消息队列通信。除此之外，完全不共享状态从而保证转发线程之间不会

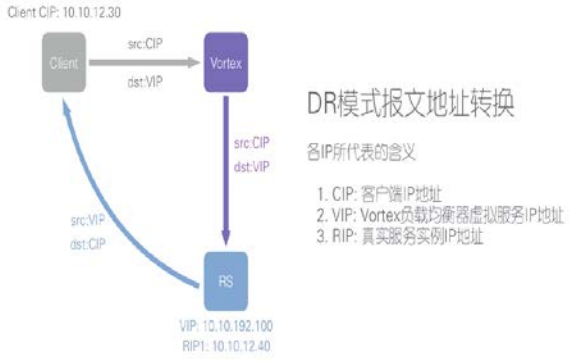


图 11：基于 DR 转发方式

互相影响。Vortex 在设计时尽可能的减少指针的使用、采用连续内存数据结构来降低 cache miss。通过这样一系列精心设计的优化措施，Vortex 的单机性能远超 LVS。单机性能横向测试比较，参见下图：

基于 DR 转发方式

LVS 支持四种转发模式：NAT、DR、TUNNEL 和 FULLNAT，其实各有利弊。Vortex 在设计之初就对四种模式做了评估，最后发现在虚拟化的环境下 DR 方式在各方面比较平衡，并且符合我们追求极致性能的理念。

DR 方式最大的优点是绝佳的性能，只有 request 需要负载均衡器处理，response 可以直接从后端服务器返回客户机，不论是吞吐还是延时都是最好的分发方式。

其次，DR 方式不像 NAT 模式需要复杂的路由设置，而且不像 NAT 模式当 client 和后端服务器处于同一个子网就无法正常工作。DR 的这个特性使他特别合适作为内网负载均衡。

此外，不像 FULLNAT 一样需要先修改源 IP 再使用 TOA 传递源地址，还得在负载均衡器和后端服务器上都编译非主线的 Kernel Module，DR 可以 KISS(Keep It Simple,Stupid) 地将源地址传递给后端服务器。

最后，虚拟化环境中已经采用 Overlay 虚拟网络了，所以 TUNNEL 的方式变得完全多余。而 DR 方式最大的缺点：需要 LB 和后端服务器在同一个二层网络，而这在 UCloud 的虚拟化网络中完全不是问题。主流的 SDN 方案追求的正是大二层带来的无缝迁移体验，且早已采用各种优化手段（例如 ARP 代理）来优化大二层虚拟网络。

F5-BIG-LTM-12250V	$169,995	$118,996	¥773,474 元	约 87 万人民币
VIPRION 机柜 F5-VPR-LTM-C4800-AC	$54,995	$38,496	¥250,227 元	约 14 万人民币
VIPRION 单刀片: LTM B4340 NEBS (96GMemory)	$169,995	$118,996	¥773,474 元	约 174 万人民币 (4 刀片集群)

图 12：基于 DR 转发方式

结语

采用 Vortex 作为 UCloud 负载均衡产品 ULB 的核心引擎，通过一致性 Hash 算法的引入，并结合 ECMP 与 DPDK 的的服务架构，解决了利用 commodity server 实现 high availability + high scalability 的高性能软件负载均衡集群的课题。

此外，ULB 对原有功能进行了扩展，重新调整了用户交互并增加了数十个新的特性。如优化了批量管理场景下的操作效率，SSL 证书提高为基本资源对象管理。未来，Vortex 将以 ULB 为产品形态输出更多的创新理念。

关于作者

Y3（俞圆圆），UCloud 基础云计算研发中心总监，负责超大规模的虚拟网络及下一代 NFV 产品的架构设计和研发。在大规模、企业级分布式系统、面向服务架构、TCP/IP 协议栈、数据中心网络、云计算平台的研发方面积累了大量的实战经验。曾经分别供职于 Microsoft Windows Azure 和 Amazon AWS EC2，历任研发工程师，高级研发主管，首席软件开发经理，组建和带领过实战能力极强的研发团队。

附录

按 7 折折扣并考虑每年 25% 的降价幅度，上文提到的一套 12250v 价格约 87 万人民币（生产环境中需要主备 2 台），一套包含一个机柜与 4 刀片的 VIPRION 4800 价格约为 188 万人民币。

基于Kubernetes打造SAE容器云

作者 丛磊

作为国内第一个公有云计算平台，SAE 从 2009 年已经走过了 6 个年头，积累了近百万开发者，而一直以来 SAE 一直以自有技术提供超轻量级的租户隔离，这种隔离技术实际是从用户态和内核态 hook 核心函数来实现 HTTP 层的进程内用户隔离：

如图 1 所示，这种方式的好处是：

- 精准的实现租户间CPU、IO、Memory隔离；
- 可以实现进程多租户复用，从而达到超低成本；
- 完全对等部署，管理方便。

另外，这种模式的最大好处可以实现完全的无缝扩容，SAE 自动根据 HTTP 等待队列长度进行跨集群调度，用户从 1000PV 到 10 亿 PV 业务暴增，可以不做任何变更，早在 2013 年，SAE 就利用这种机制成功的帮助抢票软件完成 12306 无法完成的任务，12306 抢票插件拖垮美国代码托管站 GitHub。

但是这种模式也有很大的弊端，最主要的弊端就是：

- namespace独立性不足
- 本地读写支持度不好

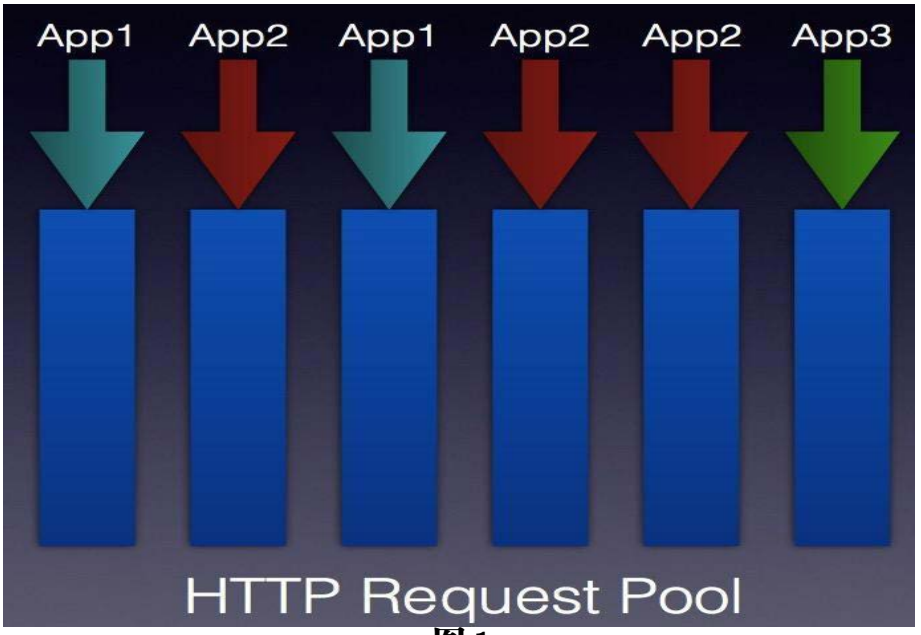


图 1

- 容易产生用户lock-in

针对于此，SAE 决定基于 Kubernetes 技术推出以 Docker 容器为运行环境的容器云，下面我们就以下三个方面展开讨论：

- Kubernetes的好处是什么；
- Kubernetes的不足有哪些；
- 针对不足，怎么改进它。

一、Kubernetes 的好处

选择一个技术，对于大型业务平台来讲，最重要的就是它的易维护性，Kubernetes 由 Go 语言编写，各个逻辑模块功能比较清晰，可以很快定位到功能点进行修改，另外，Kubernetes 可以非常方便的部署在 CentOS 上，这点对我们来讲也非常重要。

Kubernetes 提出一个 Pod 的概念，Pod 可以说是逻辑上的容器组，它包含运行在同一个节点上的多个容器，这些容器一般是业务相关的，他们可以共享存储和快速网络通信。这种在容器层上的逻辑分组非常适合实际的业务管理，这样用户可以按照业务模块组成不同的 Pod，比如，以一个电商业务为例：可以把 PC 端网站作为一个 Pod，移动端 API 作为另一个 Pod，H5 端网站再作为一个 Pod，这样每个业务都可以根据访问量使用适当数量的 Pod，并且可以根据自己的需求进

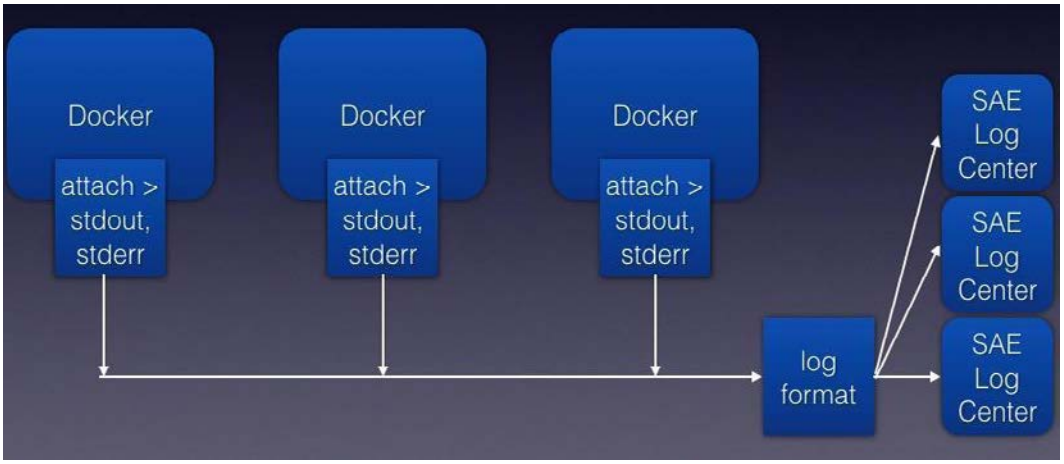


图 2

行扩容和容灾。

相同的 Pod 可以由 Replication Controller 来控制，这样的设计方便 Pod 的扩容、缩容，特别当有 Pod 处于不健康的状态时，可以快速切换至新的 Pod，保证总 Pod 数不变，而不影响服务。这种模式保证了实际业务的稳定性。

二、Kubernetes 的不足

对于目前的 Kubernetes 来讲，无缝扩容是一个比较致命的问题，不过好在社区有大量的人力已经投入力量在开发了，截止到我写稿的时间，基于 CPU 的扩容代码已经出来，也就是用户可以设定一个平均 CPU 利用率，一旦数值高于某个阈值，就触发扩容。但当我们仔细思考这个模式时，就会发现不合理的地方：CPU 利用率并不能真正反映业务的繁忙程度，也不能反映是否需要扩容。比如某些业务需要大量跟 API 或者后端数据库交互，这种情况下，即使 CPU 利用率不高，但此时用户的请求已经变得很慢，从而需要扩容。

再有，良好的监控一直是一个系统稳定运行的前提，但 Kubernetes 显然目前做的不够，先不说 Kubernetes 自身的监控，就是对于 Pod 的监控，默认也只是简单的 TCP Connect，显然这对于业务层是远远不够的，举个最简单的例子，假如某个业务因为短时间大并发访问导致 504，而此时 TCP 协议层的链接是可以建立的，但显然业务需要扩容。

还有一些其他的功能，不能算是不足，但在某些方面并不适用于 SAE，

图 3

比如 Kube-Proxy，主要是通过 VIP 做三层 NAT 打通 Kubernetes 内部所有网络通信，以此来实现容器漂 IP，这个理念很先进，但有一些问题，首先是 NAT 性能问题，其次是所有网络走 VIP NAT，但是 Kubernetes 是需要和外部环境通信的，SAE 容器云需要和 SAE 原有 PaaS 服务打通网络，举个例子，用户利用容器云起了一组 Redis 的 Pod，然后他在 SAE 上的应用需要访问这个 Pod，那么如果访问 Pod 只能通过 VIP 的话，将会和原有 SAE 的网络访问产生冲突，这块需要额外的技巧才能解决。所以我们觉得 Kube-Proxy 不太适合。

三、如何改进 Kubernetes

针对 Kubernetes 的不足，我们需要进行改进，首先需要改造的就是日志系统，我们希望容器产生的日志可以直接推送进 SAE 原有日志系统。

如图 2 所示，我们将容器的 stdout、stderr 通过 log format 模块重定向分发到 SAE 的日志中心，由日志中心汇总后进行分析、统计、计费，并最终展现在用户面板，用户可以清晰的看到自己业务的访问日志、debug 日志、容器启动日志以及容器的操作日志。

其次，既然我们不使用 Kube-

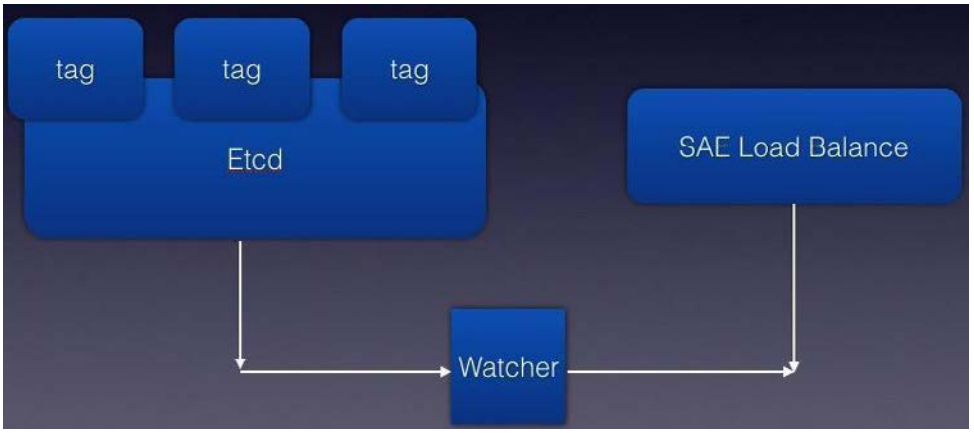


图 4

Proxy 的 VIP 模式，那么我们需要将 Pod 对接到 SAE 的 L7 负载均衡服务下面，以实现动态扩容和容灾。

etcd watcher 监控 etcd 里 pod 路径下的文件变更事件，得到事件取配置文件处理，判断配置文件里状态更新，将容器对应关系的变化同步到 SAE Load Balance，Load Balance 会在共享内存中 cache 这些关系，并且根据实际的用户请求的 HTTP Header 将请求 forward 到相应的 Pod 的对外端口上。当用户手工增加 Pod 时，Watcher 会在 100 毫秒内将新增的 Pod 的映射关系同步到 Load Balance，这样请求就会分配到新增的 Pod 上了。

当然，对于云计算平台来讲，最重要的还是网络和存储，但很不幸，Kubernetes 在这两方面都表现不够好。

网络

首先，我们来看对于网络这块 PaaS 和 IaaS 的需求，无论是 IaaS 还是 PaaS，租户间隔离都是最基本的需求，两个租户间的网络不能互通，对于 PaaS 来讲，一般做到 L3 基本就可以了，因为用户无法生成 L2 的代码，这个可以利用 CAP_NET_RAW 来控制；而对于 IaaS 来讲，就要做到 L2 隔离，否则用户可以看到别人的 mac 地址，然后很容易就可以构造二层数据帧来攻击别人。对于 PaaS 来讲，还需要做 L4 和 L7 的隔离处理，比如 PaaS 的网络入口和出口一般都是共享的，比如对于出口而言，IaaS 服务商遇到问题，可以简单粗暴的将用户出口 IP 引入黑洞即可，但对于 PaaS 而言，这样势必会影响其他用户，所以 PaaS 需要针对

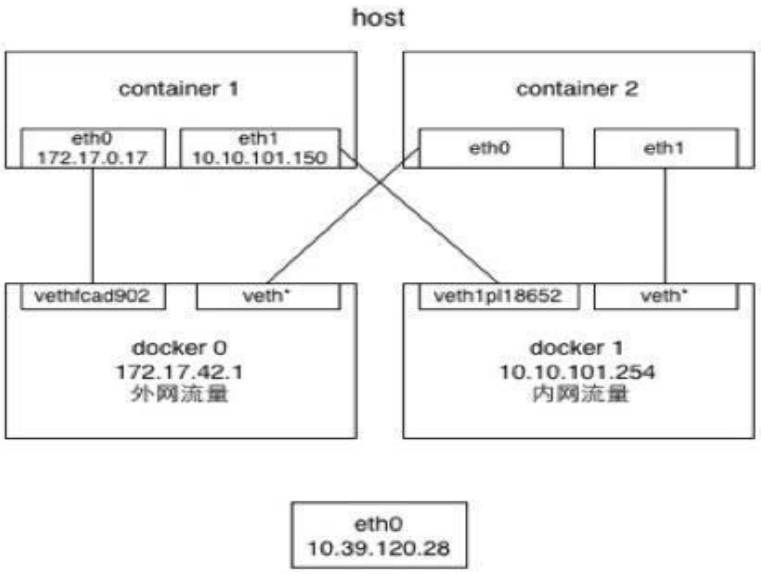


图 5

不同的应用层协议做配额控制，比如不能让某些用户的抓取电商行为导致所有用户不能访问电商网站。

目前主流的 Docker 平台的网络方案主要由两种，Bridge 和 NAT，Bridge 实际将容器置于物理网络中，容器拿到的是实际的物理内网 IP，直接的通信和传统的 IDC 间通信没有什么区别。而 NAT 实际是将容器内的网络 IP:Port 映射为物理实际网络上的 IP:Port，优点是节约内网 IP，缺点是因为做 NAT 映射，速度比较慢。

基于 SAE 的特点，我们采用优化后的 NAT 方案。

根据我们的需求，第一步就说要将内外网流量分开，进行统计和控制。

图 4 所示，在容器中通过 eth0 和 eth1 分布 pipe 宿主机的 docker0 外网和 docker1 内网 bridge，将容器的内外网流量分开，这样我们才能对内外网区分对待，内网流量免费，而外网流量需要计费统计，同时内外网流量都需要 QoS。

第二步就是要实现多租户网络隔离，我们借鉴 IaaS 在 vxlan&gre 的做法，通过对用户的数据包打 tag，从而标识用户，然后在网路传输中，只允许相同在同一租户之间网络包传输。

当网络包从容器出来后，先经过我们自己写的 TagQueue 进程，

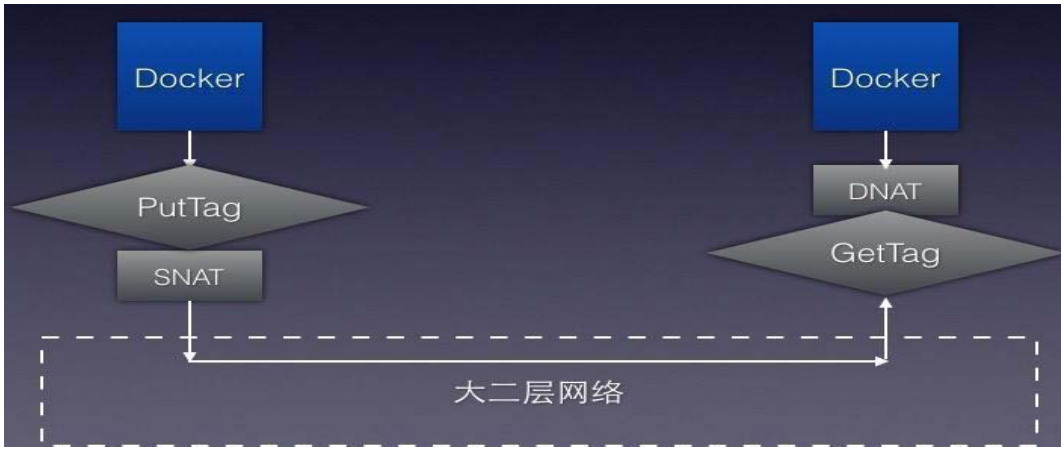


图 6

TagQueue 负责将网络包加上租户 ID，然后网络包会被 Docker 默认的 iptables 规则进行 SNAT，之后这个网络包就变成了一个可以在物理网络中传输的真实网络包，目标地址为目标宿主机 IP，然后当到达目标时，宿主机网络协议栈会先将该网络包交给运行在宿主机上的 TagQueue 进程，TagQueue 负责把网络包解出租户 ID，然后进行判断，是否合法，如果不合法直接丢弃，否则继续进行 Docker 默认的 DNAT，之后进入容器目标地址。

除去 Pod 间的多租户网络，对外网络部分，SAE 容器云直接对接 SAE 标准的流量控制系统、DDOS 防攻击系统、应用防火墙系统和流量加速系统，保证业务的对外流量正常

存储

对于业务来讲，对存储的敏感度甚至超过了网络，因为几乎所有业务都希望在容器之上有一套安全可靠高速的存储方案，对于用户的不同需求，容器云对接了 SAE 原有 PaaS 服务的 Memcache、MySQL、Storage、KVDB 服务，以满足缓存、关系型数据库、对象存储、键值存储的需求。

为了保证 Node.js 等应用在容器云上的完美运行，容器云还势必引入一个类似 EBS 的弹性共享存储，以保证用户在多容器间的文件共享。针对这种需求，Kubernetes 并没有提供解决方案，于是 SAE 基于 GlusterFS 改进了一套分布式共享文件存储 SharedStorage 来满足用户。

图 7 所示，以 4 个节点（brick）为例，两两一组组成 distributed-replicated volume 提供服务，用户可以根据需求创建不同大小的 SharedStorage，并选择挂载在用户

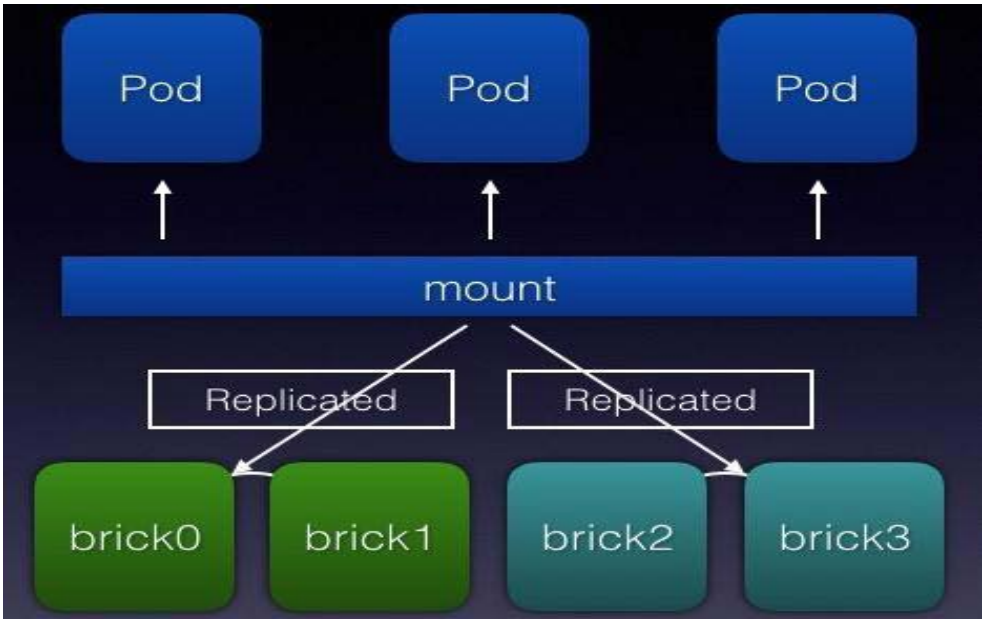


图 7

指定的文件目录，mount 之后，就可以像本地文件系统一样使用。我们针对 GlusterFS 的改进主要针对三个方面：1，增加了统计，通过编写自己的 translator 模块加入了文件读写的实时统计；2，增加了针对整个集群的监控，能够实时查看各个 brick、volume 的状态；3，通过改写 syscall table 来 hook IO 操作，并执行容器端的 IO Quota，这样防止某个容器内的应用程序恶意执行 IO 操作而导致其他用户受影响。

通过 SharedStorage 服务，用户可以非常方便的就实现容器热迁移，当物理机宕机后，保留在 SharedStorage 数据不会丢失，Kubernetes 的 replication controller 可以快速在另外一个物理节点上将容器重新运行起来，而这个业务不受影响。

总结

Kubernetes 是一个非常优秀的 Docker 运行框架，相信随着社区的发展，Kubernetes 会越来越完善。当然，因为SAE之前有比较完备的技术储备，所以我们有能力针对 Kubernetes 目前的不足做大量的改进，同时，也欢迎大家来 SAE 体验运容器平台，http://www.sinacloud.com/sc2.html。

洪强宁
宜信的PaaS平台基于Calico的容器网络实践

作者 魏星

容器云面临的网络挑战

在传统的 IDC 的架构里面网络是很重要的事情，在虚拟机环境中网络的发展已经有很多成熟的解决方案，现在为什么还在研究新的方案呢？因为云计算。云计算在 2013、2014 年之前的时间段，其主要的发力点在虚拟机。但是从容器技术出现之后，虚拟机被更轻量的容器方式颠覆，从而大幅降低开发、运维、测试和部署、维护的成本。当然也带来了很多在虚拟机里没有面临过的问题。

在网络层面，容器的本质是几个被封装了的进程。这必然导致每个容器都会有自己完整的网络栈、有自己的 IP。所以整个云的基础设施——集群里有大量的网络和 IP 要管理，并且进程的启动和结束是与容器伴生的，进程生灭的频繁程度超过了虚拟机几个量级。创建一个虚拟机会开启很多程序，程序之间彼此通信，但和虚拟机外部的通讯是少很多的。而容器的诞生，从网络管理角度来说，微服务架构呼之欲出。

用容器做迁移的时候，如果一台机器宕机了，对应用而言希望是无感知的。因此，如果容器所在的主机出故障，就带来了新的问题，即容器 IP 的任意漂移。在创建虚拟机的时候，IP 地址的分配是一种静态方式；但是在容器上面，IP 地址的分配通常是动态分配。一个容器启来的时候，通常我们不会知道它的 IP 是什么，只有起来之后才知道。

在容器云的场景下，我们倾向于把多个应用放在一个应用上运行，这导致应用和应用之间需要做安全隔离——我们不希望一个出问题的应用会影响其他的应用。所以说容器之间需要决定哪个容器可以通信，哪个不能通信。以前在虚拟机的时代，通过大量分层可以解决，但容器环境里这件事情的难度提高了。

所以我们选择网络解决方案的需求也发生了变化，最大的要求是容易管理。网络间通信量大了，不能够承受太大的损耗。而当发生容器间迁移的时候，需要一种机制，来保证迁移后的 IP 不会发生变化。同时，它的解决方案是需要细粒度的安全策略。

容器本身是一个单机软件，并没有为跨主机的网络考虑太多。跨主机容器互联的解决方案最典型的是两台主机之间通过一个交换节点实现连通。在主机内起十到几十个主机的规模，考虑到主机上有没有足够的 CPU 和内存，在这个主机上做静态 IP 规划几乎是不太可能。这时候我们就引入 SDN 技术——用软件的方式定义网络，软件数据发生变化时网络就相应地发生变化。

常见 SDN 方案

简单的谈一下 Docker Bridge，关于主机内部的网络管理工作。每个容器里面都有一个虚拟网卡，和主机上的虚拟网卡做配合，所有容器内的虚拟网卡都可以和主机通信。通过端口映射，我调用对方的容器服务的时候，不能使用该容器直接的地址，必须使用它主机的地址。因为有了这样一个转发，网络通讯的性能有所损耗。当然，还有一个问题更严重，访问你的容器先要搞清楚你的主机是什么，这样网络通讯会跳来跳去。不但麻烦，还会带来端口冲突。每创建一个容器会绑定一个端口，怎么管理好这些端口是一个很大的挑战。

第二个解决方案是 Flannel，这个想法很好。每个主机负责一个网段，在这个网段里面分配一个 IP 地址。访问另外一台主机的时候，通过网桥到达主机上的 IP 地址，这边会有一个设备，程序会把你发的包读出来，去判

断你的目标地址是什么，归哪台机器管。还会把你的 IP 包外面加一个 UDP 包，发到目标地址。收到之后，会把前面的包扔掉，留下来的就是目标容器地址。这个方法有几个问题，第一个是要做封包的操作。第二个问题是每个主机上的容器是固定的，容器的不同主机之前的迁移必然带来 IP 的变化。

再来说说 Weave，他的思路是共享 IP 而非绑定。在传输层先找到目标地址，然后把包发到对端，节点之间互相通过协议共享信息。Flannel 和 Weave 的特点是类似的，都用的是 UDP 或者是 VxLAN 的技术。事实上使用 UDP 性能是很差的，VxLAN 和 UDP 差不多，它有一个网络切隔，而且在里面可以跑二层协议。还有一种是 IPIP 封包，直接把一个 IP 包封装在一个 IP 包里面。

以上不难看出，原生网络性能比较差，使用 UDP 的时候性能损失在 50% 以上，使用 VxLAN 也会有 20% ～ 30% 的损耗。所以我要在内核上封包，使用硬件来解决这些问题。而且容器和容器之间出现通讯故障，调试的时候非常困难。

关于封包

回过头来想一下，我们为什么要封包？其实它是改包，主要解决的问题是同一个问题。即容器网络里，主机间的不知道对方的目标地址，没有办法把 IP 包投递到正确的地方。

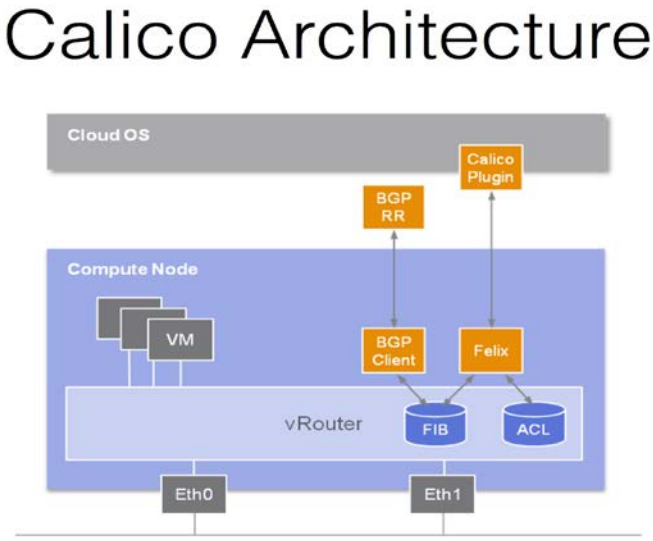
传统的二层网络是用路由来互相访问，不需要封包。但是路由规则怎么维护？利用 BGP（基于 TCP 之间两两连接）搞一个分布式的路由集群能否满足规模？

Metaswitch：容器内部配一个路由指向自己宿主机的地址，路由规则下发，这是一个纯三层的网络没有封包，接近原生网络性能。

在容器里面我们怎么做呢？我们发现了 Calico 项目。不同主机上的每个容器内部都配一个路由，指向自己所在的 IP 地址；每台服务器变成路由器，配置自己的路由规则，通过网卡直接到达目标容器，整个过程没有封包。

Calico

所以这是一个纯三层网络，没有引入一个 DP，没有封包。在主机内部做另外一个容器，可以三条到达终端，你可以知道是谁出了问题，调试的时候很容易，很好管理。容器内的应用数据传出来，和二层完全隔离，对于我们绝大多数的应用来讲只需要三层就够了，很少有应用处理二层。



那么，路由交换是不是很难呢？用传统的 BGP 技术就可以实现。这个协议在大规模应用下是一个很好的场景。而且 BGP 有一个自治域的概念。在这个场景下会有一个问题，路由之间的信息交换实际上是基于 TCP 协议，每两个之间都有一个 TCP 连接，规模大了以后复杂度非常高。所以在这种场景下，大概是几百个主机直接使用 BGP 的方式来做；再大一点的话，选择分布式互联，这样可以把网络规模继续扩大，最后能够支持到几万台机器。如果不需要这些策略，不做商业逻辑和业务逻辑，只是用 BGP 来做路由互联的话，其实还是很简单和高效的。

金融业务的具体实现

宜信内部的做法是用 powerstrip 实现，还有一个是手工去做，启动容器直接用 Docker，然后自己再去设置 IP。如果容器之间都互通，创建容器的时候都通过配置环境变量，会创建一个 Profile，配置好 iptables 以及 ipset 可以保护我们的服务器。再稍微复杂一点的话，可以修改 ACL 规则，如果有很多个 APP，每个 APP 都需要访问数据库，每增加一个 IP 都要修改配置这样显得太麻烦，我们可以通过给所有的 APP 打标签的方式来解决这个问题。

配置容器的三种方法：

- 用powerstrip来Hijack劫持Docker的API，这个思路很有意思。
- calicoctr container命令。
- libnetwork驱动（需Docker 1.9以上）。

因为做的是金融业务，性能其实不是最吸引人的，ACL应用是宜信关注的重点。比如在宜信的场景下做云平台的时候，会有应用的概念。一个应用有多个容器，互相可访问；不同应用的容器互相不能访问；一个应用对另外一个应用提供服务，作为服务容器并且规定哪些可以访问。所有的应用容器能够访问集群，应用容器不可以访问自己所在的集群的任何一台服务器。总之，我们要实现应用可以访问主机上指定的服务，可以访问除本网段之外的其他网段。

Calico使用总结

Calico的应用场景主要是IDC内部，推荐部署在二层网络上，这样所有路由器之间是互通的。这种场景是大二层的解决方案，所有服务器都在里面。但大二层主要的问题是弹性伸缩的问题。频繁开关机的时候，容器启停虽然不影响交换机，但容易产生广播风暴。网络规模太大的时候，一旦出现了这样的问题，整个网络都会有故障，这个问题还没有解决的特别好。

Calico作为一个纯二层的应用问题不大。我们可以把集群分成了很多网段，对外是三层网络的结构。集群内部分成多个自制域，比如一个机柜是一个自制域，之间通过交换路由的方式实现在三层容器到容器的互通。瓶颈在于容器的路由信息容量。所以说本质上它不是一个应对互联网规模的解决方案，但对宜信场景是够用的。

我们实施的时候遇到一些坑，比如容器在启动时没有网络、操作ETCD的时候BGP配置频繁加载，等等，最后我们多一一解决了。我们也发现了driver方案功能方面的弱点，现在还继续使用劫持API的方法。如果要在云上使用，如果支持BGP，直接可以用；如果不支持BGP可以用IPIP。现在英特尔的网卡支持两个协议，一个是GRE、一个是VxLAN。如果真的把它放在云上，我们可以考虑一下把VxLAN引进来，性能上的差别不是很大。因为SAE之前有比较完备的技术储备，所以我们有能力针对Kubernetes目前的不足做大量的改进，同时，也欢迎大家来SAE体验运容器平台，http://www.sinacloud.com/sc2.html。

廖春毅：再造一个WinTel联盟

作者 魏星

【导读】从斯坦福大学开始在网络方面创业的公司有 3 个，分别是 Nicira（后被 VMware 以 10.5 亿美元收购）；另一个是 Big Switch；再有一个便是 Pica8。起初 Nicira 专注在应用的部分，Big Switch 专注在控制器方面，而 Pica8 则在做 Switch 的部分，是交换机厂商中最早做 SDN 的团队。Pica8 创始团队自 2009 年始便与“SDN 之父”、斯坦福大学 Nick McKeown 教授的研究小组合作，专注于 SDN 技术的研发，潜心三年于 2012 年初推出第一款产品，主要专注在 SDN 和白牌机上。此外，Pica8 还服务于 Facebook 开源的 OCP 项目，并在 2015 年的 OCP 大会上亮相。

在过去的 25 年里，除了网速越来越快网络技术（协议、功能、框架等方面）本身并没有什么大的进步（当然，存储技术的进步也不多）。2008 年的时候在手机市场出现了一个巨大的变化——软件和硬件分离、应用跟系统分离。这种变化在 2009 年蔓延到

了网络设备市场，在摩尔定律作用下，Broadcom、Altera（后被Intel以170亿美元收购）、Marvell等商用芯片更新换代的速度远远超过了软件性能升级的速度，软件和硬件的分离使得白牌机机市场骤然出现，白牌机不但让产品价格大幅降低，还催生了新的协议——也就是SDN技术的出现。硬件标准化之后大量应用不断涌现，其多样化对SDN客户而言意义十分重大。

SDN也是笔者关注的重点，机缘巧合之下笔者与Pica8创始人之一的廖春毅有了一次交流，于是有了以下的内容。

一家没有销售团队的公司

InfoQ：请您简单介绍一下自己和Pica8的团队，以及研发团队的占比情况？

廖春毅：全公司大概有70人，北京研发中心有40～50人、美国有25人左右。

海外研发团队侧重跟合作伙伴的沟通多一些，跟客户的接触多一些；很多标准化的工作也是有海外团队来负责和完成的。国内团队则更专注在产品研发上，相对来说更核心一些。

InfoQ：Pica8招聘技术人员的途径有哪些？比较注重哪些方面素质？

廖春毅：目前仍以网络招聘为主；在招聘的时候比较看重基本的网络技术功底、以及对新技术的狂热。在过去的6个月我们团队增加了一半的人手，目前的目标是100人。

InfoQ：我看到Pica8领导团队里有Business Development的角色，也有Marketing的角色，这两者的区别与定位是什么？此外，为什么设有Product Management但是却没设销售的岗位？

廖春毅：首先我想说的是，Pica8的组织架构特别有湾区的风格，这与公司组织架构的设计有关。简单地说，我们设置了对内（合作伙伴）和对外（教育市场）两种角色。

目前白牌机市场十分复杂，我们的产品定位是对内而言，因此Product Management做的主要是这部分工作。Marketing的作用是对外教育市场，主要是告诉客户什么是SDN、为什么需要SDN，现在的Marketing以美国为主，在国内并没有。

Business Development研究的是商业模式。具体说是研究到底该怎么卖产品，并不像大家一般理解的是对外的商务拓展，跟销售是有区别的，BD是研究的是卖什么、卖给谁、以及怎么卖，然后决定引入哪家销售合作即可。基于此，Pica8不需要销售人员。

研发没有长期规划只有短期路线

InfoQ：Pica8 花了 3 年开发 PicOS，直到 2012 年 2 月才推出第一款产品。能谈谈这 3 年研发过程中比较难忘的经历吗？

廖春毅：最开始，跟斯坦福合作的时候我们双方都不清楚要做什么，我们花了很多时间讨论是不是开发一个新的协议，也花了很多时间兼容二层和三层的网络。Pica8 最初借了点钱运营了一个小团队，2011 年底的时候钱花完了。这时我面临一个选择，要么卖公司、要么卖产品。我打电话给杜林（Pica8 VP、国内研发团队负责人）说，必须推出一款产品啦。可是杜林跟我说，产品还没有好。但是没有办法啊，于是在 2012 年初我们找了几个大客户，基本上是斯坦福出面买了第一批产品，公司才有钱周转继续运营了下去。

我现在回想起来会觉得，创业开始的时候并不知道会走向哪里，也并不知道未来的产品会成为什么样子。

InfoQ：如果 2011 年底的 Pica8 不缺钱，你们的研发进程会不会继续下去？

廖春毅：最初的时候正是因为没有钱所以我们的研发才拖了 3 年——因为借来的钱你会省着花。债主不关心你怎么花，这样我们有时间、没压力才把东西做好了。你看很多做 SDN 的技术公司后来都没了，因为最初大家都不知道产品要做什么，做基础设施一定要有耐心。这跟做互联网不大一样，互联网讲求快速迭代。

2012 年 9 月我们进行了第一次融资，2014 年第二次融资，我们的产品在市场上表现良好，因此资金对我们来说也不是问题。

InfoQ：研发团队里产品经理与工程师的话语权怎么分配？怎么决定优先开发哪个新功能或者特性？

廖春毅：在 Pica8 里，这完全是产品经理说了算。PM 要跟市场合作，要跟销售沟通，也要跟工程师沟通。PM 是一个团队，做的是沟通、权衡、决断。沟通是对方面的，既要从销售人员那里听客户的声音，也要跟工程师沟通实现需求的细节，还要跟市场部门做好下一步的推广计划。当出现分歧的时候（蛮经常的），PM 要做的就是权衡得失，作出决断。

InfoQ：研发过程中会有很多来自客户的需求，产品的研发规划通常是怎么做的？

廖春毅：我们的长远规划其实是一个愿景，而每一版的功能、每一次的迭代都是短期的——大概 6 个月的路线图。通常我们无法做全年的

研发计划，因为市场的变化太快，比如 2014 年的时候大家都嚷着要 SDN，2015 年这个声音突然就变成了要白牌机。如果非要按照市场的变化来做规划，往往会拖垮开发团队。

InfoQ：您能说说自己最喜欢的 Pica8 产品的特性吗？

廖春毅：业界普遍认为，OpenStack 的 VM 很好，存储还可以，但网络是痛点。在我看来目前市场上所有的网络解决方案都有一个无法扩展的缺陷。比如以 VMware 为代表的一派，用虚拟机解决网络的问题，以思科为代表的一派，用硬件解决网络的问题——但都要买思科的。Pica8 的网络系统加解决方案可以让白牌机解决扩展性、安全性和功能的问题。在这一方面我们已经跟国内的运营商合作，这个痛点的解决是 Pica8 最引以为傲的。

InfoQ：那您能否透露一下 2016 年的研发规划中，重点的目标或者说方向有哪些？

廖春毅：我觉得有三个大的目标吧。其中第一个部分是白牌机标准化，要让 Pica8 的 OS 可以跑在所有的白牌机上，兼容所有白牌机产品；第二是聚焦几个新的技术，SDN 与传统网络的兼容；第三是要能给客户提供解决方案的实现和实施。

InfoQ：您在谷歌的时候参与了 OpenFlow 的研发。目前 OpenFlow 协议成为 SDN 南向接口的通用标准之一，您能谈谈 PicOS 的南向接口的开发吗？

廖春毅：南向接口技术其实很多，比如 VXLan 也是。目前 Pica8 兼容了所有的南向接口，我们花了三年的功夫解决了这一个问题。这也是为什么那么多做 SDN 的公司研发一年就把产品推向市场，但是不久公司就关门的原因之一吧。客户的场景是多样的，要兼容很多东西，只卖单一的技术产品不能解决多元和复杂的问题。Pica8 用多元化的解决方案，使得芯片中最重要的功能都可以被客户开发利用，从而赢得了市场。

InfoQ：南向面对的是物理硬件，北向面对的是软件应用。统一的北向接口标准可以降低在不同控制器上开发应用的成本。PicOS 在北向接口方面做了哪些工作？

廖春毅：我个人觉得北向是个伪命题。做网络的公司首先假设客户会先买一个控制器，然后再决定在控制器上面做什么功能。但是现有的控制器都是基于开源的技术，没有一个商品化，为什么？北向接口没法闭源！

北向接口的复杂度太高了，并且北向接口又跟应用直接对接，应用会

频繁访问这些接口，根本没有办法在短时间内把它定义并封装成一个稳定的接口（做这件事会累死人），所以只能开源。这样一来控制器不会商品化而只能变成开源项目，开源项目商品化的典范是RedHat，大家看看RedHat做的事情就能明白过来了。

InfoQ：关于东西向接口的标准，目前在业界中还未得到重视，仅OPenNFV一个项目组在关注这个方向。您能谈谈这个话题吗？

廖春毅：自从视频应用出现后，网络流量南北与东西向的比例从1:10增长到1:100甚至更大。现在一个网络请求进来，会产生几何数量级的东西向流量。2012～2013年之后，东西向网络技术在BGP下面出现了P2P利用，这么做其实是为了安全（云厂商多租户的隐私与信令风暴问题）。

所以在我看来，唯一的结论是，东西向的流量会越来越重要。至于技术发展要看具体的应用场景，比如SNS应用和视频应用的链路其实很不一样，所需要的网络技术也大为不同。

乱拳打死老师傅

InfoQ：主推NFV的厂商更愿意去做Controller，但硬件厂商并不愿意做白牌机。PicOS为什么选择从白牌机做起？

廖春毅：白牌机是趋势，硬件厂商是否愿意并不能改变这一方向。纵观个人PC的发展历史不难发现这个规律，康柏、微软正是顺应了这一潮流才站了起来。这里我想说一下“典范转移”的原理，过去的典范已经不能适应时代了，新的典范将统治世界。仔细想想Wintel联盟的强大吧，大型机正是被这个联盟扫进了历史的垃圾桶里。

为什么Pica8要选择做白牌机？我常讲“打群架”的理论——乱拳打死老师傅。诺基亚是怎么倒下的？不是被苹果击倒的、也不是被谷歌打趴下的，而是被安卓联盟——摩托罗拉、三星、联想、HTC等等所有安卓阵营（乃至富士康、夏普等整个产业链）的人打败的。现在做网络的公司任何一个都无法跟思科竞争，即便是华为也不行，如果大家结成一个联盟会怎样？坦白讲，对于硬件厂商来说，做白牌机的利润还是很高的。

InfoQ：目前业界有各类开源SDN解决方案，如Calico、OpenDayLight、Neutron DVR、Dragonflow等。您怎么看这些技术？能简单说说Pica8跟现有的SDN解决方案相比有哪些优势和劣势吗？

廖春毅：SDN的组成有三个部分，首先是盒子，其次是运行在盒子里的

操作系统，最后是控制器——控制很多盒子，这样才能架构一个完整的SDN网络。Pica8不做控制器，只做OS，白牌机加上我们OS就变成了一个可用的盒子。从生态的角度来说，基于开源技术的控制器都是APP，就像苹果手机应用商店里的那样，而APP的开发周期和成本相对来说都较低，到时候会有成百上千的应用。

Pica8是跟安卓一样，我们会不断去寻找自己的联盟。最终的形态是，下面是硬件厂商，上面是应用，联盟里的伙伴越多对我们越有利。而这个联盟正是我们的目标和优势。

InfoQ：目前Pica8的主要市场在海外，您对国内市场怎么看？国内外市场有哪些差异？

廖春毅：国内市场的特点是潜力巨大，需求高。但目前Pica8面临着两方面的问题，第一是我们在国内没有品牌影响力，第二是我们的商业模式在国内有待验证。

国内市场除了三大运营商和BAT等大互联网公司外，银行等金融企业也是主要客户。企业市场要认同我们的商业模式才可以。在我们看来，切入中国市场最快最好的方法是掌握技术，而不是拥有品牌。因为掌握品牌并不能做技术上的区分（比如华为和华三这样），因此我们在中国的策略是跟品牌商合作。比如华为，我有技术你有品牌，为什么不合作呢？Pica8这么小又不是系统商，华为、华三不会视Pica8为竞争对手，这样一来中兴、浪潮、联想等都可能成为我们的合作伙伴。这样我们的联盟和生态就构建出来了。

随着云计算SDN的发展，大的品牌商必然会切入网络市场，业务云化一定是有存储、有网络的。网络的重要性会越来越被人认可。之于我们的模式，这并不是谁的idea更优的问题，也不是能力的问题，最后是执行力的问题。

美团云的网络架构演进之路

作者 朱雯俊

在传统的观念里，美团似乎一直被认为是家提供吃喝玩乐的团购平台。但其实，从深入电影、外卖等领域起，美团就早已不再是一家团购公司了，打开今天的美团APP：电影、外卖、机票、酒店、上门、甚至周边游等多重垂直业务均被囊括其中。

2015 年年底，美团与大众点评宣布合并，合并后新美大的年活跃用户量达到 1.7 亿，高速增长的业务压力和巨额交易量的背后，是美团云提供的技术支持，让其保持着平稳运营。

作为领先的 020 电商云和大数据解决方案提供商，美团云在 2015 年入选了“TOP100 年度技术创新案例”。而今天我们要分享的，则是美团云的网络架构从最初到现在，是如何一路演进而来的，在这个过程中，又产生了有哪些产品和哪些思考。

做云是水到渠成的事

总有人会问美团为什么要做云？

做好电商才是美团该走的路。其实不然，亚马逊、阿里都是从电商起家的，而他们两家分别是国外、国内体量最大的，所以，大规模的电商网站具有把云做好的天然优势：

一方面，云的核心技术一定是由规模驱动的，大规模的电商在这方面有更深的积累。因为电商的特点除了流量大，其流量峰值波动也非常大，比如一些特殊的节日，用户访问量的峰值就会很高，所以电商在资源的弹性调度方面有更多的经验。2012 年，美团始逐步创建自己的私有云平台，2013 年 5 月正式对外推出公有云服务。2015 年更是扩建了新的数据中心，并推出了更多的组件服务。

另一方面，美团拥有的大数据相关实施经验，使得美团云能够对外提供更有针对性的大数据融合解决方案。

从技术角度来讲，美团网是一家完全云化的电商平台，规模体量居国内最大。目前美团的交易量仅次于阿里集团，只不过，阿里巴巴的电商业务绝大部分并未完全跑在阿里云上，而美团网所有的业务和交易，从 2013 年开始就完全跑在美团云上。

这个过程中，美团云在虚拟化、运维等方面积累了相当多的技术经验。同时，美团云对外输出的不仅是底层 IaaS 的云服务，更有大数据解决方案。因此，美团做云是一件水到渠成的事，美团云希望成为美团网技术积累对外输出的窗口，为更多的创业者、中小企业包括正在进行”互联网＋”的传统企业提供基础设施云服务，解决大家在技术方面的后顾之忧，而能够专心业务发展。

从私有云到公有云

美团网早期架构是从私有云做起的。目标是，资源云化和快速交付。值得一提的是，美团云从一开始就没有完全选用 OpenStack，而是决定自研云平台。原因在于当时 OpenStack 并不成熟，只有个别组件比如 glance、keystone 是合适的，所以在虚拟化、网络层，美团云进行了自主研发。

现在看来，这样做是对的。因为 OpenStack 偏向私有云，如果当初完全基于 OpenStack，现在做公有云将比较困难。但美团云选择自研云平台，结合自身业务，所以现如今能够平稳地支撑着所有业务。

当处于私有云的阶段时，主要的事情是把资源动态管理起来，对访问控制和资源隔离没有做太多要求。最初，美团云主要通过账号登陆管理、日志进行事后审计。私有云之后，推出的是办公云。办公云主要针对研发、测试人员，进行内部的测试使用。在

这个阶段，美团云已经开始为公有云做准备，建立了账号体系、计费系统等这些功能。

办公云的存在，在现在看来有一个很大的好处，就是每一个上线公有云的功能都会先在办公云上线，保证每一个功能的迭代都是稳定可靠的。也就是说，办公云实际提供了一个真实的线上测试环境。办公云之后，美团云对外推出了公有云服务。

早期的公有云和办公云的架构大体类似，拥有更用户友好、更完善的计费和消息系统、开放 API 等。其中，公有云最早的底层网络特点有几个，一是网络都是千兆网络，对软件性能要求不高。二是底层采用 VLAN 大二层，通过 OVS 控制器对用户进行隔离。由于早期流量不是太大，千兆的流量用 OVS 来控制尚可，控制器性能不够的情况尚且不多。但随着用户数量的增多，以及使用量的变大，后续开始出现问题。这也恰恰促成了美团云进入全新的网络升级时代。

从微观角度来讲，早期的公有云存在一些问题。首先，在稳定性上，内外网都是一根网线单上连一个交换机，一个地方出问题整个网络就会出问题。其次，外网、内网、管理网都是一根网线，这是在没有冗余的情况下，如果要做冗余的话，就要变成六根网线，成本太高昂。其三，千兆网络渐渐开始不能满足用户需求。还有一些隐藏问题，比如当时所有的用户都是在交换机的一个 vlan 网络下面。

理论上来说，这样是可行的。但实际上，交换机对 VLAN 的支持能力限制了网络规模的扩展，用户数量受到限制。再比如软件隔离占用宿主机计算资源，可能会出现响应不了或者抢占用户 cpu 的情况。同时，在这个网络下想实现用户自定义网络（vpc）就非常困难，灵活性低。

因此，在经过了不断地改进后，美团新的公有云网络架构在物理链路、主机网络、网关、控制器四个纬度上全面升级，大大提高了整体网络性能。

四个纬度上的性能释放

首先，从物理链路来看，性能方面，美团云实现了万兆互联；其次，在核心上实现了双机冗余，不会因为某个物理环节问题，导致网络不能启动；第三，采用了 TOR 交换机堆叠，双 40G 上联，随着日后网络流量的增加，可以再扩展。此外，在网线的选择上，美团云还采用了 10G Base-T 的电口万兆网络，这个技术比较新，很多交换机厂商都还没有这样的设备。但是它的成本较低，运维起来也会更方便。另外，在机房建设的过程中，

美团云还使用了一些目前业界领先的技术，比如核心机柜封闭冷通道、预端接，对成本的节省都是百万级的。

机房出口挖断了怎么办？同城多个互联网数据中心（IDC）之间，通过边界网关协议（BGP）来进行备份和冗余。当有一个机房的网络断掉的时候，会通过边界网关协议的流量自动转移到另一个机房。

但是底层的物理链路是万兆，不代表上层能把万兆利用起来。我们花了更多的精力，解决如何把万兆网络利用起来的问题。一部分是网关，就是整个网络出口的部分，比如 DPDK 技术。DPDK 技术目前是被主流使用的技术方案，对释放网络性能有较大帮助。另一种，预留 1-2 个处理器（core）接受数据，一个处理器根据自己的逻辑负责处理控制信息，1-2 个处理器负责收包，其余处理业务，自己处理数据分发。

在实际使用中，美团云根据两种模型的优势，分别都有选择。在浮动 IP 网关、负载均衡网关、 DDoS 清洗设备三个部分，实现了全面的 DPDK 化，同时在四层网络上，能够并发 1000w 连接情况下新建连接 100w / s。

“以最小代价解决最大问题”

当网关不是瓶颈的时候，流量就能够自由通到主机上，所以接下来就是通过主机网络释放性能。美团云最早使用的 OVS V1.1 版本，在千兆网络下可行，但万兆网络下性能远远不够。升级到 V2.3 后平台后，Megaflow 对高并发情况下性能有数量级的提升，创建能够满足要求。

但另一个问题出现了，在单流的情况下，对万兆网卡的利用率仅为 50%。随后在升级到 V2.4，支持 DPDK 版本后，美团云进一步提升了单流转发性能。在新版本的 OVS 下，只要 10% 的计算资源就可以提供万兆的网络能力，网络数据处理不影响用户计算资源。这样一来，就解决了宿主机的物理网络瓶颈。

在控制层面，有两个选择，一个是传统工具 eptables/iptables，二是 OVS 的方案。所谓 OVS 的控制方式，是配置流表，交由控制器处理。控制器决定是否放行，动态地下发对应流表，在 OVS 控制器对数据包进行过滤和处理过程中，美团云开发了软件层面的解决方案。针对单播，通过对 SYN 包检查，下发流表，并对每个不匹配的 UDP 包进行检查。

需要注意的是，由于发送端较难控制，而接收端对每个包处理，容易造成控制器队列积压。因此，美团云采用下发临时流表的方式解决积压问题，或者通过设置限流阈值，进行快

速恢复。

但是软件层面的解决方案无法根本解决积压的问题，因此下一阶段的迭代就是在硬件层面进行隔离，通过 VXLAN 对用户进行隔离。说到选择 VXLAN，就要提到对 SDN 方案选用的一些思考：在底层的万兆物理链路之上，美团云选用了 Overlay 的网络架构。

简单来说，Overlay 的架构弹性灵活，业务与物理链接和端口分离，这就意味着网络不再受限于物理上的连接和端口数量，可以按照资源池的概念来分配网络资源。而 Underlay 作为整个 SDN 框架的基础，充分吸取和延续了过去长期积累的物理网络优势，稳定可扩展。一方面 ARP/OSFP/BGP 仍然值得信任，另一方面相关领域的运维专业人才相对储备也较多。在参考了业界最新的实践经验后，美团云选用了 VXLAN 的解决方案。

要做就做行业标杆

上述是在物理链路、主机网络、网关、控制器方面释放性能上，美团云所做的尝试。再上层就是让用户可以灵活地自定义自己的网络。为了应对灵活性的挑战，美团进行了相应的处理，比如分布式的 DNS。

在传统网络下，一般使用默认的 DNS 服务器地址，并通过源 IP 区分用户。但是在用户定义网络（vpc）的情况下，用户的地址是可以重复的。所以用户识别方面，需要将 VXLANID 的用户信息嵌入 DNS 数据包。另外在用户网络中，DNS 服务器的地址也是自定义的，所以实际的 DNS 服务需要使用 Underlay 地址，这里面就需要做地址的转换和映射。

总体而言，新公有云的网络结构全面升级为万兆网络层面，管理网做 Bonding，用户的内网外网 overlay 在管理网。VPC 层面，通过 VXLAN 隔离用户，并实现自定义的网络。最后对外提供丰富的产品功能，比如浮动 IP/ 负载均衡，对象存储 / 块存储，RDS/Redis 等。

未来，运维自动化的程度会进一步提高。通过 openflow 或者 netconf 等通信手段提取到控制器上，进一步整理和分析后，能够形成可视化的网络路径图，实现更高效的网络运维管理。

这些就是美团云网络架构一路演进的过程，在这个过程中，美团云的团队成员始终秉承着“以最小代价解决最大问题”的思路，将软件和硬件相结合，通过开源与自研，高效地实现了网络架构的迭代，成为了行业标杆，并为千万用户提供更稳定、可靠的基础设施云服务。

青云SDN/NFV2.0架构剖析

作者 陈海泉

【编者按】对于青云来说，SDN/NFV2.0是一个新的突破。早在2013年，青云在第一代公有云产品中就上线了SDN技术。随着用户量越来越大，私有网络里面的 VM 数量超过一定的数量级的时候，性能问题逐渐成为瓶颈。在2015年下半年，青云经过重点研发，推出了 SDN/NFV 2.0。

SDN 就是软件定义网络。当然也不是所有网络定制一定要软件来实现，因为有很多硬件方案也可以做到 SDN 的效果。青云 QingCloud 用软件定义来实现虚拟网络。

VPC 是什么意思呢？VPC 网络是 QingCloud 环境内可以为用户预配置出的一个专属的大型网络。在 VPC 网络内，用户可以自定义 IP 地址范围、创建子网，并在子网内创建主机 / 数据库 / 大数据等各种云资源。

正是因为云计算需要虚拟网络，也需要 VPC 。所以我们还需要定义一个 SDN 方案解决这两个需求，现有的

图 1

SDN 方案主要分成两个方向：

一是用软件来定义，但是用硬件来实现。比如某些带 SDN 功能的交换机，把它采购进来，部署到产品里，用硬件厂商提供的 API ，在上面提供 SDN 功能。

二是 NFV ，就是网络功能虚拟化，用软件的方式来实现，用软件的交换机和路由器，把他们组织起来成为一个软件实现的 SDN 。其代表有 VMware NSX、Juniper OpenContrail 等等。

QingCloud 在 SDN 方案的选型上也做过讨论，用软件还是用硬件方案？其中考虑的问题主要是以下三个方面：

第一，成本。在公有云上面大家拼的是成本，谁的硬件成本低，谁就能把价格降到最低。如果我们采用现有的硬件方案，在网络设备上面增加了很多投资，并且我要采购的不是一个或者两个，而是一批。

第二，设备依赖。我们的私有云卖的是软件，客户可以按照偏好选择自己的硬件，假如 QingCloud 的 SDN 绑定了某款硬件产品，那我们在面对用户的时候，可能连招标的机会都没有，因为人家压根就没有办法用你的硬件。

第三，情怀。对于工程师来说，大家都想把产品做得更优秀。其实，软件跟传统快递行业非常的接近，为

图 2

什么这么说。因为网络中的交换机、路由器，其实跟快递行业里的快递员和包裹集散中心非常相似，一般包裹给快递员以后，快递员会发给一个快递集散中心，这里可以查询包裹应该被送到哪个地方，然后再将包裹交给快递员，送到用户那里。顺丰在中国应该是最好的快递公司之一，因为它把转运环节都做全了，只有方方面面都能够控制才能实现压倒性的优势。因此，我们如果把数据包转发的每个流程都控制到，就有可能在系统上面做到最优，采用硬件设备使用这些功能的话，最后带来的是同质化，跟竞争对手相比不会有任何的优势。

综合以上三方面的原因，我们决定开发一套新的 SDN/NFV 2.0 方案，取代 1.0 。

既然定了要自己做一套新的方案，怎么去实现？我们做了一些总结，新的产品需要满足传统 SDN 的需求。第一，数据封装。也就是实现一个基本的虚拟网络；第二，实现控制平面。二层、三层的网络数据进行路由规则的同步，然后下发到虚拟的交换机和路由器里面去，控制做到 ARP 泛洪抑制；第三，实现数据平面。除了 DVR 之外，还提供了虚拟边界路由器。

除此之外，还需要增加我们需要的 2.0 方面的功能。

第一，VPC 主机直接绑定公网 IP 。私有云用户大量依赖基础网络，要

图 3

求 VM 直接绑定公网 IP ；

第二，负载均衡器。可对进入流量进行分流，出流量经由多台 Virtual Gateway （虚拟网关）分担负载，单 IP 可承载 1 TB 出流量。同时，4—7 层完全透明；

第三，IP 不变。支持可以无限水平扩展的基础网络，并保持高可用及高性能，VM 任意迁移，IP 地址保持不变；

第四， VPC 和物理网络连接。

下面分别解释刚才说的那几个基础实现。

首先解释虚拟网络。在一些大公司里会提供一种叫内部邮递的服务。公司员工之间可以发送一种快递，比如要给财务部门某同事发一个报价单，会查他的工位，知道他坐在哪，比如 184-323-534 。然后准备一个大信封，把要填的单子放在里面。我不需要知道这个人是在北京，还是在上海，我把这个信封交给公司的收发室，这个收发室会对这个信封进行重新封装，因为他有此员工的具体地址，然后把具有新地址的信封交给外包快递公司。放到云计算里，这就是一个虚拟网络。可以允许用户自己定义一个地址，然后进行传输，为了让它在三层网络里传输，可以再进行封装，再套一个包，写上新的地址。根据外包的内容和里层包的内容把这个数据包发送到对应的信息那里。

图 4

虚拟网络依赖于拆包、分包。现在采用的方案主要是比较流行的 VXLAN，因为 VXLAN 有一系列的优势。第一，隧道连接一组物理机，由于 VXLAN 的数据包在整个转发过程中保持了内部数据的完整，因此 VXLAN 的数据平面是一个基于隧道的数据平面；第二，使用 UPD 协议，当数据包交给网卡的时候，网卡根据这个数据的包头，用不同的网卡队列。这样把包交给不同的 CPU 处理，提升性能；第三，是比较有争议的 Flood & Learn 自动管理虚拟网络，VXLAN 可以进行泛洪学习，当 VTEP 收到一个 UDP 数据报后，会检查自己是否收到过这个虚拟机的数据，避免组播学习。

通过以上几点，我们觉得 VXLAN 不错，但是仔细的去想，就发现它有两个非常大的不足。一个是组播协议，大规模部署会受硬件设备组播路由限制；第二，泛洪学习的机制，会把原来在二层广播的 ARP 包扩大到三层网络，这样随着规模扩大，广播越来越多，会严重的浪费带宽资源。所以，我们既要使用 VXLAN ，又要消灭它的不足，所以我们设置了 SDN 控制器，通过我们自己设置的规则，取代它自有的泛洪学习规则。

那么这个控制器需要多少个呢？我之前曾经了解过一些用户生产环境里用到的控制器，通常只有一个。它负责整个集群中所有节点的规则，这

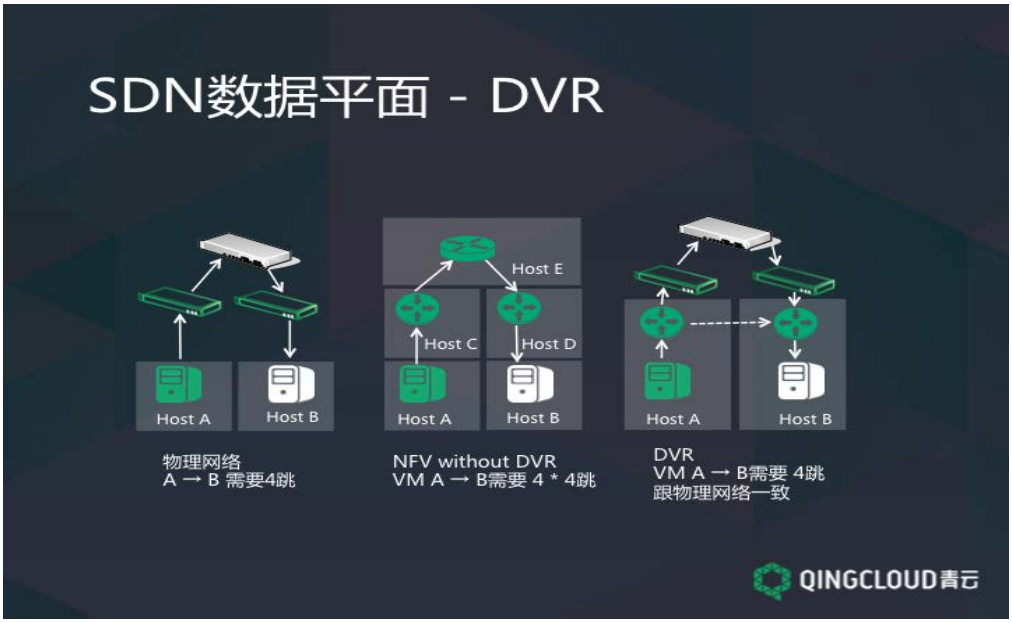


图 5

么做造成一个问题，当集群创建、销毁、迁移的时候，需要把规则同步到整个集群所有的节点中，这样随着用户秒级创建资源的需求，同步规则的流量就会相当相当大。所以我们做了一个分布式控制器，不仅把控制器分布到 VPC ，还分布到虚拟网络里头。每个 VPC 有 253 个虚拟网络。

刚才说了虚拟网络和控制器，第三点 SDN 需要做的就是控制数据平面，其作用就是把数据包真的从一个地方拷贝到另外一个地方。传统的数据平面，比如 OpenStack 通常会用 OVS ，OVS 会有一个问题，它会把数据交给另外一个程序，这样会带来一个性能的下降，而我们的方案完全改变了这个问题，自己做了一些 Kernel 的改动，将转发放到 Linux Kernel 上，而且我们是用 NFV 来实现的。

同时我们引入了一个新的功能叫做 DVR （分布式虚拟路由器）。通过上面的图解释一下我们为什么需要 DVR。左边是这张是物理拓扑图，物理世界中 A 和 B 通信，需要把信息发送到 A 的交换机，然后到路由器，然后路由器转给 B 的交换机，B 的交换机再发送给 B ，A 和 B 通常需要 4 跳才能发一个数据包。

我们 1.0 的时候，也是用 NFV 实现的 SDN ，我们会模仿物理世界，虚拟出虚拟的路由器和交换机提供给用户。如果 A、B、C、D、E 这五个设

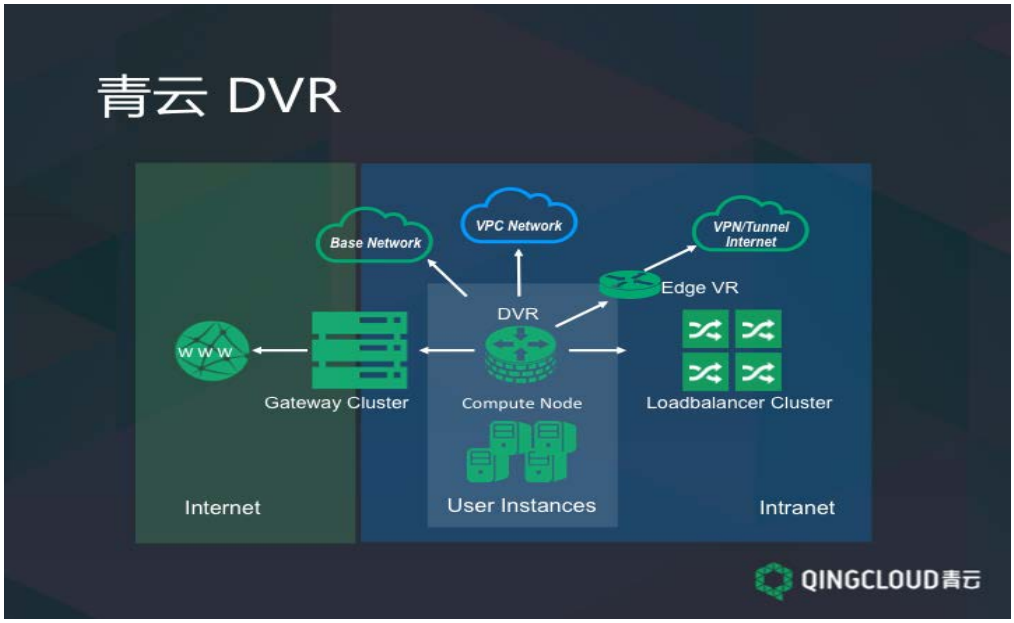


图 6

备分别位于五个不同的虚拟机上，在逻辑上 A 的包经过C、E、D、B 才能到，逻辑上是四跳。但是虚拟设备每一跳都要通过物理设备去交换，而物理设备每一条四跳，这样总得转发量实际上需要 16 跳。这也就是为什么 1.0 的性能总是上不去。为了解决这个问题，我们引入了 DVR ，从 A 到 B 还是这样，两个 DVR 之间直接交换一下数据就可以了，因为在逻辑上有一跳，所以总数跟原来物理设备一样，四跳完成一个数据包的转换，这样性能就可以非常接近物理机的性能，从而可以组成一个大的虚拟网络。

QingCloud 的 DVR 除了实现 VPC 简单的功能之外，它实际上是一个复杂的东西。因为除了保持自己跟自己的虚拟网络，还需要有其他四个方向：第一个就是网关需求，我们需要提高公网 IP 存储量，希望 DVR 把包发到公网网关；第二是 VPC 的虚拟机要能跟硬件设备进行高度的互访。因为我们私有云用户的机房里，不止有 QingCloud 的东西，还有 Oracle 的数据库、F5 的路由器等等，假如我们让用户把这些业务放到虚拟网络里，虚拟网络就要跟硬件网络进行高速的互访。第三是 VPC，可以让用户定义 255 个 C 段，加起来可以有 60000 多个虚拟机。第四是，我们还提供了一个边界路由器，可以让用户虚拟资源跟远程的 IDC 之间做一个互通。最后一个就是一个负载均衡器集群，我

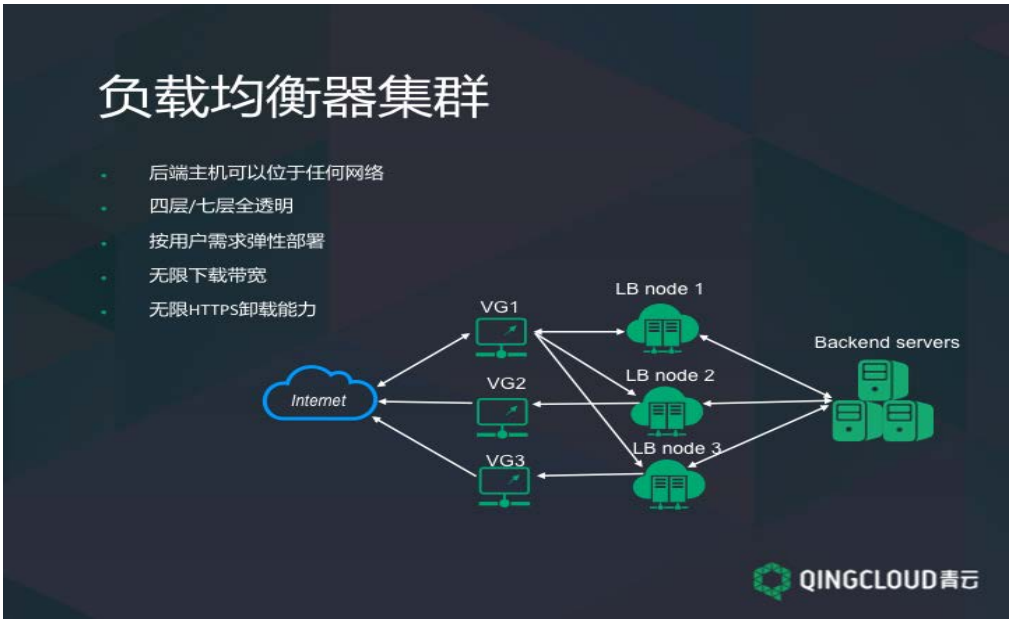
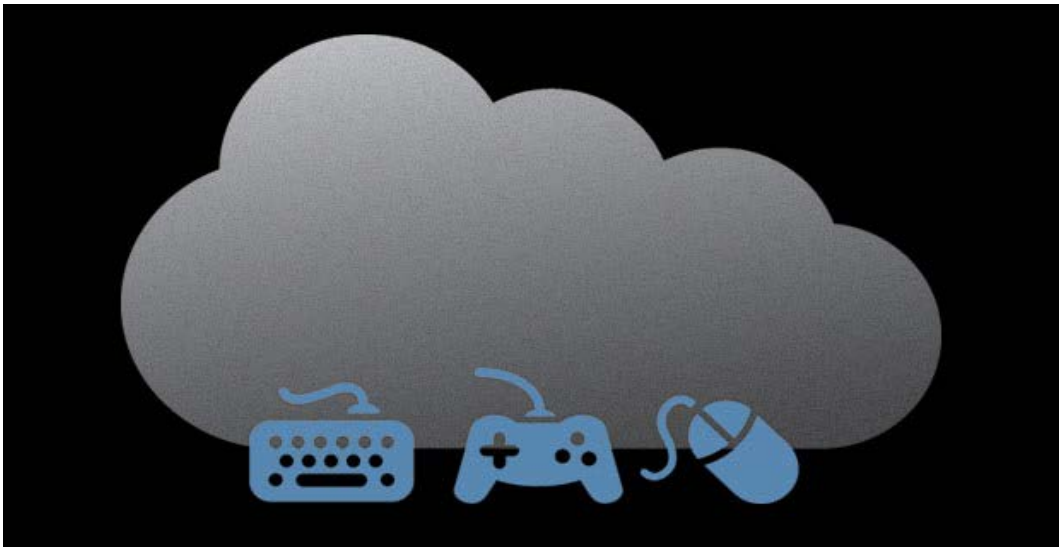


图 7

们的 DVR 就是一个平台，让用户的主机和负载均衡器直接相连。

我们做的最后一个功能就是负载均衡器集群。设计是这样，我们有一个网关集群连着因特网。比如我有一个 IP 1.1 ，实际上是绑定在 VG 1 这里，VG 1 会做第一次的流量转发，会把流量按照用户定义的负载均衡器节点数量转发到自己私有的负载均衡器节点里（1、2、3），它的特点就是，返回流量不需要经过进来的这个网关，而是经过自己对应的不同物理网关发送到因特网。因为当 VG 1 能力受到限制的时候，假如我们所有流量都从它回去的时候，它自己的网络带宽实际上就是整个集群的能力。而我们把它分散之后，就可以做到，出去的流量几乎是没有限制。只要我们的 VG 设备有多少，它的带宽就会有多少，因为流量不需要从默认的线路回去。同时随着用户拓展负载均衡器节点的数量，也扩展了 HTTPS 的卸载能力，并且我们做到了 4 层 / 7 层的完全透明，也就是说用户通过因特网访问到他们业务的时候，我们在所有转发过程中，都会保留其原地址，用户这边得到的包是直接来自因特网用户的 IP 地址。

YY游戏云平台Cloud 2.0网络设计分析



作者 风河

前言

YY游戏Cloud 2.0的开发背景详见《YY游戏私有云平台实践》。在Cloud 2.0里，虚拟网络的架构和实现是重中之重，本文主要谈及网络设计部分。网络设计一个核心功能是实现租户网络（VPC），我们采用VxLAN技术来构建VPC。考虑到性能和稳定性，使用带SDN功能的硬件交换机来完成VXLAN的offload和routing。同时，游戏运营有特殊的业务需求，例如云网关功能。处于同样的考虑，采用硬件防火墙来实现云网关，包括南北向的NAT和Floating IP。在这个方案里，还支持不同VPC对共享数据区域的访问，也就是东西向的访问，同样使用硬件防火墙的NAT来实现。

在由客户端vSwitch、接入交换机、核心交换机、防火墙组成的虚拟网络里，数据流程跟传统网络没有太大的差别，但是数据结构非常不同。比如为了支持VPC，接入交换机要设

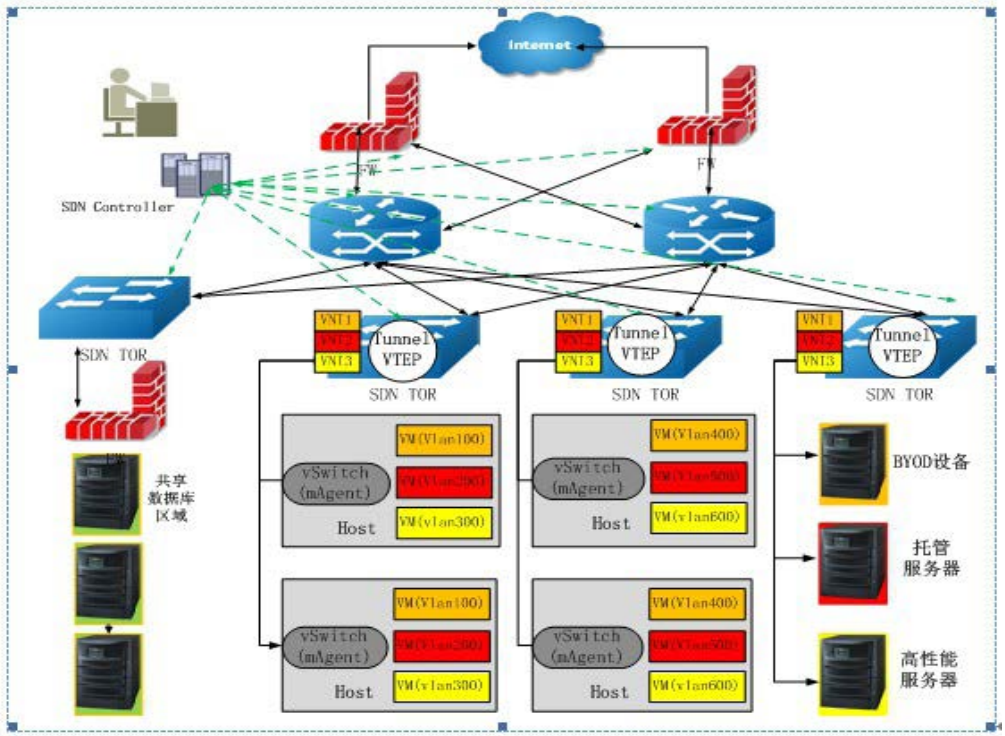


图 1

置 VNI（VXLAN 网络标识），核心交换机上要配置 VRF（虚拟路由器），防火墙上要开启 vSys（虚拟子系统），它们处理的数据包不同于传统网络。而它们处理包的方式、转发路径，则取决于控制器的实现。在本文里，将对虚拟网络架构、数据转发流程、以及 SDN 控制器进行一个较为详细的描述。

虚拟网络架构

这个架构的主要组成部分如下：

SDN TOR：虚拟网接入交换机，负责 VXLAN 的封装和转发；

SDN Core：虚拟网核心交换机，负责 VXLAN 的三层网关和路由；

Firewall：防火墙，负责南北向的 NAT 网关，包括 Floating IP；

SDN Controller：SDN 控制器，负责整个虚拟网络的配置管理；

vSwitch：运行在计算节点上的虚拟交换机组件。

核心技术指标

在这个架构中需要实现的核心技术指标有：

- VXLAN功能offload到硬件交换机中实现；

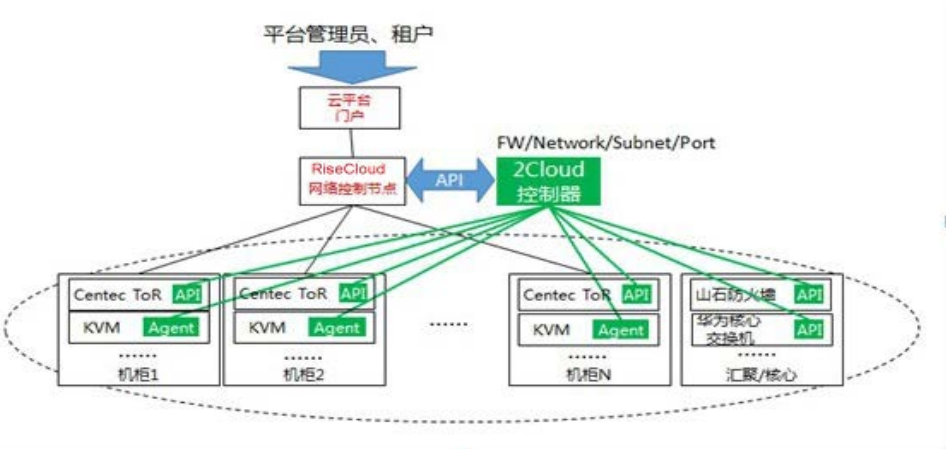


图 2：控制器架构

- VPC内L2 VTEP，offload到SDN TOR；
- VPC内L3 VTEP，在接入侧offload到SDN TOR，在网关侧offload到Core Switch；
- VXLAN VNI和 VLAN 实现解耦，VLAN根据物理位置和VNI进行动态映射；
- NAT和VPN做在硬件防火墙；
- 每个Core Switch支持至少4K租户（取决于核心交换机能力），可以通过横向扩展，支持更多租户；
- 通过在vSwitch或者TOR上启用ARP代理，避免ARP广播到物理网络；
- 流表通过控制器预先下发，而不是通过动态学习；
- Floating IP使用硬件防火墙实现，提高转发性能；
- 除了虚拟机外，还支持物理机接入；
- 控制器支持运行时无缝动态增加、删除网元设备；
- 控制器和Agent不依赖OpenStack组件，可独立部署和实现；
- OVS和Agent运行的宿主机环境，稳定支持Ubuntu 14.04 LTS操作系统和对应的最新内核(linux-image-generic)。

SDN 控制器

云平台门户调用RiseCloud API，完成跟虚拟网络的对接。RiseCloud

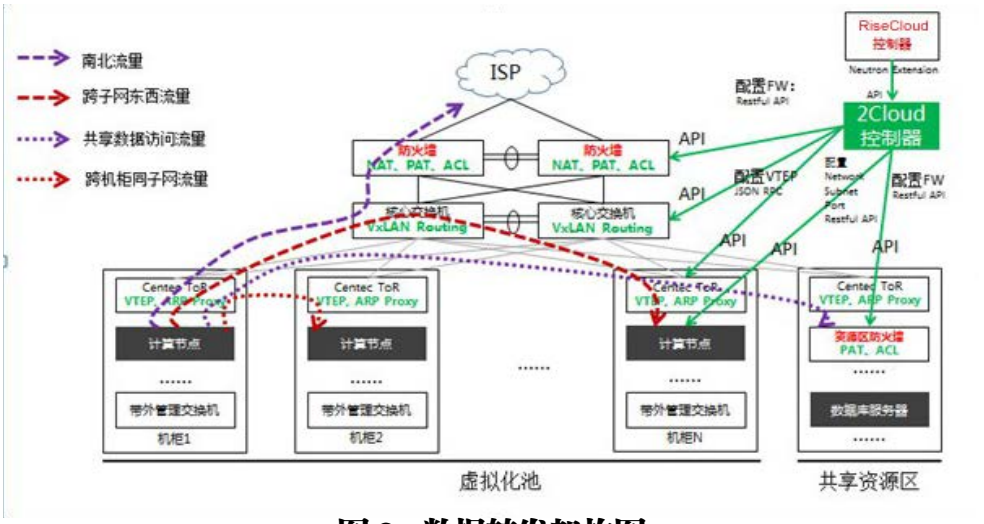


图 3：数据转发架构图

API 是 YY 自己实现的、高度抽象的网络控制器接口。它北向是一套简单的 API，类似于 OpenStack 的 Neutron API；南向跟厂家设备、厂家控制器以及跟第三方控制器集成，共同完成对虚拟网络的管理。

控制器功能

- 交换机、物理服务器、VM、租户网络等信息映射关系管理。
- VM端口配置、ACL、QOS等。
- 租户网络配置管理，租户二层网络创建、删除、变更等。将VXLAN功能实现在TOR交换机上，并随资源变化能够动态进行更新。
- 三层路由转发功能，为每个租户创建三层路由vGW，可实现ACL、Routing、QOS等功能。在核心交换机上为每个租户创建一个VRF，通过VXLAN Routing实现租户路由转发。
- 南北网关功能，为每个租户实现一个南北向网关，可实现防火墙规则、NAT、PAT功能。通过专业硬件防火墙，为每个租户创建一个VR，实现地址管理、NAT规则等独立操作。
- 共享数据区域访问，为租户提供对共享数据区域访问（IP地址、端口访问）方式，通过防火墙PAT功能，按需进行动态地址和端口映射。
- 网络监控，监控交换机、物理服务器、虚拟机、虚拟端口、流量等信息，汇总至Controller进行统一呈现。

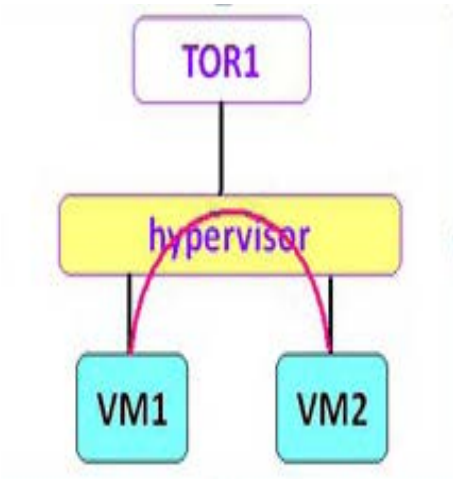


图 4：同 TOR 同 hypervisor 内同网段 VM 转发

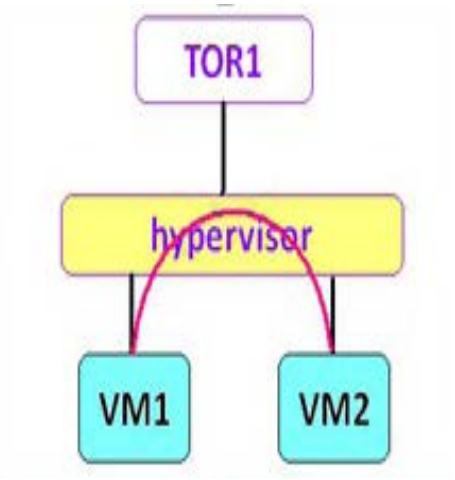


图 5：同 TOR 不同 hypervisor 内同网段 VM 转发

- 诊断分析，提供配置检查、内网发包诊断、租户内网数据抓包分析、按协议/应用/地址等进行流量分析功能。

数据转发面架构

- 南北流量

如图 2 所示，租户子网南北流量路径为：

ToR <-> VXLAN <-> 核心 <-> (GW1) 防火墙 <-> NAT/PAT

- 跨子网东西流量

如图所示，租户跨子网东西流量路径为：

ToR <-> VXLAN <-> (GW3) 核心 (GW3) <-> VXLAN <-> ToR

- 跨机柜同子网流量

如图 2 所示，租户跨机柜通子网流量路径为：

ToR <-> VXLAN <-> ToR

- 共享数据访问流量

如图所示，租户共享数据访问流量路径为：

ToR <-> VXLAN <-> ToR <-> (GW2) 防火墙 <-> PAT

注：以下网关类型分别为：

1. GW1：南北网关
2. GW2：资源区网关
3. GW3：东西网关

L2 流量转发过程

同 TOR 同 hypervisor 内同网段

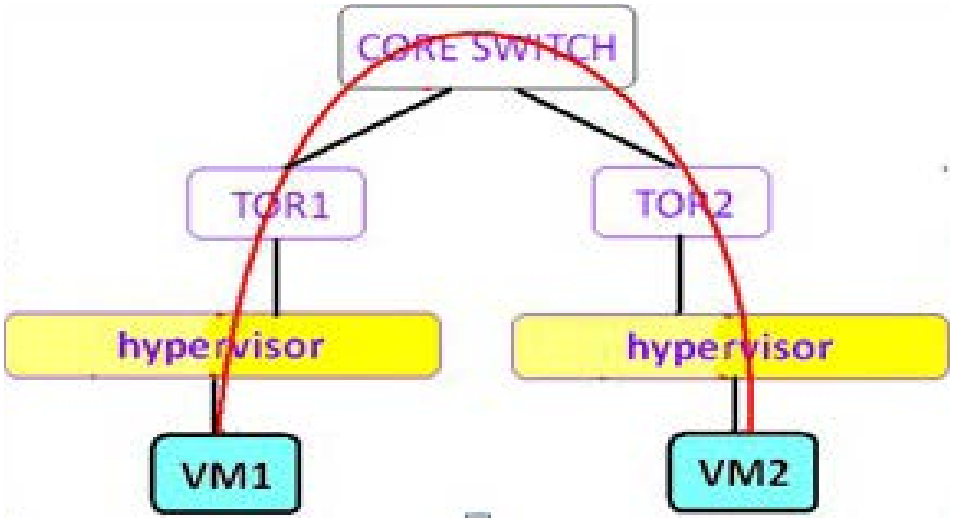


图 6：不同 TOR 不同 hypervisor 内同网段 VM 转发

VM 转发。

主要利用 Open vSwitch 的流表进行转发，这时需要 SDN 控制器向 Open vSwitch 下发对应的流表。对于 arp 报文，需要匹配 arp request 从 hypervisor 和 TOR 连接的端口送出，将报文送到 TOR 上，由 TOR 的 arp proxy 代理完后发回来。arp reply 匹配 vlan + macda 后，送到正确的 VM 上完成 arp 通信。数据报文匹配 in_port + macsa 完成 tenant VM 的识别后，送到另外一张 L2 转发流表，该流表匹配报文的 macda 后送到指定的 VM 完成数据报文通信。

同 TOR 不同 hypervisor 内同网段 VM 转发

主要利用 Open vSwitch 的流表和 TOR fdb 进行转发，这时需要 SDN 控制器向 Open vSwitch 下发对应的流表。对于 arp 报文，需要匹配 arp request 从 hypervisor 和 TOR 连接的端口送出，将报文送到 TOR 上，由 TOR 的 arp proxy 代理完后发回来。arp reply 匹配 vlan + macda 后，送到正确的 VM 上完成 arp 通信。数据报文匹配 in_port + macsa 完成 tenant VM 的识别后，送到另外一张 L2 转发流表，该流表匹配报文的 macda 后送到 hypervisor 和 TOR 连接的端口。报文送到 TOR 后，根据 fdb 表完成二层数据转发，完成数据报文通信。

不同 TOR 不同 hypervisor 内同网段 VM 转发

主要利用 Open vSwitch 的流表和 TOR tunnel offload、core switch fdb 进行转发，这时需要 SDN 控制器

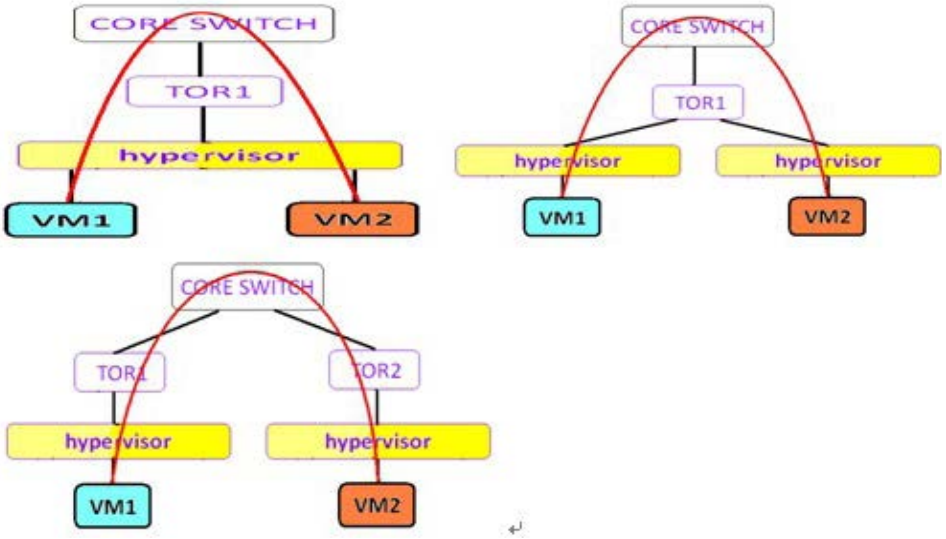


图 7：不同 TOR 不同 hypervisor 内同网段 VM 转发

向 Open vSwitch 下发对应的流表。对于 arp 报文，需要匹配 arp request 从 hypervisor 和 TOR 连接的端口送出，将报文送到 TOR 上，由 TOR 的 arp proxy 代理完后发回来。arp reply 匹配 vlan + macda 后，送到正确的 VM 上完成 arp 通信。数据报文匹配 in_port + macsa 完成 tenant VM 的识别后，送到另外一张 L2 转发流表，该流表匹配报文的 macda 后送到 hypervisor 和 TOR 连接的端口。报文送到 TOR 后，根据下发 vlan 和 vni 的对应关系，封 VXLAN 报文送到 core switch，在 core switch 上根据 fdb 表转发 VXLAN 报文，送到对端 TOR 上完成数据通信。

L3 流量转发过程

主要利用 Open vSwitch 的流表和 TOR tunnel offload 进行转发，这时需要 SDN 控制器向 Open vSwitch 下发对应的流表。对于 ARP 报文，需要匹配 ARP request 从 hypervisor 和 TOR 连接的端口送出，将报文送到 TOR 上，由 TOR 的 ARP proxy 代理完后发回来。ARP reply 匹配 VLAN + macda 后，送到正确的 VM 上完成 ARP 通信。数据报文匹配 in_port + macsa 完成 tenant VM 的识别后，送到另外一张 L2 转发流表，该流表匹配报文的 macda=core switch mac 后，送到 hypervisor 和 TOR 连接的端口。报文送到 TOR 后，根据下发 VLAN 和 VNI 的对应关系，封

VXLAN 报文送到 core switch。core switch 收到 VXLAN 报文后，根据报文里的 VNI 信息，找到对应的 VRF 信息（一个 VRF 对应于一个虚拟路由器），查找相应的路由（看 core switch 支持情况，32 位主机路由还是网段路由），将对端网段 VNI 信息再封成 VXLAN 报文送出。TOR 上解封装后，翻译 VNI 和 VLAN 的映射关系，送到 hypervisor 上查流表进行转发。

总结

YY 游戏 Cloud 2.0 建设是一项全新的尝试，我们让不同厂家的硬件设备和驱动、第三方控制器、YY 自己的 RiseCloud 控制器、YY 云平台业务系统有机的整合起来，组成一个高性能、高可靠的虚拟网络系统。在这个过程中所取得的成功经验和失败的教训，我们也乐于分享，期望对国内企业的私有云建设有所帮助。同时感谢合作厂家包括华为、H3C、云杉在技术方案、测试设备等方面对我们的大力支持。

如果您在 SDN、QEMU/KVM、Ceph、Libvirt 任何一项有长足的经验，都欢迎与我们联系：me@fenghe.org，YY 游戏云平台欢迎各路有志之士加盟。