A Programmable Overlay Router for Service Provider Innovation

Bruce Davie
Cisco Systems, Inc.
1414 Massachusetts Ave.
Boxborough, MA 01719
bsd@cisco.com

Jan Medved
Cisco Systems, Inc.
170 W Tasman Dr.
San Jose, CA 95134
jmedved@cisco.com

ABSTRACT

The threat of commoditization poses a real challenge for service providers. While the end-to-end principle is often paraphrased as "dumb network, smart end-systems", the original paper makes a more subtle argument about appropriate distribution of functionality among endpoints and intermediate systems. Functions may be implemented in the network for performance reasons, and when they offer value to a wide range of applications without inhibiting the correct operation of applications that do not need these functions. In this context, we describe a prototype platform for experimentation with novel, useful functions "inside" the network. This programmable platform allows service providers to innovate quickly and to deploy new functions within the network when it makes sense. By implementing the platform as an overlay, service providers can assist those applications that benefit from added functions such as caching and streaming support, without interfering with the correct and efficient operation of other applications that do not need them. Service providers can also leverage their detailed topological knowledge and ability to control network resources, features that would be difficult for conventional overlays. Programmability of the platform enables features to be extended or composed with other pieces of software, by either the service providers or third parties.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Network Topology; C.2.6 [**Internetworking**]: Routers

General Terms

Design, Performance

Keywords

Overlays, programmable routers, end-to-end argument

1. INTRODUCTION

It is often asserted that the end-to-end principle [21] mandates that networks should be "dumb" and that most functionality should be provided by end-systems. In fact, that assertion considerably oversimplifies the arguments of the original paper. We quote the original argument here:

> The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)

We will return to consider this argument more closely later in the paper, but for now it is worth noting that it is hardly a blanket prohibition on the implementation of functionality inside the network. We note in particular the parenthetical statement at the end: a case can be made for "in-network" functionality when it provides a performance enhancement, even if certain functionality is "incomplete".

The "networks should be dumb" mantra is problematic for service providers[1] in a competitive environment, because it effectively requires all service providers to provide the same service — basic packet transport. This leads to the notion that the service provider is offering a *commodity* service, in which providers differentiate themselves primarily through price. While selling a commodity need not be unprofitable (as demonstrated in recent memory by the oil companies), many if not most service providers would prefer to differentiate themselves from their competition by offering more compelling services. Thus, the idea of adding valuable functionality to the network itself is potentially appealing to service providers[2]. As discussed below, however, there are abundant downside risks to the addition of in-network functionality, and thus service providers should exercise careful judgment when moving beyond simple packet transport.

This paper proposes an approach by which service providers can both add new functionality to the network and experiment with the utility of such new functionality, without encumbering those applications and users that have no need of such functionality. At the heart of this approach is a prototype platform which we refer to as a "programmable overlay router". We argue that an overlay provides an appropriate means for experimenting with new functionality "in" the network, in part because it allows the underlay

[1]We use the term "service provider" broadly to encompass providers of networks including both Internet service providers (ISPs) and Telecommunications service providers (telcos).

[2]The economics of the service provider business is much too complex for a thorough discussion in this paper. Furthermore, we have no desire to step outside our area of expertise by holding forth on economic topics.

to be kept simple and stable. Meanwhile, programmability of the overlay devices enables experimentation with new functionality at relatively low cost.

The concept of using overlay networks to enhance the functionality of an underlying IP network is not new (see, for example, [17].) The contribution of this paper is to focus on *service provider-hosted* overlays. This contrasts with prior work that has typically assumed that overlays are operated by an administrative entity distinct from the one operating the "underlay" IP network. Allowing the same entity to operate both the underlay and the overlay presents both challenges and opportunities. The focus of our work has been to tackle those challenges and exploit the opportunities.

The organization of the paper is as follows. Section 2 outlines some of the challenges faced by service providers as they attempt to add useful functionality to their networks, using the end-to-end principle as a guide. In Section 3 we describe an overall architecture to address the challenges using a programmable overlay. Section 4 presents a prototype implementation of this approach and gives an overview of one example application that we have built using the platform. Finally, we discuss the related work that has informed our ideas, and present some concluding remarks.

2. CHALLENGES FOR SERVICE PROVIDERS

Service providers who attempt to add functionality to their networks above and beyond simple packet transport face a number of challenges. If functions which are not generally useful to applications are added, they add cost and complexity without providing sufficient benefits to justify the costs. This is at the heart of the end-to-end argument: if, for example, the only way for an application to truly ensure reliability is to implement its own reliability mechanisms, there is not much value in providing partial reliability mechanisms inside the network, and there is a cost to doing so. Hence, one challenge for service providers is to identify those mechanisms that are better provided in the network itself than by the endpoints.

A more subtle challenge arises when we consider the problem of innovation. As observed in a paper that re-examines the end-to-end argument in a modern context [8]:

> The edge orientation for applications and comparative simplicity within the Internet together have facilitated the creation of new applications, and they are part of the context for innovation on the Internet.

Put another way, the simplicity of the Internet's core fosters innovation by enabling a rich diversity of new applications to be deployed at the edges, requiring no modification to the network core itself. Conversely, if every new application were to require some service provider involvement before it could be deployed, innovation would likely be stifled.

In summary, service providers would like to add more functionality to their networks in the hope of "adding value" and differentiating themselves from their competition. In doing so, however, they must avoid adding needless cost and complexity and also avoid creating barriers to innovation. In the following section, we will describe an approach by which service providers can achieve an appropriate balance between these apparently conflicting goals.

3. TECHNICAL APPROACH

Our technical approach to addressing the problems described above can be summarized as follows: service providers can *supplement* their core network infrastructure with a *programmable overlay*. They continue to offer a simple, robust IP packet delivery service thus ensuring that edge-based innovation can continue to happen as before. At the same time, they can offer enhanced functionality and services using the overlay for those applications and users that stand to benefit from such enhancements. By making the overlay programmable, they can begin with a small set of relatively well-understood and well-defined functions, which can be subsequently added to and combined to produce new functionality. This enables service providers to experiment with new functionality at relatively low cost, retaining only the functionality that demonstrates long-term value. In a sense, this is analogous to the way application developers experiment with new ideas at the edge today, with the most successful applications surviving and continuing to grow.

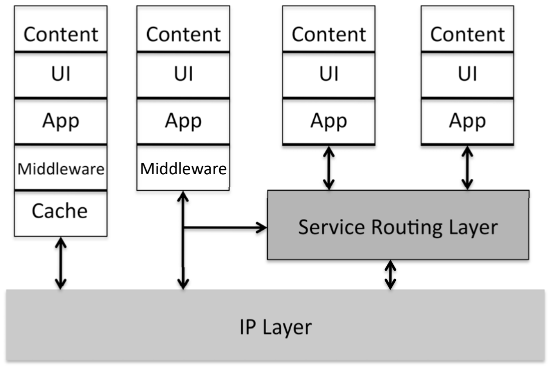

Figure 1: The Service Routing Layer

Figure 1 illustrates the proposed approach at a high level, using the example of content delivery applications. In this example, all the applications consist of some content, a user interface (UI), and some application layer functionality/protocols (App). On the far left, the application also includes some "middleware" (e.g., naming and lookup functions) and a caching function. The service provider-hosted overlay implements a set of functions which we refer to as the "service routing layer". The application on the left completely bypasses this layer, running directly over IP as normal. Other applications, such as those on the right-hand side, make use of supplemental functionality provided by the service routing layer. This functionality might include, for example, overlay routing, naming services, caching of popular content, etc. Other applications might choose to implement some functionality on their own while selectively drawing on functionality provided by the service routing layer.

The goal of implementing the service routing layer as part of the service provider's infrastructure is to facilitate the deployment of new applications by either the service provider or by cooperative third parties. For example, a company wishing to develop and deploy a new peer-to-peer television service could utilize the peer-to-peer overlay and caching functions provided by the service provider, while focusing the majority of its energy on obtaining compelling content and developing an effective user interface. This contrasts with the "over the top" model, in which the aspiring P2P TV company would also have to develop its own caches and overlay, while also doing all the other things necessary to produce a compelling application. The key question, of course, is which functions have enough general applicability to merit inclusion in the service routing layer. The following sections describe our approach to answering this question in more detail.

3.1 Service provider-hosted overlay

The idea of using overlays to incrementally add functionality to existing networks can be traced back at least as far as the Mbone[3] [11], and programmable overlays became popular around a decade ago, motivated in part by concerns about "ossification" of the Internet [14]. PlanetLab [16] is perhaps the most widely known programmable overlay network.

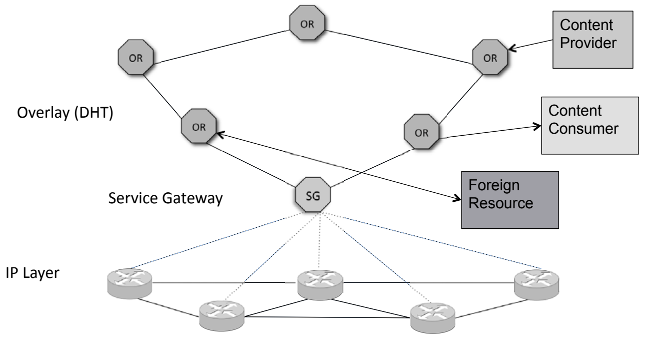

Figure 2: Programmable Overlay Network Architecture

Overlays have generally been operated without the involvement of the service provider who operates the "underlay", but clearly our goal here is to enable the service provider to be directly involved in the delivery of higher layer services. In this context, the service provider benefits from using an overlay because it enables selective addition of new, higher-layer services for the benefit of those applications that find them useful, while other applications continue to operate on the unmodified underlay — the IP layer.

A programmable overlay can be contrasted with programmability at the IP layer itself. An overlay clearly limits our ability to change the operation of the IP data plane, although small-scale experimentation with new data plane functionality is still possible, much as in the Mbone. However, since the IP layer needs to be kept robust and simple, it is attractive to be able to make modifications to the services offered by the network without risking disruption at the IP layer. We also observe that many modern applications could benefit from enhancements above the IP layer, such as caching and support for streaming media. (We return to this subject below in section 3.4.) In fact, we believe that there is likely to be more need for innovation above the IP layer, making a programmable overlay an attractive choice. Also, since the demand for the new, higher-layer services is likely to be small initially compared to the demand for traditional IP packet delivery, it is appealing to be able to deploy a small number of devices in the overlay, which can then be scaled up as demand dictates[4].

3.2 Topological proximity

When an overlay is operated by an entity that is independent of the provider of the underlay, there is typically a certain amount of information hiding between the layers. For example, PlanetLab nodes are ignorant of the detailed IP-layer topology interconnecting the nodes. By contrast, a service provider who operates his own overlay on top of his own network infrastructure has the liberty to exchange information between the layers. One or more of the overlay nodes may, for example, listen to the IP-layer routing protocols to obtain an accurate picture of the network layer topology. Equipped with such information, an overlay node may then answer a range of queries related to the distance or routing cost between various entities in the network. We refer to this as a "topological proximity" function.

A topological proximity function can be used in a variety of ways. For example, if the service provider is offering a caching service, it can be used to determine which of several possible cache locations is the best one from which to serve a particular cached item. Alternatively, applications that operate outside of the service provider's control, such as peer-to-peer file-sharing applications, can request assistance from the proximity function to help them perform peer selection. This sort of capability has recently attracted considerable attention in the IETF and elsewhere [6][23][15].

As illustrated in Figure 2, the proximity function can provide a logical connection between the overlay and the network layer. We implement the proximity function in a subset of the overlay routers (labeled Service Gateway in the figure), where it participates as a passive listener in the routing protocols (IS-IS, OSPF, and BGP). It may then communicate with its peers in the overlay to answer their queries related to network topology. We are also defining an interface to the proximity function that could be used by a range of clients [1].

One might reasonably ask whether this inter-layer communication actually provides a benefit when compared to "traditional" overlays that have no access to detailed information from the underlay. For example, the RON overlay [2] successfully uses probing to establish the suitability of various paths across the underlay. While we are not aware of any comprehensive comparative study that answers this question, there is evidence that explicitly providing topological information to some classes of applications such as peer-to-peer file sharing can both improve the performance of those applications and reduce their consumption of network resources [6][23]. Further exploration of the value of the topological proximity function is a focus of our ongoing work.

3.3 Network Control

Just as a service provider-hosted overlay has more access to topological information than a separately managed overlay, the service provider also has potentially more ability to control properties of the network. For example, an overlay node operated by the service provider may be able to signal to the network that a particular quality of service is needed for a given flow, using mechanisms such as RSVP [24] or Differentiated Services [7]. If the underlay supports MPLS traffic engineering, the overlay could explicitly signal to the underlay to establish a particular traffic engineering path meeting a set of specified constraints. Similarly, GMPLS could be used to establish optical paths in a suitable underlay network[5].

It is noteworthy that deployments of QoS technologies such as RSVP and Diffserv have been largely limited to single domains [9]. Thus, the fact that the overlay nodes can be treated as being in the same administrative domain as the underlying IP network raises the possibility of greater usage of QoS capabilities at the IP layer. Our current implementation makes use of Diffserv in this way. RSVP brings more solid QoS guarantees at the cost of more state, which would be most logical for a relatively small class of high-value applications.

To simplify the task of developing applications that run in the overlay and might take advantage of these network layer functions, one could imagine abstracting the low-level network capabilities

[3]The Internet may itself be counted as a famous example overlay on top of the PSTN.

[4]Significant experimentation with new data-plane functionality is not precluded by our approach, it is simply not our focus. Issues related to the use of overlays for data-plane experimentation are well discussed in [17].

[5]An approach along these lines is being explored by the DRAGON project [10].

behind a more developer-friendly API. For example, something along the lines of the Microsoft generic QoS API could be provided in the overlay nodes and used by applications to invoke the network-layer QoS mechanisms, with the low level details (e.g., DSCP values, RSVP messages) being controlled in response to higher level goals (e.g. "gold", "silver", "bronze" QoS levels).

3.4 Built-in functions

While the programmability of the overlay is essential to enable new capabilities to be added and experimented with, it makes sense to include some number of "built-in" functions that are likely to have wide utility. At the time of our implementation, we identified a relatively small set of functions that enable a wide range of applications to be supported. In the following sections, we discuss how these functions can be extended and composed with others to produce more complex applications and services.

The set of functions that we identified was influenced heavily by the prevalence of streaming video in today's networks. The functions include:

- Distributed hash table (DHT). DHTs are well established as a scalable and resilient technique for constructing overlay networks. We use a DHT based on Pastry [19] to construct our overlay; we also offer DHT storage and look up functions as a building block service, in a similar manner to OpenDHT [18]. We store structured descriptors in the DHT to support a variety of functions, such as maintaining pointers to content replicas in the distributed cache, and to point to instances of replicated services. The DHT provides a unified naming system for all objects (content, services, etc.) and a robust, scalable means to route requests to overlay nodes that can service them – it forms the heart of the "service routing" layer. The DHT may also be used simply to store opaque data.

- Proximity. The topological proximity function was described above. It is used by other functions such as caching, to enable the nearest replica of an object in the cache to be selected. It is also made available as a service, to be used by applications that can benefit from the topological knowledge of the service provider (e.g., peer-to-peer applications).

- Caching. Caching is probably one of the clearest examples of a piece of functionality that a service provider is well equipped to offer. Caching requires that cached copies of objects are located close to where they are needed; the service provider has the necessary footprint to place caches where they will be most effective, and also has the detailed knowledge of the network topology to enable intelligent choices to be made among multiple cache locations.

- Streaming support. With video becoming an increasingly dominant component of network traffic, it may be useful not only to be able to cache video files in the overlay devices but also to be able to stream video directly from these devices to end-users. While the proliferation of different streaming formats complicates the situation, just supporting a couple of popular formats is likely to provide significant benefit, and support for other formats can always be added.

- Events. Scalable event notification is another useful building block for a wide range of applications, and is readily implemented on top of a DHT-based infrastructure [20].

The next section describes how these building block functions may be extended and composed to implement a rich set of applications and features.

3.5 Extensibility and open APIs

The extensibility of our platform is critical to enabling service providers to innovate and experiment with new functionality. While any programmable platform is by definition extensible, we have taken a number of steps to make it *easy* to extend the platform. These steps include:

- Well-defined APIs using Web services. All of the built-in functions described in the preceding section are made available as Web services. The APIs are formally described using WSDL (Web Service Definition Language), thus making it fairly straightforward for external applications to access the functions described above[6].

- Service registration and replication. New services and functions can be added to the system either through the addition of code running on the platform itself, or on external systems (see "foreign resources" below). These services can be published using a service descriptor mechanism. Service descriptors are stored in the DHT. The location in the DHT is determined by a hash of the service name, and the descriptor itself contains a list of URLs for instances of the service. Thus, for example, a new service could be implemented on some subset of the programmable overlay routers, and a descriptor for that service would be stored in the DHT, pointing to the nodes at which the service is available. This means that a popular service can be widely replicated, and consumers of the service can be directed to a nearby instance of the service using the proximity functionality. Note that this ability to replicate a service enables a service provider to experiment with new functionality and increase the amount of resources allocated to that function if it turns out to be popular.

- Support for "foreign resources". We use the term "foreign resources" to refer to functionality that is implemented on systems other than the overlay routers. For example, a service provider may wish to make use of functionality provided by some third party (e.g., persistent storage provided by Amazon Web Services [4]) or to access other services implemented elsewhere in his network. The overlay nodes provide a proxy function, to create the illusion (from the perspective of an end user) that these remote functions are actually implemented in the overlay routers. End users interact with the proxy function running on an overlay router, and the proxy function communicates with the remote server that implements the foreign resource. This provides a level of isolation between the foreign resources and the customers, and creates the appearance of a seamlessly integrated application.

  If multiple instances of a foreign resource are available (e.g., multiple data centers providing persistent storage), the proximity function can be used to select among the different instances.

These capabilities, combined with some virtualization support described below, considerably simplify the task of adding functionality to the network.

3.6 Virtualization

The programmable overlay routers can run code from a variety of sources. In addition to the "built-in" functions described above, service providers may develop their own code to run on the overlay nodes; they may also wish to execute code provided by third parties, content providers, etc.

[6]Our initial implementation uses SOAP; a RESTful implementation is on the roadmap.

In this environment, some degree of isolation between different pieces of code is clearly required. One option would be to run multiple virtual machines on a single programmable overlay router, providing strong isolation between code from different parties. This in turn could be achieved in a number of ways, such as a hypervisor environment, container-based virtualization similar to PlanetLab, or a high-level language-based VM such as Java. Soltesz et al. [22] discuss some of the trade-offs among these approaches in terms of isolation, performance, etc.

4. IMPLEMENTATION

We have implemented a prototype version of the architecture described above. Our implementation runs on generic server-class PC hardware (with copious amounts of disk and RAM) and is built on a standard Linux environment. It includes one or more instances of an embedded off-the-shelf Web application server that handles, among other things, processing of Web Services API requests and the upper, more complex layers of system infrastructure. Lower layers of the infrastructure, time-critical functions and data movement functions, such as the cache control daemon, the routing protocol daemons, the HTTP server, the DHT daemon, and streaming engines, are implemented in C for better efficiency.

We opted to use a Java VM to provide the application virtualization support described in the previous section, although we expect that a real-world deployment would quite likely require the higher level of isolation provided by container-based VMs or a hypervisor. Thus, if one wishes to extend the capabilities of the system by deploying code on the overlay router itself, that code today is written in Java. Of course, code written in any language may be deployed on external systems, and integrated into an application that uses the overlay router platform, by treating the external system as a Foreign Resource, as described above. Ultimately we expect that a range of language environments would need to be supported to fully realize the potential of the programmable overlay.

Management of both the system infrastructure and the deployed applications is achieved using Java Management Extensions (JMX). In addition to the "native" JMX management interface, the system also provides SNMP and a command line interface similar to Cisco IOS®. System management is implemented in a separate JVM. The JMX management API enables use of 3rd party node management tools and emerging open-source and commercial network monitoring tools.

4.1 An example application: Social TV

To illustrate how the capabilities of the overlay router have been used to rapidly deploy a novel application, we provide a quick overview of an application that we developed to demonstrate the system.

Figure 3 gives a high-level overview of a "social TV" application. From a user perspective, the application supports groups of users watching the video content while exchanging messages, recommendations, etc. with each other. It is implemented as a distributed application that has both client and server ("back end") components. The server components are deployed into a selected set of nodes and provide APIs utilized by clients. The APIs provide login, content browsing and search, program guide, user content upload and social functions, such as instant messaging, content rating, comments, etc.

Operation proceeds as follows. Initially, a user logs in to a specialized "portal node", which, after querying the proximity function, redirects the user to the nearest overlay router. The user downloads the necessary client software (a browser plug-in). The client software then proceeds to interact with software running on the

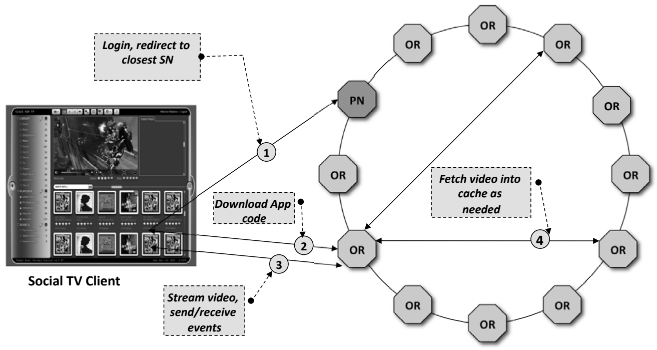
Login, redirect to closest SN
Social TV Client
Download App code
Stream video, send/receive events
Fetch video into cache as needed
1
2
3
4
PN
OR

Figure 3: Example Application

nearest overlay node. The overlay provides routing of client requests to backend servers, content caching and delivery, and uses the event notification service as the basis for instant messaging.

Observe that this application uses most of the built-in functions described in Section 3.4. The benefits of implementing the application in the overlay include: reduced latency; efficient use of network resources (popular content traverses the backbone only once per "point of presence", where it is then cached); better isolation between backend servers and clients (which communicate only with nearby overlay nodes); and the ability to scale up the application by replicating it on as many overlay nodes as necessary.

5. RELATED WORK

As already noted, the programmable overlay architecture draws heavily on prior research. In some respects, it is most similar to PlanetLab [16]. Our work builds on PlanetLab's programmable overlay model by adding a number of built-in functions (described in section 3.4) to form the foundation of the "service routing layer". In this respect, we also build on the foundation of OpenDHT [18], adding several other built-in functions above and beyond the DHT infrastructure. In contrast to both of these efforts, we have tried to integrate the overlay more closely with the network layer through the use of topological proximity and allowing the overlay to control network resources. By allowing select overlay nodes to passively listen to network layer routing, for example, we enable applications to make intelligent choices among replicated content, services, etc. based on the location of these replicas in the network. This enables service providers to take advantage of their network infrastructure in a way that would be challenging for more traditional overlay networks.

Akamai [13], along with other content distribution networks (CDNs), is a salient example of the effective use of overlays to add functionality to an IP network, providing benefits to a wide range of applications. These overlays are again not closely integrated with the service provider infrastructure, nor do they offer the programmability or open interfaces to facilitate easy extensibility by either the providers or by third parties.

Our work also suffers from a potential disadvantage compared to separately managed overlays. Most service provider networks have limited geographic scope, whereas overlays such as PlanetLab and Akamai have close to global footprints. Enabling some sort of peering among providers at the content or application level would seem to be a necessary step to address this concern, and is a focus of ongoing work. Our initial efforts in this area build on the "peering peer-to-peer" work of Balakrishnan *et al.*[5]

It is interesting to compare our approach with "cloud computing" as exemplified by services such as Amazon Web Services [4] and Google's App Engine [12]. In the sense that our programmable overlay provides a general-purpose computation environment that is located in "the cloud" and available to a wide range of users, it could be considered a form of cloud computing environment. (As widely noted, cloud computing is a term with many definitions — see [3] for some relatively clear discussion on this topic.) A main point of contrast between our approach and the mainstream cloud computing providers is the attempt to locate computation "inside" the network specifically to take advantage of the service provider's footprint and resources.

6. CONCLUDING REMARKS

Service providers face a difficult challenge if they attempt to move beyond the world of "commodity" packet transport by adding potentially valuable functionality to their networks. Not only do they risk adding costly functionality that applications cannot or do not use; the additional complexity may also make their networks less robust. Furthermore, they risk stifling innovation if new applications cannot be readily added without changes to the network.

Our proposed way out of this bind is the use of a programmable overlay, or a "service routing" layer. A stable, robust IP layer is preserved, while useful "in-network" functionality is performed by the overlay. By making the overlay devices fully programmable, we enable service providers to experiment with new functionality and to rapidly develop new applications and services using such functionality. By giving the overlay routers access to the network layer topology information, requests for replicated content or services can be routed to the nearest replica, thus providing better service to users and more efficient network utilization than in a conventional overlay. Overlay nodes also have the ability to control network resources (e.g. QoS settings), which may prove advantageous compared to separately managed overlays.

7. ACKNOWLEDGMENTS

A large team contributed to the architecture and implementation of the system described in this paper. We would like to thank the entire team including Manish Bhardwaj, Pawel Lipka, Steven Luong, David Powell, Stefano Previdi, Radovan Semancik, George Suwala, Albert Tian, Robert Varga and Tomasz Wysocki. We also thank Tony Bates and Mark Carroll for their support of the project. Finally, we thank the reviewers for their thoughtful and constructive comments on the paper.

8. REFERENCES

[1] O. Akonjang, A. Feldmann, S. Previdi, B. Davie, and D. Saucez. The PROXIDOR service. Internet draft, work in progress, March 2009.

[2] D. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. 18th ACM Symposium on Operating Systems Principles*, Banff, Alberta, Canada, October 2001.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical Report EECS-2009-28, EECS Dept., UC, Berkeley, 2009.

[4] Amazon Web Services home page. http://aws. amazon.com/.

[5] H. Balakrishnan, S. Shenker, and M. Walfish. Peering peer-to-peer providers. In *4th International Workshop on Peer-to-Peer Systems*, Ithaca, NY, Feb. 2005.

[6] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang. Improving traffic locality in bittorrent via biased neighbor selection. In *Proc. ICDCS*, 2006.

[7] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. *Request for Comments* 2475, Dec. 1998.

[8] M. S. Blumenthal and D. D. Clark. Rethinking the design of the Internet: the end-to-end arguments vs. the brave new world. *ACM Trans. Internet Techn.*, 1(1):70–109, 2001.

[9] B. Davie. Deployment experience with Differentiated Services. In *Proc. RIPQOS Workshop, SIGCOMM*, Karlsruhe, Aug 2003.

[10] DRAGON (dynamic resource allocation via GMPLS optical networks). http://dragon.east.isi.edu.

[11] H. Eriksson. Mbone: The multicast backbone. *Commun. ACM*, 37(8):54–60, 1994.

[12] Google App Engine. http://code.google. com/appengine/.

[13] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, 1997.

[14] National Research Council. *Looking Over the Fence at Networks*. National Academy Press, Washington D.C., 2001.

[15] J. Peterson and A. Cooper. Report from the IETF workshop on P2P infrastructure, May 28, 2008. Internet draft, work in progress, February 2009.

[16] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proc. HotNets-I*, Oct. 2002.

[17] L. L. Peterson, S. Shenker, and J. S. Turner. Overcoming the Internet impasse through virtualization. In *Proc. Hotnets-III*, Nov. 2004.

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

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

[20] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. In *Networked Group Communication, Third International COST264 Workshop (NGC'2001)*, Nov. 2001.

[21] J. Saltzer, D. Reed, and D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, Nov. 1984.

[22] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. C. Bavier, and L. L. Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *Proc. EuroSys*, Lisbon, Portugal, 2007.

[23] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. G. Liu, and A. Silberschatz. P4P: provider portal for applications. In *Proc. ACM SIGCOMM*, Seattle, WA, 2008.

[24] L. Zhang, S. Deering, D. Estrin, S. Schenker, and D. Zappala. RSVP: A new resource reservation protocol. *IEEE Network*, 7(9):8–18, Sept. 1993.