Understanding Tradeoffs in Incremental Deployment of New Network Architectures

Matthew K. Mukerjee
Carnegie Mellon University
mukerjee@cs.cmu.edu

Dongsu Han
KAIST
dongsuh@ee.kaist.ac.kr

Srinivasan Seshan
Carnegie Mellon University
srini@cs.cmu.edu

Peter Steenkiste
Carnegie Mellon University
prs@cs.cmu.edu

ABSTRACT

Despite the plethora of incremental deployment mechanisms proposed, rapid adoption of new network-layer protocols and architectures remains difficult as reflected by the widespread lack of IPv6 traffic on the Internet. We show that all deployment mechanisms must address four key questions: How to select an egress from the source network, how to select an ingress into the destination network, how to reach that egress, and how to reach that ingress. By creating a design space that maps all existing mechanisms by how they answer these questions, we identify the lack of existing mechanisms in part of this design space and propose two novel approaches: the "4ID" and the "Smart 4ID". The 4ID mechanism utilizes new data plane technology to flexibly decide when to encapsulate packets at forwarding time. The Smart 4ID mechanism additionally adopts an SDN-style control plane to intelligently pick ingress/egress pairs based on a wider view of the local network. We implement these mechanisms along with two widely used IPv6 deployment mechanisms and conduct wide-area deployment experiments over PlanetLab. We conclude that Smart 4ID provide better overall performance and failure semantics, and that innovations in the data plane and control plane enable straightforward incremental deployment.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.2.2 [**Computer-Communication Networks**]: Network Protocols

Keywords

incremental deployment; network architectures; future Internet architecture

CoNEXT'13, December 9-12, 2013, Santa Barbara, California, USA.

http://dx.doi.org/10.1145/2535372.2535396.

1. INTRODUCTION

Internet Protocol version 4 (IPv4) has been the internetworking protocol for decades. However, as networking technologies advanced, new versions [13], new Internet architectures [17, 33], and new features (multicast [12], QoS [6], etc.) have been introduced to address shortcomings of IPv4. These new designs are not compatible with IPv4, so deployment is challenging. Since a global "flag day" where all machines upgrade simultaneously is not a viable solution, *incremental deployment* mechanisms are required to realize the benefits of these new designs.

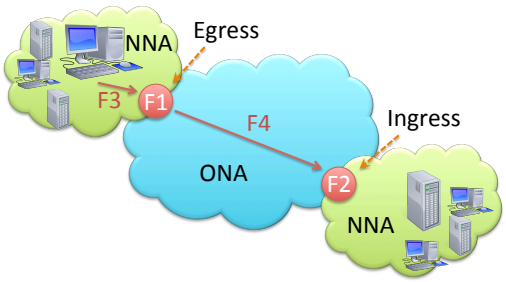


Figure 1: Disjoint NNA Clouds

In this paper, we define incremental deployment of a network-layer protocol as the ability for two hosts using a new network-layer architecture (**NNA**, e.g., IPv6) to be able to communicate with each other before they have an entirely NNA path between them. Figure 1 shows a common scenario where the path between a pair of hosts in a source and destination network consists of an old network architecture (**ONA**, e.g., IPv4). We focus specifically on the incremental deployment of network-layer protocols as their deployment has traditionally been the most difficult due to the "narrow waist"-model of the Internet. This means that the scope of this work is focused solely on changes at the network layer (e.g., IPv6, XIA [17], NDN [33], MobilityFirst [27]), and not higher-layer protocols such as DNSSEC or S-BGP. Although other factors such as policy and economic incentives are significantly important to a protocol's initial deployment, we narrow our scope to the technical aspects of deploying the protocol incrementally, providing the initial steps towards a holistic principled design.

Unfortunately, most new architectures do not consider incremental deployment as their first-order objective [17, 33].

Thus, incrementally deploying these new protocols over the current Internet requires external support mechanisms that are described as part of neither the old nor new protocols. The most widely seen example of this is IPv6's deployment over IPv4, which makes use of a large number of proposed incremental deployment mechanisms, such as Teredo [18], 6to4 [8], and 6rd [14]. However, despite this variety of external deployment mechanisms, IPv6 traffic accounts still for a small fraction of all web traffic (typically less than 0.2% [22]) and the majority of this IPv6 traffic (80% [1]) uses an entirely native path, eschewing incremental deployment entirely. The lack of use of these mechanisms suggests that there may be significant need for further improvement.

To better understand and improve the mechanisms for incremental deployment, we categorize the existing mechanisms into a design space based on how each design answers these four questions:

- How and when to select an egress gateway from the source NNA network
- How and when to select an ingress gateway into the destination NNA network
- How to reach the egress gateway of the source NNA network from the source host
- How to reach the ingress gateway of the NNA destination network from the source NNA network

This problem breakdown allows us to 1) systematically examine the design choices made by existing deployment mechanisms, 2) guide the design of a new deployment mechanism, and 3) easily compare different mechanisms and characterize the tradeoffs. In fact, in studying the design space, we identify that an interesting part is left unexplored. Thus, this paper explores the design space more fully by describing two new classes of mechanisms and generalizing two previously explored classes of mechanisms.

The four (existing and new) classes and their instances are as follows:

1. Static Tunnels: 6in4 [28], AYIYA [26], TSP [4]
2. Address Mapping: 6to4 [8], 6over4 [7], 6rd [14], Teredo [18]
3. Flexible Addressing: 4IDs (new)
4. Smart Control Plane: Smart 4IDs (new)

To characterize the real-world performance tradeoffs of the four classes, we implement a representative mechanism from each class and perform wide-area experimentation over PlanetLab [11]. We explore how the choices made in each class directly impact host performance (path latency, path stretch, and latency seen by applications) as well as failure semantics (failure detection and recovery time) through a quantitative analysis. We additionally provide a qualitative analysis of management and complexity overhead of each mechanism. Path latency and stretch provide insight into the quality of the path chosen by each mechanism, whereas application latency shows the path's impact on hosts. Failure semantics and management/complexity overhead present a fuller picture of the effort needed to use these mechanisms, which is often left out in analysis.

Our results shows that the new Smart 4ID-based approach outperforms previous approaches in performance while simultaneously providing better failure semantics. We contend that the our mechanism performs better because it leverages innovations in the data plane (flexible addressing) and the control plane (centralized local controller) rather than relying solely on traditional ideas (routing, naming, etc).

In summary, this paper makes the following contributions:

1. We improve the understanding of incremental deployment by identifying the key decisions that have to be made and describing the entire design space.
2. We identify two new approaches, 4ID and Smart 4ID. The 4ID approach utilizes flexible addressing in the data plane, and the Smart 4ID approach additionally leverages an intelligent control plane for better performance.
3. We perform qualitative and quantitative comparisons of four representative mechanisms and show that our new Smart 4ID approach outperforms the others.

We explain the design space (in §2) using IPv6 deployment scenarios to provide a familiar environment for the reader and to clearly relate the design to previous work that has focused almost exclusively on IPv6 deployment. However, our techniques and observations apply to the deployment of any NNA. In fact, the implementation and evaluation of our techniques use XIA [1], which is an new network architecture that is radically different from either IPv4 or IPv6.

The rest of this paper is organized as follows. First, we examine the problem we wish to solve and create a design space of possible solutions (§2). Next, we examine where previous approaches fit within that design space, using IPv6 deployment as a case study (§3). Then, we explore the design space and develop two new approaches in (§4). We then systematically evaluate previous designs with our two new approaches over PlanetLab (§5) before concluding (§6).

2. DESIGN SPACE

2.1 Requirements and Deployment Scenarios

Incremental deployment implies that hosts can communicate before the entire path is upgraded. This means that part of the path between the two hosts uses some other ONA [5, 15, 25, 30]. There may be multiple different ONAs in use between the NNA hosts, but for the sake of clarity we assume a single ONA technology. For example, this ONA could be IPv4 and the NNA could be IPv6, XIA [17], or MobilityFirst [27].

Thus deployment mechanisms need to satisfy two requirements: the mechanism needs to provide global reachability for all NNA hosts in disjoint networks using the mechanism, regardless of the characteristics of the path between a pair of NNA hosts (a correctness requirement) and the mechanism should do so without introducing undesirable side effects, like increasing path stretch (a performance requirement).

In order to gain a clear picture of common issues with deployment mechanisms, we define some common scenarios.

Disjoint NNA Clouds: One of the most common incremental deployment scenarios involves two hosts in disjoint

[1] `http://xia.cs.cmu.edu`

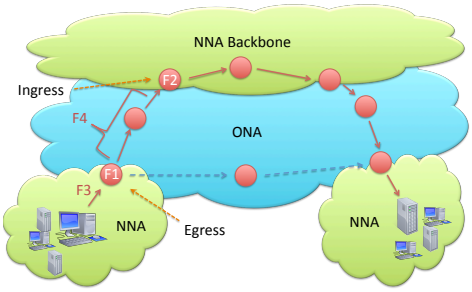


Figure 2: Backbone-based

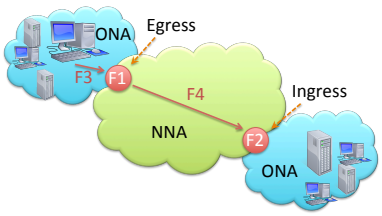


Figure 3: Disjoint ONA Clouds

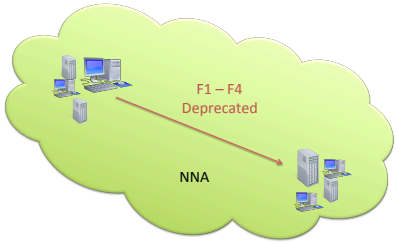


Figure 4: Merged Clouds

NNA "clouds" (consisting of one or more ASes or even a single machine) wishing to communicate over an ONA (See Figure 1).

As shown, the two gateways (an "egress gateway" **F1** and an "ingress gateway" **F2**) translate messages between NNA and ONA formats on behalf of hosts in the NNA networks, as "dual-stack routers" that understand both protocols. The labeled functions **F1 - F4** define specific tasks (egress and ingress selection, and egress and ingress location) and are further explored in § 2.2.

Backbone Topologies: Some deployment mechanisms [4, 26, 28] opt for more logically centralized systems such as backbones (e.g., 6bone for IPv6 or Mbone for multicast), where packets first must travel to a backbone router before reaching their destination (see Figure 2). A better solution would have packets travel directly to their destination (Figure 1), as diverting packets through additional infrastructure impacts performance by increasing latency as well as increases the potential for failures.

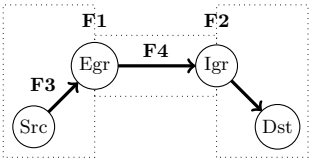
Disjoint ONA Clouds: After an NNA is widely used, the ONA becomes the architecture that has difficulty being supported by the network (Figure 3). This seems like a rather distant scenario but is commonplace in places like China where new IPv4 addresses are already scarce, but IPv6 addresses are plentiful [32]. Although this paper discusses incremental deployment in terms of disjoint NNA clouds, the approaches are general enough to work for disjoint ONA clouds as well.

Merged Clouds: As time passes, a larger percentage of the path between NNA hosts contain NNA routers, until eventually the previously disjoint NNA clouds are now merged (Figure 4). Deployment approaches need to take this into account to prevent unnecessary processing overhead when the path is fully NNA. Many current solutions don't account for this scenario and thus can only be used during the initial stages of deployment and must be manually shut off.

Each scenario provides a different set of challenges to a deployment mechanism over its lifetime. Although Figure 1 is the most common scenario, the rest of the scenarios must also be adequately explored. In particular, we will see how different design choices can affect how deployment mechanisms handle the scenario presented in Figure 4.

2.2 Problem Breakdown

Although the scenarios presented in Figures 1 - 3 initial seem quite different, we note that they all contain the same high-level form:



Where "Src" and "Dst" are the source and destination hosts respectively, and "Egr" and "Igr" are the egress gateway and ingress gateway respectively. Although there are many properties we would like a deployment mechanism to preserve (e.g., security of a packet, the user's intent, etc.), at the most fundamental level, a deployment mechanism ultimately requires locating an ingress and egress and providing a means of reaching both. We focus specifically on reachability, providing explicit functions that must be satisfied:

F1 ***Select an egress*** *router for packets leaving the source network.*

F2 ***Select an ingress*** *router for packets entering the destination network.*

F3 ***Reach the selected egress*** *router within the source network.*

F4 ***Reach the selected ingress*** *router within the destination network.*

These functions are general enough to be applied to any arbitrary NNAs being deployed over any arbitrary ONA that does not provide any built-in support for incremental deployment, such as IPv4.

Two important aspects determine the high-level design of an incremental deployment mechanism: 1) *where* these functions are implemented and 2) *when* they are answered.

Where: These functions can be implemented either in the control plane or in the data plane. Within each category, there are multiple solutions for implementing the functions.

- **Control Plane:** Hard state at Gateway (**F2**, **F4**), (Dynamic) Routing (**F1**, **F3**), Chosen by Local Controller (**F1**, **F2**, **F3**)
- **Packets (Data Plane):** Packet carries egress and/or ingress identifier (**F3**, **F4**)

When: Gateway selection process can be either *early* or *late* in the forwarding process.

- **Early Binding:** Gateway Selection happens explicitly as part of packet creation (**F1**, **F2**)
- **Late Binding:** Gateway Selection happens implicitly during forwarding (**F1**, **F2**)

	Egress Selection	Ingress Selection	Egress Location	Ingress Location	Problems
Old Approaches:					
Static Tunneling	Fixed	Fixed	Routing	Hard-state at Egress	Hard-state (**Flex.**); Manual Setup (**Manage.**); Path Stretch (**Perform.**)
Address Mapping	None	Naming (Encoded in Identifier)	Routing	Per-packet via Addresses	Node Bound to own Ingress (**Flex., Perform.**); Must update DNS (**Manage.**)
New Approaches: **Flexible Addressing**	None	Naming (Encoded Separately)	Routing	Per-packet via Fallbacks	ONA Route Tables (**Perform., Manage.**); Source Selects Ingress (**Perform.**)
Smart Control Plane	Chosen by Local Controller	Naming (Encoded Separately)	Per-packet via Fallbacks	Per-packet via Fallbacks	Early Binding to Egress (**Flex.**)

Table 1: Comparing Approaches for Incremental Deployment

We believe that all previous deployment approaches can be described by this model and that many schemes that implement these functions identically are not meaningfully different. We will show that this design space includes additional designs that have not been used in practice. Furthermore, we believe that all future deployment mechanisms must provide ways to implement these functions and, thus, also fall within this design space.

2.3 Evaluation Criteria

In order to provide fair comparison between deployment approaches we define criteria with which to evaluate them. Although the most important criteria for a deployment mechanism is satisfying reachability, we assume that all designs by definition must handle this. Thus, we instead focus on raw *performance*, *flexibility*, and *management* overhead. In the context of deployment mechanisms, raw performance would include additional incurred latency and path stretch. Flexibility is the ability for the approach to adapt to changes in topology over time (as illustrated in our deployment scenarios) as well as failures. Management is how much human interaction is involved during setup, failures, and maintenance. We examine each approach we present based on this criteria.

3. PREVIOUS APPROACHES

In order to better understand the impact of the design choices, we take deployment of IPv6 (NNA) over IPv4 (ONA) as a case study, as it is the most widely studied network architecture deployment. We compile the results into Table 1. Although most previous work in this area has focused on IPv6 deployment over IPv4, the approaches we present are general in nature and can be applied to arbitrary NNAs. In our evaluation of these approaches (§5) we implement these approaches using XIA [17], an architecture radically different from IP, as the NNA.

While the IPv6 standard itself is fairly stable, there are numerous (and varying) methods for deployment over the existing IPv4 Internet [4, 7, 8, 18, 26, 28]. Broadly put, these mechanisms fall into two major categories: static tunneling-based approaches and address mapping-based approaches. We evaluate these these two categories, explaining how their design decisions directly affect their raw performance, flexibility, and management overhead.

3.1 Static Tunnels

Static tunneling approaches provide direct connections between specific disjoint network clouds by explicitly keeping state at specific edge gateways to form a "static tunnel" between the two clouds. Creating static tunnels between all clouds that wish to communicate is not scalable; thus, these approaches often comes with an additional assumption – a backbone topology model. Internetworking has a long history with using backbone topologies for emerging technologies, e.g., MBone [29] or 6bone[2]. Having a backbone allows gateway routers to keep state for one static tunnel regardless of the number of disjoint clouds, as opposed to one static tunnel per disjoint cloud.

Examples of static tunneling mechanisms include 6in4 [28], AYIYA [26], and TSP [4]. The differences between them are mostly manual (6in4) versus automatic (TSP) tunnel setup and whether IPv6 can be encapsulated in transport layer protocols (AYIYA). Static tunneling approaches are not as widely used as they once were given that 6bone was decommissioned in 2006[3], however they are the "traditional" approach to IPv6 deployment, and, thus, provide a good point of comparison.

3.1.1 Design Decisions

F1 : The egress is a single gateway holding tunnel state.

F2 : The ingress is a single gateway on the other side of the tunnel also holding tunnel state.

F3 : The egress is reached by routing; all packets not destined for hosts in the local network are drawn towards the egress gateway due to routing.

F4 : The ingress is reached by encapsulating packets based on a destination address stored within the egress gateway as tunnel state.

3.1.2 Tradeoffs

The major benefit of this design is simplicity, but it comes at the cost of additional overhead. Getting onto the network is as simple as tunneling to a backbone node, as the backbone facilitates communication between hosts.

Performance: Performance can be a major issue if a backbone node is not geographically close to the source or destination, causing unnecessarily high path latency/stretch and therefore application latency.

[2] `http://www.gogo6.com/page/6bone`
[3] `http://www.gogo6.com/page/6bone`

Flexibility: If a single tunnel entry point goes down, an entire network can lose external connectivity. At this point, the tunnel needs to be reestablished by reconfiguring router state.
Management: Static tunneling in the IPv6 world generally requires users (be they individuals or network operators) to be directly involved in a tunnel setup process. Typically this process involves contacting a "tunnel broker" in order to establish an endpoint for the other side of the tunnel. Tunnel brokers generally have direct access to the backbone of the Internet itself via connections to Points of Presence (PoPs) around the world (SixXS[4], Hurricane Electric[5], etc.). After failures, tunnel reestablishment is possibly manual.

3.2 Address Mapping

Address mapping approaches cover many of today's most widely used IPv6 deployment mechanisms. All of these mechanisms share the property that each host encodes an IPv4 address of an ingress router, within their IPv6 address. This provides a simple way for each packet to carry an ingress address, which can be used to do per-packet encapsulation at the egress router. This approach directly contrasts with the hard-state stored at routers in the tunneling approach as each packet now contains the "state" needed to forward to the ingress.

Examples of address mapping schemes include 6to4 [8], 6over4 [7], 6rd [14], and Teredo [18]. 6rd and/or 6to4 are the most widely used transition mechanisms [1] as they tend to be the most flexible; however, Teredo holds the majority of registered IPv6 *addresses* on the Internet [20]. Additionally, 6to4 support is included in modern versions of both Windows and Mac OS X; whereas, Teredo is only included in Windows. The schemes themselves differ by focusing more on NATs (Teredo) or using general IPv6 prefixes (6rd).

We take 6to4 as an example implementation, to highlight how these mechanisms work. All 6to4 hosts have IPv6 addresses starting with the 2002:: prefix. The next four bytes in their IPv6 address encodes the IPv4 address of the ingress router they wish to be reached from. For example, a host with the address prefix 2002:C000:0101::/48 wishes to be reached through an ingress gateway with IPv4 address 192.0.1.1 (i.e. C000:0101).

3.2.1 Design Decisions

F1 : The egress is not selected, but gateways advertise prefixes, and, thus, get selected during routing.

F2 : The ingress is encoded within the destination address as given by name lookup.

F3 : The egress is reached by relying entirely on routing for the given destination address prefix.

F4 : The ingress is reached by encapsulating individual packets on the fly based on the IPv4 address stored in destination IPv6 address within the packet.

3.2.2 Tradeoffs

This approach is very clean: receivers get to decide directly what IPv4 address they prefer to be reached from and publish this information to DNS. However, having hosts bind to an IPv6 address that encodes an IPv4 address within it causes numerous issues.
Performance: Binding an endhost to a specific single ingress address can hurt performance depending on the location of the source relative to the destination in the network. A better solution would allow for the ingress to change dynamically depending on the location of the source.
Flexibility: Conflating the ingress IPv4 address and destination IPv6 addresses causes issues as topologies change over time, in addition to needing to rebind after an ingress failure. However, having each packet carry the information it needs to reach the destination makes the approach more flexible than static tunneling.

6to4 and Teredo egress gateways always provide default routes for their respective traffic. Traffic always goes to the egress closest to the source. As clouds merge over time (as explained in § 2.1), these gateways will continue to pull packets destined for endhosts that previously were in disjoint clouds out into the IPv4 network, without need. 6rd addresses this somewhat as routes can be changed by ISPs, but this provides unnecessary management overhead. A proper solution would be to provide flexibility to choose between using IPv4 addresses and IPv6 addresses during forwarding.
Management: if the specific ingress that a host is using goes down, then the host first needs to pick a new gateway, then change its IPv6 address to encode that gateway (and thus break all current connections), and finally update DNS to point its new IPv6 address and wait for DNS cached records to expire.

3.3 Native IPv6

Although not a deployment mechanism, Native IPv6 does account for the majority of IPv6 traffic received by large-scale content distribution networks [1]. Native IPv6 requires an end-to-end IPv6 path from source to destination. Historically, few ISPs offer such service, but this has been changing recently. Both Comcast in the US and KDDI's fiber service in Japan have offered native IPv6 support alongside their normal IPv4 support, giving all customers the option of native dual-stack support. There are two big conclusions to draw from this: ISPs are introducing native IPv6 support, implying that more and more hosts are IPv6 enabled, and that transition technologies are not being used in practice when destinations (such as content providers) support native IPv4 as well.

4. MECHANISM DESIGN

As we've seen there are a wide variety of ways to perform the four functions from § 2.2. If we were to think of the optimal deployment mechanism, we would like it to be efficient, in that it would always pick the optimal egress/ingress pair in a dynamic fashion; thus, avoiding potential failures and handling all scenarios in § 2.1 elegantly. This mirrors our previous discussion of the need for both correctness and efficiency in a given mechanism. The mechanism should also be general enough to apply to arbitrary new network architectures (e.g., IPv6, XIA [17], MobilityFirst [27]). We explore how recent research in network architecture can enable new deployment mechanisms before pinning down two distinct new approaches: **Flexible Addressing** and **Smart Control Plane**.

[4] `http://www.sixxs.net/`
[5] `http://www.he.net/`

Option Number	Egress in Addressing	Ingress in Addressing	Provider	Pros	Cons
1	None	ONA Address	Naming	Late Binding on Egress	Non-optimal paths; ONA Route Tables in NNA
2	NNA Address	ONA Address	Local Controller; Naming	Optimal paths; Scalable	Early Binding on Egress and Ingress
3	NNA Address	ONA Prefix	Local Controller; Naming	No Early Binding	Have to send to ONA prefix
4	NNA Address	Name	Local Controller	No Early Binding	High-speed name lookup
5	NNA Address	None	Local Controller	Scoped Tunneling	Not Scalable

Table 2: Unexplored Options in the Design Space. An example deployment would be IPv6 as the New Network Architecture (NNA) and IPv4 as the Old Network Architecture (ONA).

4.1 Exploring the Design Space

Of the previous approaches (§ 3), we note that static tunneling approaches lack flexibility in picking egress/ingress pairs and introduce fate-sharing issues due to the hard state at these gateways (see Table 1). Address mapping approaches work around this by providing dynamic per-packet encapsulation, but introduce new failure semantics by binding node addresses to their ingress' address. Address mapping schemes are considered stopgap solutions and, thus, can not deal with continued use over the lifetime of NNA's deployment (see § 2.1 and Figure 4). Both approaches are similar in what basic concepts they leverage in their mechanisms: routing, naming, and addressing. We contend that recently introduced concepts in the network architecture community can provide a much clearer path to incremental deployment. We explain three concepts: separation of addresses in packets, fallback-based forwarding, and centralized control, before showing how they can be used to enable new deployment mechanisms.

4.1.1 Introducing Modern Concepts

Separation of Addresses: While address mapping approaches provide better performance, they fundamentally can not provide optimal path selection in all scenarios because they conflation of ingress and destination addresses. This makes it difficult to update the ingress and destination independently. If the architecture allows for a way to have **separate addresses** within a packet [17, 27], then many of the shortcomings of address mapping schemes can be fixed. We could use these distinct identifiers to independently encode an ONA address for the ingress into the destination cloud (a sort of locator), while separately encoding the NNA address for the destination (an identifier), both within a packet header. Having separate addresses in a packet allows for simple switching of the ingress addresses after failure, without needing the destination to rebind to a new NNA address. In addition, separating these addresses provides a clear division both semantically and in terms of management and troubleshooting.

IPv6 can not support separate ingress addresses in a straight-forward way, however NNAs like MobilityFirst [27] provide space in a packet header to store both an endhost identifier (i.e., the destination NNA address) as well as a endhost locator, which we could overload as an ingress ONA address in a fairly straight-forward way. XIA [17] provides further support by allowing alternate means of communication (i.e., ingress ONA addresses) to be added to the network at any point in time, seamlessly. Both architectures can support this simple switching of ingress addresses, but XIA in particular makes the process very clean.

Fallbacks in Forwarding: A further shortcoming of address mapping schemes is their inability to handle changes to the network over long periods of time (see § 2.1 and Figure 4). In order to adequately handle merging of networks over time, the ability for each router to choose between forwarding based on the ingress ONA address or the destination NNA address is crucial. In its most straightforward form, we can easily handle merging of NNA clouds by having all routers attempt to forward based on the destination NNA address unless they don't have an entry in their table for it, in which case they "**fallback**" to the ONA and encapsulate the packet using the ingress ONA address and continue to forward it. IPv6 and NNAs like NDN [33] and MobilityFirst [27] do not have this capability built directly into the network-layer. XIA [17] does allow for fallbacks, using its flexible DAG-based addressing scheme.

Centralized Local Control: A large thrust in the network architecture community has been towards the centralized control of networks, referred to as "Software Defined Networks" [9, 16, 21]. Centralization can be very useful to incremental deployment as well as centralization of the local network graph at a local controller within a source NNA cloud can allow the local controller to pick better ingress/egress pairs for a given source/destination pair. Having a much wider view of the network when compared a purely distributed approach, it is much easier for a local controller to provide better selections, but we leave the details of this process to other works in this area [2, 3].

4.1.2 Exploring the Options

Having both separate addresses and fallback-based routing allows us to explore the design space more fully. What would a mechanism look like if we included both the ingress and egress addresses in packets? Could we include something other than an ingress address in a packet (e.g., a prefix or a name)? We explore five such options (see Table 2) below and examine the first two in more detail in the following subsections.

Option **(1).** relies on routing for egress selection and naming for ingress selection. Providing the *optimal* ingress for an arbitrary source-destination pair would be difficult for naming to do in a scalable fashion. A reasonable solution would be to have the name server return multiple potential ONA ingress addresses for a given name (as DNS does today) and allow the host to choose one. The chosen ingress

ONA address(es) are encoded into the packet as a separate address, which is a form of early binding.

Given the destination address, this option selects the egress location via routing, similar to address mapping. However, this scheme provides a better egress gateway selection by dynamically selecting the egress based on the chosen ONA ingress address. This is done by mirroring ONA route announcements into the source NNA cloud. This allows for automatic selection of the best egress address for the specific ONA ingress address, rather than a "one-size-fits-all" solution found in most address mapping schemes. However, this requires all NNA routers in the source cloud to maintain an ONA route table, adding additional complexity.

Option **(2).** still uses naming to obtain ingress, but uses a centralized local controller (similar to an SDN) to find the best egress. This removes the major downfall of **(1.)** as it eliminates the need for ONA routing tables in all routers. The local controller in the source NNA network selects and routes to the best egress depending on the ingress ONA address chosen. This can be done by either monitoring the external route advertisements or directly probing the external ONA path from all possible egress points. Thus, only gateway routers need to be aware of the ONA protocol. This provides much better performance, but weakens our failure recovery model as it causes early binding to an egress address for individual packets. However, it is possible for a local controller to continually scan for failures in the network and subsequently inform endhosts to change which egress they use after an egress failure.

Option **(3). and (4).** additionally removes the early binding issue of Option **(2)**. However, **(3).** requires being able to send to a packet to an ONA network prefix, which may not be possible in the given ONA. **(4).** requires either fast name lookup during forwarding or name lookup caching, which would be very difficult for arbitrary-length names.

Option **(5).** is markedly different in that it would require egress routers to be able to forward to an ingress without having any information about it in the data plane. Thus, this information would essentially come from state on the router, like in the static tunneling approach. The major difference here is that providing an egress address scopes the tunnel to a particular path configured at that egress.

4.2 Pinning Down New Mechanisms

From these five options, we believe the first two provide the best tradeoffs. The first option (which we refer to as **Flexible Addressing**) uses address separation and fallbacks, to provide an approach similar to address mapping schemes that removes the poor failure semantics. The second option (which we refer to as **Smart Control Plane**) improves on the first by providing optimal paths (thus better performance) through centralization of the control plane. We devise an example mechanism in each category and compare them to the previous approaches in § 3.

4.2.1 Flexible Addressing

As explained in § 4.1.2, flexible addressing is similar to address mapping schemes, but differs in that it stores the ingress ONA (e.g., IPv4) address as a separate address in the packet, in addition to using fallbacks during forwarding.

We build a specific mechanism in this space, the **4ID**, that retrieves a set of possible ONA ingress addresses from the naming server, picks one, encodes it within a packet and then sends packets out, using routing to locate a proper egress. As previously explained, to find the proper egress all routers need to maintain an additional ONA route table, forcing the network to be explicitly linked to the ONA at a global scale, an obvious design issue.

Performance: Having individual endhosts decide which ingress to use for a given destination proves difficult, as the endhosts have a limited view of the network. Thus this selection can lead to poor performance.

Flexibility: Separating the addresses allows for simple changing of the ingress address without needing to change the destination NNA (e.g., IPv6) address. Additionally, fallbacks handle all deployment scenarios in § 2.1, providing graceful deprecation of the mechanism. Having all the data needed in each individual packet provides flexibility.

Management: In order to provide proper egress selection, ONA route tables need to be stored on each NNA router, providing additional complexity in management.

4.2.2 Smart Control Plane

Smart control plane style approaches build off of flexible addressing approaches, but further improve upon them by removing the need for ONA (e.g., IPv4) routing tables on routers in the NNA (e.g., IPv6) source cloud, by using centralized local control. Our specific mechanism in this space, **Smart 4IDs**, works by having a logically centralized local controller make informed decisions as to which egress/ingress pair should be used between a given source/destination pair.

Performance: As a local controller has a much wider view of the network, it is much easier for it to provide better selections of egress/ingress pairs, but we leave the details of this process to other works in this area [2, 3].

Flexibility: As this approach builds off of flexible addressing, we maintain all the benefits (being able to change addresses after failures, handling all deployment scenarios) through use of separate addresses, fallbacks, and data being stored in-packet. However, this approach loses some flexibility due to binding each individual packet to an egress.

Management: Having to maintain a local controller provides some overhead, however it is centralized.

4.3 Discussion

Differentiating from Previous Approaches: We have shown that our example mechanisms, 4IDs and Smart 4IDs can provide optimal path selection while simultaneously gaining better failure semantics when compared to previous approaches. The core of these mechanisms lie in their use of recent research in data plane technology (Using sets of addresses in packets and fallbacks during forwarding) and control plane technology (logically centralized local controllers).

These technologies are incredibly powerful in that they represent a fundamental change when compared to previous mechanisms. Previous mechanisms rely entirely upon time-tested familiar concepts (routing, naming, hard-state, etc). As we've seen, all of these mechanisms have major drawbacks. Thus, we propose that new concepts (i.e., using sets of addresses, fallbacks, and local controllers) are much better suited to support incremental deployment. We contend that *any* network architectures that can support these new technologies can also easily provide straightforward incremental deployment. In §5 we show that levering these new

	Performance		Flexibility		Management	
Static Tunneling	X	Stretch from Backbone	X	Hard State at Egress	X	Manual Setup
Address Mapping	X	Bound to Ingress	X	Bound to Ingress	X	Updates require DNS Propagation
			✓	Per-packet Data		
Flexible Addressing	X	ONA Routing Table	✓	Separate Addresses	X	ONA Routing Table
	X	Source Selects Ingress	✓	Fallback-based Forwarding		
			✓	Per-packet Data		
Smart Control Plane	✓	Local Controller	✓	Separate Addresses	-	Local Controller
			✓	Fallback-based Forwarding		
			✓	Per-packet Data		
			X	Early Binding to Egress		

Table 3: How the Approaches Satisfy the Evaluation Criteria

technologies in the data plane and the control plane provide much better performance and failure semantics.

Multiple Technologies: Our smart 4ID mechanism has the added benefit that the only nodes within the network that need to understand *anything* about the ONA are the gateway routers (which would need to interoperate with the ONA anyways) and the local controller. Thus, the rest of the NNA network can evolve completely independently of the core network "fabric", allowing for greater changes in diversity and functionality within the network as explained in Fabric [10]. If we take XIA [17] as an example ONA and IPv4 as an example NNA, we would be effectively building is a layered version of the Fabric model: XIA would be the "edge" with an IPv4 "fabric", where IPv4 acts as an "edge" with an MPLS "fabric" at its core. Thus, any upgrades to IPv4 network (say to IPv6, multicast, DTN, or new architectures like SCION [34]) can be done independently of XIA, only requiring necessary upgrades at the local controller and the gateway routers that interface with the newly upgraded network, thus allowing multiple technologies to be in use. Smart 4IDs however do not cover a scenario where a source NNA cloud (e.g., XIA) is connected to multiple different ONA technologies (e.g., IPv4 and IPv6) directly. While possible with a truly flexible addressing scheme, such as XIA's DAG based addressing [17], we leave the analysis of such a mechanism to future work.

Fault Tolerance vs Throughput: If we assume that an architecture has enough flexibility to encode multiple potential ONA ingress addresses within a single packet (for example XIA [17]), then this provides an interesting tradeoff between fault tolerance and throughput. Allowing multiple ingresses addresses within a packet (to be used in the event of a failure of the first ingress) allows for very strong resilience to failures, but the overhead of including these additional addresses in *every* packet can greatly lower throughput, especially as the number of additionally included addresses gets close to the MTU size. Conversely, an architecture could allow for the endhost to choose to not include an ingress address at all (lowering the total bytes in the header), allowing for increased throughput at the cost of resilience to failures. This flexibility of an endhost to choose between fault tolerance and throughput at a per-packet level warrants further evaluation, which we leave to future work.

Other Deployment Concerns: Although we focus specifically on the technological concerns of incremental deployment of new network-layer architectures, there are other significant concerns that prevent rapid adoption. Both the economic model [24] an NNA has, as well as its ability to enforce policy [31] are rapidly becoming first-order concerns when designing NNAs. However, even the NNA with the most complicated policy handling and economic incentives must still have the basic ability to provide reachability from some source to some destination. Thus, we contend that the mechanisms we discuss are general enough to apply to these NNAs as well, but we leave the analysis of and integration with such networks to future work.

5. EVALUATION

To evaluate the different general approaches we've presented (Static Tunneling, Address Mapping, Flexible Addressing, and Smart Control Plane) in Table 1, we pick four specific mechanisms that we think are representative of each of the four categories: 6in4 [28], 6rd [14], 4IDs, and Smart 4IDs. We quantitatively and qualitatively compare the four mechanisms in terms of their performance (path latency, path stretch, latency presented to applications), their flexibility (failure detection and recover time), and their management/complexity overhead, using PlanetLab [11], a large-scale global test bed. We wish to find a mechanism that provides optimal path selection (i.e. low path latency, path stretch, and application latency) that simultaneously providing good failure semantics and low management/complexity overhead. We see that Smart 4IDs are the clear winner in terms of their performance and failure semantics, while providing fairly low management overhead. We summarize the qualitative portion of our results in Table 3.

5.1 Topology Setup and Methodology

We conduct our experiments using approximately 200 PlanetLab nodes across the United States. For each metric, we simulate approximately 100 deployments over each of the four mechanisms. As XIA [17] provides both the support for distinct addresses as well as network fallbacks within its flexible DAG addressing scheme, we choose to implement all 4 mechanisms on top of it, for simplicity of comparison.

For the 6in4 (static tunneling) approach, we create a backbone similar to the Abiline backbone [19]. We create the topology seen in Figure 5, using the red nodes. We construct a static tunnel between each pair of connected nodes in the backbone using a UDP socket. From the backbone hosts' perspective, they are connected via a single hop, essentially forming an overlay. This is reasonable, as backbone nodes would typically be connected by long, direct links to their immediate neighbors.

We then randomly select two hosts from the remaining PlanetLab nodes and connect each to their respectively closest backbone node, based on latency. We can see an example

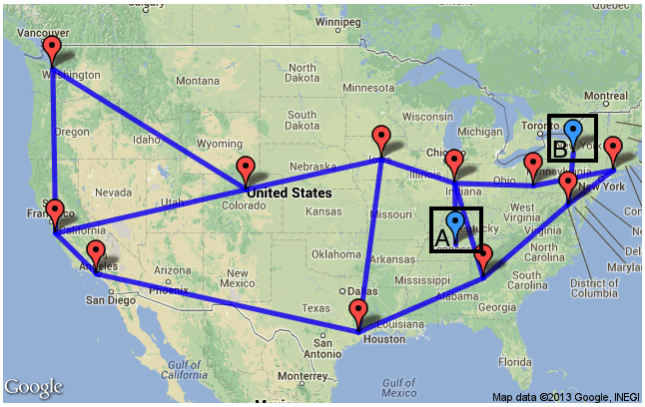


Figure 5: An Example Topology used during 6in4 Experiments

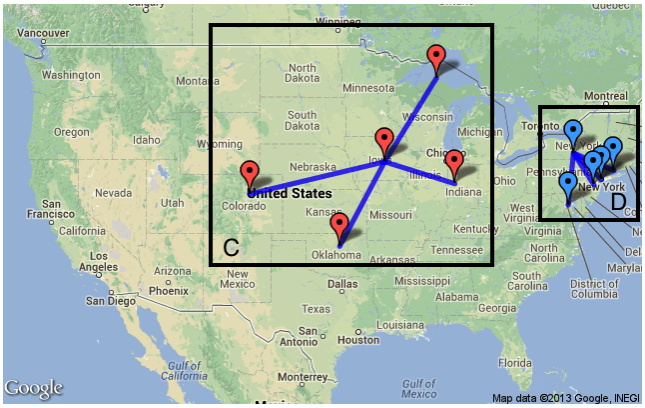


Figure 6: An Example Topology used during 6rd/4ID/Smart 4ID Experiments

of this with the blue nodes in Figure 5, labeled "A" and "B". This simulates a typical 6in4 scenario as these two nodes can now talk to each other through the backbone.

For the other three mechanisms (6rd, 4IDs, and Smart 4IDs), we start with a different topology, as they do not utilize a backbone. For these three scenarios, we pick 5 specific nodes as a constant network deployment and then test by constructing a second network of 5 random nodes, and gather statistics based on the three different mechanisms. This models a single NNA cloud (the control network) wishing to communicate with hosts in a variety of different NNA clouds (the test networks) through the broader ONA Internet. Although the nodes in the test network are random, they maintain the constraint that they are less than 15 ms away from the central node under test.

Our constant network deployment are the red nodes in Figure 6, labeled "C". We choose a star topology as our interest is gateway selection rather than local network configuration. Our randomly selected network (seen in blue in Figure 6 and labeled "D") also utilizes a star topology for the same reason. The nodes at the center of each star are the source and destination nodes being evaluated.

We evaluate the three remaining mechanisms in this topology as follows: as 6rd embeds an ONA ingress (IPv4) address within an NNA (IPv6) address, we model this by having all test networks use the same ingress into the control

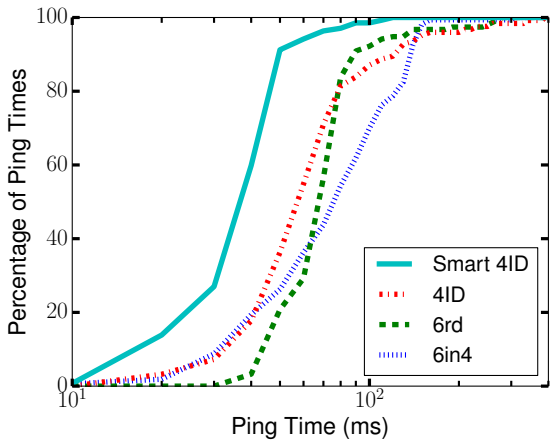


Figure 7: Latency using Different Deployment Mechanisms

network (located in Denver), regardless of location. This models 6rd as a host's NNA (IPv6) address encodes a specific ingress' ONA (IPv4) address that is used in each packet.

For our 4ID approach, the global name server returns a list of ingresses, of which we pick from with equal probability, similar to anycast.

Finally, our Smart 4ID approach can make use of a centralized intelligent control plane to find optimal paths, so we model this in our evaluation by having the local controller in each domain do a pair-wise ping between all possible gateways to select the gateways from both networks that are closest to each other.

5.2 Microbenchmarks

We conduct microbenchmarks over each experimental deployment for each of the four mechanisms by measuring the latency and hop count. For latency we measure the end-to-end path latency by taking the average over 50 pings between the source and destination hosts under test. We drop the 10% of data with the highest latency to remove transient outliers from our data due to errors such as scheduling artifacts.

To measure hop count, in the 6in4 (backbone) case, we count the number of IPv4 hops between the test nodes and their closest backbone node and then add the number of hops between nodes within the backbone. The number of hops within the backbone will be low, as we consider each backbone node to be directly connected to each of its neighbors. For the other three mechanisms, we measure the number of hops between the node under test and its actual gateway ISP router (by finding the first router in traceroute not of the same domain), when conducting a traceroute to its selected egress. We add this to the actual hop count between the selected ingress / egress gateway pair. We do this to provide a realistic estimate of the number of hops to reach the egress router within the source network.

We can see in the results for our latency experiments (Figure 7) that the data is roughly as we expect: 6in4 (backbone) performs worse than the more direct approaches, and smart 4IDs perform the best. We see that on average, Smart 4ID outperforms 6in4 by halving latency, and is very close to halving 6rd's latency as well. Interestingly enough, 4IDs perform very similarly to 6rd. This is due to 4IDs requiring

	Local Failures	Remote Failures
6in4	New Tunnel Setup (seconds)	New Tunnel Setup (seconds) + BGP Propagation Time (minutes)
6rd	BGP Propagation Time (minutes)	DNS Update Propagation Time (hours)
4IDs	BGP Propagation Time (minutes)	Select new Ingress (instant)
Smart 4IDs	Select new Egress (instant)	Select new Ingress (instant)

Table 4: Local and Remote Failure Recovery Times

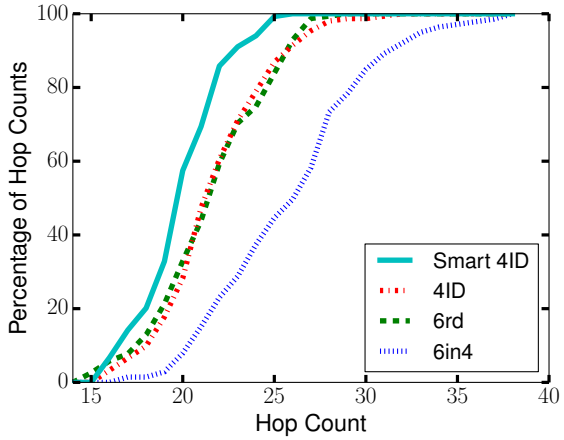


Figure 8: Path Stretch using Different Deployment Mechanisms

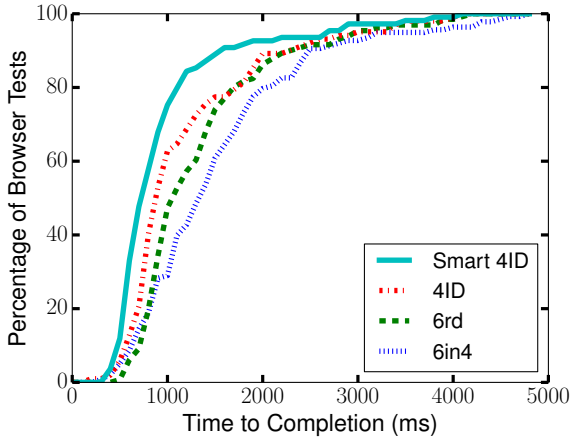


Figure 9: Webpage Latency using Different Deployment Mechanisms

an endhost to select which ingress to use rather than always selecting the same ingress (as with 6rd). In our evaluation, endhosts pick from the possible ingresses at random in 4ID experiments, thus we expect the average over many trials to be similar to 6rd for random deployments.

This decrease in latency seen in smart 4IDs when compared to other approaches directly impacts application performance but also points to the fact that packets are spending an unnecessarily long time in the network in other approaches, implying wasted network capacity and management overhead.

We see similar results for hop count (Figure 8). On average the path stretch of needing to first reach the backbone in 6in4 causes an increase of 37% when compared to Smart 4IDs. Again, 4IDs and 6rd perform similarly, due to the random ingress selection at endhosts in our 4ID experiments. An increase in hop count underlines the same information that latency does: Smart 4IDs greatly cut down on the number of routers that need to process the same packet, saving network operates both capacity and management overhead, as well as reducing potential points of failure.

5.3 Application Workloads

To provide a picture of end-to-end performance tradeoffs of the various approaches, we conducted a web page fetch over each deployment experiment and recorded its time to complete. We host the XIA main page[6] along with all sub-resources on one node under test. The remaining node fetches the main page from the first node as well as all sub-resources.

[6] `http://www.xia.cs.cmu.edu/`

The results are shown in Figure 9. Here we can clearly see the impact that small differences in path latency have on applications. Again, the results are not surprising; Smart 4IDs once again see clear benefits but 6in4 is quite far behind, greatly impacting the perceived speed of the network.

5.4 Failure Semantics

We evaluate the failure semantics of each model by comparing the time required to detect remote gateway failures in addition to providing a sense of the time scale for recovery from local and remote failures. For detecting remote gateway failures, we assume that hosts are aggressive and will report a remote gateway failure if the ingress it wishes to use does not respond to pings after 2*RTT seconds. We plot this in Figure 10. We can see that 6rd, 4ID, and Smart 4ID perform similarly, but 6in4 performs much worse. This is due to the fact that in the first three schemes, the remote gateway is exposed to the source as part of the addressing scheme, making it possible to check the status of the gateway. However, in static tunneling schemes like 6in4, these gateways are not directly exposed to the source node and thus full end-to-end status checks must occur, resulting in a significantly longer detection time.

For failure recovery, we present the time-scales required in Table 4. We can see that for 6in4, a failure requires the reestablishment of a tunnel. In the worst case, this could require network manager intervention, but we assume that tunnel reestablishment happens automatically over a few seconds. BGP propagation time shows up in a few cases, where nodes need to be informed of a new egress or tunnel endpoint in schemes that are reliant on routing. We assume

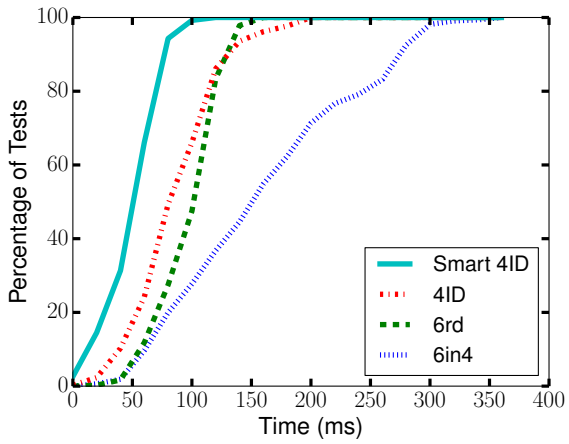


Figure 10: Detection Time for Remote Failures

this propagation time to be close to one or two minutes, as seen in [23].

The major differentiating factor here is 6rd's name server update after an address change. As 6rd addresses embed an ONA (IPv4) address, changing this address after gateway failure requires a change to DNS to update this ingress address. DNS propagation time is highly variable depending on caching timeouts. Typically this could be on the order of tens of minutes to hours.

4IDs avoid this issue entirely as the ONA (IPv4) addresses contained in the packet are distinct from the destination's address. Changing the ingress address after failures solely involves replacing the ingress node address in the packet, as opposed to binding to a completely new address. This has the added benefit of allowing higher-layer sessions (e.g. TCP) to stay active across ingress failures.

We can see that clearly Smart 4IDs provide the best failure semantics as a local egress failure can be recovered from by having the local controller inform hosts to use a new egress. 4IDs, however, rely on routing to locate a suitable egress (like 6rd), and thus must wait for new route propagation. Additionally, like 4IDs, smart 4ID nodes can be given a set of possible ingresses and thus can select a new one immediately, once an ingress goes down.

5.5 Management and Complexity

In addition to quantifiable metrics like performance and failure semantics, an important aspect of any architectural deployment is the amount of overhead in management and complexity. The specific issues we focus on are: which nodes in the network can cause end-to-end failures, where state is stored, what new devices need to be deployed, and what existing services need to be extended.

We examine each mechanism in turn and find that more complicated schemes like Smart 4IDs add some complexity to the network, but do so in a way that is easy to manage.

In **6in4** end-to-end communication can fail if the tunnel fails, potentially requiring manual recover, but TCP connections can persist across failures. Hard-state is stored at the tunnel gateways, but all other routers and hosts store nothing. No additional infrastructure is required beyond the tunnel gateways.

In **6rd** ingress failure causes a host to rebind to a new address, thus breaking TCP sessions, and requires a naming update. Hosts must know their own ingress, which is essentially soft-state. No additional services need to be deployed, but routing needs to be updated over time to track changes in topology, causing issues with deprecation over time.

4IDs can cope with path failures by simply choosing a new ingress address, even preserving TCP sessions. All source network routers need to keep track of an ONA (IPv4) routing table, potentially adding a lot of additional state. No new services need to be deployed, but the name server needs to be extended to return ingress ONA (IPv4) addresses.

Smart 4IDs can also cope with path failures by simply updating their egress and ingress addresses, also preserving TCP sessions. One additional dependence is created as each node relies on its local controller for gateway selection. Only gateway routers and the local controller need to keep additional ONA (IPv4)-related state, greatly reducing the complexity from 4IDs. This approach does require a new piece of infrastructure, the local controller. Like in 4IDs, the name server would need to be extended to return ingress ONA (IPv4) addresses.

While it is clear that Smart 4IDs provide the best performance while simultaneously providing clean failure semantics, the mechanism's impact on management and complexity is not as straightforward. While it greatly reduces complexity in the network from 4IDs, having the additional management overhead of a local controller when compared to simpler schemes 6in4 makes it harder to say that Smart 4IDs are uniformly better. We argue though that even in comparison to 6in4, some aspects of management are reduced, because the solution is *localized*. In the case of 6in4, failures require reestablishing tunnel endpoints in both networks, requiring coordination between multiple parties. Thus, although Smart 4IDs introduce local controllers into the network, there management overhead is not high as they are entirely localized.

6. CONCLUSION

We have shown, despite the wide variety of deployment mechanisms in use today, they only differ in how they answer four fundamental questions: selecting an egress, selecting an ingress, reaching the egress, reaching the ingress. Even with such variety, we see multiple approaches in this design space that are curiously absent from real-world deployment. Two approaches we describe are particularly intriguing and we explore them further.

We find that these two approaches borrow from work in the network architecture community. The first deployment approach "Flexible Addressing" leverages the flexibility in the data plane to encode ingress ONA addresses within a packet as a *separate* address from the destination's address, as well as use "fallbacks" to flexibly choose between addresses during forwarding time in the source network, providing better failure semantics. Our second deployment approach, "Smart Control Plane" introduces an intelligent control plane to provide both an ingress address *and* egress address within the packet to be used at forwarding time, further reducing complexity in the network.

Incremental deployment of network architectures is nontrivial. We have shown that the performance differences in wide-area experiments are rather striking. The mechanisms we propose provide new alternatives, but more im-

portantly we introduce a fundamental shift away from using solely familiar concepts (routing, naming, etc) to utilizing both newly introduced data plane and control plane concepts. These new concepts provide insight into a much more straightforward path to incremental deployment.

Acknowledgments

We thank David Naylor for his continued feedback on this project as well as the rest of the XIA team. We are also grateful to our shepherd Michael Schapira and the anonymous reviewers for their feedback. This research was supported in part by the National Science Foundation under award CNS-1040801.

7. REFERENCES

[1] Akamai. IPv6: What the Transition Means for Content and Application Delivery. Technical report, 2012.

[2] A. Akella. *Endpoint-Based Routing Strategies for Improving Internet Peformance and Resilience*. Ph.D., Carnegie Mellon University, Sept. 2005.

[3] D. G. Andersen. *Improving End-to-End Availability Using Overlay Networks*. Ph.D., Massachusetts Institute of Technology, Feb. 2005.

[4] M. Blanchet and F. Parent. IPv6 Tunnel Broker with the Tunnel Setup Protocol (TSP). RFC 5572 (Experimental), Feb. 2010.

[5] Y. Blanpain, H.-Y. Hsieh, and R. Sivakumar. The Incremental Deployability of Core-Stateless Fair Queueing, 2001.

[6] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633 (Informational), June 1994.

[7] B. Carpenter and C. Jung. Transmission of IPv6 over IPv4 Domains without Explicit Tunnels. RFC 2529 (Proposed Standard), Mar. 1999.

[8] B. Carpenter and K. Moore. Connection of IPv6 Domains via IPv4 Clouds. RFC 3056 (Proposed Standard), Feb. 2001.

[9] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: taking control of the enterprise. *SIGCOMM Comput. Commun. Rev.*, 37(4):1–12, Aug. 2007.

[10] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian. Fabric: a retrospective on evolving SDN. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 85–90, New York, NY, USA, 2012. ACM.

[11] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, July 2003.

[12] S. Deering. Host extensions for IP multicasting. RFC 988, July 1986. Obsoleted by RFCs 1054, 1112.

[13] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), Dec. 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946.

[14] R. Despres. IPv6 Rapid Deployment on IPv4 Infrastructures (6rd). RFC 5569 (Informational), Jan. 2010.

[15] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. M. Maggs, K. C. Ng, V. Sekar, and S. Shenker. Less Pain, Most of the Gain: Incrementally Deployable ICN. 2013.

[16] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4D approach to network control and management. *SIGCOMM Comput. Commun. Rev.*, 35(5):41–54, Oct. 2005.

[17] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan, and P. Steenkiste. XIA: efficient support for evolvable internetworking. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, NSDI'12, pages 23–23, Berkeley, CA, USA, 2012. USENIX Association.

[18] C. Huitema. Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). RFC 4380 (Proposed Standard), Feb. 2006. Updated by RFCs 5991, 6081.

[19] Internet2. Abiline Network. Technical report, 2005.

[20] M. Karir, G. Huston, G. Michaelson, and M. Bailey. Understanding IPv6 Populations in the Wild. In M. Roughan and R. Chang, editors, *Passive and Active Measurement*, volume 7799 of *Lecture Notes in Computer Science*, pages 256–259. Springer Berlin Heidelberg, 2013.

[21] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[22] C. Labovitz. Six Months, Six Providers and IPv6. Technical report, 2011.

[23] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet routing convergence. *IEEE/ACM Trans. Netw.*, 9(3):293–306, June 2001.

[24] P. Lin, J. Zhang, Y. Chen, and Q. Zhang. Macro-femto heterogeneous network deployment and management: from business models to technical solutions. *Wireless Communications, IEEE*, 18(3):64–70, 2011.

[25] J. Martin, A. Nilsson, and I. Rhee. The incremental deployability of RTT-based congestion avoidance for high speed TCP Internet connections. SIGMETRICS '00, pages 134–144, New York, NY, USA, 2000. ACM.

[26] J. Massar. AYIYA: Anything In Anything, 2004.

[27] S. C. Nelson, G. Bhanage, and D. Raychaudhuri. GSTAR: generalized storage-aware routing for mobilityfirst in the future mobile internet. *MobiArch*, 2011.

[28] E. Nordmark, R. E. Gilligan, and I. Inc. Basic Transition Mechanisms for IPv6 Hosts and Routers. RFC 4213 (Proposed Standard), 2005.

[29] K. Savetz, N. Randall, and Y. Lepage. *MBONE: Multicasting Tomorrow's Internet*. IDG Books Worldwide, Inc., Foster City, CA, USA, 1st edition, 1995.

[30] D. Thaler and B. Aboba. What Makes For a Successful Protocol? RFC 5218 (Informational), July 2008.

[31] A. Thierer. Are "Dumb Pipe" Mandates Smart Public Policy? Vertical Integration, Net Neutrality, and the Network Layers Model. In T. Lenard and R. May, editors, *Net Neutrality or Net Neutering: Should Broadband Internet Services be Regulated*, pages 73–108. Springer US, 2006.

[32] J. Wu, J. H. Wang, and J. Yang. CNGI-CERNET2: an IPv6 deployment in China. *SIGCOMM Comput. Commun. Rev.*, 41(2):48–52, Apr. 2011.

[33] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, K. Claffy, K. Dmitri, D. Massey, C. Papadopoulos, T. Abdelzaher, L. Wang, P. Crowley, and E. Yeh. Named data networking (NDN) project. Technical Report NDN-0001, PARC, 2010.

[34] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen. SCION: Scalability, Control, and Isolation on Next-Generation Networks. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 212–227, Washington, DC, USA, 2011. IEEE Computer Society.