An Experiment in Deploying Next Generation Network Protocols

Hitesh Ballani*, Andrey Ermolinskiy‡, Sylvia Ratnasamy†, Paul Francis*
*Cornell University †Intel-Research ‡UC Berkeley

***Abstract*— This paper presents IP$_{dmux}$ – a network-layer infrastructure that serves as a general-purpose deployment vehicle for next-generation network protocols. For each new network protocol, IP$_{dmux}$ provides a network-level access path between *any* end-user and the (approximately) closest router supporting the new protocol. IP$_{dmux}$ thus ensures that even partial deployments of new protocols are easily accessible by the global Internet user population.**

We present the design and implemention of IP$_{dmux}$ which we then use to experiment with three next-generation IP architectures – IPv6, FRM (a new protocol for global network-layer multicast) and *i3* (a rendezvous-based network architecture). Our experiences suggest new network-layer architectures can easily leverage IP$_{dmux}$ to aid their deployment. Moreover, our evaluation through simulation, measurement and wide-area deployment indicates that even small-sized IP$_{dmux}$ deployments can yield reasonable end-to-end performance for partially deployed next generation architectures.

I. INTRODUCTION

Motivated by mounting security and reliability concerns, the research community recently launched a large-scale effort to enable more comprehensive revisions to the Internet's internal plumbing[41]. In addition to reinvigorating research on improved network architectures, this effort has highlighted the importance of sustained evolvability as an architectural goal in itself.

Several recent studies explore this question of evolvability and offer insight on different fronts – design principles [47], [8], [2], implementation methodologies [23], [7], [2], [10], economic models [21], [22], deployment strategies [19], [31] and so forth. These insights however have not, to date, been translated into a practical realization of a network layer that actively assists with transitions between successive protocol generations. Our goal in this paper is to take a first step in this direction.

We present the design, implementation and use of IP$_{dmux}$, a network-layer infrastructure that serves as a deployment vehicle for new network protocols and services. End-users, even those in legacy networks, who wish to use a new network protocol can send their packets to IP$_{dmux}$, which then routes it to the (approximately) closest router supporting the new protocol. Key to IP$_{dmux}$ is the implementation of in-the-network protocol demultiplexing that serves as a simple "access" mechanism connecting users and new service providers. In effect, end-users or applications need only specify the network protocol (and optionally the service provider) they wish to use and it is the IP$_{dmux}$ infrastructure that takes care of routing packets to the appropriate provider for the requested service. IP$_{dmux}$'s key strength thus lies in providing global access to new network deployments in a manner that is flexible and general-purpose. Specifically,

- IP$_{dmux}$ supports different network-layer solutions and providers – small or large scale, single or multi-provider, native or overlay, commercial or experimental – and allows these to simultaneously co-exist.
- IP$_{dmux}$ ensures *all* Internet users can access *any* offered service (not just those supported by their local ISP).
- the implementation and use of IP$_{dmux}$ itself is agnostic to the specifics of any particular deployment testbed, next-generation architecture and/or provider.

In this paper we focus on IP$_{dmux}$ as a means of evolving the Internet Protocol (IP). This is due not only to the central role IP plays in the overall architecture but also because IP has proven singularly resistant to evolution. While the last two decades have seen a proliferation of new networking technologies both above and below IP (IP-IP, GRE, L2TP, and PPTP tunnels, wireless, MPLS, DNS redirection, NAT, firewalls, overlays), IP itself – in the sense of IP addressing, packet formats, service guarantees, interfaces and (to a somewhat lesser extent) routing – has remained mostly unchanged. The three major efforts to enhance IP itself, IP Multicast, IPv6, and Diffserv, have been far more painful and less successful than anticipated.

The specific contribution of IP$_{dmux}$ is a framework and infrastructure that supports multiple, co-existing end-to-end network layers. The immediate payoff is the enabling of experimental network protocol research, for instance in GENI. Over a longer term however, IP$_{dmux}$ can be viewed as a first-cut at an Internet engineered for easy evolution. Seen in this manner, IP$_{dmux}$'s support for multiple co-existent network-layer architectures represents something of a departure from accepted Internet design wisdom. A cornerstone of the Internet architecture has been the adoption of a single (mostly) ubiquitously supported IP 'dialtone'. This one-protocol-for-all model simplifies global interoperability but also limits flexibility and the room for innovation. We conjecture the IP$_{dmux}$ model might instead represent the best of both worlds. Like IP, it offers a narrow and ubiquitous 'point-of-interoperability' (namely the interface to IP$_{dmux}$). Unlike IP, however, it gives endpoints access to a wide range of end-to-

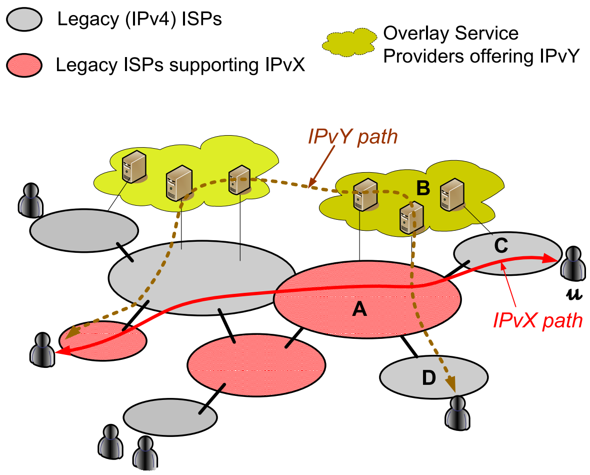

Legacy (IPv4) ISPs
Legacy ISPs supporting IPvX
Overlay Service Providers offering IPvY
IPvY path
B
C
A
u
IPvX path
D

Fig. 1. *Achieving global user access to new IP protocols IPvX and IPvY.*

end network services rather than impose a single lowest-common-denominator network service. Such conjectures are however undoubtedly premature – our effort in this paper is as yet but a small-scale experiment pursuing one possible approach to evolution and much experimentation remains before we can hope to converge on the right approach to enabling protocol evolution.

The remainder of this paper is organized as follows: Section II explains the overall IP$_{dmux}$ approach and its relation to prior work. We present the detailed design, implementation and deployment of IP$_{dmux}$ in Sections III and IV respectively, discuss our experience with deploying a few next generation protocols in section V, evaluate IP$_{dmux}$ in section VI and conclude in Section VII.

II. APPROACH

IP$_{dmux}$ combines elements from several recent research proposals. In this section, we first present the rationale and overall approach for IP$_{dmux}$ in Section II-A and then relate this approach to prior work in Section II-B.

A. Rationale and System Overview

Borrowing from [31], we use **IPvN** to generally denote a next-generation IP protocol. Figure 1 depicts the high-level functionality we seek to achieve – providers of both legacy IPv4 and different IPvN solutions coexist and can be either existing ISPs (as in IPvX), new overlay providers (IPvY), or combinations thereof. Our task is to ensure *all* IPvN end-users anywhere can exchange IPvN packets whether or not their local access providers support IPvN. Figure 1 denotes two such sample paths for IPvX and IPvY. The authors in [31] term this property *universal access* and note that universal access is key to enabling inter-provider competition for two reasons. First, universal access helps break the chicken-and-egg problem between demand and deployment – *e.g.*, Microsoft does not write IP Multicast-based applications because the population of users with multicast connectivity is limited while ISPs do not deploy IP Multicast as they see little user demand to justify the deployment costs. Secondly, universal access ensures end-users are no longer limited to the IP service supported by their local access providers and this user choice fosters provider competition [39], [8], [2].

Because universal access serves to transition between generations of IP protocols and providers, it is crucial that the mechanism that implements universal access impose minimal constraints on, and make minimal assumptions about, the nature of future IPvNs and the nature of agreements between end-users, IPvN providers and legacy providers. Moreover, any universal access mechanism cannot require any special-purpose support from legacy domains. These constraints are quite limiting in practice. For example, consider two seemingly straightforward options for universal access in Figure 1. In the first, IPvN domains (*e.g.*, A and B) could notify legacy domains (*e.g.*, C and D) of their existence as IPvN providers; IPvX packets from user u are then tunneled to A by u's legacy provider C. However this implies IPvN awareness at legacy domains and the existence of new IPvN-aware agreements between legacy and IPvN providers and hence universal access is easily gated by a lack of support from legacy providers.

A different approach might be to directly configure the end-user (u) to tunnel IPvX packets to A. This raises the question of how u determines the appropriate configuration. One possibility is that u enters an explicit agreement with A akin to its existing agreement with provider C. However, requiring such agreements raises the bar to adoption as demand can no longer be driven by users simply adopting IPvN-enabled software and applications; instead, adoption requires users to switch or add on service providers. Moreover, such agreements complicate the continued growth of IPvN deployment as users would have to explicitly switch providers in order to leverage new deployments (e.g., when C deploys IPvX).[1]

Another alternative might have users make use of an application-layer configuration service[13], [38]. However this requires that IPvN providers share fine-grained router-level topology information with a third-party provider (or the third-party provider use probes to determine this) at the application-layer which raises security and scalability issues (as discussed in [13] and [31]) and makes them unsuitable for router-level packet rerouting.

---

[1] It is important however that universal access mechanisms not preclude such explicit agreements as these might be important in certain IPvN scenarios – *e.g.*, IPvN providers that want access control for charging purposes. While we will allow such configuration, we just do not require it, thus leaving the door open to alternate charging models. For example, schemes that involve charging application providers (CDNs, source-based charging in multicast) or having IPvN revenues fall out of the increase in traffic and existing ISP settlements [22] or demand-first strategies [21] do not require that IPvN providers *directly* charge the IPvN end-user.

In exploring solutions to universal access, [31] translates this need for generality into respecting two key constraints:

(1) assume no special support from legacy networks (which in turn implies universal access should be achievable in current networks).
(2) neither require nor preclude new contractual agreements (e.g., IPvN user to IPvN provider, IPvN-'aware' peering between IPvN and legacy providers, *etc.*).

While [31] focusses on the deployment of new IPvN by incumbent ISPs, our goal with IP$_{dmux}$ is to support a broader variety of deployment efforts; in particular, experimental deployments[41] or new overlay-based providers[2], [35], [1]. We thus add the following requirement:

(3) allow a range of models for IPvN providers – single or multi-provider, native or overlay-based, commercial or experimental.

The discussion in [31] makes the case for IP anycast as uniquely suited to serve as the mechanism for universal access under constraints (1) and (2). IP anycast, defined in RFC 1546[30], is an IP addressing mode whereby multiple (possibly geographically disperse) hosts are assigned the same IP address, with the result that IP routing delivers packets destined for the anycast address to the nearest[2] such host. **IP Anycast, unlike IP Multicast, works without any changes to unicast routing** as routers do not distinguish between multiple routes to the multiple different hosts and multiple routes to the same hosts – each router merely retains the path to the closest of the hosts that share the anycast address. [31] proposes using IP anycast for the deployment of an IPvN as follows: each IPvN is associated with a well-known legacy IPv4 anycast address *a* and all IPvN routers advertise *a* into the IPv4 (legacy) routing protocol. IPvN users then tunnel all IPvN packets to *a* and legacy routing delivers these packets to the closest IPvN router without requiring any IPvN awareness by legacy domains and without any IPvN-provider-specific configuration by users.

While elegant, the straightforward application of IP anycast as described in [31] is less amenable to our third constraint as it requires that each IPvN provider undertake the advertisement and maintenance of anycast reachability. For non-ISPs, this can be a substantial overhead for the following reasons:

- In practice, because the BGP routing infrastructure only accepts prefixes longer than $/24$, each IPvN deployment must obtain a $/24$ unicast prefix and AS number from address agencies such as ARIN, RIPE[51], *etc.* Obtaining $/24$ prefixes, particularly for research purposes, is a time-consuming process made increasingly difficult by the dwindling IPv4 address space. This overhead can be a significant practical (if not technical) barrier to experimental and small-scale deployment efforts.
- Once obtained, this prefix must be advertised into the BGP routing infrastructure from *every* anycast-accessible site. Again, for ISPs that are already in the business of running and managing BGP, this is easily achieved. However, for non-ISP, overlay-based deployments, this requires that each site contact their upstream provider to advertise the anycast prefix and ISPs are often not particularly amenable to research-oriented BGP peerings [48]. This site-by-site negotiation represents a non-trivial overhead [28] and in fact has been the major bottleneck to scaling our IP$_{dmux}$ deployment.
- Finally, as our evaluation shows, a robust IP$_{dmux}$ infrastructure that provides fast failover requires planned and carefully engineered deployments (due to the interactions with global BGP) while a well-hosted provisioned deployment can be expensive (*e.g.*, assuming 5Mbps "burstable" bandwidth at each anycast box, the cost of deploying 50 boxes at a commercial co-location facility could amount to approximately $105,000 in start-up costs and a monthly fee of $35,000). For individual (particularly research) IPvNs, this cost and engineering overhead can be a significant barrier.

Given the above overhead, we chose to accommodate constraint (3) by amortizing the anycast deployment costs across multiple IPvN deployments. Borrowing Ballani and Francis' proposal for a single global IP anycast service[5], we decouple the anycast and IPvN components of an IPvN deployment and share a single IP anycast infrastructure across different IPvNs.

Figure 2 depicts the high-level operation of IP$_{dmux}$ and Figure 3 shows the overall router-level path IPvN packets traverse. The IP$_{dmux}$ infrastructure is associated with a single network prefix (dmux-prefix). The IP$_{dmux}$ infrastructure itself consists of a network of IP$_{dmux}$ routers each of which runs BGP to advertise and maintain reachability to the dmux-prefix. IPvN providers register their existence with the IP$_{dmux}$ service. Clients tunnel all IPvN packets to dmux-prefix and regular (legacy) IP routing delivers these packets to the closest IP$_{dmux}$ router. An IP$_{dmux}$ router looks up the particular protocol (*i.e.*, IPvN) specified in an incoming packet and forwards it to the appropriate registered IPvN router.

Our architecture thus allow all IPvN deployments to easily achieve universal access by simply piggybacking on the universal access to IP$_{dmux}$ itself. The tradeoff is the additional detour introduced due to routing through IP$_{dmux}$ *i.e.*, packets flow as source→IP$_{dmux}$ →IPvN →dest rather than source→IPvN →dest. Our evaluation in Section VI explores this issue in depth.

---

[2] Nearest according to the routing metrics used by the unicast routing protocols.

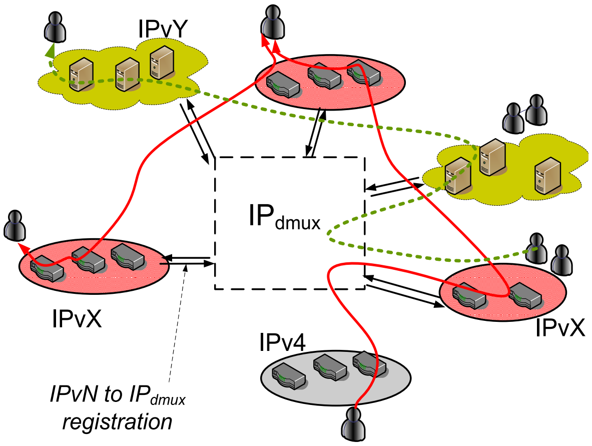


Fig. 2. *Logical view of IP$_{dmux}$ architecture. IPvN providers register with IP$_{dmux}$, users in legacy domains send IPvN packets to IP$_{dmux}$ and IP$_{dmux}$ redirects packets to the appropriate IPvN provider. Figure shows to sample IPvX paths and one IPvY path.*

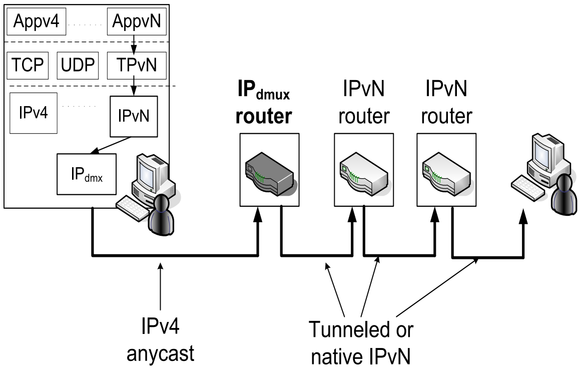


Fig. 3. *Packet forwarding path using IP$_{dmux}$.*

B. Related Work

IP$_{dmux}$ is most closely related to the ideas on evolvable Internet architectures in [31] and Ballani *et al.*'s proposed approach to building a global IP anycast service. Specifically, we borrow from [31] the goal of universal access and the recommendation of anycast as the means by which to implement universal access. However, we generalize these ideas to accommodate a broader range of deployment efforts. This leads us to decouple the anycast and IPvN components of a deployment and reuse a single anycast service across IPvN deployments. IP$_{dmux}$ also contributes the detailed design, implementation and deployment of the framework outlined in [31].

IP$_{dmux}$ can be viewed as a particular use of the PIAS system described in [5]. We contribute the deployment, evaluation and application of an anycast service such as PIAS.

GENI, the Global Environment for Network Innovations[41] is a broad initiative aimed at amplifying research efforts in innovative network technology. Our motivation in supporting experimental deployment efforts, is inspired by the GENI vision as articulated by Anderson *et al.*[2] and IP$_{dmux}$ could serve as the means by which end-users access GENI deployments. At the same time, we deliberately avoid entrenching any testbed or architecture specific components into the design of IP$_{dmux}$. This allows us to support deployment efforts by incumbent ISPs as well as GENI-based providers. We believe this is key to eventually moving to a network that has, built into it, hooks that allow incumbent providers to more easily transition between protocol generations.

PlanetLab[7] (eventually GENI) offers a testbed that could be used for the deployment of new IPvNs. IP$_{dmux}$ does not provide a testbed for the deployment of IPvNs but instead merely reroutes user traffic to different IPvN deployments including, but not limited to, IPvNs deployed over testbeds such as PlanetLab. PlanetLab and IP$_{dmux}$ are thus complementary and in theory PlanetLab could be used to host IP$_{dmux}$ routers.

The Overlay Convergence Architecture for Legacy Applications (OCALA)[19] is an architecture to support legacy applications over overlays networks. In some sense, IP$_{dmux}$ addresses a problem complementary to that of OCALA in that we support new (*i.e.*, IPvN-based) applications running at hosts located in legacy networks rather than legacy applications running in new networks. Consequently, a major issue for IP$_{dmux}$ is that non-legacy traffic must be redirected to an upgraded provider in a manner transparent to the user's legacy network. While the ideas from OCALA and IP$_{dmux}$ could potentially be combined into a more comprehensive transition framework, we do not explore this possibility in this paper.

Finally, IPv6 has defined several mechanisms for tunneling an IPv6 host on an IPv4 network into the IPv6 infrastructure. One of them uses IP anycast [17], while the other uses explicitly configured tunnels [11]. To our knowledge, the latter is more commonly used than the former. Based on informal conversations with people involved in IPv6 deployment, we believe the main reason for this is that users of the anycast approach had no way of knowing which IPv6 server might handle their packets. As a result, if the IPv6 service fails, for instance, the user has no idea where to go for assistance. IP$_{dmux}$ overcomes this by allowing the user to specify the IP$_{dmux}$ and IPvN provider. Also, IP$_{dmux}$ puts the transition mechanisms proposed in [17] to general use, thereby building a general-purpose deployment vehicle for different IPvNs, not just IPv6.

III. DESIGN

The overall operation of IP$_{dmux}$ involves four key entities: (1) IP$_{dmux}$ providers, (2) IPvN providers, (3) legacy network providers and (4) client end-users. In this section we lay out the required functionality and design options for each. Because IP$_{dmux}$ requires no change to existing legacy networks, we need only discuss operations within IP$_{dmux}$, IPvN networks and IPvN end-users.

Our discussion assumes an IP$_{dmux}$ infrastructure of moderate scale operated by a single administrative entity. This is partly for ease of exposition but also because this is the regime our current implementation targets. Given the early stage of research in this area, we prioritized simplicity and ease of implementation over extreme scalability. Section III-D discusses available design alternatives that we could pursue in the future should the need for greater scalability arise.

A. IP$_{dmux}$ providers

IP$_{dmux}$ serves to connect end-user clients to IPvN providers via network-layer rerouting. As described in Section II-A, IPvN providers register with IP$_{dmux}$ while client traffic is delivered to IP$_{dmux}$ via anycast; IP$_{dmux}$ then forwards received packets to the appropriate registered IPvN provider. IP$_{dmux}$ must thus support the following functionality:

- establishing anycast reachability
- establishing and maintaining IPvN provider information
- per-client IPvN router selection

The above comprise the key *control plane* operations of IP$_{dmux}$. In addition, the IP$_{dmux}$ *data plane* is responsible for de-tunneling IPvN packets sent by clients and tunneling them to the IPvN router selected by the control plane. We discuss each of these operations in turn.

Establishing anycast reachability: To allow clients in legacy networks to reach IP$_{dmux}$ via anycast, we associate the IP$_{dmux}$ service with an entire IPv4 unicast address prefix (*dmux-prefix*) and have IP$_{dmux}$ routers advertise this dmux-prefix into the IPv4 inter-domain routing fabric. For example, an ISP serving as IP$_{dmux}$ provider would configure some of its border routers to advertise the dmux-prefix through their BGP peerings. Similarly, a non-ISP provider of IP$_{dmux}$ (as in our case) advertises the prefix through a peering arrangement with the BGP-speaking router at the site hosting the IP$_{dmux}$ router. Thus, the set of all IP$_{dmux}$ routers form an IPv4 anycast group and any packet destined to an IPv4 address from the dmux-prefix will be delivered by legacy IPv4 routing to the IP$_{dmux}$ router closest to the traffic source.

Establishing and maintaining IPvN registration information: To receive IPvN client traffic, IPvN providers need only register with an IP$_{dmux}$ gateway. We assume a central authority that issues certificates to legitimate IPvN providers. For example, with IPv6 this task could fall to regional address allocation agencies such as RIPE and ARIN that issue IPv6 address blocks to providers. Likewise, PlanetLab administration might issue these at the time of account creation. Each IPvN router presents this keying material when registering as an IPvN provider. An IPvN router that wishes to receive tunneled IPvN packets from clients (*i.e.*, an IPvN "border" router) registers with its closest IP$_{dmux}$ gateway using a registration interface of the form: **register (proto_id, router_id, credentials)**. Here `proto_id` identifies the IPvN protocol (*e.g.*, IPv6, IPvMcast, IPvRON), and `router_id` the address of the registering IPvN router. In our design, `router_id` is a `(router_address, provider_id)` tuple where `provider_id` identifies the specific IPvN provider the registering router belongs to (recall that IPvN deployments may be multi-provider). The `provider_id` field allows clients the option to specify the first hop IPvN provider they'd like to be routed to and hence allows IP$_{dmux}$ to support IPvNs that require (possibly out-of-band) associations between IPvN users and providers. The `router_address` is the IPv4 address of the registering router. Since we run our test IPvNs over PlanetLab which imposes certain restrictions on the use of raw sockets, our implementation tunnels IPvN packets over UDP-IPv4 and consequently our `router_address` also includes the UDP port number of the registering router.

The **register** messages are sent to a well-known anycast address in the dmux-prefix that is set aside for registration purposes. Anycast delivery ensures an IPvN router registers with the IP$_{dmux}$ router closest to itself.

As described above, IPvN routers register with a single IP$_{dmux}$ router; however, since IPvN packets from clients may arrive at any IP$_{dmux}$ router, an IP$_{dmux}$ router must disseminate the information about IPvN routers registered with itself to other IP$_{dmux}$ routers. There are a variety of strategies for achieving this – broadcast, gossip-based propagation, DHT-based indexing – that offer typical tradeoffs between factors such as update overhead and rapid update propagation. For simplicity, our implementation has IP$_{dmux}$ routers establish full-mesh connectivity amongst themselves and disseminate registration information over this mesh. Hence, when a IPvN router registers or de-registers with a IP$_{dmux}$ gateway, the gateway informs all other IP$_{dmux}$ gateways of this event. We choose this as we expect IPvN routers, like most router infrastructures, are likely to suffer relatively low churn. Section III-D discusses DHT-based solutions for storing this registration information in a more scalable manner.

IPvN router selection: Ideally, the detour imposed by IP$_{dmux}$ should be minimal so that end-users see acceptable performance. Consequently, IP$_{dmux}$ tries to deliver packets from a given IPvN client to its closest IPvN router. While many latency estimation techniques exist (topology mapping[12], virtual coordinates[27], reactive probing[38],beaconing[20]), our use of IP anycast lends itself to a particularly lightweight solution that allows us to avoid having to discover or maintain the distances between all [IPvN client, IPvN router] pairs. An IPvN

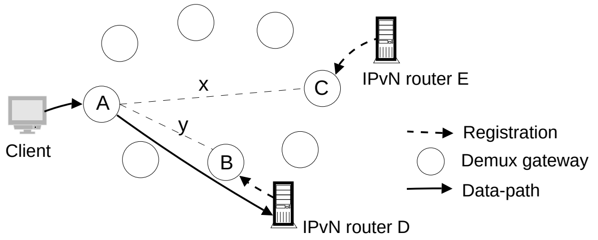


Fig. 4. Gateway A forwards IPvN packets to the IPvN router registered with the gateway closest to itself. In this case, y<x and so, IPvN packets are forwarded to IPvN router D which registered with gateway B.

client uses anycast to reach the IP$_{dmux}$ router closest to itself and similarly, an IPvN router uses anycast to register with the IP$_{dmux}$ router closest to itself. Hence, an IP$_{dmux}$ router can forward IPvN packets to an IPvN router registered with the IP$_{dmux}$ router closest to itself and still hope to offer good proximity. Figure 4 illustrates this process. We evaluate the quality of proximity offered by such an approach in section VI-B.

This approximation implies IP$_{dmux}$ routers need only maintain the pair-wise distances between themselves. In our implementation, each IP$_{dmux}$ router uses periodic active measurements to measure the latency to all other IP$_{dmux}$ routers and uses these measurements to determine which IPvN router to forward client packets to. Similarly, in the non-default case where the client specifies a desired first-hop IPvN provider, an IPvN router will forward the packet to the closest IPvN router whose registered `provider_id` field matches the `provider_id` field in the received packet. Note that for scalability IP$_{dmux}$ routers need not maintain the list of all IPvN routers for each type of IPvN but instead can maintain only the closest of the set of IPvN routers that share the same `provider_id`.

In summary, using the above process every IP$_{dmux}$ router discovers and maintains a set of proximal IPvN routers for every registered `protocol_id` and `provider_id`.

Forwarding in the data plane: Client packets arrive at an IP$_{dmux}$ router tunneled in UDP-IPv4. An IP$_{dmux}$ router's *data plane* is responsible for de-tunneling IPvN packets sent by clients and then tunneling them towards the IPvN router chosen by the control plane. The particular IPvN protocol type and the optional specification of an IPvN provider are carried in an IP$_{dmux}$ shim header inserted at the client between the IPv4 tunneling header and the IPvN packet itself.

The IP$_{dmux}$ gateway then tunnels IPvN packets over UDP-IPv4 to IPvN routers (as mentioned earlier, this allows us to run IPvN routers on PlanetLab machines). To provide the IPvN router with the IPv4 address of the originating client[3], we also include the client's IP address and UDP port in this tunneling header.

If the IP$_{dmux}$ router control plane has multiple equally close IPvN routers, it consistently selects one by hashing the client's IP address over the set of possible IPvN routers. This balances the traffic to different IPvN routers but maintains a consistent client-to-router mapping so as to avoid aggravated packet reordering.

B. IPvN providers

An IPvN provider, for example an ISP, that wants to provide transit for IPvN traffic does so by simply registering with an IP$_{dmux}$ gateway. For this, the ISP designates some of its IPvN-enabled routers to serve as ingress for IPvN traffic and these then register with an IP$_{dmux}$ gateway. These routers send their registration requests to the well-known registration address in the dmux-prefix. The use of IPv4 anycast for registration not only ensures IPvN routers register with their closest IP$_{dmux}$ router but also greatly simplifies the IPvN-side configuration needed to accommodate changes in the IP$_{dmux}$ deployment – *no* change is needed to IPvN routers as IP$_{dmux}$ routers fail or IP$_{dmux}$ providers upgrade their deployment with new IP$_{dmux}$ routers. An IPvN router uses periodic keepalives to maintain its registration with IP$_{dmux}$ and can deregister simply by falling silent.

Given our goal of not precluding an explicit association between IPvN providers and IP$_{dmux}$ providers, we also allow IPvN routers to register with specific IP$_{dmux}$ providers. For this, IP$_{dmux}$ routers also support registration at their regular unicast addresses (not derived from dmux-prefix), thus providing IPvN providers with the flexibility to choose the IP$_{dmux}$ provider they register with.

In addition to its interaction with the IP$_{dmux}$ service, an IPvN router must, of course, participate in the IPvN-specific operations such as the construction and maintenance of the IPvN topology. IP$_{dmux}$ does not impose any restrictions on the internal operation of an IPvN and hence a discussion of IPvN-specific operations is orthogonal to the issue at hand. However, an "ingress" IPvN router does need special handling for packets received from an IP$_{dmux}$ gateway – such packets must be detunneled before applying the IPvN routing scheme to the encapsulated IPvN packet.

Note that IP$_{dmux}$ imposes remarkably few requirements or restrictions on an IPvN deployment: "ingress" IPvN routers need only maintain their registration with IP$_{dmux}$ and de-tunnel packets received from IP$_{dmux}$ but no change is required at non-ingress routers (in fact, these are altogether unaware of IP$_{dmux}$) or to the IPvN packet format.

---

[3] this ensures that we do not place any restrictions on the IPvN design. For example, this might be useful if the IPvN uses the source IPv4 address in its routing decisions.

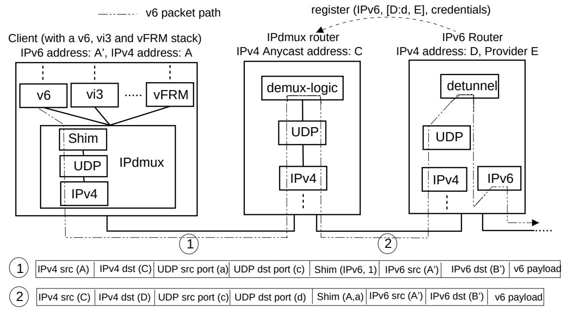

v6 packet path
register (IPv6, [D:d, E], credentials)
Client (with a v6, vi3 and vFRM stack)
IPv6 address: A', IPv4 address: A
IPdmux router
IPv4 Anycast address: C
IPv6 Router
IPv4 address: D, Provider E
v6
vi3
vFRM
Shim
UDP
IPv4
IPdmux
demux-logic
UDP
IPv4
detunnel
UDP
IPv4
IPv6
①
②
① IPv4 src (A) | IPv4 dst (C) | UDP src port (a) | UDP dst port (c) | Shim (IPv6, 1) | IPv6 src (A') | IPv6 dst (B') | v6 payload
② IPv4 src (C) | IPv4 dst (D) | UDP src port (c) | UDP dst port (d) | Shim (A,a) | IPv6 src (A') | IPv6 dst (B') | v6 payload

Fig. 5. IPv6 packets from a IPv6-enabled client reach a nearby IPv6 router through a IP$_{dmux}$ router

C. IPvN clients

A client with an IPvN-enabled stack tunnels its IPvN packets over UDP-IPv4 and sends them to an anycast address from the IP$_{dmux}$ dmux-prefix. The client also inserts an IP$_{dmux}$ shim header between the UDP header and the IPvN packet. This shim header carries the `protocol_id` field that identifies the IPvN protocol in question (IPv6, IPvRON, *etc.*) and the optional `provider_id` field (introduced in Section III-A). If used, we assume the client discovers the appropriate `provider_id` value through some out-of-band communication with IPvN providers. If the client is agnostic to the particular choice of IPvN provider, it sets the `provider_id` field in the shim header to 1 in which case IP$_{dmux}$ selects a proximal IPvN router as described above. Client packets are thus delivered to an IP$_{dmux}$ gateway that forwards them on to the IPvN network. Figure 5 illustrates an example IPv6 client with an IP$_{dmux}$-compatible network stack (in effect, IP$_{dmux}$ is implemented using UDP-IP tunneling with the addition of a IP$_{dmux}$-specific shim header). It also shows the traversal of IPv6 packets from the client to a first hop IPv6 router through an IP$_{dmux}$ gateway.

Finally, IPvN clients must also have a local-discovery protocol that allows them to check for IPvN support in their immediate network. For example, in the case of IPv6, the Neighbor Discovery protocol [26] offers such functionality. With this, clients in IPvN-enabled networks no longer need to use IP$_{dmux}$ when their immediate physical network supports IPvN. This thus provides a transition path where the use of IP$_{dmux}$ for a particular IPvN is naturally phased out as the deployment of IPvN spreads.

We note that, as in the case of IPvN providers, the IP$_{dmux}$ framework requires little complex mechanism at clients and the mechanism required is independent of the specifics of the particular IPvN protocol.

D. Design extensions

Our flat design described so far includes two choices that limit scalability. The first is the global dissemination of IPvN registration information and the second is the all-pairs latency measurements between IP$_{dmux}$ routers. For a large-scale but single-provider IP$_{dmux}$ deployment, these are easily fixed. DHTs [36], [33] can be used to achieve scalable storage and retrieval of registration information in a straightforward manner. Each registration entry can be stored in the DHT indexed by both the `protocol_id` field and a combination of the `protocol_id` and `provider_id` field (the latter being used to support client-to-IPvN-provider associations) with standard DHT techniques used to reduce the latency of internal DHT operations[9], [32]. Likewise, the use of all-pairs latency measurements can be replaced by the use of network coordinates[27]. Each IP$_{dmux}$ router independently computes its network coordinates and every registration entry in the DHT is augmented with the network coordinates of the IP$_{dmux}$ router that received (via IP anycast) the registration request from the registering IPvN router in question. Both DHTs and network coordinates have been extensively researched in recent years and we expect that their incorporation into IP$_{dmux}$, if needed, would be fairly simple.

Similarly, scaling to a multi-provider IP$_{dmux}$ can be achieved by moving to a two-level hierarchy of DHT deployments similar to those employed by systems such as ePOST[25] and DOA[37]. Such systems cluster nodes (in our case IP$_{dmux}$ routers) belonging to the same IP$_{dmux}$ provider into a single “level 2” provider-specific DHT and have a few representative nodes participate in a “level 1” inter-provider DHT. An entry (in our case registrations) inserted at a node in provider P is then inserted in P’s level-2 DHT and the global inter-provider DHT. This arrangement ensures administrative autonomy as each IP$_{dmux}$ provider has complete freedom to organize its level-2 IP$_{dmux}$ domain as desired (*e.g.*, complete mesh as we describe or a traditional DHT). This two-level arrangement also offers the appropriate fate-sharing between an IP$_{dmux}$ provider and its clients and registrants[37], [25].

In addition, an ISP acting as IP$_{dmux}$ provider can also inject the dmux-prefix into its intra-domain routing fabric in addition to the inter-domain one. This ensures that its own clients are dependent on intra-domain routing for failover (when an IP$_{dmux}$ router fails), and hence achieve fast failover.

IV. DEPLOYMENT AND IMPLEMENTATION

The previous sections detailed our design; we discuss a few additional implementation and deployment details here.

A. IP$_{dmux}$ Routers

Implementation: We have implemented an IP$_{dmux}$ router based on the design from Section III. In order to advertise the dmux-prefix into the IPv4 routing

Gateway	Host-Site	Host-site AS#	Upstream AS
128.84.154.99	Cornell University	26	WCG
12.155.161.153	Intel-Research Berkeley	2386	ATT
195.212.206.142	Intel-Research Cambridge	65476	ATT-World
12.108.127.148	Intel-Research Pittsburgh	2386	ATT
12.17.136.150	Intel-Research Seattle	2386	ATT

TABLE I

IP$_{dmux}$ deployment comprising of five gateways. Each of these advertise the dmux-prefix (204.9.168.0/22) through a BGP peering with their host-site.

fabric, each IP$_{dmux}$ router runs the quagga [49] software router and maintains an EBGP peering with the BGP-speaking router of the hosting site. The IP$_{dmux}$ software itself is a single-threaded, event-driven, user-level C++ program consisting of $\approx$ 2000 lines of code that utilizes libasync[24].

For fate-sharing between BGP reachability and the IP$_{dmux}$ router, we use the SNMP interface of the quagga router to ensure that the dmux-prefix is advertized only when the IP$_{dmux}$ software is running. When the IP$_{dmux}$ router starts up, it contacts a central server to determine information about other IP$_{dmux}$ gateways. It then communicates with other operational gateways to determine registered IPvN routers. The IP$_{dmux}$ router implements the various functions outlined earlier – determining the latency to all IP$_{dmux}$ routers, soft-state maintenance of state for all such registered IPvN routers, disseminating registration changes to other IP$_{dmux}$ routers and so forth. Our implementation also includes per-IPvN-router dampening of registration updates to ensure unstable IPvN routers don't lead to a deluge of updates across the IP$_{dmux}$ infrastructure. This bounds the churn in registration updates.

Deployment: In order to deploy IP$_{dmux}$, we obtained a /22 prefix (204.9.168.0/22) and AS number (33207) from ARIN and deployed IP$_{dmux}$ routers at the five sites list in Table I. At each site, our IP$_{dmux}$ router injects our AS number and prefix into the global IPv4 BGP routing fabric. Effectively, our deployment constitutes an (admittedly very small-scale) Autonomous System that is currently accessible to the global Internet population. An IPv4 packet to an address in the 204.9.168.0/22 range will be delivered to the closest of the above sites with "closest" determined by BGPv4 routing metrics and policies.

Our deployment is as yet very small in scale. The biggest hurdle in growing the deployment has been the site-by-site negotiation with upstream ISPs (ATT, WCG) to clear the advertisement of our dmux-prefix. Our experience on this front has been highly variable. At Intel-Berkeley, this approval was obtained almost immediately with a 5 minute phone call to ATT's support while the ISP contact at Intel-Cambridge required almost two months to approve the advertisement. Note that we do not require these upstream ISPs to actively inject our prefix in the BGP fabric but only propagate the advertisement from our IP$_{dmux}$ router onwards. Instead, the approval from the ISPs is only due to the access control ISPs often enforce on the AS numbers and network prefixes they expect to see advertised from customer sites. To accelerate this site-by-site deployment, we are currently in the process of deploying IP$_{dmux}$ over NLR [45]. To start off, we will deploy IP$_{dmux}$ routers at the 6 NLR POPs currently available and will add more routers as more NLR POPs are commissioned.

Further details about our deployment as well as the requirements for (much welcome) participant sites are available at [4].

B. IPvN clients

IP$_{dmux}$ requires that IPvN packets be captured on the client and tunneled to a IP$_{dmux}$ gateway. Ideally, this would be implemented with client kernel modules for each supported IPvN. For example, the `sit` IPv6-over-IPv4 tunneling module in Linux can be used to do so for IPv6. However, requiring kernel modules at the client side severely restricts the set of hosts we can use for our deployment and evaluation and in particular would prevent us from running IPvN clients over PlanetLab. We thus implement the client-side IP$_{dmux}$ functionality in a user-level daemon that uses *libiptc* for transparently capturing the requisite packets using Netfilter[46] hooks and *libipq* for user-level queuing of the captured packets. This daemon then attaches the shim header and tunnels packets out to IP$_{dmux}$ as described in Section III.

V. Deploying new network protocols Using IP$_{dmux}$

The previous sections described the design and implementation of IP$_{dmux}$. To test the generality of IP$_{dmux}$, we selected three very different next-generation protocols and deployed these using IP$_{dmux}$. We briefly describe these experimental deployments in this section. We report on the measured performance of these protocols later in Section VI but note that, given our task, the deployability of these protocols via IP$_{dmux}$ is probably more indicative of the value of our system than measurement numbers.

A. Free Riding Multicast (FRM)

FRM [50] is a new approach to implementing IP Multicast aimed at simplifying inter-domain multicast routing. FRM's core idea is to avoid complex tree construction protocols by instead using a multicast variant of source routing wherein the routing tree is computed from existing BGP unicast routes. We deployed a virtual FRM network on PlanetLab made up of 20 FRM routers organized as 6 FRM-enabled domains that run BGP with the FRM extensions amongst themselves. These FRM routers allow end-hosts or clients to join multicast groups and deliver multicast packets to them in a manner similar to

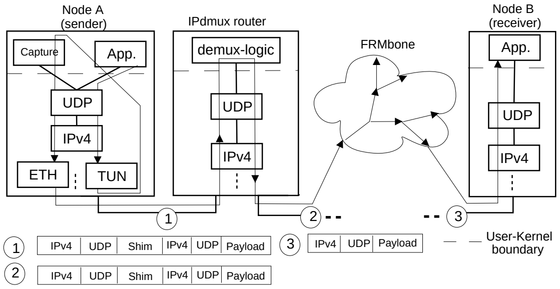


Fig. 6. Traversal of multicast packets from source A to one of the receivers B through IP$_{dmux}$ and FRMbone.

traditional IP Multicast. In effect, the deployment (which we call the FRMbone) is similar to the MBone [29] with the difference that it uses FRM techniques internally for route and membership discovery between domains. FRM leaves the multicast service model unchanged and hence works with no change to end-user network stacks (as most OSes support IGMP and multicast APIs). However clients whose ISPs do not support IP multicast still need some mechanism by which to reach the FRMbone. We used the IP$_{dmux}$ deployment described in section IV to enable clients in non FRM-enabled domains to access the FRMbone as described below.

Each FRM router in FRMbone registers with one of the five IP$_{dmux}$ gateways using IPv4 Anycast and keeps this registration alive during the period of its operation. To access the FRMbone through IP$_{dmux}$, FRM clients must tunnel their multicast packets to IP$_{dmux}$. This is done using the client-side capture-and-tunnel daemon described in section IV-B. Unfortunately, due to PlanetLab restrictions on user access to *iptables*, the daemon does not work for PlanetLab machines operating as FRM clients. We thus implemented a separate user-level daemon to capture user multicast packets using the TUN device [16] on PlanetLab machines.

For any given FRM client, all multicast packets, including messages to join and leave multicast groups, are routed to a nearby FRM router through IP$_{dmux}$. Figure 6 illustrates packet traversal from a multicast application on PlanetLab node A to a multicast group member (at PlanetLab node B) through our deployment. Thus, using this approach clients are able to efficiently and robustly access the FRMbone and run unmodified multicast applications. Ensuring client access to the FRMbone was relatively easy using IP$_{dmux}$– the IP$_{dmux}$-specific functionality (registration, keepalives etc.) that we added to the original FRM router implementation is only $\approx$ 60 lines of C code, while the client-side daemon consists of only 320 lines of code. (The non-PlanetLab version of the client packet capture module is implemented as a library and contains $\approx$ 250 lines of code).

B. Other IPvNs

In addition to FRM, we experimented with two other end-to-end protocols: IPv6[44] the next-generation IP architecture currently under deployment and *i3*, a research network architecture that offers a rendezvous-based communication abstraction built over a DHT routing layer[35]. In each case, we ran our IP$_{dmux}$ registration and detunneling modules on the IPv6 (i3) router and the capture-and-tunnel module on the clients. These IP$_{dmux}$-specific operations at both the router and client are almost entirely agnostic to the specifics of the particular IPvN and hence follow along the lines of the FRM deployment described above. As part of these experiments, we were able to achieve round-trip communication with IPv6 (using `ping6`) and with *i3* (using the send-private-trigger and recv-private-trigger programs shipped with the *i3* distribution) between 2 clients, each in a network that does not support IPv6 or *i3*.

VI. EVALUATION

In this section we evaluate the performance and usability of IP$_{dmux}$. We're primarily interested in two key performance aspects of IP$_{dmux}$. The first is quantifying the overhead (latency and throughput) due to the detour IP$_{dmux}$ imposes on the packet forwarding path. This overhead is affected by many factors including the efficiency of our implementation, the scale of our IP$_{dmux}$ deployment, the scale of an IPvN deployment and the router-level topology of the Internet itself. To better isolate these different factors, we measure IP$_{dmux}$ performance within a LAN environment, in our wide-area IP$_{dmux}$ deployment and using trace-driven simulation. We calibrate the overhead over IP$_{dmux}$ in isolation and as seen by an end-user IPvN application (a constant bitrate VAT-like tool running over FRMbone). We explore the above in Sections VI-A and VI-B.

The second performance question we explore in Section VI-C is the failover behavior due to IP$_{dmux}$ operating at the IP routing layer. The failure of an IP$_{dmux}$ router disrupts the end-to-end communication between two users and the repair time is subject to the time BGP takes to converge to the next closest IP$_{dmux}$ router. We experiment with our IP$_{dmux}$ deployment to measure the impact of such failure on the performance seen by an end-user multicast application. To the best of our knowledge, this is the first experimental study looking into the impact of IP anycast on end-to-end connectivity.

A. Microbenchmarks

In this section we present microbenchmarks for our implementation. Our experiments were conducted on 1.7 GHz Pentium IV machines with 512 MB of RAM running Linux 2.6.10 located on a single switched LAN. Given our user-level implementation, we use `gettimeofday` for timing measurements and perform 100 runs of each

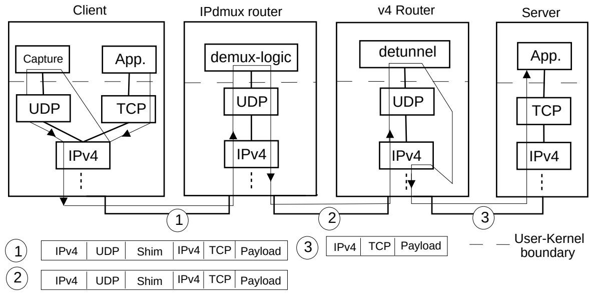


Fig. 7. Packet traversal for IPv4 through IP$_{dmux}$ as used in our experiments

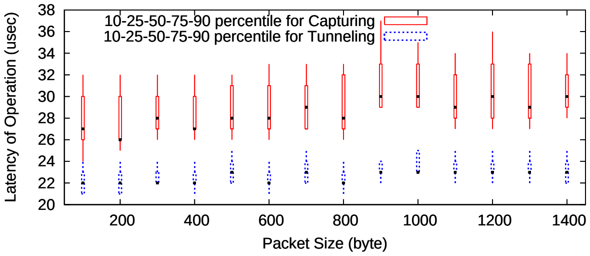


Fig. 8. Percentiles for the latency of capturing and tunneling with varying packet sizes

experiment. In order to use legacy measurement tools, we reuse IPv4 doubles as the IPvN protocol (i.e., N=4). Figure 7 shows how packets traverse from a legacy TCP client application to a server application in our experiments.

Client-side overhead: IP$_{dmux}$ involves capturing IPvN packets on the client and tunneling them to an IP$_{dmux}$ router. This is done by the capture daemon described in section IV. We measure overhead of performing these operations at the user-level. Figure 8 shows the percentiles for the latency of both the capture and tunneling operations for varying packet size[4]. The median capturing latency is ≈29usec while the median tunneling latency is ≈23usec. These numbers are along expected lines with the capture latency slightly higher as it involves the transfer of the packet from the user-space (client application) to the kernel and back to the user-space (capture daemon) in contrast to a single transfer from user-space to the kernel for tunneling. Also, both latencies increase slight with increasing packet size.

LAN and WAN experiments: We used LAN experiments to analyze the overhead due to our implementation on end-to-end characteristics. To this end, we measured the round-trip latency and TCP throughput in the following scenarios:

1) *Client-app* → *Server-app*: this provides the baseline measurements.
2) *Client-app* → *Client-capture* → *Server-app*: packets from the client application are captured and then re-injected. Hence this scenario includes the overhead of capturing the packets.
3) *Client-app* → *Client-capture* → *IP$_{dmux}$ router* → *IPvN router* → *Server-app*: packets from the client application are captured and tunneled to a IP$_{dmux}$ router which forwards them to a IPvN router and so on (as shown in figure 7). Packets on the reverse path travel directly from the server to the client. Given that both the IP$_{dmux}$ router and the IPvN router are implemented in user-space, packets need to enter the user-space at both the intermediate hops.
4) *Client-app* → *Client-capture* → *IPvN router* → *Server-app*: to better isolate the overhead due to the IP$_{dmux}$ router implementation, we also measured a scenario with just a IPvN router between the client and the server.

Scenario	1	2	4	3
Latency (usec)	174	218	523	834
Throughput (Mbps)	92.1	90.3	45.7	30.6

TABLE II
MEDIAN ROUND-TRIP LATENCY AND THROUGHPUT FOR SCENARIOS 1-4.

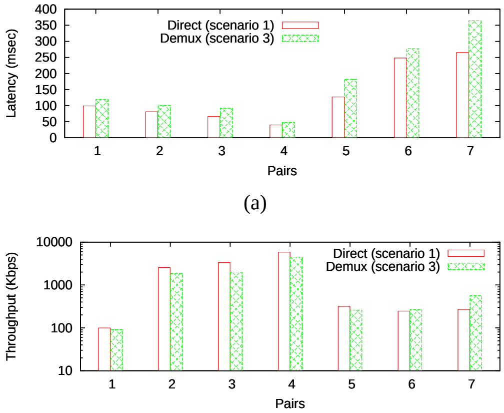
We used `ping` to measure the round-trip latency and `iperf`[43] to measure the TCP throughput for the four scenarios. Table II shows the median results of these measurements. The increase in the latency from scenario 1 to 2 is in accordance with the results in the last section. The addition of the IPvN router and the IP$_{dmux}$ router along the path add delays of ≈310 usec each; this can primarily be attributed to the packet going up and down the network stack at the intermediate hops. The fact that the client and the server are on the same LAN implies that the round-trip latency between them (for scenario 1) is very small. Hence, the relative increase in latency due to the IP$_{dmux}$ router and the IPvN router is very high which explains the sharp drop in throughput[5]. More importantly, in the wide-area context, the propagation latency of the detour through the IP$_{dmux}$ router greatly dominates the overhead due to our user-space implementation.

To verify this conjecture, we characterize the overhead due to IP$_{dmux}$ in the wide-area by measuring the round-trip latency and TCP throughput for randomly chosen pairs of PlanetLab nodes that route through our five node IP$_{dmux}$ deployment. Figure VI-A shows the measured latency and throughput values for scenarios 1 and 3 for seven such pairs. The plotted values are a median over 10 runs. Figure 9(a) shows that for the chosen pairs of nodes, going through the IP$_{dmux}$ deployment causes a latency overhead ranging from 8 to 55 msec. As expected, this overhead is much higher than the overhead in the LAN setting. However, since the relative increase in

---

[4] we restrict ourselves to 1400 byte packets to avoid fragmentation of the tunneled packets

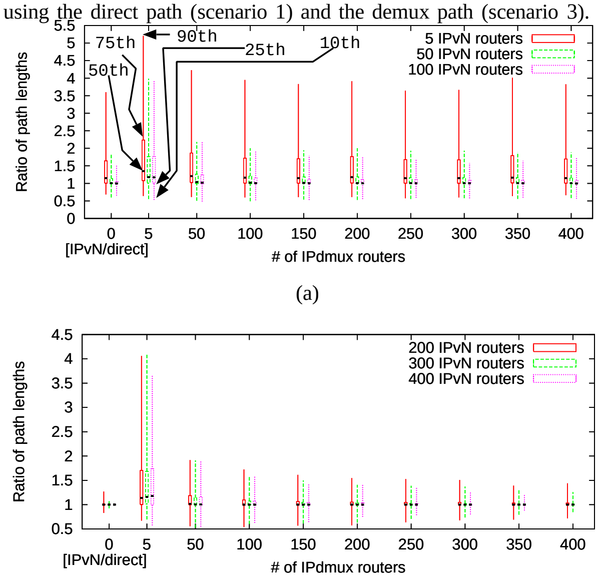
[5] We also note that an in-kernel implementation of the IP$_{dmux}$ router (and the IPvN router) would avoid most of the delays and hence, offer much better performance even in a LAN setting.



(a)

(b)

Fig. 9. Latency and (median) Throughput for pairs of PlanetLab nodes using the direct path (scenario 1) and the demux path (scenario 3).



(a)

(b)

Fig. 10. Percentiles for the ratio of length of the IPvN and the IP$_{dmux}$ path to the direct path length with varying number of IPvN routers and IP$_{dmux}$ routers (the first routing scheme is used).

latency is not as high as the LAN setting, the reduction in throughput due to the detour through the IP$_{dmux}$ deployment is much less. For the chosen pairs, going through the IP$_{dmux}$ deployment causes a throughput drop ranging from -288 Kbps to 1350 Kbps. Note that in some cases, the IP$_{dmux}$ path provides better throughput than the direct path in spite of having a higher latency. We conjecture that this might be due to lower loss on the IP$_{dmux}$ path in these cases.

B. IP$_{dmux}$ Stretch

With IP$_{dmux}$, packets from clients reach a vN router through a IP$_{dmux}$ router. In this section, we use simulations and measurements to evaluate the impact of this detour on the length of the end-to-end path.

1) Simulation study: For the simulation, we use a subset of the actual tier-1 topology of the Internet, as mapped out in the Rocketfuel project [34]. This subset consists of 22 ISPs, 687 POPs, and 2825 inter-POP links (details in [40]). The use of just the tier-1 topology can be justified on two grounds. First, a large proportion of traffic between a randomly chosen [client, server] pair on the Internet would pass through a tier-1 ISP. Second, such a simulation gives us an approximate idea about the overhead that a IP$_{dmux}$ deployment restricted to tier-1 ISPs would entail.

The topology was annotated with the actual geographical distance between POPs (in Km). We then used SSFNET[52] to simulate BGP route convergence. This allowed us to construct forwarding tables at each of the POPs and hence, determine the forwarding path between any two POPs.

The simulated IP$_{dmux}$ deployment involves placing a variable number of IP$_{dmux}$ routers and IPvN routers at randomly chosen POPs. These POPs are referred to as *router POPs* and *IPvN POPs* respectively. Next we consider a number of [client, server] pairs. For each such pair, we choose a POP through which the client's packets enter the topology (the *client POP*) and a POP through which packets leave the topology towards the server (the *server POP*). We simulated packet traversal between the client POP and the server POP for three different scenarios:

1) *direct path*: the native IPv4 path between client and server POP. This test reflects a global deployment scenario.
2) *IPvN path*: the path through the IPvN POPs assuming the IPvN deployment has its own prefix which is advertised by all IPvN providers. Packets from the client POP are delivered by anycast to an IPvN POP (the IPvN POP closest to the client POP) and thereon through IPvN routing to the server POP. This test captures the penalty due to partial deployment.
3) *IP$_{dmux}$ path*: the path through the IP$_{dmux}$ service. This captures the penalty due to partial IPvN deployment *and* IP$_{dmux}$.

Note that simulating the traversal of packets from the first IPvN POP to the server POP in scenarios (2) and (3) requires us to make assumptions about the routing scheme between IPvN routers. While a variety of routing schemes are conceivable, we used two simple, yet realistic, schemes in our simulations. In the first scheme, packets reaching the first IPvN POP are directly delivered to the server POP along the IPv4 path. This would be the case when the IPvN functionality requires a single IPvN router to merely modify some IPvN header field. In the the second scheme, packets reaching the first IPvN POP are routed to the IPvN POP closest to the server POP before being delivered to the server POP.

Figure 10 plots the 10th, 25th, 50th, 75th and the 90th percentiles for the ratio of the length of the IPvN path and the IP$_{dmux}$ path to the length of the direct path for

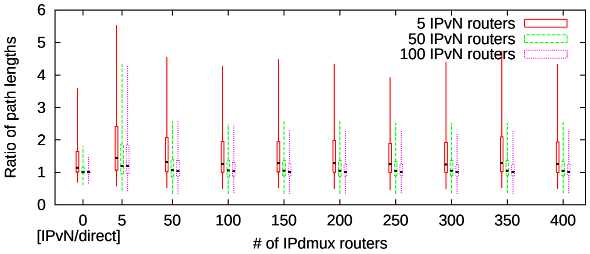


Fig. 11. Percentiles for the ratio of the length of the IPvN and the IP$_{dmux}$ path to the direct path length with varying number of IPvN routers and IP$_{dmux}$ routers (the second routing scheme is used).

varying number of IPvN routers and IP$_{dmux}$ routers while using the first routing scheme.[6] For a given number of IP$_{dmux}$ routers and IPvN routers, we simulated 100000 runs. Note that varying the number of IP$_{dmux}$ routers has no impact on the IPvN path and hence, the set of bars with zero IP$_{dmux}$ routers represents the ratio of the IPvN path length to the direct path length.

The figures show a decline in the overhead of the IP$_{dmux}$ path as compared to the direct path as the number of IP$_{dmux}$ routers increase, though the decline tapers off for higher number of routers. The overhead of the IP$_{dmux}$ path shows similar patterns with increasing number of IPvN routers. Note that the well-documented non-optimal nature of inter-domain routing[1] is reflected in the cases where the IP$_{dmux}$ path turns out to be shorter than the direct path.

Figure 10(a) shows that with a deployment of just 100 IP$_{dmux}$ routers (a mature deployment might encompass 50 times more POPs), and 100 IPvN routers, the median stretch for the IP$_{dmux}$ path is 1 with the 90^{th} percentile being 1.9. Similarly, the stretch of the IPvN path for a 100 IPvN router deployment has a median value of 1 and a 90^{th} percentile of 1.4. While such low median stretch values might seem surprising, they can be explained in terms of the latency distribution of nodes across the Internet. Gummadi et. al.[14] argued that the latency distribution as seen from a “typical” node applies to most of the Internet. Consequently, in case of the IPvN path with a 100 router IPvN deployment, the latency from the client POP to the closest IPvN POP (since it is chosen by anycast) can be approximated by taking the minimum of a 100 independent samples from the distribution presented in [14]. It is due to this reason that the median IPvN path length turns out to be the same as the direct path length. The same also applies to the latency from the client POP to the IP$_{dmux}$ router and from the IP$_{dmux}$ router to the IPvN router in case of the IP$_{dmux}$ path. Thus, even for moderate sized IP$_{dmux}$ and IPvN deployments, the impact of the detour on the end-to-end path length should be (and appears to be) acceptable. The figures also show the IP$_{dmux}$ path is not much longer than the IPvN path in IP$_{dmux}$ deployments with more than 200 routers. We believe that the advantages of a shared IP$_{dmux}$ infrastructure as compared to individual IPvN deployments advertising their own prefix far offset this overhead.

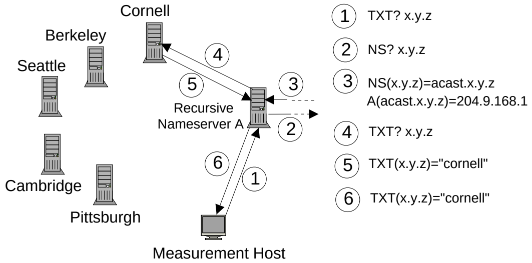


Fig. 12. Determining the router accessed by nameserver A through IP Anycast. Querying Nameserver A for the TXT record for x.y.z returns ”cornell” and hence, packets to the dmux-prefix (204.9.168.0/22) from Nameserver A are routed to the router at Cornell. In our actual measurements, we used domain ”anycast.anycast.guha.cc” as x.y.z

Similarly, figure 11 shows the ratio of the IPvN and the IP$_{dmux}$ path to the direct path length with the second routing scheme[7]. While the overhead with this second scheme is higher due to the extra IPvN hop, the variation of the overhead with different number of routers and IPvN routers is the same as that with the first scheme.

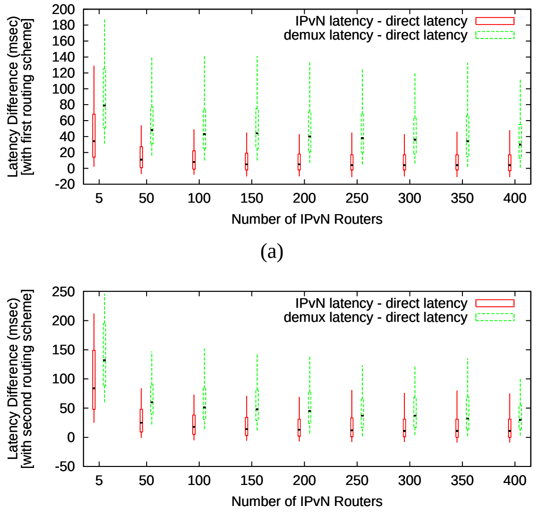
2) Measurements for hypothetical IPvN deployments: We also used active measurements to evaluate the stretch imposed by our five router IP$_{dmux}$ deployment on end-to-end paths. We used the King[15] technique to utilize recursive DNS nameservers as vantage points in our measurements. Using this approach, we measured the pairwise latencies between 518 widely distributed nameservers leading to a total of 133903 measured latency values.

In this study, we wanted to determine the overhead that our IP$_{dmux}$ deployment would impose if some of these nameservers were operating as IPvN routers and IPvN clients. However, this required us to determine the particular IP$_{dmux}$ router that is accessed by each of these nameservers through IPv4 anycast and the latency of doing so. For this purpose, we created a domain name (*anycast.anycast.guha.cc*) with a NS record pointing to an address (*204.9.168.1*) in our dmux-prefix. We also configured a `BIND` [42] nameserver on each of our IP$_{dmux}$ routers to authoritatively answer DNS queries for the domain anycast.anycast.guha.cc. Specifically, the nameserver at each router was configured to return a router-specific string in response to a TXT type DNS query for the domain. Hence, querying a recursive nameserver for the TXT record for anycast.anycast.guha.cc returns the string specific to the router that is accessed by the nameserver when it sends packets to the dmux-

---

[6] We plot the ratio instead of stretch (the difference in the length of the two paths) as it is difficult to gauge the relevance of the stretch being equal to x kms.

[7] results with 200, 300, 400 IPvN routers show a similar pattern and hence, are not shown



(a)

(b)

Fig. 13. Percentiles for the difference between the demux path and the direct path latency and the difference between the IPvN path and the direct path latency for our five router IP_{dmux} deployment with varying number of IPvN routers. 13(a) uses the first routing scheme while 13(b) uses the second routing scheme.

prefix. Figure 12 shows a nameserver A such that packets to the dmux-prefix from A are routed to the router at Cornell and illustrates the process by which we determine this. Further details about this process are available at [3]. Using this approach, we determined the IP_{dmux} router accessed and the latency of doing so for all the nameservers in our study.

Given the data collected above, we assumed some number of randomly chosen nameservers to be IPvN routers. Also, we generated [client, server] pairs by assuming one nameserver to be the client and another to be the server. For each such [client, server] pair, we calculated the latency of the direct path, IPvN path and the IP_{dmux} path (as defined in section VI-B.1) with both the routing schemes described in section VI-B.1. For a given number of IPvN routers, we determined the aforementioned latencies for 50 randomly chosen [client, server] pairs. Figure 13 plots the 10^{th}, 25^{th}, 50^{th}, 75^{th} and the 90^{th} percentiles for the difference between the demux path and the direct path latency and the difference between the IPvN path and the direct path latency for varying number of IPvN routers.

The figures show that the difference of the IP_{dmux} path and the direct path latency reduces as the number of IPvN routers increase, though the reduction tapers off with increasing number of IPvN routers (the simulation results had shown a similar trend). The same also applies to the difference between the IPvN path and the direct path latency. Figure 13(a) shows that with the first routing scheme, our current deployment imposes a $\approx$43 msec overhead on the end-to-end path in the median case and a $>$141 msec overhead for about 10% of the [client, server] pairs. Hence, in spite of having just five routers, our current deployment would offer performance that can be termed as acceptable for most kinds of IPvNs. The figure also shows that the IPvN path imposes just a $\approx$8 msec median overhead in the same setting. This suggests that even modest-sized IPvN deployments can offer very good performance. More to the point, our simulations have shown that the IP_{dmux} path length comes close to the IPvN path length as the number of IP_{dmux} routers increase. Hence, by deploying more routers we should be able make the impact of the detour through IP_{dmux} on end-to-end performance negligible.

Percentiles	10^{th}	25^{th}	50^{th}	75^{th}	90^{th}
(FRM -direct) latency	-23.3	76.8	140.8	182.9	220.5
(IP_{dmux}- direct) latency	14.5	29.13	147	229	441

TABLE III

PERCENTILES FOR THE LATENCY OVERHEAD OF THE FRM AND THE IP_{dmux} PATH AS COMPARED TO THE DIRECT PATH.

3) Measurements for an actual IPvN deployment: We also measured the stretch imposed by our IP_{dmux} deployment on end-to-end paths through the FRMbone. The goal here was to determine the overhead on multicast clients that use the FRMbone through IP_{dmux} as compared the clients that access the FRMbone directly. Hence, unlike the last two sections where we varied the number of IPvN routers or the number of IP_{dmux} routers or both, the number of IPvN routers (twenty FRM routers) and IP_{dmux} routers (five) is fixed in these measurements.

For these measurements, we randomly chose pairs of nodes on PlanetLab. For each pair [C1, C2], we determined the round-trip latency [8] through the FRMbone in two scenarios. In the first scenario (FRM-latency), we manually directed each client's multicast packets to the FRM router that is closest to it. This represents the ideal scenario if the FRMbone were to advertise its own anycast prefix. In the second scenario (IP_{dmux}-latency), each client's multicast packets reach the FRMbone through the IP_{dmux} deployment and then onwards to the other client. We also measured the round-trip time for the direct IPv4 path (direct-latency) between C1 and C2 using `ping`.

We performed this experiments for 15 pairs of PlanetLab nodes chosen at random. Table III shows the percentiles for the difference between the FRM-latency and the direct-latency and the difference between the IP_{dmux}-latency and the direct-latency. As can be seen, using the IP_{dmux} infrastructure to direct clients to the FRMbone imposes a very small overhead in the median case as compared to a scenario where the FRMbone itself advertizes an anycast prefix. This agrees with the simulation and measurement results presented earlier and further buttresses the case for IP_{dmux}.

C. IP_{dmux} failover

As a network layer service, IP Anycast *transparently* routes around failure and hence packets from IPvN clients

---

[8] to determine the roundtrip latency between [C1, C2], we built a multicast-ping as follows: C1 joins group1 and acts as a sender to group2 while C2 joins group2 and acts as a sender to group1

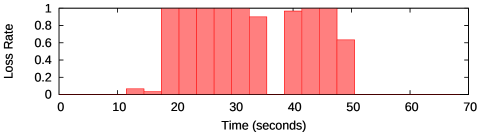


Fig. 14. Variation of loss rate with time for a multicast sender whose packets were being delivered to FRMbone through the IP$_{dmux}$ router at Cornell. The router fails at t≈18 seconds resulting in the sudden spike in the loss rate.

IPdmux router Site	Recovery Period (sec)	Stabilization Period (sec)	No. of multicast senders measured
Cornell	18.1	42.5	12
Berkeley	12.0	12.0	12
Cambridge	28.1	28.1	12
Pittsburgh	5.4	5.4	12
Seattle	4.1	4.1	12

TABLE IV

MEDIAN RECOVERY AND STABILIZATION PERIODS FOR PLANETLAB NODES SENDING MULTICAST DATA THROUGH OUR FRMBONE-THROUGH-IP$_{dmux}$ DEPLOYMENT.

that were using a failed IP$_{dmux}$ router are, (after the delay due to BGP route convergence) routed to the next closest IPvN router. In this study, we measure the impact of such router failures on the end-to-end communication between IPvN clients.

To this end, we determined the impact of IP$_{dmux}$ router failures on multicast communication between PlanetLab nodes that were using our FRMbone-through-IP$_{dmux}$ deployment. For each router in our IP$_{dmux}$ deployment (say A), we determined a few PlanetLab nodes such that packets to the FRMbone transit-address from these nodes are routed to router A. We then made these PlanetLab nodes send packets at a rate of 10 packets per second to a multicast group with two members. These packets are routed through router A to the FRMbone and finally, to the two group members. At this point we induced a failure by shutting down the router. Note that this also involves withdrawing the dmux-prefix that was being advertised into BGP by the router. At the receiver, we determined the time it takes for packets from each sender to start appearing again.

Figure 14 shows the variation of loss rate over time at one group member for packets sent by one particular PlanetLab node (that was using the Cornell router) when the Cornell router failed. As can be seen, the failure is marked by a sharp increase in the loss-rate to 100% and ≈18 seconds later, packets start appearing at the receiver again. However, there is some more instability a couple of seconds later and packet reception returns to normal after a total of 43 seconds. Further investigation revealed that the BGP convergence process induced by the failure of the Cornell router causes this PlanetLab node to be routed to a different router before settling on the final router. Given this, we defined two metrics to capture the impact of a router failure: *recovery period* or the time after a failure for the first packet to arrive at the receiver and *stabilization period* or the time is takes for the loss-rate to return and stabilize to pre-failure levels.

We performed these failover measurements separately for all five IP$_{dmux}$ routers. Table IV shows the median recovery period and the median stabilization period for the PlanetLab nodes whose multicast packets were being routed through the respective routers. The routers at Cornell and Cambridge have a distinctly higher recovery and stabilization period than the other routers. We believe that this is because of the fact that the three other routers have the same upstream provider (ATT). Hence, the failure of one of these routers induces a convergence process restricted (mostly) to the ATT network and the multicast senders that were using the failed router quickly failover to one of the other two routers. While these are very preliminary results from a small IP$_{dmux}$ deployment, they do point to the need for ensuring that each IP$_{dmux}$ provider deploy more than one IP$_{dmux}$ router to attain fast convergence.

Further, one can also imagine IPvNs or IPvN applications that find a 5-10 second recovery period too high and require even faster convergence. IP$_{dmux}$ can achieve this by decoupling machine failures from the BGP advertisement of the dmux-prefix through a clustered deployment where each IP$_{dmux}$ router comprises of a cluster of machines (as described in [5]). To conclude, these observations suggest that a robust IPv4 anycast service that provides fast failover requires a well-engineered deployment and hence, it makes a lot of sense to amortize the deployment effort across multiple IPvNs (as is the case with IP$_{dmux}$).

VII. CONCLUSION

This paper presents IP$_{dmux}$ – a network-layer infrastructure that allows even partially deployed network architectures to achieve connectivity to the global Internet user population. IP$_{dmux}$ achieves this through a combination of network-layer anycast and protocol demultiplexing. The use of anycast allows clients and service providers to use IP$_{dmux}$ in a robust, configuration-free manner and implies that IP$_{dmux}$ is incrementally deployable with no changes required to routers or routing protocols. The use of protocol demultiplexing allows multiple end-to-end network services to coexist without requiring complex interoperability. We implement IP$_{dmux}$ and use it to experiment with the deployment of 3 different network architectures. Our experiments validate the basic feasibility of our IP$_{dmux}$ approach while our results indicate that the inefficiency of interposing IP$_{dmux}$ on the path between end-users and service providers is reasonable for even small-scale IP$_{dmux}$ deployments.

Because IP$_{dmux}$ serves the GENI community, a deployment path forward is that GENI fund the effort (or something similar). The path forward for commercial deployment is less clear although certain possibilities exist. One is that of a standalone IP$_{dmux}$ deployment with IPvN providers paying for IP$_{dmux}$ service. Another is that Tier-1 ISPs deploy IP$_{dmux}$ for the added revenue they stand to gain due to the increased traffic pulled into their networks over paying customer-provider links. The most realistic however might be that one of the IPvN protocols developed through GENI will find commercial traction. Were this to happen, and assuming that that IPvN would be already using IP$_{dmux}$, it is likely that the coupling of the IPvN and IP$_{dmux}$ would lead to their joint adoption. Once done, the necessary hooks for deploying new IPvNs will have found their way into end-user software thus clearing the way for deploying still more IPvNs. Ultimately, this could lead to the use of IP$_{dmux}$ as a generic narrow interface to an Internet that supports multiple, varied, competing and complementary network services.

REFERENCES

[1] Andersen, D., Balakrishnan, H., Kaashoek, F., and Morris, R. Resilient overlay networks. In *Proc. of the ACM Symposium on Operating Systems Principles* (2001).

[2] Anderson, T., Peterson, L., Shenker, S., and Turner, J. Overcoming the Internet Impasse through Virtualization. *IEEE Computer 38*, 4 (2005).

[3] Ballani, H. Anycast Deployments: A Meaurement Study, April 2006. `pias.gforge.cis.cornell.edu/intel-any-measure.pdf`.

[4] Ballani, H. PIAS and IPdmux deployment, April 2006. `http://pias.gforge.cis.cornell.edu/deployment.php`.

[5] Ballani, H., and Francis, P. Towards a global IP Anycast service. In *Proc. of ACM SIGCOMM* (August 2005).

[6] Carpenter, B., and Moore, K. RFC 3056 - Connection of IPv6 Domains via IPv4 Clouds, February 2001.

[7] Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., and Bowman, M. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review 33*, 3 (July 2003).

[8] Clark, D. D., Wroclawski, J., Sollins, K. R., and Braden, R. Tussle in cyberspace: defining tomorrow's internet. In *Proc. of ACM SIGCOMM* (2002).

[9] Dabek, F., Li, J., Sit, E., Robertson, J., Kaashoek, M., and Morris, R. Designing a DHT for low latency and high throughput. In *Proc. of Symposium on Networked Systems Design and Implementation* (2004).

[10] David Taylor and Jonathan Turner. Diversifying the Internet. In *Proc. of Globecom* (2005).

[11] Durand, A., Fasano, P., Guardini, I., and Lento, D. RFC 3053 - IPv6 Tunnel Broker, January 2001.

[12] Francis, P., Jamin, S., Jin, C., Jin, Y., Raz, D., Shavitt, Y., and Zhang, L. IDMaps: a global internet host distance estimation service. *IEEE/ACM Trans. Netw. 9*, 5 (2001).

[13] Freedman, M. J., Lakshminarayanan, K., and Mazires, D. OASIS: Anycast for Any Service. In *Proc. of 3rd USENIX/ACM Symposium on Networked Systems Design and Implementation* (2006).

[14] Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., and Stoica, I. The impact of DHT routing geometry on resilience and proximity. In *Proc. of ACM SIGCOMM* (2003).

[15] Gummadi, K. P., Saroiu, S., and Gribble, S. D. King: Estimating Latency between Arbitrary Internet End Hosts. In *Proc. of the SIGCOMM Internet Measurement Workshop* (2002).

[16] Huang, M. VNET: PlanetLab Virtualized Network Access, Oct 2005. `http://www.planet-lab.org/doc/vnet.php`.

[17] Huitema, C. RFC 3068 - An Anycast Prefix for 6to4 Relay Routers, June 2001.

[18] Huitema, C. Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs), February 2006.

[19] Joseph, D., Kannan, J., Kubota, A., Lakshminarayanan, K., I.Stoica, and Wehrle, K. OCALA: An Architecture for Supporting Legacy Applications over Overlays. Tech. Rep. UCB/CSD-05-1397, University of California, Berkeley, 2005.

[20] Kommareddy, C., Shankar, N., and Bhattacharjee, B. Finding Close Friends on the Internet. In *Proc. of IEEE ICNP* (2001).

[21] Kumar, S., and Swaminathan, J. M. Diffusion of Innovations under Supply Constraints. Operations Research, Aug 2001.

[22] Laskowski, P., and Chuang, J. Network Monitors and Contracting Systems: Competition and Innovation. Under Submission (anonymized due to blind reviewing), Jan 2006.

[23] Loo, B. T., Condie, T., Hellerstein, J. M., Maniatis, P., Roscoe, T., and Stoica, I. Implementing Declarative Overlays. In *Proc. of ACM SOSP* (2005).

[24] Mazières, D., Kaminsky, M., Kaashoek, M. F., and Witchel, E. Separating key management from file system security. In *Proc. of the 17th ACM Symposium on Operating Systems Principles* (1999), pp. 124–139.

[25] Mislove, A., and Druschel, P. Providing Administrative Control and Autonomy in Peer-to-Peer Overlays. In *Proc. of IPTPS* (2004).

[26] Narten, T., Nordmark, E., and Simpson, W. RFC 2461 - Neighbor Discovery for IP Version 6 (IPv6), December 1998.

[27] Ng, T. S. E., and Zhang, H. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proc. of INFOCOM* (2002).

[28] Norton, W. B. Internet service providers and peering. In *Proc. of NANOG 19* (June 2000).

[29] Ohta, H., and Eubanks, M. MBONE Deployment (mboned) charter, Mar 2006. `http://www.ietf.org/html.charters/mboned-charter.html`.

[30] Partridge, C., Mendez, T., and Milliken, W. RFC 1546 - Host Anycasting Service, November 1993.

[31] Ratnasamy, S., Shenker, S., and McCanne, S. Towards an Evolvable Internet Architecture. In *Proceedings of SIGCOMM 2005* (Aug. 2005).

[32] Rhea, S., Godfrey, B., Karp, B., Kubiatowicz, J., Ratnasamy, S., Shenker, S., Stoica, I., and Yu, H. OpenDHT: a public DHT service and its uses. In *Proc. of ACM SIGCOMM* (2005).

[33] Rowstron, A., and Druschel, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)* (Heidelburg, Germany, 2001).

[34] Spring, N., Mahajan, R., and Wetherall, D. Measuring ISP topologies with rocketfuel. In *Proc. of ACM SIGCOMM* (2002).

[35] Stoica, I., Adkins, D., Zhuang, S., Shenker, S., and Surana, S. Internet Indirection Infrastructure. In *Proc. of ACM SIGCOMM* (2002).

[36] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A Scalable peer-to-peer lookup service for Internet applications. In *Proc. of ACM SIGCOMM 2001* (Aug. 2001).

[37] Walfish, M., Stribling, J., Krohn, M., Balakrishnan, H., Morris, R., and Shenker, S. Middleboxes no longer considered harmful. In *Proc. USENIX OSDI* (San Francisco, CA, December 2004).

[38] Wong, B., and Sirer, E. G. ClosestNode.com: An Open-Access, Scalable, Shared Geocast Service for Distributed Systems. *SIGOPS Operating Systems Review 40*, 1 (Jan 2006).

[39] Yang, X. NIRA: A new Internet routing architecture. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture* (Karlsruhe, Germany, Aug. 2003).

[40] Zhang, X., Wang, J., and Francis, P. Scaling the Internet through Tunnels, 2005.

[41] Global Environment for Network Inovations, Apr 2006. `http://www.geni.net/`.

[42] Internet Systems Consortium, Apr 2006. `http://www.isc.org/`.

[43] Iperf, Apr 2006. `http://dast.nlanr.net/Projects/Iperf/`.

[44] IPv6: The Next Generation Internet!, April 2006. `http://www.ipv6.org/`.

[45] National Lambda Rail, April 2006. `www.nlr.net`.

[46] Netfilter project page, April 2006. `www.netfilter.org`.

[47] NewArch Project, Apr 2006. `http://www.isi.edu/newarch/`.

[48] Private Communication with Bill Woodcock, Jan 2006.

[49] Quagga Routing Suite, Apr 2006. `http://www.quagga.net/`.

[50] Revisiting IP Multicast. Under Submission (anonymized due to blind reviewing), Jan 2006.

[51] RIPE Network Coordination Center, Apr 2006. `http://www.ripe.net/`.

[52] SSFNET, Apr 2006. `http://www.ssfnet.org/`.