# Approaches for IPv4 as a Service

Brian Field Ph.D.
Comcast

*Abstract*

*IPv6 has been gaining traction both as a necessary service provider technology and as an increasing carrier of Internet traffic. Within Comcast, dual-stack has been an important step to help IPv6 deployment mature and grow. As IPv6 traffic grows and as IPv4 traffic diminishes, we would like to avoid the need to make software changes to enable network features, while also avoiding hardware operational costs that accompany the treatment of IPv4 as native transport in our infrastructure. As such, we are looking at ways for our next generation infrastructure to incrementally evolve towards an IPv6 focused infrastructure. We realize we must continue to support IPv4 for the foreseeable future. Our our thinking is to support IPv4 as a Service (IPv4aaS) — meaning we will carry IPv4 traffic over a "lean" IPv6 focused network infrastructure. In this paper, we detail the insights and learnings Comcast acquired in evaluating technologies that enable or support IPv4aaS over a lean core infrastructure. We also describe how these technologies synergize with current Software Defined Network (SDN) methodologies.*

## INTRODUCTION

The unmistakable rise of IPv6, both as a necessary service provider technology and as an increasing carrier of Internet traffic, prompted Comcast to take a leadership position in IPv6 deployment. Comcast operates a 100% dual-stack network, with a significant portion of our cable modem infrastructure strictly using IPv6. As part of this effort, Comcast has developed and shared with the Internet community enhancements to OpenStack that support IPv6 for virtual machine (VM) and related infrastructure.

Within Comcast, dual-stack represents an important step to the deployment, maturation and growth of IPv6. None the less, we are looking at ways for our next generation infrastructure to natively support *only* IPv6.

We believe this direction will result in a number of benefits. One such infrastructure benefit would be a reduced Forward Information Base (FIB) size, because the network will only need to carry IPv6 routes. This translates into much less high-speed memory required on routing line cards, which reduces the cost to build them, and the energy they consume. Less memory on router line cards also translates into more compact form factors, which makes for better port densities. Further, less memory means less heat generation, which lowers overall environmental costs.

If we are able to totally eliminate IPv4 as a native citizen in the network, we can also eliminate IPv4-specific features in the router configs— which, in turn, reduces both our configuration and operational overhead. Further, we could operate router code revisions that only contain IPv6 capabilities— which yields more efficient code, fewer bugs, and fewer unintended operational interactions with our equipment vendors. It follows that a simpler router environment lowers the bar for new vendors to enter the eco-system and makes it more plausible to seamlessly mix and match differing vendor router platforms.

While we certainly see value in making the infrastructure IPv6 only, we can't abandon IPv4 — even as its traffic share decreases. By supporting IPv4 as a Service (IPv4aaS), we can carry IPv4 traffic over our "lean" network infrastructure.

## Lean, IPv6 Focused, & Lean IPv6 Networks

The technologies we present in the subsequent sections represent different capabilities and solve slightly different problems. We define a "lean" network as one with a relatively small FIB table, with a goal of keeping the FIB size small, to reduce capital and facility expenses. A lean network can be dual-stack.

An IPv6-only network is one in which the network infrastructure only supports the IPv6 protocol natively. The goal of this network architecture is to drive towards a IPv6-only feature set; a small FIB is not a specific optimization.

A "lean IPv6 core" is one where the underlaying network has the characteristics of reduced FIB size, and is a single stack network, namely based on IPv6.

Before considering the technology options that enable a lean or IPv6-focused network, we need to consider how much IPv4 traffic currently exists on our network. For instance, if we attempt to move to an IPv6-only network today, how much IPv4 traffic might need to be carried, via an overlay technology?

To determine the traffic load, we used Netflow information processed through Deepfield[1] to show how much traffic was traversing the Comcast Backbone. The amount of traffic observed over a five-day period is show in Figure 1. In the following sections, we have opted to use qualitative terms to characterize the load on our network rather than specific values.

Figure 1 shows that the Comcast backbone carries, at peak, multiple Tb/s of traffic. Even during non-peak hours, the backbone carries traffic a significant IPv4 traffic load. This is a sizeable amount of traffic to carry via an alternative technology were we to attempt to migrate to an IPv6 only network.

Thus, in the near term, it might not be cost effective to transition all of this IPv4 traffic to an overlay technology in order to obtain an IPv6 only infrastructure. However, there is value in understanding the specific traffic load, by IPv4 prefix, as a means to determine if we can incrementally evolve to a lean network core.

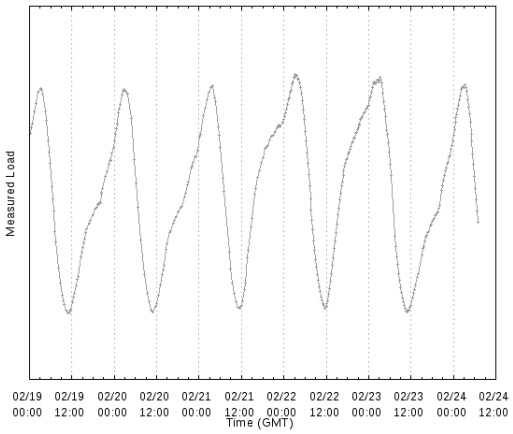


**Figure 1: Measured Load**

## IPv4 FIB and Per Prefix Usage

Our IPv4 Internet routing table contains a mix of Comcast local IPv4 routes (for internal connectivity) and Internet routes (routes received from peers that provide the Internet-wide connectivity). As of Q1 2015, the size of the IPv4 routing table is 575,000 routes. Each route has a "prefix" and a "mask" and consumes a FIB entry.

Operators of ISPs around the world tend to differ in business and operational directions, and those directions tend to influence how an ISPs announces its IPv4 address space to the Internet. ISPs (us amongst them) often have limited means to manage these received announcements, while still maintaining full Internet connectivity.

While full Internet connectivity is a requirement for the backbone, it wasn't clear how much traffic was being delivered to each IPv4 route in the FIB.

Using Deepfield, we performed an analysis on the IPv4 traffic observed on the backbone. Specifically, we configured Deepfield to monitor and report on the amount of traffic observed per IPv4 prefix contained in the routing table. We configured Deepfield to report data on five-minute intervals, meaning the information returned as average over a five-minute window.

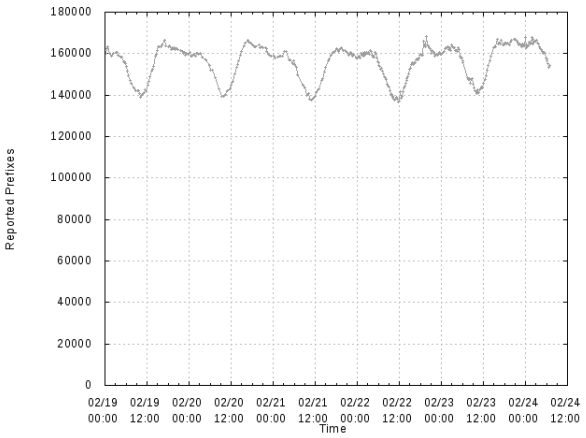


**Figure 2: Observed Prefix count per Sample**

Figure 2 shows the number of prefixes Deepfield reported in each five-minute window, observed during a five-day stretch in February 2015. As it turns out, the number of prefixes varies between 140k (in the very early morning hours in the U.S.) to just over 160k prefixes during the day and into the early peak hours.

Recall that the full Internet routing table is about 575k routes, and at peak, Deepfield sees traffic of only 160k prefixes. This means that Deepfield *doesn't* see traffic for 415k prefixes (or more) in each 5-minute sample.

Part of the reason we might not see traffic to these prefixes is related to our Netflow sampling configuration (1 packet per 8000). However, not seeing traffic to many prefixes (415k, in our case) likely indicates that there is very little traffic to those prefixes in the first place.

The first observation from this data: Nearly three quarters (72%) of the IPv4 prefixes in our FIB have no measureable traffic.

Details on the Prefixes Observed

We continued our analyses by looking at the load reported for the prefixes within one five-minute snapshot. We picked a snapshot near peak load -- specifically, the 3:15 GMT snapshot on 2/20/2015. Figure 3 displays analysis from this five-minute snapshot.

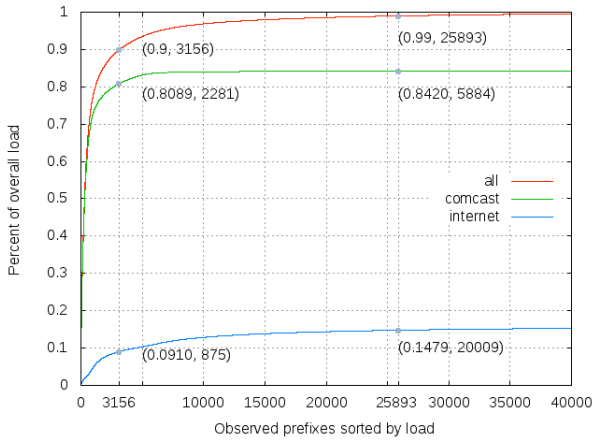


**Figure 3: Cumulative Load Percent**

Specifically, the graph in Figure 3 shows the cumulative load observed from the most loaded prefixes, as a percent of the total overall load.

The first item of note: While the snapshot analysis contained 160,102 prefixes, the graph reports information for the 40k most heavily loaded prefixes (x-axis). The graph doesn't show the remaining 120,102 other prefixes because while there is some load to them, it's insignificant compared to the total load.

A number of additional key points are highlighted on the graph. First, the 3,156 most heavily loaded prefixes account for 90% of the total traffic on our backbone (red line). Recall that the backbone FIB consisted of 575k routes. What this means is that 0.0054% of the IPv4 prefixes carry 90% of the load in

this snapshot. The 3,156 most heavily loaded prefixes are derived from 2,281 Comcast prefixes (green line) and 875 Internet prefixes (blue line).

A second highlight involves the point at which 99% of the traffic is carried. The top 25,893 prefixes carry 99% of the overall total load. These prefixes are made up of 5,884 Comcast prefixes and 20,009 Internet prefixes. What this means is that while our full Internet routes consist of 575k routes, 99% of the IPv4 load is carried via 25,893 prefixes—or 4.5% of the FIB. The inverse of this is that 95.5% of the prefixes in our full route FIB carry just 1% of the load, roughly a few hundreds of Gb/s.

What this analysis suggests is that a very lean FIB architecture is possible today—meaning we could significantly reduce the size of the IPv4 FIB with very little traffic (relative to the overall traffic load) being carried in an overlay IPv4aaS infrastructure.

More details on the relationship between various possible lean FIB sizes and the corresponding load that would be carried in an overlay network are provided in the Figure 4.

Figure 4 shows the amount of traffic that would need to be carried in an overlay IPv4aaS network, if the underlay (physical network) IPv4 FIB was limited to a number of lean sizes. While not specifically shown on the graph, the y-axis maximum value is on the order of a few 100s of Gb/s—so all lines represent loads well below this. Take, for instance, the red line in Figure 4, which corresponds to the IPv4 FIB of 25k prefixes. The graph shows the amount of traffic that would be in the overlay if the IPv4 FIB was limited to 25k prefixes. As we increase the size of the underlay FIB, we can carry more prefixes and hence the load to the overlay network decreases.

In this analysis, we choose which routes to put into the underlay FIB -- and thus what traffic would be carried in the overlay network. To simplify the operation of the overlay network, we opted to pack the underlay FIB with routes in a particular order. The order in which we selected routes to put into the underlay FIB was 1) our routes (highest loaded prefixes, 2) Internet aggregate routes (highest loaded prefixes first), and 3) Internet most specific routes (highest loaded prefixes first).

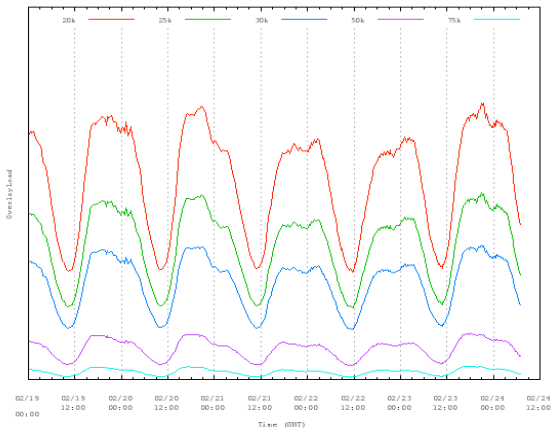


**Figure 4: FIB sizes and Overlay Load**

By putting all Comcast prefixes into the underlay FIB, the overlay network would only contain Internet prefixes and thus would only need to operate in one direction. This simplifies the routing architecture needed to pull traffic into the overlay network. This approach does, however, increase the load on the overlay network, because we service several thousand Comcast prefixes with very little traffic (e.g. customers with static address prefixes.) Internet aggregates were added before more-specifics, as they tend to be catchall announcements that subsume the load of the more-specifics, if the more-specifics disappeared from the upstream peer. This approach results in a higher amount of traffic in the overlay network, but we think it a reasonable tradeoff, because it makes the overlay network architecture simpler.

## Traffic Patterns Enable Thin FIB

The data collected and corresponding analysis strongly suggests that today's IPv4 traffic patterns easily lend themselves to a Thin FIB underlay solution, without requiring excessive resources to build an overlay network. As shown in Figure 4, one could deploy routers with IPv4 FIBs holding well under 100k entries, and the overlay network would only need to carry upwards of a few 100s Gb/s at peak. Note that this overlay load would be sourced from our regional networks, such that any individual regional network would source only a portion of this load.

This means that a collection of cloud infrastructure resources, located within each regional network, should be sufficient to support the overlay infrastructure.

Next, let us examine the different technologies which could be used to create this overlay network.

## IPV4AAS TECHNOLOGY OPTIONS

There are multiple technologies that could be used as the overlay mechanism. In this paper we focus our analysis and work on two technologies, namely MAP and LISP.

### MAP

MAP (Map Address and Port) is an approach that enables a customer to have access to a global IPv4 address, but removes the need for this IPv4 address be natively carried by the underlying infrastructure. (The customer home network is assumed to be dual stack.) The customer home gateway (HG) is assigned an IPv6 address and a provisioned with a set of MAP rules. The combination of the assigned IPv6 address and the MAP rules are used by the HG device to algorithmically derive two things: Its globally (possibly shared) IPv4 address, and the range of ports it can use for its NATP (Network Address and Port) function. Because MAP is used to connect IPv4 islands over an IPv6 infrastructure, the MAP rules also define how to map the IPv4 addresses into IPv6 addresses. Two approaches are defined, one using encapsulation (MAP-E) and a second using translation (MAP-T).

### MAP-E

The key aspect to MAP-E[2] is that traffic between IPv4 islands is fuly encapsulated in an IPv6 header. For traffic to an end-point outside of the MAP domain, packets must be encapsulated and sent to a MAP Border Router (BR), which then de-encapsulates the packet and forwards the IPv4 packet natively. When both IPv4 end-points are within the MAP domain, it is possible for the sending end-point to derive the destination end-point's IPv6 address, based on the mapping rules provided. In this case, traffic need not transit through a BR. It can be sent to the remote end-point directly, where it is de-encapsulated and IPv4 processing occurs. In the current MAP-E Internet draft, the model is that the HG device would have a default route to an identified BR for off-domain traffic.

### MAP-T

In MAP-E, the IPv4 packet (header and payload) is encapsulated within a newly created IPv6 header. This extra level of overhead is avoided in MAP-T[3], because in MAP-T, the IPv6 header is synthesized from the original IPv4 header. The IPv4 payload (UDP, TCP) is then carried directly within the synthesized IPv6 header. In this approach, the IPv4 addresses are embedded into the synthesized IPv6 header. Much like the MAP-E system, for end-points within a domain, traffic can be directly transmitted. For an off-net end-point, the traffic must be sent to a Border Router (BR) device, which performs the de-translation process—meaning it takes the IPv6 header info and derives from it the original IPv4 header. The IPv6 payload

is then attached to the IPv4 header, and the reconstituted IPv4 packet is then forwarded by the BR.

Analysis of MAP

In both MAP approaches, the goal is to eliminate the carriage of IPv4 packets over the underlying infrastructure. A consequence is that it effectively enables a HG based stateless version of a Carrier Grade NAT—meaning an IPv4 address can be shared by many individual customers. The MAP solution is IPv4-specific, meaning its applicability is to carry IPv4 within an IPv6 infrastructure. It can't be extended to non-IPv4 payloads. MAP lends itself to an IPv6-focused network -- but doesn't eliminate the need for a portion of the network infrastructure to be fully IPv4-aware.

In order for a MAP solution to carry the same traffic engineering capabilities as our fully routed IPv4 network today, the BR devices likely need to be interconnected via a IPv4-based infrastructure, with corresponding IPv4 routing mechanisms in place. MAP can eliminate the need for IPv4 routing in the domain covered by MAP --it can enable a lean core in that portion of the network. However, the practicality of lifting the current scale of our IPv4 traffic into a MAP solution would be a serious undertaking. Further, the MAP solution is geared towards a solution where the domain edge is the customer home gateway (HG)—which means the HG device must support the MAP functionality.

LISP

An IPv4 address is both an identity (what an application binds to) and a location (used to announce how to reach an end-point.) The semantics of an address being both an identity and locator contributed to the scaling issues we see with IPv4 today. LISP[4] (Location Identity Separation Protocol) has its roots back in the late 1990s, when an active discussion thread about the "Next-Generation" Internet Protocol led to the development of IPv6. During the discussions leading up to the creation of IPv6, it was suggested that this "Next-Generation" IP address should have new semantics. Specifically, it was suggested that the identity of a device itself did not imply the location of the device.

In the end, IPv6 carried forward the same semantics location-identify semantics of IPv4. However, researchers continued to see value in technologies which decoupled a device's location from its identify. This thinking evolved into a protocol called LISP.

LISP really consists of two functions: an encapsulation protocol and a control plane. The encapsulation mechanism allows the carriage of one address family, inside another address family. LISP could, for instance, encapsulate IPv4 inside IPv4—if one wanted to create a lean IPv4 infrastructure. In this paper, we will examine LISP encapsulating of IPv4 packets inside of IPv6 packets.

The second LISP component is a control plane—a mechanism by which one determines the current, best remote-end point for the outer encapsulation layer address. The default LISP control plane is called DDT (Delegated Database Tree). It's a DNS-like query mechanism, where the LISP encapsulation engine receives a packet, looks at its destination, and then queries the DDT tree to determine the remote end point to send the packet to. The structure of the DDT tree is similar to DNS, in that there's a root node, and then portions of the "name space" are delegated to sub-nodes, with further sub-delegation possible. The LISP encapsulation engine performs the iterative queries through the DDT that determine the best current remote end-point and its IP address. Once this end-point IP address is determined, the packet is encapsulated and sent to the remote end-point. Much like DNS, scaling is

achieved by caching information learned from the DDT tree during the lookups. Like DNS, the cached information has a TTL (time-to-live) value to prevent the encapsulator from having stale information. Besides a TTL value associated with cached entries, the LISP encapsulation mechanism can also perform "probes" to remote end-points, to make sure they're still reachable and functioning correctly.

LISP Analysis

LISP is well suited to enable a lean core, carrying one address family as an overlay to another address family, operating as the underlay. For instance, one could build a IPv6-focused underlay and use the LISP encapsulation and control protocol for an IPv4 overlay. Overlays could also be created for L2 VPNs or L3 VPNs.

One of the downsides to LISP is the query-based control plane, in that the time to detect a remote end-point failure is a function of a cached TTL value, or how often probes are sent. One powerful aspect to the route announcement paradigm (not present with the DDT control plane) is that it is push-based. Changes to the routing system get pushed asynchronously, which makes it well suited in a dynamic environment.

Given LISP's ability to enable a lean IPv6 focused core for both IPv4 and other address families, the remainder of the paper details a LISP-based network overlay solution.

To address the issues with LISP's DDT control plane, we instead considered a routing approach --except this routing model leverages aspects often associated with the SDN. Details of an SDN-powered LISP solution are presented in the following sections.

IPV4AAS

Given the insights from our data analysis, we can incrementally move to a lean IPv6 focused architecture by carrying the IPv4 prefixes with no or little traffic on them via an overlay network. We propose that as a starting point, only the unused or low bandwidth internet prefixes be carried in the overlay network. As such, we detail an overlay network architecture that initially operates only for a subset our Internet traffic.

The overlay architecture can be built on our elastic OpenStack cloud infrastructure, and will require the cloud infrastructure to migrate from national data centers, down into the regional networks, and into the larger hub site locations. This evolution of the cloud eco-system closer to the edge (both hub site edge and peering edge) is being driven by multiple service and business requirements—IPv4aaS is but one of the service drivers for an edge-based cloud infrastructure

Architecture

In order to lift traffic onto this overlay network, we will deploy two or more "Ingress Tunnel Routers" (iTR) per regional network. An iTR is a LISP term for a device that takes native traffic (in our case, IPv4 traffic), performs a look-up on the destination address to determine the remote end-point (in our case an IPv6 address), and then encapsulates and forwards this encapsulated packet. The iTR can operate as VM within the regional network edge cloud infrastructure.

The encapsulated traffic is addressed to an Egress Tunnel Router (eTR). Its responsibility is to de-encapsulate the packet, then forward it, based on the inner IPv4 packet contents. eTRs can be co-located with our peering routers, and can also be VMs operating out of our cloud infrastructure. This high layer architecture is depicted in Figure 5.

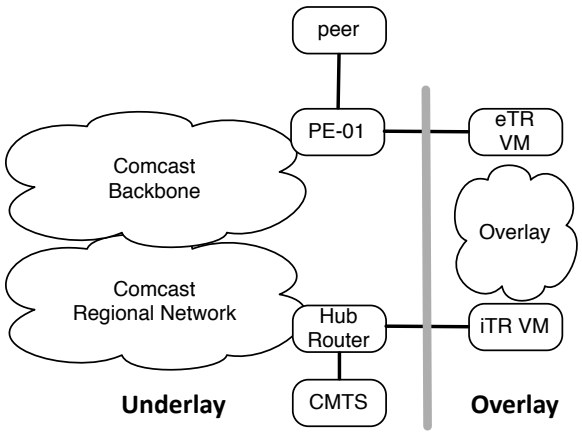


Figure 5: IPv4aaS Logical Architecture

## Routing Architecture

In order to pull traffic into the overlay network, the iTR will need to inject routes into the underlay network. At the time of this writing (winter 2015), we are evaluating a number of routing designs, and present one of the approaches below.

Consider the overlay network as depicted in Figure 6. A peer will send IPv4 prefixes to the Comcast underlay network's peering router PE-01. PE-01 then sends all received routes to the overlay network eTR router. PE-01 also sends a subset of these routes towards the core underlay network's backbone routers. The selected underlay routes are the combination of Comcast prefixes, and the highest loaded Internet prefixes. Let's call these the Thin Fib Routes (TFRs). The TFRs are propagated from the backbone routers into each regional network and down into each hub site router, effectively populating the underlay network's IPv4 FIB.

Netflow data from the underlay network allows us to determine loads on the prefixes in the underlay network. For the overlay network, we are looking at how Openstack's Ceilometer[5] could be used as an alternative paradigm for Netflow-like data collection and back-end data analysis. Further, we are looking at how Openstack's Heat[6] might allow us to adjust iTR or eTR capacity as needed.

These collective systems would continue to monitor load in the underlay and overlay and over time adjust the prefixes in the underlay accordingly.

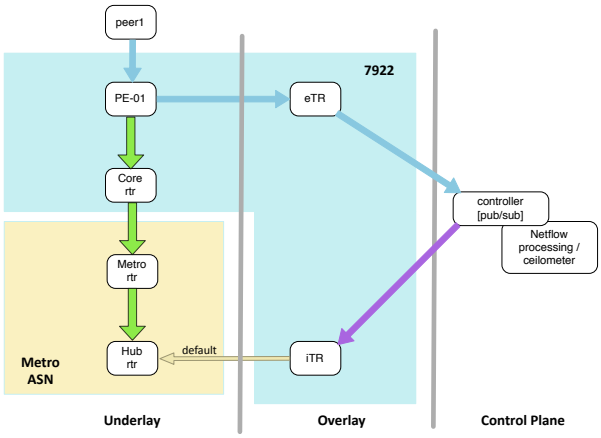


Figure 6: IPv4aaS Routing Architecture

## BGP, JSON and HTTP

The eTR router takes the received BGP[7] (Border Gateway Patrol) announcements and translates them -- from classic BGP protocol format to BGP semantics encoded within a JSON structure. The eTR wraps the BGP JSON announcement information into an encompassing JSON structure that contains information for LISP. Specifically, the encompassing LISP JSON structure provides the RLOC (Routing Locator) which is the IPv6 address of the eTR.

What we are effectively doing is associating a BGP IPv4 prefix announcement with an IPv6 Next Hop. Note that this is not possible with BGP today, but, these semantics are easily encoded via the JSON logic performed on the eTR. The eTR then performs an HTTP PUT of this JSON object to a Pub/Sub entity. One can think of this Pub/Sub logic as being part of an SDN controller. The controller can process the incoming announcements, and adjust the set of announcements sent to an iTR. The iTR then receives the set of JSON

based BGP IPv4 prefixes with an IPv6 Next Hop, and performs classic BGP path selection. For each chosen IPv4 prefix, the iTR LISP forwarding engine is programmed with the target eTR IPv6 RLOC address. The iTR announces a default route, via classic BGP, to the hub router in the regional routing system.

The hub router will see the set of more specific TFRs and a default from the iTR. This way, traffic only gets pulled into the overlay iTR if the packet doesn't match against any routes in the underlay network. We envision the overlay network, and the SDN controller, to be part of the backbone ASN (Autonomous System Number). This enables the backbone operational team to adjust the amount of load which traverses the overlay network based on the routes present in the underlay FIB, and based on business rules applied by the SDN controller on the routes and associated BGP attributes sent to each iTR.

## Prototype Development Status

To date, we have developed a working prototype of a LISP overlay network, including iTR and eTRs coupled with an implementation of the routing control plane architecture described above. This includes the BGP JSON encoding with the associated LISP extensions to enable a IPv4 prefix to have a IPv6 next-hop value.

One issue that we have run into is related to limited Network Function Virtualization (NfV) capabilities in Openstack. Specifically, a key feature needed for this overlay implementation is expected in the Kilo[8] Openstack release slated to be released in April 2015.

## Alternative Encapsulations

While this work has leveraged LISP as the encapsulation technology, other encapsulation technologies could be used including Segment Routing[9] (SR). One could envision that an eTR could associated not just an RLOC with a IPv4 prefix, but also a stack of SR labels (MPLS) or Segment IDs (IPv6) that would be applied by the iTR for a prefix. Note that changes to the routing announcements and associated encapsulation details could also be performed by the SDN controller before sending the update announcement to an iTR.

## Future Work

Future work includes considering how we might use SR and the above routing based push control plane to signal service chaining semantics for specific prefixes and or possibly more specific (prefix and port) announcements.

## Summary

In this paper we have measured the loading of IPv4 prefixes carried in the Comcast full routing table, and determined that a significant number of FIB entries have little or no traffic. We used that information to suggest that a lean, IPv6-focused core network is possible today. We then then described a technology and routing architecture to support an IPv4aaS overlay network. We show that this overlay architecture can be instantiated on an OpenStack elastic cloud system, and can leverage SDN approaches to build both the routing control plane and a real-time measurement system. The measurement system ties classic network performance information (Netflow) to OpenStack SDN systems, including Ceilometer and Heat.

## Acknowledgements

We would like to thank John Leddy, John Brzozowski and Paul Mabey for their thoughts in the many discussions we have had in this subject area. Further, we would like to thank Dino Farinacci for the use of his LISP

implementation and associated software changes to support the described IPv4aaS routing architecture.

---

[1] www.deepfield.net
[2] https://tools.ietf.org/html/draft-ietf-softwire-map-13
[3] https://tools.ietf.org/html/draft-ietf-softwire-map-t-08
[4] https://tools.ietf.org/html/rfc6830
[5] https://wiki.openstack.org/wiki/Ceilometer
[6] https://wiki.openstack.org/wiki/Heat
[7] http://www.ietf.org/rfc/rfc4271.txt
[8] https://wiki.openstack.org/wiki/Kilo_Release_Schedule
[9] http://www.segment-routing.net/home/ietf