Traffic Steering: Considerations, Alternatives, and Solution Requirements
An Industry Whitepaper

Executive Summary
OpenStack has rapidly become the de facto standard open source software platform for the delivery of network functions virtualization (NFV).

Unfortunately, NFV environments present OpenStack with challenges not encountered in the traditional cloud computing environment. To use OpenStack as the software platform for NFV, plug-ins and external services must overcome these gaps:

- Overcoming Traffic Asymmetry
- Selecting Services
- Scaling Horizontally
- Subscriber Awareness
- Service Self-Healing
- Standards Compliance

In practice, most communications service providers (CSPs) fall into one of two camps: either they haven’t realized these problems exist, or they think these problems can be overcome by:

- OpenStack’s Load Balancer as a Service
- Open vSwitch and DPDK; or
- An external integrated load balancer

This paper introduces and explains the gaps encountered when applying OpenStack to NFV applications, and demonstrates that none of the three proposed alternatives above completely satisfies the solution requirements.

Finally, the paper closes by very briefly introducing a solution that does overcome the problems.
Introduction to OpenStack Networking

OpenStack has rapidly become the de facto standard open source software platform for the delivery of Infrastructure as a Service (IaaS), cloud computing, and network functions virtualization (NFV).

NFV is a carrier-led effort to move away from proprietary hardware, motivated by the desire to dramatically increase agility, to enable faster service launches, and to dramatically reduce the cost of deployment.

A topic closely related to NFV is service function chaining—a technique for selecting and steering data traffic flows through various ‘service functions’, allowing operators to deploy a wide range of services. For example, a communications service provider (CSP) may use service function chaining to intelligently steer relevant traffic through service functions like network security (e.g., DDoS scrubbers, tarpits), parental controls, and traffic optimization solutions (e.g., congestion management, TCP acceleration).

OpenStack consists of many interrelated components which are controlled through standard open OpenStack application programming interfaces (APIs). OpenStack is maintained and developed by a very active group of individuals and technologists and is backed by some of the most innovative and some of the largest organizations within the technology space.

APIs and Plug-in Architecture

OpenStack provides a set of standard open source software components. Each component is replaceable, so an operator is free to choose components from specific vendors or open source alternatives.

To ensure that compatibility and flexibility are maintained, OpenStack provides a set of standard APIs and a plug-in architecture for components (see Figure 1).

Figure 1: OpenStack API Map

[Diagram showing the interrelated components of OpenStack and their functions, such as Heat for orchestration, Horizon for providing UI, Neutron for network connectivity, Nova for VMs, Glance for image management, Swift for object storage, and Keystone for authentication and authorization.]

1. Reproduced from https://www.openstack.org/
Neutron

The Neutron service component and associated APIs are vitally important within OpenStack, as they control the networking modules. These networking capabilities allow OpenStack to connect to standard Ethernet and IP-based physical underlay networks by:

- associating the hosted cloud applications with underlying IP or Ethernet physical networking
- providing interconnectivity between hosted cloud applications; and
- providing a programmatic API allowing for the effective provisioning of networking services to hosted applications

Neutron is designed to provide connectivity as a service for the compute and networking nodes within the OpenStack infrastructure (i.e., virtual network interface cards, or vNICs) and external networking components. It does so through a series of agents which provide services to the infrastructure and hosted compute notes; these agents are designed to be replaceable and extendable for adding vendor-specific capabilities or for extending the existing Neutron capabilities as needed.

Figure 2: OpenStack Neutron logical architecture²

2. Reproduced from https://docs.openstack.org/security-guide/networking/architecture.html
OpenStack Neutron and NFV: Considerations and Limitations

Neutron was built to deliver traffic from underlying networks to hosted compute instances and services within the OpenStack environment. Unfortunately, NFV environments present Neutron with some unique challenges not encountered in the traditional cloud computing environment. These challenges expose gaps in Neutron (see Table 1) that must be filled by plug-ins and external services in order to use OpenStack as the software platform for network functions virtualization.

Table 1: Gaps in Neutron that must be filled by plug-ins and external services

<table>
<thead>
<tr>
<th>Neutron Gap</th>
<th>Requirement</th>
</tr>
</thead>
</table>
| Overcoming Traffic Asymmetry | Communications service provider networks are inherently asymmetric, as a consequence of being engineered for efficiency and reliability. By contrast, asymmetry doesn't exist in traditional cloud computing environments.  
Subscriber services (and most network services, in general) require that traffic (i.e., data packets) be presented to the service in a completely symmetric manner. |
| Selecting Services           | Traffic must be delivered to active service instances, even as they migrate, instantiate, and scale.  
Databases and similar applications don't push packets, so as long as the instance in the cloud is connected to the same network as the other cloud elements, things are generally fine.  
Network VNFs depend upon getting the right packets to the right instance while maintaining high performance. |
| Scaling Horizontally         | Traffic must be delivered to the correct service instance (i.e., VNF), balanced appropriately across instances. Ideally, traffic is delivered to service instances as they are instantiated and is removed from service instances as they reach the end of the VNF lifecycle.  
As with the previous explanation, it's all about getting the right packets to the right instance while maintaining high performance. |
| Subscriber Awareness         | Unlike many traditional cloud computing services (e.g., Salesforce, SAP, Amazon Web Services, etc.), the majority of CSP services are subscriber-centric and stateful. Successfully delivering these services via VNFs requires subscriber affinity for the duration of service life; that is, the traffic for a particular subscriber must always be directed to the same VNF instance.  
Neutron does not have built-in subscriber awareness, so this function must be provided by an external (relative to Neutron) component. |
| Service Self-Healing         | If a VNF gets into—or is approaching—an unhealthy state, then the underlying traffic handling components must be able to actively direct new connections to different VNFs, and to divert existing connections from the instance (dependent upon the failure type) to ensure service continues. |
| Standards Compliance         | Having a standards-compliant mechanism for steering traffic is the only true way to ensure cross-compatibility with multiple like and unlike VNFs.  
Though it's possible to use proprietary systems to steer traffic, the likelihood of software changes introducing unwanted behavior is high. Standards bodies like the IETF, ETSI, and 3GPP ensure a standard approach is taken to ensure interoperability of vendors. |

The rest of this paper examines several potential solutions to the problems identified above.\textsuperscript{3,4,5}

---

\textsuperscript{3} The IETF's Network Service Header (NSH) information is at \url{https://datatracker.ietf.org/doc/draft-ietf-sfc-nsh/}

\textsuperscript{4} The IETF's service function chaining information is at \url{https://datatracker.ietf.org/wg/sfc/documents/}

\textsuperscript{5} ETSI is working to define the Traffic Steering Support Function (TSSF)
**Proposed Solutions**

The following subsections examine the solutions most commonly proposed by CSPs for the gaps outlined previously:

- OpenStack Load Balancer as a Service
- Open vSwitch with DPDK
- External Integrated Load Balancer

**OpenStack Load Balancer as a Service**

The OpenStack Load Balancer as a Service (LBaaS) allows for the Layer 3 balancing of service traffic between multiple OpenStack compute instances; within the OpenStack infrastructure, the LBaaS provides basic Layer 4 load balancing (i.e., by IP/Port) within a single compute instance, but cannot extend to multiple compute instances.

LBaaS has developed significantly since the early version 1 implementation and now supports multiple Load Balancer instances, each containing multiple service listeners and controlling pools of multiple compute instance members (as shown in Figure 3). Each listener is dynamically associated with a pool of IP addresses which are balanced and health monitored as a group.

LBaaS is controlled by an extensive API, based on REST/JSON.

While LBaaS has some powerful features and benefits that make it a viable option to replace traditional service load balancing approaches (e.g., for web scaling applications), it does not overcome the gaps identified previously.

---

6. Reproduced from [https://docs.openstack.org/ocata/networking-guide/config-lbaas.html](https://docs.openstack.org/ocata/networking-guide/config-lbaas.html)
Table 2: OpenStack Load Balancer as a Service compared against Neutron gaps

<table>
<thead>
<tr>
<th>Neutron Gap</th>
<th>Solved?</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overcoming Traffic Asymmetry</td>
<td>No</td>
<td>LBaaS has no ability to overcome asymmetry.</td>
</tr>
<tr>
<td>Selecting Services</td>
<td>Partial</td>
<td>LBaaS supports service selection through the configuration of the service, but it does not support any dynamic service selection based on traffic information.</td>
</tr>
<tr>
<td>Scaling Horizontally</td>
<td>Yes</td>
<td>LBaaS supports scaling instances horizontally and supports the most common load balancing algorithms.</td>
</tr>
<tr>
<td>Subscriber Awareness</td>
<td>No</td>
<td>LBaaS is not subscriber-aware.</td>
</tr>
<tr>
<td>Service Self-Healing</td>
<td>Partial</td>
<td>LBaaS supports basic health checking and service monitoring.</td>
</tr>
<tr>
<td>Standards Compliance</td>
<td>Yes</td>
<td>LBaaS does conform to standards-based implementations, but open source components often lag in adopting the latest standards.</td>
</tr>
</tbody>
</table>

Open vSwitch and DPDK

Open vSwitch (OVS) is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It's designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (e.g., NetFlow, sFlow, IPFIX, RSPAN CLI, LACP, 802.1ag, etc.).

OVS also supports distribution across multiple physical servers.

OVS can operate both as a soft switch running within the hypervisor, and as the control stack for switching silicon. It has been ported to multiple virtualization platforms and switching chipsets.

Due to its robust capabilities, OVS is the default switch in many hypervisors, and has been integrated into many virtual management systems—including OpenStack.

Figure 4: General Open vSwitch architecture

7. Learn more at [https://openvswitch.org/](https://openvswitch.org/)
8. Reproduced from [https://docs.openstack.org/liberty/networking-guide/scenario-classic-ovs.html](https://docs.openstack.org/liberty/networking-guide/scenario-classic-ovs.html)
Native Open vSwitch forwards packets via the kernel space data path. In this kernel data path, the switching ‘fastpath’ consists of a simple flow table indicating forward/action rules for received packets. Exception packets (i.e., the first packet in a flow) don’t match any existing entries in the kernel fastpath table and are sent to the user space daemon for processing (slowpath).

After user space handles the first packet in the flow, the daemon updates the flow table in kernel space so that subsequent packets within the same flow are processed in the fastpath, rather than being sent to the user space.

By using this approach, native OVS eliminates the costly context-switching between kernel and user space for the vast majority of received packets. However, the achievable packet throughput is still limited by the forwarding bandwidth of the Linux network stack.

To overcome this bottleneck, the solution relies on the Data Plane Development Kit (DPDK)\(^9\). DPDK is a set of user space libraries that enable a user to create performance-optimized packet processing applications. In practice, DPDK offers a series of Poll Mode Drivers (PMDs) which enable direct transferal of packets between user space and the physical interfaces—bypassing the kernel network stack.

Bypassing the kernel network stack provides a significant performance boost over kernel forwarding, by eliminating both interrupt handling and traversing the stack itself.

By integrating OVS with DPDK, the switching fastpath moves into user space, and the exception path becomes the same path traversed by packets in the kernel fastpath case.

Integrating OVS with DPDK addresses some performance issues that held back kernel mode OVS, making it better suited to the performance demands of network functions virtualization (see Figure 6).

\(^9\) Learn more over at [http://dpdk.org/](http://dpdk.org/).
While the packet throughput performance is high, OVS with DPDK still doesn't overcome Neutron's gaps sufficiently to be a real-world traffic steering option.

<table>
<thead>
<tr>
<th>Neutron Gap</th>
<th>Solved?</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overcoming Traffic Asymmetry</td>
<td>No</td>
<td>OVS with DPDK does not overcome traffic asymmetry.</td>
</tr>
<tr>
<td>Selecting Services</td>
<td>Partially</td>
<td>OVS with DPDK is capable of performing basic service selection and forwarding to instances utilizing external SDN or OpenFlow controllers, for installation of forwarding rules following flow detections.</td>
</tr>
<tr>
<td>Scaling Horizontally</td>
<td>Partially</td>
<td>OVS with DPDK supports only very basic load balancing, and there isn't any instance scale-out/back.</td>
</tr>
<tr>
<td>Subscriber Awareness</td>
<td>No</td>
<td>OVS with DPDK is not subscriber-aware.</td>
</tr>
<tr>
<td>Service Self-Healing</td>
<td>No</td>
<td>OVS with DPDK does not support any service healing capability and relies on external components to reconfigure connectivity.</td>
</tr>
<tr>
<td>Standards Compliance</td>
<td>Partially</td>
<td>OVS with DPDK can be programmed to deliver SFC-like behavior, but it is not a native function.</td>
</tr>
</tbody>
</table>

It's also worth mentioning that OVS with DPDK is very resource-intensive (i.e., all or nothing on DPDK instances), and there's a complex development tree and capability mapping. The problem of being resource-intensive will likely remain until at least 2019.

Additionally, while OVS with DPDK is not constrained to a single compute chassis, getting packets from one chassis to another would introduce latency, and requiring anything of OVS with DPDK that’s more advanced that L2/L3 would increase the compute requirements.

**External Integrated Load Balancer**

Integrating an external load balancer is a common approach, as load balancer vendors begin to introduce additional capabilities to preserve the value of their existing platforms. This approach allows for potentially reusing existing load balancers and application delivery controllers (ADCs).

While there are several ways to integrate an external load balancer to an OpenStack solution, the most common method is by using ML2 Agents. Service Agents\(^{11}\) are used to connect to the MANO-connected Neutron API to the underlying network technology.

---

11. Service Agents are part of OpenStack under Neutron. Basically, you ask Neutron for a network of Type X, then it uses an ML2 agent to talk to something to set up the network (so you have agents for OVS, Linux Bridge, external switches, etc.). MANO talks to Neutron, and Service Agents translate Neutron to the actual network setup.
While integrating external load balancers or ADCs is promoted heavily by vendors—who want to remain relevant as things shift to NFV—the approach has serious shortcomings. For instance, it presents some impractical limitations in the evolution of NFV services, and it limits scalability by requiring an inline hardware platform be deployed and scaled based on network planned capacity rather than utilization—very much against the intention of NFV.

<table>
<thead>
<tr>
<th>Neutron Gap</th>
<th>Solved?</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overcoming Traffic Asymmetry</td>
<td>No</td>
<td>The load balancer or ADC expects symmetric flows, so asymmetry must still be resolved separately.</td>
</tr>
<tr>
<td>Selecting Services</td>
<td>Partially</td>
<td>Service selection can be achieved by an external system; however, additional complexity is often involved in configuring and integrating additional external components.</td>
</tr>
<tr>
<td>Scaling Horizontally</td>
<td>Partially</td>
<td>Horizontally scaling instances can be achieved by an external system; however, additional complexity is often involved in configuring and integrating additional external components.</td>
</tr>
<tr>
<td>Subscriber Awareness</td>
<td>No</td>
<td>Load balancers and ADCs are not subscriber-aware.</td>
</tr>
<tr>
<td>Service Self-Healing</td>
<td>Partially</td>
<td>Service healing can be achieved by external network components; however, doing so requires significant integration into the NFVi and MANO to monitor service instance health.</td>
</tr>
<tr>
<td>Standards Compliance</td>
<td>No</td>
<td>Load balancers and ADCs often don't consider service chaining or traffic steering as a necessary aspect of their solutions; most such solutions are built with user-defined rules per application (i.e., port) and the destination IP addresses to balance or ensure availability.</td>
</tr>
</tbody>
</table>
Conclusions

NFV environments present OpenStack with challenges not encountered in the traditional cloud computing environment. To use OpenStack as the software platform for NFV, plug-ins and external services must overcome a number of gaps.

Many communications service providers (CSPs) are looking to overcome these gaps by using OpenStack’s Load Balancer as a Service, Open vSwitch with DPDK, or an external load balancer.

The reality, though, is that none of these proposed solutions completely overcomes the gaps in Neutron exposed by NFV use cases (see Table 3).

<table>
<thead>
<tr>
<th>Neutron Gap</th>
<th>OpenStack’s LBaaS</th>
<th>OVS with DPDK</th>
<th>External Load Balancer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overcoming Traffic Asymmetry</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Selecting Services</td>
<td>Partially</td>
<td>Partially</td>
<td>Partially</td>
</tr>
<tr>
<td>Scaling Horizontally</td>
<td>Yes</td>
<td>Partially</td>
<td>Partially</td>
</tr>
<tr>
<td>Subscriber Awareness</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Service Self-Healing</td>
<td>Partially</td>
<td>No</td>
<td>Partially</td>
</tr>
<tr>
<td>Standards Compliance</td>
<td>Yes</td>
<td>Partially</td>
<td>No</td>
</tr>
</tbody>
</table>

A Real-World Solution: Sandvine’s Traffic Steering Engine

To overcome the shortcomings of the alternatives examined in this whitepaper, we created the Traffic Steering Engine\(^\text{12}\).

Sandvine’s Traffic Steering Engine addresses all of the Neutron gaps discussed previously, enabling CSPs to deploy service function chains in a straightforward, standards-compliant manner.

Additional Resources

Thank you for taking the time to read this whitepaper. We hope that you found it useful, and that it contributed to a greater understanding of how to deploy service function chains in a virtualized network.

In addition to the resources cited in this document, please consider reading these documents related to network functions virtualization and policy control, all of which are available on www.sandvine.com:

- EANTC Report: *Independent Scalability and Functionality Test: Sandvine Virtualized Traffic Steering Engine (TSE) and Virtualized Policy Traffic Switch (PTS)\(^\text{13}\)*
- Whitepaper: *Implementing Policy Control as a Virtual Network Function: Challenges and Considerations\(^\text{14}\)*
- Whitepaper: *Applying Network Policy Control to Asymmetric Traffic: Considerations and Challenges\(^\text{15}\)*
- Technology Showcase: *Breaking the Terabit Barrier\(^\text{16}\)*

If you have any feedback at all, then please get in touch with us at whitepapers@sandvine.com.

---

\(^{12}\) If you want to learn more about the Traffic Steering Engine, then please visit [https://www.sandvine.com/products/traffic-steering-engine/](https://www.sandvine.com/products/traffic-steering-engine/)


