From IaaS to PaaS to Docker Networking to … Cloud Networking Scalability

Wenbo Mao

DaoliCloud Company

www.daolicloud.com

wenbo dot mao at gmail dot com

November 22, 2014

Abstract

This whitepaper presents DaoliCloud's Network Virtualization Infrastructure (NVI) technology as a scalable cloud networking solution. It reasons about why and how the NVI technology, by means of network patching connecting **independently orchestrated** clouds, each having a **modest scale of physical realization**, can provide a useful technical enabler to truly scalability for cloud services with an affordable and sustainable service offering cost. In order to reveal cloud networking scalability problem in today's cloud networking practices, the whitepaper also provides a technology background survey on existing IaaS and PaaS networking offerings.

Keywords

Cloud Networking Scalability, Docker Networking, IaaS, PaaS, Network Virtualization, SDN, Openflow, Non-Encapsulation Overlay Network.

1. An IaaS Story

In 2006 Amazon launched Amazon Web Services (AWS) to online offer IT infrastructure as a service (IaaS). The most innovative element in AWS is selling CPU cycles in virtual machines (VMs) named Elastic Cloud Compute (EC2). That awesome business initiative inspired many followers entering into the IT resource servicing market. Eight years on, AWS remains the largest cloud service provider; revenue in 2013 exceeds the sum total that of the next five followers. However Amazon may not feel a great business achievement if contrasting its cloud revenue to its http based online business income. After 7, 8 years of EC2 in operation, AWS' revenue in 2013 only amounts to just below a negligible 3% of what Amazon harvested from http. We do not know all reasons for the slow growth of cloud business while can't help puzzling how http based business could succeed growing the firm much faster a decade back the inception of EC2. With DaoliCloud working in the technology side, let us look into and discuss some technology aspects trying to find some interpretation.

Our investigation reveals to us that EC2 VMs are tenant network isolated by labelling network packets in the underlay network data path or forward path in networking terminology. In fact, online CPU cycles sold by all other EC2 followers, and cloud IaaS orchestration software from proprietary vendors or from open source projects such as Openstack and Contrail, all use the same technology for tenant/project network isolation. With cloud computing being for sure to stay with us just like electricity utility, and networking being in the important position to connect clouds just like the power grid connecting generators of all power plants, we enter into a questioning mood: Whether or not can the seemingly uniform underlay labelling approach be hopeful to become a standard to connect clouds?

We observe that each overlay network application has actually already looked after itself for communication context isolation, also by exploiting packet labelling trick albeit in a much easier programmable way. The overlay application context isolation labelling is done at OSI L4 in a soft manner—TCP/UDP-like protocols port number or ICMP-like protocols identifier of a communication initiator, which are random numbers tagged by the overlay OS hosting the network application. It is obvious that communication context isolation implies tenant/project isolation, and hence the soft labelling effort of the overlay OS can certainly be made use of for tenant/project isolation. In Section 4 we will provide detailed technical descriptions for how to make use of context isolation for tenant/project isolation. By ignoring the overlay OS' context isolation labelling effort, networking for EC2 and for the like IaaS technologies add another underlay hard label outside the overlay packet. The use of the word "hard" means not only the fact that underlay labelling ignores the soft context isolation labelling, but also that the labelling idea is copied from networking hardware boxes vendors. Formulations for underlay hard labelling vary, but all take the same ignoring-soft-labelling approach; examples range from VLAN, MPLS, VXLAN, GRE, NVGRE, LISP, STT, to a latest standard attempt of Geneve, to name a few. Below let us comment and discuss the underlay hard labelling technology from a number of aspects.

First, underlay hard labelling may be viewed a bit of brutality toward the overlay CPU effort already spent by the overlay OS on soft labelling. This is because the ignored overlay CPU effort is actually rather smart spent. Overlay label provisioning and checking are flow based which are equivalent to the least requirement of routing forward service. The flow based routing service provisions and checks context isolation in a state machine fashion, once only per flow even though a flow may include a large number of packet frames. In contrast, an awkward consequence of ignoring this smart flow state machine based context isolation is that any underlay hard labelling outside tenant packets have to be provisioned and checked in stateless manner on per-frame basis. Such unnecessarily labor-some underlay CPU effort is obviously less brainily spent than the being brutally ignored smart overlay counterpart. The inefficiency of the underlay hard work has another vector: underlay labelling needs room in packet headers which can only be sought from resorting to packet encapsulation. Encapsulation causes packet enlargement and consequent painful packet fragmentation and reassembly follow, not only consuming additional CPU cycles and bandwidth resources, but also complicating firewall processing which should be distributed for cloud. Some networking vendor solutions, e.g., Cisco nexus 7000 and VMware's NSX, resort to ad-hoc repair for the packet fragmentation pain as a result of network non-virtualization: Use 1,600-byte MTU (maximum transmission unit) instead of the 1,500-byte established standard! In so doing, overlay OS can still keep complying the established standard of sending/receiving 1,500-byte MTU packets, in hope that underlay encapsulations shall add less than 100 bytes to the overlay packet not to cause nascent fragmentation. Unfortunately, changing networking standards have always been notoriously slow and costly due to its forklift upgrading nature, not very suitable for to be applied to trivialities such as MTU modification.

One might be dismissive on resource wastage not being a big deal in today's IT using affluent economies. Let us turn to discussing a second and worse problem. Underlay hard labelling **ruins**

network virtualization. In hard labelling, trans-cloud overlay network packets are directed to some fixed holes, e.g., "Neutron nodes" in Openstack, "Special vRouters" in Contrail, and "Nicira NVP Gateways" in NSX, to go-out and come-into clouds where underlay hard labelling, encapsulation, fragmentation, reassembly are frame provisioned, frame rechecked and frame routed. We know that underlay network is physical having hard boundaries being made of box steel casings and building concrete constructions. Such physical attributes are very hard to program for route optimization, traffic engineering, multipath distribution, or motion management of overlay entities for better resource utilization or high availability QoS, etc.

The problem of network non-virtualization in today's cloud networking practices reveals its ugliest part of the face in **poor networking scalability**. ALL IaaS orchestration platforms to date, AWS, Openstack, Azure, vRealize, Contrail, …, unfortunately take unsuccessful scale-UP approaches to orchestrating networking, unlike their very successful scale-OUT counterparts to orchestrating CPUs, memories and storage spaces. Known technologies for IaaS orchestration inevitably involve time-out-able event queuing, copy-on-write database state synchronization, write-lock file systems, cluster leader election, etc. The scale of cloud IaaS resource orchestration must be limited to a rather humble size or else large number of queuing events, frequent locking files for writing, and/or database state synchronization will be slowed down by size proportional momentum leading to orchestration chaos. Evidence of networking scale-UP and related size limits are eminent, e.g., VMware NSX for Multi-Hypervisor (NSX-MH) Network Virtualization design guide stipulates very wisely a rule-or-thumb networking scale-UP bound: one NSX network controller to orchestrate no more than 1,000 hypervisors (or a cluster of five NSX controllers to orchestrate no more than 5,000 hypervisors, see e.g., http://demo.ipspace.net/get/3.2%20-%20NSX%20Control%20Plane.mp4); this level of networking scale-UP limit seems sufficiently large for NSX's targeting market of IT as an asset ownership. The evidence is also detectable from the non-open-technology platform of AWS: (1) none of AWS's worldwide located datacenters permit trans-datacenter deployment of a VPC; (2) even within one datacenter or "AWS Region", VPC construction has quite a number of size bars for tenant self-service provisioning, beyond which the tenant has to submit offline applications for special arrangements. To the date of writing this whitepaper we have not seen any commercially successful cloud IaaS or PaaS service deployment or software offer, proprietary or open source, of which networking orchestration takes a scale-OUT approach. The scale of cloud networking has been showing a prominent difference from that of http over CDN: the former is in disconnected small islands and the latter is in globally connected World Wide Web. Probably as a consequence, cloud IaaS and PaaS businesses have long been remaining slow growing while ecommerce, social-networking, and advertisement businesses, etc., on http have long been growing in explosive speeds and still do so today. The sharp contrast has also been plainly stated in revenue reports of providers who offer both online services. If cloud networking remains in today's disconnected small islands status quo, inter-cloud standard would seemingly be in dream for long, as negotiation of underlay hard labelling would add chaos to such …

Summary of the story: Since it is a common quote using power grid utility to analog IT provisioning, let us summarize this section by asking a question: AWS EC2 does seemingly set forth the de-facto standard for power generator. Is that a standard candidate for the grid to connect power generators?

There exists an attempt for an answer: A demonstration of IT "Power grid connecting generators" has been open for beta trial run at www.daolicloud.com Virtualized, distributed and scale-out connection of independently orchestrated clouds is achieved by respectfully and honorably using overlay CPU's effort on context isolation. Detailed technical descriptions will be provided in Section 4.

2. **A PaaS Story: TuHao Version** (Tuhao: noun; origin: modern Chinese; referring to an explosive rich who spends wastefully with unrefined taste)

Not long after EC2, Google launched its value-added-on cloud service: Google Application Engine (GAE) in the form of Platform as a Service (PaaS). In a PaaS offering, IT services are provisioned by pre-installing software tools and middleware packages over CPUs and other IaaS resources to make cloud an ease of programming environment for applications programmers, so that the applications programmers can focus more on business requirements and less on IT technical requirements. Since the inception of GAE we have seen a number of PaaS offerings and projects: Force.com, Heroku, Cloud Foundry, BlueMix, …, not for an exhaustive list. But there has been a problem until a quite recent solution which we will be discussing in Section 3.

Let us consider that a PaaS offering focus on a not-too-big **set** of commonly used software tools and middleware packages. PaaS would be valuable only if it can provide an almost arbitrary **subset** of software tools from the offering **set** to satisfy the user chosen programming environment. Also, since PaaS should hide IT technical details, such as CPU capacities, RAM sizes, operating systems, storage spaces and networking QoS, etc., from being dealt with by applications programmers. Thus, a PaaS offering should combine various IaaS requirements with the software tool offering set. Then, how many different programming environments there would be out of selecting "almost arbitrary subsets of not-too-big set", and further considering the variety to be combined with the number of IaaS technical parameters to choose? Such a PaaS provider is clearly facing a combination explosion problem! (Our natural universe is combinational exploded from only 92 elements.) There are probably only two practical ways to avoid offering a vastly large combinations of programming environments. One is to require the user itself to install its chosen software packages into a small set of CPUs. That solution is in essence let the application programmer directly use IaaS. It is because installation and configuration of software on IaaS is difficult, mundane and hence unwanted jobs that exactly render PaaS the very service value over IaaS. The other solution is for the PaaS platform to pre-install a software tool in various assorted parameters of IaaS resources to let the application programmer on-demand choose them and "knit" connect its choices into the needed programming environment. With CPUs being provisioned in VMs and even organized in VPC, the so-called "knit" connection of software tool pre-installed IaaS resources is rather easy, and can be self-serviced in web service manner.

Many programmers or PaaS targeting customers, in particular a purported vast number of early cloud adopters, are so-called "long-tail" customers who are very important to commercial viability for cloud computing business. It is also believed that booming of mobile networking use cases and big data applications will create a large long-tail market of startup business customers for cloud computing. A typical long-tail customer only humbly spends a little on IT resource renting, and a vast number of such customers can aggregate a sizable business income. Another eminent attribute for long-tail is that the tastes of long-tail customers are eccentrically special and individual.

Now comes the problem. Asking a startup or long-tail programmer to rent a number of VMs each with a pre-installed software and link them to an eccentrically special programming environment, so doing would be way too wasteful. A VM is quite heavyweight, having a dedicated guest OS running on top of a host hypervisor. Whether or not a humble software in such a VM is in use of not—most likely inactive or dormant is the case due to the well-known 80:20 rule—the hypervisor must keep on job schedule provisioning CPU cycles to the VM. To sell multiple VMs linked PaaS to a long-tail customer is somewhat alike to sell a number of villas to a DiaoSi. (DiaoSi: noun; origin: modern Chinese; referring to a poor and humble youth of mediocre appearance, having no car, no house, and poor social skills and connections, yet being proud of having fighting aspiration to change current lack of fortune position.) In recent history in China, most villas have been tailor built for Tuhaos who have indeed shown a trend of buying a plural number of in one shopping spree. It would however be absolutely impossible to sell even one single villa to a DiaoSi in spite of the fact that DiaoSi youths do aggregate a vast group in the population in desperate need of housing. This ironic situation in China vividly mimics "linking a plural number of software pre-installed VMs" based PaaS offerings targeting long-tail customers while such PaaS offerings are in fact in TuHao version. Long-tail programmers do not aggregate a large size market for TuHao version of PaaS. VM based PaaS market has been in a quite dormant state until …

3. A New PaaS Story: DiaoSi Version

Circa 2013-14, Docker (www.docker.com) publicized a great observation: To partition a Linux host OS into many containers with a fine granularity to exactly fit for refined CPU need of PaaS customers. There has long been such partition technologies, e.g., Linux Container (LXC), of which Docker made a unique exploitation. The partition of the host Linux OS by Docker is done to a very fine granularity. Analogous to differentiation and integration in mathematics, various shaped needs of programming environment can be composed of by link composing a plural number of software pre-installed and fine granular CPUs. Let us try to explain why the Docker observed composition can be viewed as differentiation and integration to any shape of programming environment without causing wastage on CPU cycles.

The LXC technology virtualizes CPU in a direct-IO and host-OS application manners. (In contrast, hypervisor-based virtualization is in indirect-IO and guest-OS simulation manners.) It is easily observable on a Docker host that a container if hosting a dormant software only uses a negligible (almost zero) fraction of the host CPU cycles however the host-OS application based container can be woke up instantly into full action of service to guarantee a good user experience. It is because of this instant wakeup property that over provisioning of containers will not cause waste of CPU cycles for a service provider. This instant wakeup property exactly permits a PaaS service provider to offer PaaS services in the fashion of allowing a long-tail programmer to link a plural number of container CPUs into any shaped programming environment, each CPU is pre-installed one software package only. Such a way of offering is not economically possible for previous VM based PaaS offerings, as we have revealed in the preceding section. Thanks to the universal validity of 80:20 rule, **over provision** of containers can indeed make an economically sound business for a PaaS provider, as well as an affordable purchase for a DiaoSi customer. Likewise, the well-known high contention ratio trick permits over sale of network bandwidth, and the so-called "thin provisioning" disguised over sale of storage spaces exploited the same principle.

Docker has successfully developed an open source project to promote quick commercial deployment and adoption. Docker's container technology has succeeded the first commercial deployment of the LXC technology. In October 2014, Microsoft is also in: it announced its DrawBridge Project based container technology to provide containers for Windows servers.

However unfortunately, Docker PaaS platform still has a technical problem: Its networking lacks scalability. Let us provide an analysis on Docker's networking as follows:

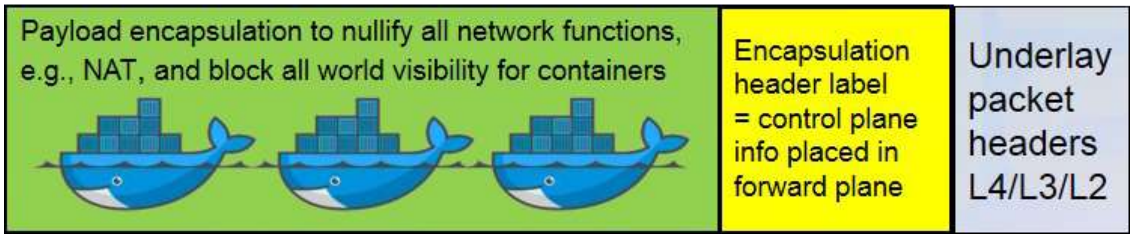
Payload encapsulation to nullify all network functions, e.g., NAT, and block all world visibility for containers
Encapsulation header label = control plane info placed in forward plane
Underlay packet headers L4/L3/L2

Figure 1: Known Technology for Trans-Docker Host Connecting East-West Networking for Docker PaaS

- Every Docker host provides L3 services for hosted containers, NAT (Network Address Translation) is one such.
- NAT has a "diode firewall" like behavior, default blocking ingress traffics; therefore, containers in different Docker hosts have no default IP connectivity.
- NAT can be bypassed by L3 encapsulation protocols: e.g., STT, VXLAN, GRE, IPsec, LISP, VPN, …, as shown in Figure 1. Google's Kubernetes for Docker orchestration with networking option of CoreOS' Rudder/Flannel, and Weave Project, are very recent attempts to trans-Docker-host connecting containers. We notice that these technologies take underlay hard labelling approach to tenant isolation, and the needed trans-host connection is via packet encapsulation technologies.
- Unfortunately, as we have discussed and explained in Section 1, these underlay hard labelling technologies and consequent encapsulation protocols **do not scale** and are **not a network virtualization technology**. They are scale-UP networking orchestration approach, and cannot provide truly scalable cloud service.
- A little unlike PaaS for DiaoSi use cases, the networking non-scalability limitation of known publicized Docker networking offerings do not seem to feature an online IaaS business.

Lack of networking scalability, known publicized Docker networking based on underlay hard labelling technologies, although can offer PaaS service at a DiaoSi scale, would likely to render the service provider very busy at and difficult in managing customers aggregation over fragmented small Docker platforms. It certainly is unsuitable for offering a container based IaaS service business for which economy of scale is absolutely the key for resource consolidation. Let us use the case of Docker containers to provide a quantitative analysis.

Let us consider the following three facts. (1) VMware NSX has very correctly stipulated a rule-of-thumb size up bound that one networking orchestration domain manage connecting no more than 1,000 hypervisors; (2) a Docker host is analogous to a hypervisor however can host 10-100 folds up the number of virtual CPUs over a hypervisor can do; and (3) networking scale-UP limit necessarily relates to the number of communication entities within the orchestration domain—in the case of networking orchestration in specific, the orchestration domain in question is roughly DHCP and ARP protocols broadcasting management region. From these three facts we can conclude that a non-network-virtualization-connected Docker cloud, i.e., one which is scaled-UP by underlay hard labelling and packet encapsulation protocol technologies, should have a rule-of-thumb size up bound of not much bigger than a handful hardware servers. At this trivial size level, it would be very hard for an IaaS provider, or even for a PaaS provider to achieve a desirable and business sustainable service economy of scale.

About a week before the time of writing this whitepaper, AWS announced EC2 Container Services (ECS) in free trial. From that announcement we cannot find useful information to judge whether ECS's networking is scalable in OUT or in UP senses, nor can we know ECS is suitable for IaaS business, a DiaoSi scale of PaaS business, or for a really large scalable PaaS business.

Time will tell.

4. **DaoliCloud's Network Virtualization Infrastructure (NVI) Technology:** Cloud Networking for Unbound and Arbitrarily Elastic Scalability

In the NVI technology, IaaS, or PaaS resources are deployed in distribution in separate and completely independently orchestrated cloud resource management platforms. Figure 2 illustrates the working principle of DaoliCloud's Network Virtualization Infrastructure (NVI) Technology.

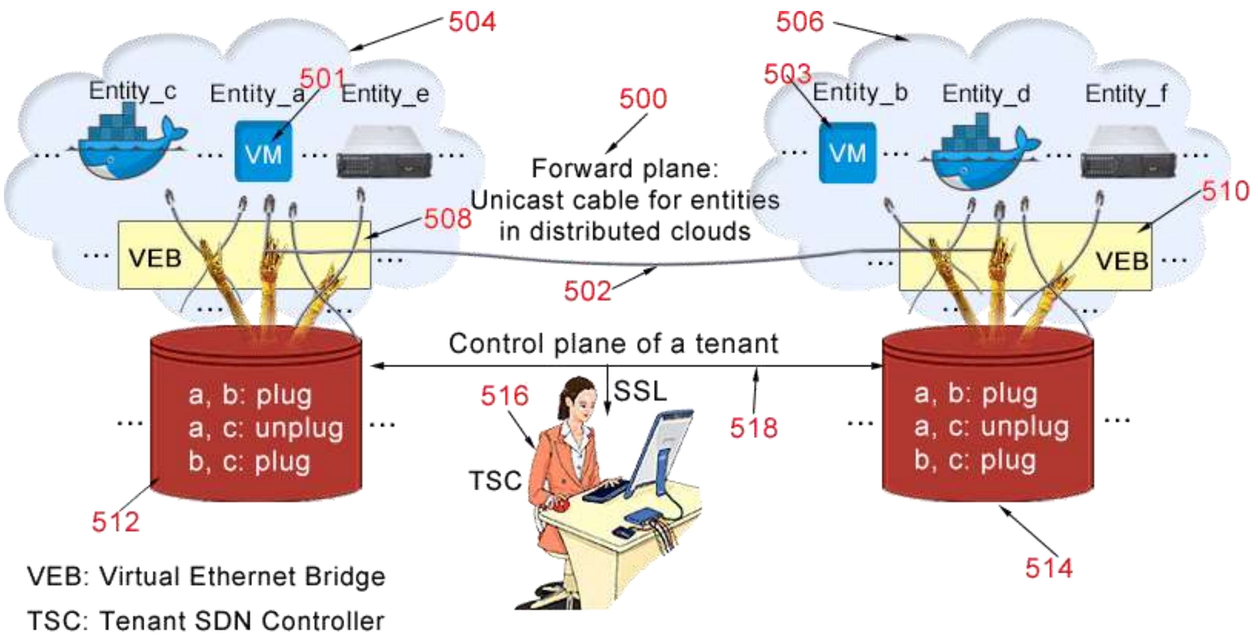

Figure 2. Network Virtualization Infrastructure (NVI) Technology

For a rather extreme view of independence and distribution in cloud orchestration, the reader may imagine each Docker host forming an independent cloud resource orchestrator, i.e., each Docker host forms a tiny sized cloud. Making cloud orchestration domain small is very desirable, because the only known technologies for orchestration of CPUs, memories, storage spaces, and networking devices inevitably involve time-out-able event queuing, copy-on-write database state synchronization, write-lock file systems, cluster leader election, etc.; these technologies simply do not scale. We have to limit the physical implementation size of any cloud orchestration platform to a rather humble scale. In fact, the rule-of-thumb stipulation that one VMware NSX controller should only orchestrate no more than 1,000 hypervisors, is very wise, demonstrating a profound knowledge level and seasoned experience of its architects.

In the NVI technology, physically independently deployed and orchestrated not-so-large-scale clouds can be tenant connected using a Tenant SDN Controller (TSC). TSC is innovated out of the Openflow technology and hence can be viewed as an Openflow controller. However as the name suggests and unlike a usual Openflow controller, TSC does not have a data path in neither clouds it helps to connect. TSC comes into action only upon a communication flow takes place between distributed entities of a tenant, in particular when the distributed entities are in independently orchestrated clouds. As Figure 2 illustrates, connection (plug) or disconnection (unplug) of a data path "unicast cable" between any pair of CPUs requiring communication service (or being denied of so) is executed by TSC in terms of programming at Virtual Ethernet Bridges (VEBs) which are ubiquitously distributed everywhere wherever there are CPUs requiring communication services, be they in forms of hardware, VMs or containers.

To architect the network scalable OUT property for NVI, data path in NVI has been carefully designed to avoid any visibility to TSC. Our solution is to avoid underlay hard labelling and packet encapsulation. In encapsulation protocols, the entire data path between two overlay CPUs is visible by an SDN controller. Thus, an SDN controller if involved in using encapsulation protocols will see an enlarged network when connecting separate network parts, in essence encountering network scale-UP. Figure 3 uses Docker containers to illustrate a respecting-and-hornoring-overlay-CPU-labelling-work-for-context-isolation and non-encapsulation technology for connecting separate and independently orchestrated clouds. The technology of course also applies to connecting VMs or even hardware servers, or mixture of everything, as long as CPUs for connection are distributed to/over where SDN plug-unplug programming can be performed.

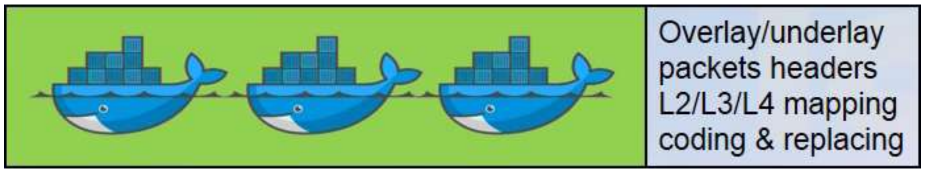

Figure 3: Trans-docker-host Connection of Entities by Respecting Overlay Application Context Isolation Labels (soft labelling)

Let us provide a description for key technology knowhow for the NVI technology:

- A novel and useful improvement to the Openflow standard.
- L2/L3/L4 header metadata mapping, coding and replacing technology (Compare packet structures in Figure 3 and Figure 1).
- The mapping and coding information is agreed upon between two servicing VEBs via the Openflow control path connecting TSC to distributed VEBs.
- Random mappings are non-secret; TSC controller can help agree upon mappings to connect separate overlay CPUs while not seeing intranets internal information of neither clouds hosting the CPUs.
- That's how notion of Tenant SDN Controller (TSC) gives rise: TSC working with independently orchestrated clouds at their respective gateways to connect distributed CPUs within.

- Minus underlay hard labelling and encapsulation, all other virtues of Openflow are kept, e.g., efficient per flow checking routing in VEB in a state machine fast-path manner, instead of inefficient per frame labelling/checking underlay hard labels (i.e., the yellow header content in Figure 1).
- Extremely efficient: Header metadata replacement can be operated in nest to eliminate MAC populating in exponential speed of reduction! Also no packet enlargement, no fragmentation, no need of DHCP or ARP broadcast via TSC, …

A more detailed technical whitepaper "One Variable to Control Them All for Openflow" is available in www.daolicloud.com That whitepaper describes not only how the NVI and TSC technologies can connect independently orchestrated clouds with network virtualization enabled elastic scaling out and in, but also manifests the technologies' other uses in network function virtualization (NFV), service chain QoS, etc.

5. Conclusion

Let us list a few key findings to conclude this whitepaper:

- Avoid physically implementing a cloud orchestration platform too large, 1K hardware CPUs per orchestration domain for the case of hypervisor CPU virtualization technology, and a handful number of such for the case of container CPU virtualization technology, would be a good rule-of-thumb scaling up limit.
- Avoid connecting clouds using underlay hard labelling and the consequent packet encapsulation technologies since they scale UP the size of the resultant network when patching connecting clouds.
- The NVI technology permits to connect globally distributed CPUs in independently orchestrated and small-physical-size-implemented clouds using an overlay communication context isolation labelling method. In so scale-OUT patching connection, an IaaS or PaaS cloud can have arbitrary scalability and elasticity, and yet still achieve good operational stability.

In this way, we have achieved for the first time in industry "One Cloud Two Orchestrators", where orchestrators can be implemented by Openstack, Cloudstack, Contrail, vRealize, Kubernetes, Azure, ACI, … All of these can be inter-cloud, and even inter-service-provider, connected using the NVI and TSC technologies which we have described in this whitepaper.

With the detailed technology description provided in this whitepaper, anyone with average networking knowledge and kernel programming skill can implement the NVI and TSC technologies, and can provide arbitrarily large scalable and elastic cloud IaaS and PaaS services.

6. Public Beta Trial

Interested reader may register a trial account to use DaoliCloud at:

www.daolicloud.com

You will see that DaoliCloud is the world's first inter-cloud connecting everything: Docker containers, VMs, hardware servers, it provides IaaS and PaaS services, and it permits tenants to self-service construct VPCs spanning a number of independently orchestrated Openstack in globally distributed datacenters.

7. Acknowledgements

I would like to thank all colleagues at DaoliCloud for their excellent implementation of NVI, for collective and sustained innovations together, and for their hardwork to have made very useful contributions.

Early and frequent discussions of the Docker PaaS technology with BAI Xiaoyong of UFIDA have proven been very beneficial and inspirational to this work.