Network Segmentation in the Cloud A Novel Architecture Based on UCC and IID

Sebastian Jeuk*†, Gonzalo Salgueiro‡, Fred Baker†, Shi Zhou*
*Department Computer Science
University College London (UCL),UK
†Cisco Systems, San Jose, USA
‡Cisco Systems, Research Triangle Park, USA

***Abstract*—Cloud Computing is known for its scalability, flexibility and on-demand workload creation. Today, cloud-enabled data centers utilize VLAN, VxLAN or GRE segmentations but these techniques, despite being widely deployed, have a variety of inherent technical and architectural limitations. In this paper we introduce a novel architecture leveraging UCC and IID for segmentation, rather than those traditionally used today (e.g., VLAN, VxLAN, etc.). The proposed architecture is entirely based on IPv6 and, for illustrative purposes only, is demonstrated using OpenStack as the cloud framework. This proposed reference architecture is based entirely on UCC and IID, two OpenStack-independent concepts, could easily be realized in outer cloud frameworks as well. UCC introduces cloud-specific traffic isolation within IPv6 extension headers. IIDs can be incorporated as a unique identifier within an IPV6 address to identify endpoints. The combination of both allows network devices to segregate traffic according to cloud service, cloud tenants and endpoint affiliation. Here, we highlight current shortcomings of existing segmentation techniques as well as define design considerations for the cloud framework in question (i.e. in this case OpenStack) to circumvent such limitations. The proposed architecture is depicted and explained in the context of a traffic flow example.**

I. Introduction

Cloud Computing, beside its obvious advantages of building highly dynamic, flexible and scalable infrastructures, also provides the means to build environments based on technology building blocks. The decisions of what technology to use are often based on the needs and requirements of the cloud provider, the services offered and its consumers or tenants. On the networking side cloud providers can choose to segment their networks based on VLANs, VxLANs or by using GRE tunnels. Each of these technologies have their own characteristics and limitations.

These segmentation techniques are often defined by the scalability of their component technologies, their configuration simplicity and their ability to effectively interoperate with other technologies governing network, compute and storage resources. VLANs, a legacy segmentation technology, are limited to 4096 segments and require a considerable additional configuration throughout the network. VxLAN, on the other hand, can scale to several million segments. However as an overlay technology adds overhead to the packet and the requirements within the network. GRE tunnels, an overlay technology similar to VxLAN, is used to interconnect the compute nodes with each other and the controller. That results in a fully-meshed GRE-tunnel topology. Within the tunnel, the tunnel-ID is used to segment traffic according to its service affiliation. A tunnel mesh is not scalable as it increases with each additional compute node. All three approaches, in addition to their specific limitations, also face challenges when isolating cloud-relevant entities based on the provider, service or tenant affiliation.

In this paper we highlight some of those shortcomings as well as solution-specific considerations of segmentation technologies.As a means for overcoming these limitations, we propose an optimized cloud architecture solution leveraging Universal Cloud Classification (UCC) and the Interface Identifiers (IIDs), while also eliminating the need for additional network segmentation technologies completely. This solution is based on IPv6 and outlines how cloud providers can build highly scalable environments with very little configuration overhead and minimal dependence on interoperability with other network technologies. This treatise underscores the power and flexibility of this proposed architecture with a solution based on the open source cloud software OpenStack. Despite being depicted with OpenStack, it is worth noting that the solution presented here is independent of any particular cloud computing framework since it is entirely based on the underlying network technologies, and consequently, is implementation independent. It defines reference architectures for all three segmentation approaches (i.e., VLAN, VxLAN and GRE).

We first discuss these segmentation strategies and highlight their limitations and difficulties followed by an introduction to UCC and IID. We then outline the proposed solution based on UCC, IID and IPv6 and highlight its novel aspects and the flexibility and advantages it offers to cloud providers. This innovative architecture is simply one of the possible use-cases of UCC. It provides the flexibility to set up a cloud environment without the need for any underlying segmentation technology, therefore eliminating their inherent limitations. The proposed architecture provides scalability, simplified configuration and network management. Additionally, it inherits the advantages of UCC by enabling network services based on the introduced identifiers.

II. Background

A. Universal Cloud Classification

The Universal Cloud Classification (UCC) scheme is part of the authors research in [1] and further evaluated in [2] and [3]. It is introduced to circumvent the shortcomings seen in state-of-the-art segmentation technologies, including VLANs, VxLANs, GRE tunnels and other more application-specific isolation.

The proposal defines three identifiers (IDs) that are incorporated into an IPv6 extension header. The IDs are structured based on the Digital Object Identifier (DOI) scheme. The first ID, called Cloud-ID, isolates traffic according to its cloud provider affiliation. It is a 4 byte long identifier separated into a registrar and provider sub-ID. To manage cloud provider IDs on a global basis we suggest the use of a registrar similar to DNS. This registrar provides the means to guarantee uniqueness in assigned IDs. The registrar sub-ID can be used to incorporate its global location. The second ID is used to define the Services run within a cloud provider. The Service-ID is 6 byte long identifier separated into sub-IDs to incorporate metadata on the data center location, the service itself and optional information. Finally, the tenant or consumer is identified by a 6-byte long Tenant-ID. Only the Cloud-ID has global significance, while the Service-ID and Tenant-ID are locally significant within the provider network identified by the specific Cloud-ID.

The novelty of the Universal Cloud Classification scheme can be summarized as follows: A hierarchical end-to-end classification scheme consisting of three IDs (Cloud/Service/Tenant) closely reflecting the internal structure of cloud environments. These IDs are carefully selected and defined to solve the classification challenges seen in Cloud Computing. The scheme can be succinctly characterized by the following high-level points:

- hierarchical
- end-to-end
- optional
- flexible and extensible
- universal
- guaranteed uniqueness

B. Interface Identifiers (IID)

The Interface Identifier (IID) is part of the IPv6 stack and defined in RFC 4291 [4]. It is used to uniquely identify interfaces on a link and typically incorporated into an IPv6 unicast address. For unicast addresses the interface identifier is required to be 64 bit long and describes the host portion of the IP address.

Uniquely assigning Interface IDs leverages the IEEE EUI-64 standard for network interface addressing. These addresses can be derived from IEEE 802 MAC addresses. OpenStack is managing and assigning unique MAC addresses to new instances. This observation allows us to use the MAC address assigned to a VM to derive the IEEE EUI-64 address and use it as a unique IID to identify tenant specific VMs. To better understand the procedure on transforming an IEEE 802 MAC address into a IEEE EUI-64 IID we first outline the structure of a MAC address and then show the mapping to 64 bit.

IEEE 802 interface identifiers use 48-bit, split into two 24-bit long sections. The first 24-bit depict the manufacturer ID while the later 24-bit define the board ID. The combination of both IDs produces a globally unique 48-bit address.

With IEEE EUI-64 the manufacturer ID is still 24-bits, however the board ID is now 40 bits long. The conversion between the 48-bit long IEEE 802 MAC address and the 64-bit long IEEE EUI-64 address happens by inserting 0xFFFE (or 1111 1111 1111 1110) in between the manufacturer and the board section. To be able to use the EUI-64 addresses now as IPv6 Interface Identifiers (IIDs) the 7th bit of the 1st byte in the EUI-64 address has to be complemented.

C. OpenStack

OpenStack is an open source cloud framework based on a modular approach to manage compute, network and storage resources in Cloud environments. The most prominent components are (1) Nova, responsible for assigning and managing compute resources, (2) Neutron, handling network related configurations and (3) Cinder, defining block-storage resources. There are approximately 10 different components, which we will not further highlight in this paper, refer to [5] for further details).

For the purposes of this paper we will briefly highlight the neutron services as they handle the virtualized networks and the interconnection to the physical data center. The Neutron framework consists of several sub-services that manage layer-3, DHCP, DNS and L2-switching capabilities. Depending upon the way OpenStack is deployed, these components use VLANs, VxLAN or GRE to interconnect compute hosts with network resources.

The following section outlines those three reference Architectures and highlights their limitations and challenges observed in production environments.

D. OpenStack Reference Architectures

This section focuses on outlining the different reference architectures as used within the open source cloud software OpenStack. These architectures, even though specific to OpenStack, highlight the implementation and usage of VLAN, VxLAN and GREs in cloud environments. OpenStack uses the word ”tenant” as a way to describe projects. In this paper, we will use ”tenant” to describe consumers of a service.

A typical OpenStack deployment [6] consists of multiple nodes that are interconnected in different ways while performing different tasks. Here, we will focus on the Neutron network node, the compute and the controller nodes. The Neutron network node is responsible for providing required network services for the OpenStack environment, including L3, DHCP and NAT functionality. The compute node is typically hosting the instances owned by the different services while the controller node is used to manage the OpenStack environment providing further services. When looking at reference

architectures using VLANs, VxLANs or GRE tunnels these components are interconnected in different ways. The functionality of OpenStack however remains the same independent of the transport protocol in use. However, depending on the technology used, different limitations and considerations are important to consider when running an OpenStack cloud. Figure 1 is used as the reference to explain the different segmentation approaches in OpenStack.

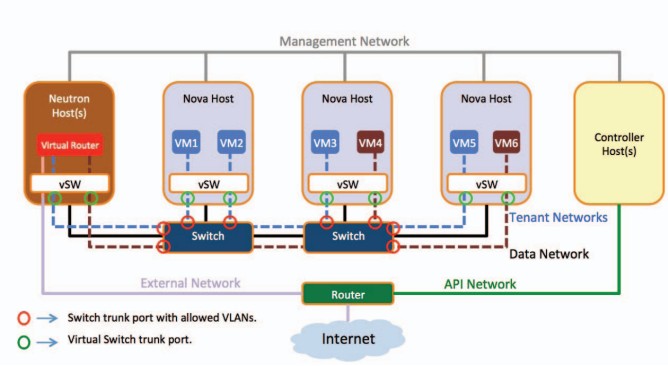

Fig. 1. Reference Architecture

In addition to OpenStack-specific components, cloud environments also require switching fabrics that physically interconnect the different OpenStack nodes. All of these components are affected by the choice of segmentation and different limitations apply.

First, we show how OpenStack uses VLANs to segment networks within a project. That is followed by similar outlines of the VxLAN and GRE architectures. Each section highlights some of the advantages and disadvantages of using each of these implementations.

A typical OpenStack implementation consists of a minimum of four different networks. The public network is used to communicate with the instances running on the compute node from outside, that can be the Internet, corporate network or end users. It provides connectivity to the globally routable address space used within the OpenStack cluster. The management network is used for communications between the different services operated by OpenStack. It provides connectivity between the network, compute and controller nodes to exchange database queries, Advanced Message Queuing Protocol (AMQP) messages and high availability information. As a pure OpenStack control plane network it is often physically separated from the private and public network. The private network allows communication within a tenant's environment, between tenant VMs. Depending on the transport protocol used, the public network and the private network are segmented using VLANs, VxLANs or GRE tunnels.

1) VLAN Architecture: In the VLAN reference architecture, segmentation is based on IEEE 802.1q. The switching fabric provides the required traffic separation for inter-node communication. In OpenStack, each network gets assigned a new Segmentation-ID, which refers to a unique VLAN-ID. These IDs are then used to segregate traffic both within and across an OpenStack project. A network in OpenStack is not bound to a specific compute host, which highlights one of the major drawbacks of using VLANs to segregate traffic. The VLAN provisioning has to be managed either from OpenStack via plug-ins (talking to the switching fabric) or via other means (e.g., switching fabric controllers, manually by the administrator, etc.).

In cloud environments, instances residing in the same 802.1q segment are typically not running on the same compute node. They can reside on any compute node with available resources. The network node and the switching fabric have to be aware of the locations of every VM in a segment to provide the correct 802.1q connectivity. In case VMs move (workload mobility) the switching fabric has to be reprogrammed to maintain connectivity. This highlights another shortcoming of VLAN based cloud architectures. The configuration is both complex and cumbersome to manage and maintain.

The third and most severe shortcoming of IEEE 802.1q is the size of the VLAN ID field, which is limited to 12 bits or 4096 VLANs. Cloud environments demand high scalability, which is extremely limited by segmenting networks using VLAN IDs.

To summarize, a VLAN based Cloud environment has the following shortcomings:

- Configuration overhead across compute, network and switching fabrics
- Complex management of VLANs, their assignments, instance location and per rack configuration
- Scalability limited to 4096 segments per environment

Having said that, though, one of the advantages of VLANs is the broad support of IEEE 802.1q on both legacy and state-of-the art network entities.

2) VxLAN Architecture: The Virtual extensible Local Area Network (VxLAN) can be defined as an L2 overlay over an L3 network. The overlay network is known as a VxLAN segment and identified by a 24-bit long VxLAN Network Identifier (VNI). Tenant traffic is segmented by VNI number, therefore only VMs within the same VNI are allowed to communicate with each other.

Even thought VxLAN solves some of the limitation cloud providers are faced with when using VLANs, it also has its own shortcomings.

- VxLAN requires a multicast environment to enable dynamic MAC-learning for discovery and as means to establish the tunnels. Cloud switching fabric often are not enabled for multicast or do not support the underlying technologies such as PIM or IGMP.
- The MAC-in-IP encapsulation of VxLAN requires a 1600 byte MTU to accommodate the 24-bit header. As a consequence all network devices carrying VxLAN traffic have to support jumbo frames.

3) GRE Architecture: GRE, or Generic Routing Encapsulation, as defined in RFC 1701 encapsulates any protocol in any other protocol. It is considered an overlay or tunnel protocol that allows interconnecting private subnets over a public network by encapsulating IP datagrams within IP datagrams.

OpenStack uses the IP-in-IP tunneling approach to separate tenant networks between hypervisors on different compute nodes and the network node. As a results for every tenant network a separate tunnel has to be created, causing control plane overhead on the compute/network node and the switching fabric. Additionally, due to the IP-in-IP encapsulation the MTU size in the network has to be increased.

Beside the technical considerations when deploying GRE tunnels, experience shows that cloud providers do not deploy GRE tunnels as frequently as the VLAN reference architecture.

III. UCC + IID Network Architecture

As highlighted in the previous sections, current network reference architectures deployable in an OpenStack environment have several limitations, ranging from scalability to configuration overhead.

A. Overview

We propose a solution based on Universal Cloud Classification (UCC) and the Interface Identifiers (IID) that leverage IPv6 to eliminate the need for any Layer 2 segmentation technologies. The proposed base architecture solely relies on UCC and IIDs to isolate traffic flows based on their service, Tenant and VM affiliation. Not only do we eliminate the current architectural limitations but also segment cloud traffic according to cloud-specific characteristics, rather than relying on legacy protocols.

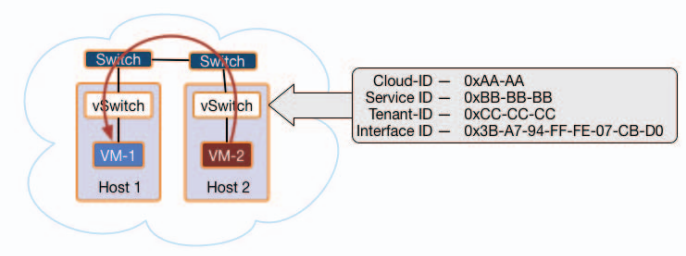

Fig. 2. UCC + IID Based Classification

Figure 2 depicts a simple cloud architecture with two compute nodes, each hosting a vSwitch. The nodes are interconnected using a physical switching fabric. The vSwitch is the first network entity that inspects the packets and matches and isolates them according to their Cloud-ID, Service-ID, Tenant-ID and IID affiliation. The vSwitch uses these identifiers to apply network policies against the traffic flow to define behavior.

First, we will highlight some of the critical design elements for Cloud environments. This is followed by an overview of IPv6 design details describing address types and how they can be used in a cloud environment. To better understand the new architecture and how it eliminates the need for other segmentation technologies, we outline a traffic flow example based on UCC and IID.

B. Design Elements

A newly proposed architecture should reflect the cloud-specific requirements including traffic isolation based on cloud elements (cloud services, cloud tenants and VMs).

The architecture should segment traffic at least as good as current technologies while also removing their limitations. Here, we try to tackle scalability, configuration management overhead and other relevant issues currently seen in OpenStack architectures.

Additionally, the introduced segmentation approach should not increase any security concerns over current technologies. This includes privacy, spoofing and other related concerns.

Based on the shortcomings highlighted for VLAN, VxLAN or GRE deployments we define a couple of critical considerations to take into account when designing an OpenStack Network Architecture.

Here, we assume that the cloud network environment is entirely based on IPv6. Companies such as Google and Facebook are moving towards IPv6-only data centers within the next one to three years [7] [8]. Such moves demonstrate the feasibility and imminent unavoidability of IPv6 in highly scalable and dynamic cloud deployments.

One of the most critical elements of this architecture is its complete lack of dependence on any of the legacy encapsulation technologies. If desired, cloud operators can choose to deploy additional segmentation technologies, but this would be for optimization purposes only. These methods are however not required for the proposed design.

To summarize, our newly proposed OpenStack based network architecture fulfills the following design requirements:

- Data center entirely based on IPv6 (no longer requiring dual-stack support)
- No need for legacy segmentation technologies (VLAN, VxLAN, GRE)
- Cloud-specific classification
- Improved segmentation while tackling current limitations
- Equivalent or superior security over current segmentation technologies

Based on these requirements, we outline the proposed architecture over the next sections.

C. IPv6 Design Details

IPv6 defines multiple address types that can be assigned to a single interface. The following list highlights the most common ones in order of their usability scope.

- Link Local Address (LLA): This address type is typically configured automatically and used for hosts to communicate on their local segment only. Similar to RFC 1918 IPv4 addresses the Link Local Address is not routable, hence routers would never forward these addresses outside a certain segment. A link local address always starts with FE80.
- Unique-Local Address (ULA): This address contains the Interface identifier and should be globally unique. Its usage however should be limited to local communications. A unique-local address always starts with a prefix

of ”1111 110” followed by the L bit set to 1 (locally assigned), hexadecimal defined as ”FD”. The first 8 bits are followed by a 40-bit Global ID and a 16-bit long Subnet-ID. The remaining 64-bits are used for the Interface Identifier (IID).

- Global Unicast Address (GUA): A globally unique and routable unicast address that is equivalent to IPv4’s public address. This address type is split into a 48 bit global routing prefix, a 16-bit subnet-ID and a 64-bit Interface Identifier (IID). The Global Routing Prefix is assigned specific to autonomous systems.

The OpenStack kilo release, [9] introduces several key features for IPv6 support in cloud-enabled data centers. Address assignment for tenant networks is now supported using Stateless Address Auto-configuration (SLAAC) [10] or DHCPv6 [11]. In addition, the Kilo release introduces support for provider networks with Router Advertisements (RAs) [12] messages generated by an external router.

In our proposal we make use of the OpenStack features allowing the assignment of several IPv6 address prefixes to a single interface. By default, an interface receives an LLA to handle traffic within its local segment. Additionally, that interface can also be assigned one or more GUAs for end-to-end connectivity.

We also leverage the enhancement within Kilo that eliminates the need for NAT by assuming that each OpenStack instance gets at least one globally routed address and can communicate directly using pure L3 routing. This removes the additional processing overhead within the ”neutron-l3-agent” OpenStack service performing NAT on north-south or inter-subnet traffic.

The advancements in the latest OpenStack release, Kilo, make it possible to design a network architecture solely based on IPv6. Combining this with the UCC and IID identifiers creates a novel OpenStack IPv6 based architecture.

D. Discussion

The most prominent advantage of the introduced architecture is the capability to provide visibility within a cloud network down to the consumer (tenant) level allowing network services and related policy enforcement on a per-tenant basis.

The proposed architecture is based on UCC and IIDs while leveraging OpenStack as a possible cloud framework. UCC is defined to eliminate shortcomings currently seen in segmentation approaches by introducing a Cloud-ID, Service-ID and Tenant-ID that can be used end-to-end, provides globally unique identifiers and reflects the typical cloud hierarchy.

By combining UCC with the IID of a VM’s interface provides adequate classification details to match each flow by its source, including the service, tenant and VM it originated from. This allows intermediate network entities to classify flows accordingly and isolate them from any other tenant traffic.

By using IPv6 as the addressing scheme for endpoints the architecture eliminates the need for Network Address Translation (both static and dynamic) and allows multiple addresses (both local and global) per VM interface.

In addition to highlighting the architectural advantages we also outline opportunities for further investigation.

wide-spread adoption of UCC

To realize the capabilities of the proposed architecture there needs to be a willings to abandon incumbent classification technologies.

While there has been extensive testing on both UCC [2] and IID independently, the architecture as a whole has not been verified in production environments. The authors, however, strenuously emphasize that the introduction of the combination of UCC and IID data sets have no impact on cloud performance, scalability and other metrics.

E. Flow Example

To better understand how flows can be distinguished based on their UCC + IID information we depict a flow example for a data center internal traffic stream.

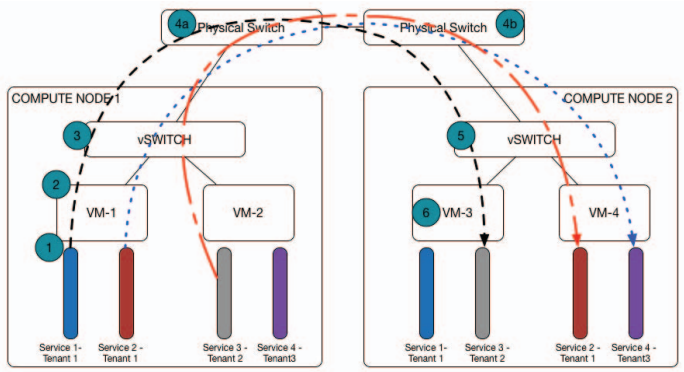

Fig. 3. UCC + IID Enabled Flows

Based on figure 3 we define three different traffic flows. Here we focus our explanation on Traffic flow (1) between tenant 2 of service 3 and service 2. The diagram depicts a typical cloud-enabled data-center including the physical switching fabric. Two compute nodes are used to host VMs used by different services (these VMs can offer SaaS or PaaS applications). The compute node also contains some flavor of a virtual switch. The nodes are interconnected via two physical switches forming the fabric. We highlight six points within the virtual and physical network to show how UCC+IID segmentation is designed and used.

Traffic flow 1 originates on VM-1 hosted on Compute Node 1. Tenant 1 of service 1 is trying to access information of service 3.

1) Universal Cloud Classification information are defined on the originating VM 1. Here, the Service-ID is set to service 1 while the Tenant-ID identifies tenant 1. These details are incorporated into the hop-by-hop IPv6 extension header so that all intermediate devices are able to classify traffic accordingly.
2) Every flow leaving a VM incorporates the Interface Identifier that is included in the VMs IP address. This is done using SLAAC.

3) The virtual switch, here OVS, can use the classification information included within the IP header to distinguish traffic flows and authorize flows between source and destination. As OpenStack is aware of MAC address assignments and therefore of Interface Identifiers it can program the virtual switches to permit/deny flows accordingly. Additionally, the Service-ID and Tenant-ID information are used to isolate traffic on a per-service and per-tenant basis
4) Here, (4a) and (4b) are similar but define ingress and egress behaviors on the physical switch. All three classification information are inspected at the ingress port on the physical switch. This enables the switch to define forwarding decisions per-service, tenant and IID.
5) After the stream is forwarded to compute node 2, the virtual switch inspects the header information to gather the IDD, Service-ID and Tenant-ID details. These can be used to authorize traffic between the source and the local destination. In addition, service and/or tenant specific policies can be applied locally before the traffic is send out to the VM.
6) As the service and tenant information are source specific they can be used to verify flow information and authenticity.

The above points demonstrate how the proposed solution realizes network segmentation on a per-service, consumer and endpoint basis.

IV. Conclusion

Cloud Computing is a scalable, flexible and highly-dynamic way providing resources to services and their tenants. One of the key networking requirements in cloud environments is the provisioning of scalable and isolated tenant networks. In todays cloud-enabled data centers segmentation is achieved using either VLANs or overlay technologies such as VxLANs or GRE tunnels. In this paper we highlighted the shortcomings cloud providers face when architecting the scalable and dynamic cloud using VLANs, VxLANs or GRE tunnels.

OpenStack is an open source cloud orchestration solution consisting of several projects. Each project is used to manage, administer and implement tenant resources such as compute, storage or networks. On the network side, OpenStack currently deploys three different reference architectures based on either VLANs, VxLANs or GRE tunnels. These technologies are used to segment tenant networks both within the virtual networks but also on the physical switching fabric.

Based on the listed limitations we identified several design requirements for a novel approach in architecting cloud environments. We argued that a new network design should be (1) solely based on IPv6, (2) shouldn't require any legacy segmentation approaches, (3) should classify and isolate traffic cloud specific, (4) should tackle current limitation and (5) provide equivalent or superior security characteristics.

Based on the authors previous research, Universal Cloud Classification, this paper introduces an IPv6 OpenStack based network architecture and Interface Identifiers. Here, we try to show both the feasibility and usefulness of the Universal Cloud Classification approach while also defining a novel approach to architecting network resources.

This paper shows how Universal Cloud Classification (UCC) can be used in conjunction with Interface Identifiers to create a network architecture that does not rely on other segmentation technologies. UCC introduces three identifiers to isolate traffic according to its cloud provider, service and tenant affiliation. These IDs, incorporated into an IPv6 extension header, tackle limitations of current segmentation technologies.

To conclude, the proposed solution enables (1) visibility within a cloud network down to the consumer level, (2) is based on UCC + IID introducing a classification scheme reflecting the typical cloud hierarchy and (3) by using IPv6 eliminates the need for NAT by assigning globally routable addresses.

References

[1] S. Jeuk, J. Szefer, and S. Zhou, "Towards cloud, service and tenant classification for cloud computing," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, May 2014, pp. 792–801.

[2] S. Jeuk, G. Salgueiro, and S. Zhou, "Universal cloud classification (ucc) and its evaluation in a data center environment," in *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, Dec 2014, pp. 469–474.

[3] ——, "A novel approach to classify cloud entities: Universal cloud classification (ucc)," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, May 2015.

[4] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," RFC 4291 (Draft Standard), Internet Engineering Task Force, Feb. 2006, updated by RFCs 5952, 6052, 7136, 7346, 7371. [Online]. Available: http://www.ietf.org/rfc/rfc4291.txt

[5] (2015 (accessed 18 May, 2015)) Openstack.org. [Online]. Available: https://www.openstack.org/

[6] Oracle. (2012 (accessed 17 May, 2015)) Cloud reference architecture. [Online]. Available: http://www.oracle.com/technetwork/topics/entarch/oracle-wp-cloud-ref-arch-1883533.pdf

[7] (2014 (accessed 18 May, 2015)) Case study: Facebook moving to an ipv6-only internal network. [Online]. Available: http://www.internetsociety.org/deploy360/resources/case-study-facebook-moving-to-an-ipv6-only-internal-network/

[8] A. Tore, "The case for ipv6-only data centers," url, 2012, http://fud.no/talks/20120209-V6_World_Congress_2012-The_Case_for_IPv6_Only_Data_Centres.pdf.

[9] N. Yechiel. (2015 (accessed 15 May, 2015)) What's coming in openstack networking for the kilo release. [Online]. Available: http://redhatstackblog.redhat.com/2015/05/11/whats-coming-in-openstack-networking-for-the-kilo-release/

[10] S. Thomson, T. Narten, and T. Jinmei, "IPv6 Stateless Address Autoconfiguration," RFC 4862 (Draft Standard), Internet Engineering Task Force, Sep. 2007, updated by RFC 7527. [Online]. Available: http://www.ietf.org/rfc/rfc4862.txt

[11] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)," RFC 3315 (Proposed Standard), Internet Engineering Task Force, Jul. 2003, updated by RFCs 4361, 5494, 6221, 6422, 6644, 7083, 7227, 7283. [Online]. Available: http://www.ietf.org/rfc/rfc3315.txt

[12] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," RFC 4861 (Draft Standard), Internet Engineering Task Force, Sep. 2007, updated by RFCs 5942, 6980, 7048, 7527. [Online]. Available: http://www.ietf.org/rfc/rfc4861.txt