IEICE TRANS. COMMUN., VOL.E89–B, NO.9 SEPTEMBER 2006

INVITED PAPER *Special Section on Networking Technologies for Overlay Networks*

Towards a Service Oriented Internet

Jaideep CHANDRASHEKAR[†a)], **Zhi-Li ZHANG**[††b)], **Zhenhai DUAN**[†††c)], *and* **Y. Thomas HOU**[††††d)], ***Nonmembers***

SUMMARY Today's Internet remains faithful to its original design that dates back more than two decades. In spite of tremendous diversity in users, as well as the sheer variety of applications that it supports, it still provides a single, *basic*, service offering—unicast packet delivery. While this legacy architecture seemed adequate till recently, it cannot support the requirements of newer services and applications which are demanded by the growing, and increasingly sophisticated, user population. The traditional way to solve this impasse has been by using overlay networks to address individual requirements. This does not address the fundamental, underlying problem, i.e., the ossification of the Internet architecture. In this paper, we describe the design of a new *Service Oriented Internet* framework that enables the flexible and effective deployment of new applications and services. The framework we describe utilizes the existing IP network and presents the abstraction of a *service layer* that enables communication between *service* end-points and can better support requirements such as *availability*, *robustness*, *mobility*, etc., that are demanded by the newly emerging applications and services.

key words: *service-oriented, overlays, architecture*

1. Introduction

Over the course of the last two decades, the Internet has been transformed from an academic network to a ubiquitous communication infrastructure. Today's Internet is very much an integral part of society, underlying many key commercial, cultural and social activities. In stark contrast to this transformation, the architectural design of the Internet remain unchanged since its beginning. As was the case when first deployed, the Internet today supports the same (single) basic communication paradigm: *best-effort* unicast packet delivery, or stated differently—packet delivery between two fixed end interfaces. While this basic design has so far endured, this is no longer the case; the requirements of emerging services and applications are very hard to support (given the design). The tension arises from the fact that these new applications, which are quite hard to deploy (effectively) with today's Internet architecture, are crucial for the future growth and evolution of the Internet. The traditional communication paradigm, i.e., that of communication between fixed end hosts is somewhat outdated. For instance, when people perform a search engine lookup, say using Google, any of the thousands of Google servers can be be pressed into service; the identity of the particular server is irrelevant from the user's point of view. Thus, we argue that communication between service end-points is a more powerful paradigm than the traditional model.

However, supporting such a paradigm is very hard with today's Internet. Consequently, application requirements such as *availability*, *reliability*, *mobility*, *quality of service*, etc., are incredibly difficult to support. Take for instance the notion of *availability* in the context of a video streaming application. Supporting availability, in this case, requires the ability to migrate a streaming session to a different server during the lifetime of the video session. However, in the existing design, a binding between the client and a server needs to be established before any video frames are transferred. If the particular server were to fail, the session will abort, even though there may be other available servers that can serve up the same video frames.

The traditional way to address such requirements has been to deploy various *ad-hoc* mechanisms piecemeal. Examples include the deployment of content distribution networks, application specific overlay networks, etc. However, it is important to realize that these approaches are intended as a short term solution and do not address the underlying problem.

In this paper, we present a new architecture, which we term the "Service Oriented Internet" or SOI, that is best described as an *efficient, generic, unifying framework* to easily allow new services and applications to be deployed. SOI uses the underlying IP fabric to actually carry the bits and presents the abstraction of a *service layer*, that forwards packets between *service* end points. In the design of the SOI architecture, we introduce three key abstractions: (1) the notion of a *service cloud*, which is simply a collection of service entities that are deployed by a service provider. The simplest example would be a cooperating hierarchy of web proxy servers; (2) a new *two-level, location-independent* addressing scheme; and (3) a new abstract service layer that is used to forward packets (to the appropriate service end-points).

2. Service Oriented Internet

Overlay networks have emerged as an effective way to implement functionality which otherwise would require signif-

Manuscript received January 23, 2006.

[†]The author is with Intel Research/CTL, USA.
[††]The author is with University of Minnesota, USA.
[†††]The author is with Florida State University, USA.
[††††]The author is with Virginia Tech, USA.
a) E-mail: jaideep.chandrashekar@intel.com
b) E-mail: zhzhang@cs.umn.edu
c) E-mail: duan@cs.fsu.edu
d) E-mail: thou@vt.edu

DOI: 10.1093/ietcom/e89–b.9.2292

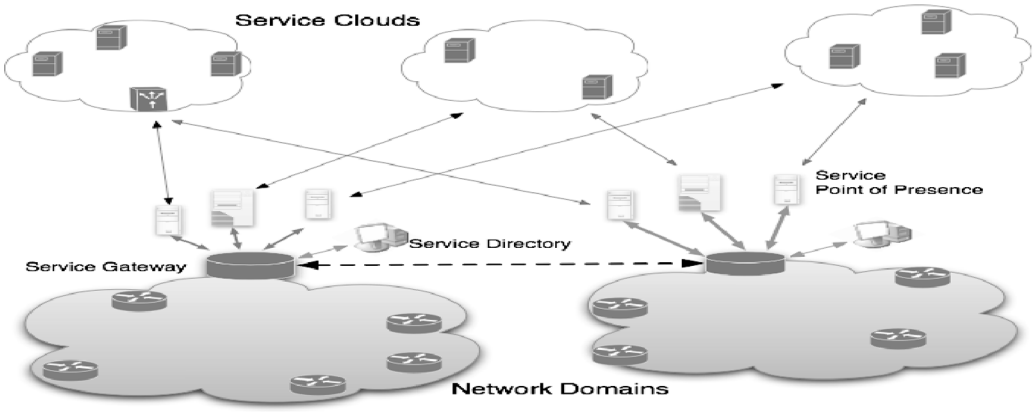


Fig. 1 Illustration of the SOI architecture.

icant change at the IP layer; they can be realized with very little overhead or infrastructure support. A set of end-nodes can form an overlay network without any additional support from the underlying network (or the ISP's carrying traffic between the nodes). However, this transparency comes at some cost. Firstly, by being completely *oblivious* of the underlying network layer, there are certain inefficiencies that cannot be avoided; very often, an *overlay neighbor* could actually be very far away in terms of the IP level network. Second, in most cases, overlays provide services (or realize applications) that are mandated on a well behaved underlying network. Presently, ISP's do not differentiate between traffic forwarded over an overlay and other traffic sharing the same network. Providing handles for the network to identify, and differentiate, packets carried over an overlay, is important from the view of supporting QoS requirements. Third, if we were to imagine a number of overlay networks on the same underlying network, each of them would have to replicate some common functions. For example, consider a situation where overlay *A* provides a streaming video service and overlay *B* is used for multicast video conferencing. Both overlays deliver real-time traffic; hence, both are likely to perform active measurements to support path selection and forwarding. Clearly, this replication is clearly inefficient. The obvious improvement is to decouple the active measurement component from the overlay operation and allow the different overlays to share a common *measurement infrastructure*. A similar idea has been discussed in [1], where the authors advocate a *routing underlay* that takes over the common tasks.

The underlying idea that lies behind our architecture is that services can be deployed as overlays, but to address the performance limitations of the overlays and to ensure support for the requirements of newer applications, we also need an underlying infrastructure which addresses the specific shortcomings of the traditional overlay paradigm. In the rest of the paper, we focus on the details of such an infrastructure.

We distinguish between *data transport networks*, which roughly correspond to the existing autonomous systems (and the IP networks), and *service overlay networks* (SON), each of which provides a well defined service (or set of services). The role of the data transport networks is to provide bit-pipes to the service overlay networks that ride on top. On the other hand, service overlay networks, provide specific value-added services to subscribers. For instance, we can think of VoIP service clouds, or content distribution clouds (think of the entire Akamai infrastructure as a cloud). Each of these are operated by service providers and generally have several distributed locations at which they interface with the data networks. Requests from clients are routed over the data network to the nearest (or most appropriate) point of entry into a particular service cloud, and subsequently served by some host inside the cloud. This high level description is depicted in Fig. 1, with the data networks shown towards the bottom of the figure and the service clouds near the top. Note that the framework defines how data is routed to the *border* of the service clouds, but not how it is handled (or transported) internally. Service providers can use arbitrary mechanisms that best satisfy their goals and purposes.

The *logical decoupling* between the data network domains and the service networks allows the independent evolution of each. This logical independence is an artifact of completely separating the addressing, routing and forwarding mechanisms in the two realms. A service cloud could implement each of these mechanisms as best suits its needs. There are three elements that are key to this separation, namely: a new *naming and addressing* scheme that is a significant departure from the existing IP addressing scheme, *service gateways (SG)*, and *service points-of-presence (S-PoP)*.

2.1 Key Abstractions

The SOI architecture is built on top of the existing IP infrastructure, and provides *a common platform* for flexibly deploying new Internet services and effectively supporting their diverse requirements. The architecture is based on three key abstractions, described below.

Service Clouds are collections of service entities (servers, proxies, caches, etc.) that are deployed over the Internet

to collectively and collaboratively provide a set of application/information services to users. Each of these clouds is a "virtual service overlay network" that is commonly owned and managed by a single provider or a consortium of application service providers, and it relies on the underlying IP data network domains for data delivery across the Internet[†]. Each service cloud has one or more entities that interface with the infrastructure; we refer to these as *service points-of-presence* (S-PoPs). Objects enter or exit a service cloud *only* via its S-PoPs.

Service-oriented addressing scheme: The central idea of the SOI architecture is a new two-level addressing scheme that provides *location-independent* identification of service clouds and objects within these clouds. Each service cloud is uniquely identified by a fixed-length *service id* (**sid**); and an object within a service cloud is specified by a (generally variable-length) *object id* (**oid**). The syntax and semantics of **sid** are *globally defined* and *centrally administered*, just like today's IP addresses (or rather network prefixes). On the other hand, the syntax and semantics of **oid** are defined by individual service providers (the mapping to host is scope limited to the service cloud), and thus are *service-specific*.

Service (routing/delivery) layer: Underlying the SOI architecture is a new *service layer* abstraction that, logically, sits just above the IP network layer in the protocol stack. Corresponding to the two-level $\langle$**sid**, **oid**$\rangle$ addressing scheme, the service layer introduces two new *network elements* with distinct functions: *service gateways* (SGs) and *service points-of-presence* (S-PoPs). SGs can be viewed as extensions of the underlying network domains, and are typically deployed at the edge of a network domain. They are responsible for routing and service delivery across network domains, i.e., overlay forwarding is performed by SGs using the **sid** in the address[††]. S-PoPs are the *interface* points of a service cloud with the network domains, and are thus logically a part of the service cloud (and hence are **oid**-aware). They are responsible for delivering objects within a service cloud. SGs and S-PoPs work together to support flexible end to end delivery.

Data destined for any particular service cloud must necessarily transit at least one SG (and S-PoP). Note that the former is owned and operated by individual network domains. This provides a way for the network domain to accurately identify and track traffic meant for the overlay networks.

3. SOI Architecture

In this section, we present the key components of the proposed SOI architecture, describe basic operations and walk through a typical transaction.

3.1 Addressing and Name Resolution

The name resolution stage returns return a two level $\langle$**sid**, **oid**$\rangle$ address. A key observation in the design is that the two identifiers are resolved independently and at different locations. Seen at a very high level, the **sid** mapping is performed external to the service cloud, while the **oid** mapping is performed inside the cloud. The advantages of this will become clear shortly.

Under the proposed SOI architecture, each application/information service provider who wants to deploy services over the Internet is assigned a single fixed-length (32 bit) service id, which is administered by a central authority. This is a departure from the IP addressing scheme, where a "cloud" (network domain) is assigned a contiguous range of addresses (address block or network prefix). Each service cloud can be *roughly* thought of as corresponding to an organization currently having a second tier (e.g., yahoo.com, msn.com, real.com) or third tier (e.g., shop.msn.com, nu.ac.cn) domain name[†††]. Such domain names will be retained in our SOI architecture as the names of service clouds, and are referred to as *service names*. To resolve the service name of a service cloud to its assigned **sid**, we can reuse the current DNS infrastructure (extending it so that names resolve to **sid**'s), or build a similar *service name resolution* system. The specific details of the service name resolution are out of the scope of this paper. It is important to note that caching the mappings locally requires only a small amount of memory—the number of service names is significantly smaller than the number of domain names in the current DNS system, and moreover, service-name-to-**sid** mappings are essentially *static*. Hence, service name resolution can be done with very little overhead (on the shared infrastructure).

In contrast to the **sid** space, the **oid** space is defined by each individual service cloud, with its own syntax and semantics. This gives each service cloud the most flexibility and efficiency for defining its own *object naming and addressing* system. It also offloads many service-specific functions (e.g., object resolution, internal routing, load balancing, etc.) to individual service clouds, which leads to a socially optimal solution. Providers that want more complicated mechanisms to perform the name resolution are free to do so, but only within the cloud. In addition, hiding the syntax and semantics of a service cloud's **oid** space from outsiders makes it more secure. This makes it very difficult for an attacker to lanch a DoS attack targeting a particular server, since the *corresponding* **oid** *can be dynamically remapped.*

Service Layer: For convenience, we refer to a service-layer protocol data unit as a *service object*. Figure 2 shows an abstract representation of a service object header. The

[†]Note that the separation between data transport domains and service clouds is purely logical. It is possible, though not required, that the service cloud forwards data internally over the existing IP infrastructure.

[††]Note that the IP address is still used to forward packets between neighboring SG's.

[†††]This is not a strict requirement, and is just a suggestion that reflects our belief that most current service providers fall into these categories.

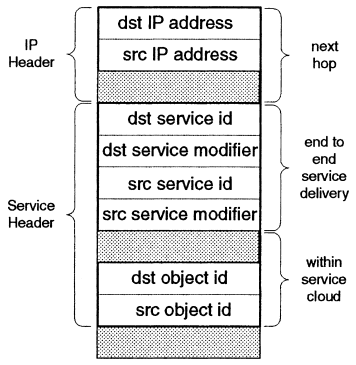


Fig. 2 Service object header format.

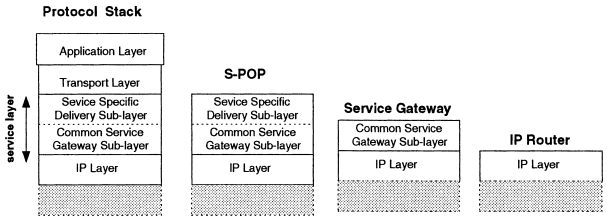


Fig. 3 Service layer and the SOI protocol stack.

header is partitioned into two logical sections, the **sid** part and **oid** part. Associated with both destination **sid** (**dst sid**) and source **sid** (**src sid**) is an additional 32-bit *service modifier*, which is defined by a service cloud to influence the forwarding of service objects. The service modifier contains two types of information: *S-PoP attribute* and *service attribute* (see Fig. 5 for an example). The S-PoP attribute describes the properties of S-PoPs, and in general contains two sub-fields, an *S-PoP level* and an *S-PoP id*. For example, using S-PoP attributes, a service cloud can organize its S-PoPs in a certain hierarchy to best meet its needs[†]. The service attributes are used to indicate a preference for different service classes, next hops, etc. Multiple service attribute *sub-fields* can be defined as appropriate. We illustrate this with an example in Sect. 4.

When a service object is generated, both the **sid** and **oid** parts of the header are filled *appropriately* by an application program (e.g., a browser). Figure 3 shows the relative position of the *service layer* in the protocol stack. Also shown are the layers of the stack that are interpreted by the different entities along the path. This should clarify that the service layer lies above the IP layer, and is independent of it. The service layer consists of two sub-layers: the *common service gateway* layer where only **sid**'s dictate how objects are forwarded among service clouds; and the *service-specific delivery* layer where **oid**'s are used in the forwarding decision (inside a service cloud).

Service Gateway: The data plane function of an SG is to forward a service object to an appropriate next-hop on the path to the destined service cloud (either an adjacent S-PoP, or another SG), using the **dst sid** (or perhaps both **dst sid** and **src sid**) and associated service modifier(s). For this purpose, each SG maintains a *service routing* table (similar to an IP routing table), constructed by participating in the *service gateway routing protocol* (SGRP), the control plane function of an SG. The service routing table contains mappings from a **dst sid** (and, if specified, an associated service modifier) to a next-hop SG/S-PoP (specified by IP address). From operational stand point of view, we expect the SGs to be deployed by the Autonomous Systems.

Service Point-of-Presence: An S-PoP plays two major roles: 1) it cooperates with SGs to route and forward service objects to/from the service cloud it proxies for; and 2) it cooperates with other S-PoPs in the service cloud to route and forward a service object within the service cloud. The latter role is determined by the *service-specific* routing protocol and forwarding mechanisms employed by the service cloud. The internal operation of the service cloud will not be addressed here, but an example is discussed in Sect. 4.

Service Gateway Routing Protocol: This protocol is responsible for constructing the forwarding (or service routing) tables on all the Service Gateways. The protocol is similar in scope to the Border Gateway Protocol [2] in the sense that it distributes "reachability." BGP distributes reachability to end network domains identified by prefixes; in contrast, SGRP distributes reachability to service clouds, identified by service id's. While we use BGP as a starting point in the design of SGRP, we incorporate further design principles that help avoid some of the associated problems: slow convergence, lack of support for traffic engineering, excessive routing churn, scalability, etc. In the rest of this section, we briefly describe the key aspects of SGRP.

At a very high level, SGRP involves three distinct operations: first, when a new S-PoP is deployed, it needs to *register* with nearby SG(s); this inserts a direct entry in the SG's forwarding tables; second, the SG that receives the registration can now *deliver* packets meant for the particular service cloud and this mapping—between the SG and the service id corresponding to the newly deployed S-PoP—needs to be propagated to all the other SG's; and third, the SG's exchange topology state messages to construct and maintain a *graph* of SG's over which SGRP messages are exchanged. In the following, we discuss each of these distinct operations.

S-PoP registration and advertisement: When a new S-PoP of a service cloud is deployed in the Internet, it must announce its presence so that SG's may begin to direct traffic (specifically service pdu's) to the said S-PoP. This is done by the S-PoP registering itself with SGs that it is adjacent to[††]. In the registration process, the S-PoP will tell the nearby

[†]This is but one possible interpretation. Since the SG does not need to understand the exact semantics of the modifiers, the service cloud can define them appropriately.

[††]This adjacency is *logical* and not physical; messages are exchanged over the IP network and might go over multiple router hops. However, it is reasonable to expect that service providers will deploy S-PoP's *near* existing SG's.

SGs about two things: the **sid** of the service cloud it is proxying, and a set of *supported service modifiers*. The latter describe specific capabilities satisfied by the said S-PoP (distinct S-PoPs may support different capabilities for the same service). For instance, consider the example of accessing web content over a cellphone. Clearly, the content will have to be customized to the (minimal) cellphone interface and requires client specific behavior from the web server. Service modifiers can be used to differentiate between S-PoP's that can serve content to cellphones, and those that cannot. Then, requests originating from cellphones can be directed to the appropriate servers. Thus, the S-PoP registration creates a mapping between the SG, the service id of the cloud and the service modifiers. Service pdu's will be forwarded to the S-PoP, by the SG, only if the service modifiers match the published capabilities.

Importantly, the service modifiers are opaque to the SGs, i.e., SGs do not associate any semantic meaning with the registered modifiers. They simply treat them as "patterns," or regular expressions, that are matched as a condition to forward packets to a particular S-PoP. The single exception to this case is the *null* service modifier: this particular expression matches everything, i.e., any traffic (destined to the service cloud) can be forwarded to an S-PoP that publishes a null service modifier. This "opaqueness" allows service providers to associate a range of forwarding behaviors using distinct modifiers.

Service reachability propagation: After the initial registration, the SGs have a forwarding pointer to the local S-PoP. This information is then disseminated to the other SGs in the network using *service reachability advertisements*. SRA's are constructed based on the registration information from the adjacent S-PoPs. Simply, an SRA specifies all the service clouds that can be reached via a particular SG (which *originates* the SRA). In other words, it contains an IP address (for the SG) and a set of service id's. SRAs are disseminated over the "network" of SGs using a form of reliable flooding; each SRA received at an SG inserts a pointer into the SG's (service) forwarding table, corresponding to a set of service ids being reachable through the neighbor the SRA was received from. Importantly, an SG may receive the SRA's for a service id from multiple neighbors; this causes multiple entries, perhaps supporting distinct service modifiers, to be inserted into the SG's forwarding table. An SG forwards SRAs that it receives, perhaps after aggregating (or even filtering) them. This form of information hiding is key in reducing the number of updates to be sent: an SG with multiple paths to reach a service cloud need not generate updates for changes to few of them; it needs to do so only when it knows of no other paths.

Importantly, we decouple *reachability*, as described by particular SRAs, from *topology*. The former simply qualifies a particular S-PoP (and the adjacent SG) to receive traffic for a service id, while the latter is concerned with the existence of a path to the SG in question. Reachability and topology, which are conflated in the case of BGP, change at very different time scales: topology changes are likely to result from failures (and repairs) between ASes, while reachability changes are due to S-PoP failures or administrative changes. Recent studies of global BGP routing reveal that the large majority of updates correspond to events of the latter class, while link failures between ASes is a far rarer event. Significantly, as is also pointed out in [3], decoupling these notions significantly decreases churn. The implicit principle in SGRP is to aggressively distribute reachability changes; service reachability messages disseminated using a form of reliable flooding. Topology changes are distributed more conservatively.

SG topology map construction: Messages and data objects in the service layer, i.e., between SG's, are forwarded over a *logical* SG graph; logical because SGs bear IP addresses and are organized into an overlay network on top of the IP network. SG's establish adjacencies by exchanging messages with each other. Note that since these may be over multiple IP hops, the necessary address information may have to be distributed in other ways: either statically configured or distributed using BGP announcements. However, once SGs establish the necessary adjacencies, they exchange *topology state advertisements* to construct and maintain the logical SG graph. The connectivity information described by these messages will include, in addition to the status of the adjacency, attributes describing dynamic properties such as delay, effective bandwidth, etc. Importantly, and in contrast to BGP, the protocol is soft-state; periodic messages update the dynamic attributes of the adjacencies.

A very important consideration in designing the protocol are the existing commercial relationships between ASes. A consequence of this is that an SG might have only an approximate topology map. This may lead to inefficient choices when the end SG, for a packet being forwarded, is far away. However, this is less of a concern when S-PoP's are widely deployed and there is a good chance of a nearby S-PoP being available to receive the packets.

4. Example

In this section we use an example, namely multimedia content distribution, to demonstrate the key features of our architecture. This example is particularly relevant because the service provider must support a range of service-types and object instances to be delivered over the same infrastructure. In addition, such a service would benefit from flexibility in the SGs to forward traffic over a set of next-hops. The *service modifiers* that we describe allow just such a capability; next-hops may be associated with specific modifiers, or alternatively, an SG may forward the same packet to multiple nexthops (to support *swarm* style forwarding). In general, the service provide may dictate specific forwarding behavior (by controlling how service modifiers are announced by S-PoPs).

Consider a service cloud that provides multimedia content delivery services. To support such an application effec-

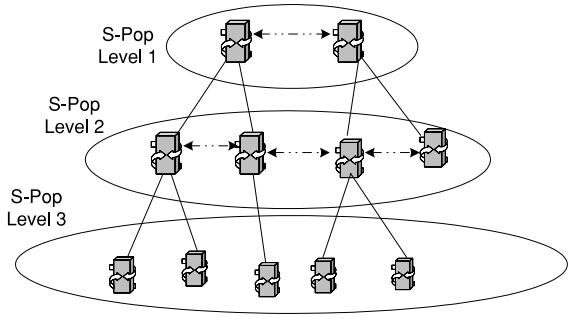


Fig. 4 A three-level S-PoP hierarchy.

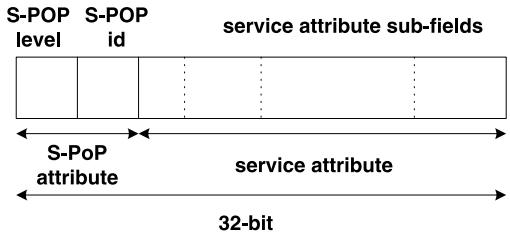


Fig. 5 Service modifier.

tively, the service cloud deploys a collection of S-PoPs organized in a 3-level hierarchy as depicted in Fig. 4. At the top of the hierarchy (level 1) are central S-PoPs, which are the front-ends to replicated *central* servers with a *complete* object repository. The intermediate level (level 2) are regional S-PoPs which are the front-ends to proxy servers that have a *partially replicated* object repository. At the bottom level (level 3) are local S-PoPs which are the front-ends for *local cache* servers. The local cache servers are only deployed inside network domains with large user bases. Hence not all level-2 S-PoPs have level-3 S-PoPs attached. The service cloud uses a one-byte field to specify the S-PoP attribute (see Fig. 5), of which a 2-bit sub-field indicates the S-PoP level and a 6-bit sub-field indicates the S-PoP id within a level. S-PoP level 0 and S-PoP id 0 are *default* values, which are used to represent *wild-card* matching.

To efficiently deliver its content using the S-PoP hierarchy, the service cloud defines a 2-bit *service attribute* sub-field to specify the *cacheability* of its content: popular (i.e., highly cacheable), normal (cacheable, the *default* value), rare (cacheable, but generally not cached), and dynamic (*not* cacheable). Popular content is preferably serviced from a local cache via a level-3 S-PoP if there is one close by, otherwise via a level-2 S-PoP. Normal content is generally serviced from a proxy server via level-2 S-PoP, while rare content from a central server via a level-1 S-PoP. Request for dynamic content is preferably processed by a proxy server via a level-2 S-PoP, which is responsible for forming the dynamic content, retrieving appropriate content from a central server if necessary. These guidelines for content delivery can be represented as a set of bit-pattern matching rules using the *S-PoP level* and the *cacheability* service attribute sub-field.

The S-PoPs register with the neighboring SGs of the underlying network domains, and advertise their presence and service capabilities (represented by a set of bit-patterns for the service modifiers it can handle). SGs formulate service reachability advertisements (SRAs) for the service cloud and propagate them (perhaps after filtering or aggregating. From SRAs that it receives, an SG builds entries in its (service) routing table. It should be emphasized that SGs do *not* need to understand the *syntax* and *semantics* of service modifiers defined by individual service clouds. All that is required is *the ability to manipulate regular expressions and perform table look-ups*.

The cacheability service attribute of content can be embedded in an HTML (or XML) page publicized by the service cloud, and filled accordingly by a client program when a request is generated. Upon receiving a request for a popular object of the service cloud, an SG will forward it to a nearby level-3 S-PoP (a local cache), if one exists. On the other hand, requests for other content will always be forwarded to a level-2 S-PoP, or a level-1 S-PoP if there is one close by. If a request for a popular object cannot be satisfied by a local cache (i.e., a *cache miss*), the level-3 S-PoP will automatically re-direct the request to a nearby level-2 S-PoP by changing the value of the S-PoP level sub-field from 3 to 2, and forwarding it to a nearby SG. If a level-3 S-PoP fails, a nearby SG, upon learning of the failure, will cease forwarding requests to it, and instead will forward them to a nearby level-2 S-PoP. In case of a level-2 S-PoP failure, an SG can automatically forward requests to another level-2 or level-1 S-POP. In addition, an *overloaded* level-2 S-PoP can perform load-balancing by re-directing requests to a *lightly-loaded* level 2 S-PoP by specifying its S-PoP id (instead of the default value 0) in the S-PoP id sub-field.

5. Related Work

We introduce the abstraction of a *service layer* that takes care of the service delivery from end to end. A somewhat similar notion is described in [4] where the authors advocate a "content layer" that forwards packets based on the resource name (that will be carried in packets). Given that names are generally unconstrained in length, this is somewhat unrealistic. There has been considerable research carried out in the area of using overlay networks to realize applications that are otherwise hard to deploy *natively*; for instance, multicast [5], [6], multimedia broadcast distribution [7], resilient routing [8] and even content distribution. However, these suffer from scalability and performance issues. Our architecture provides a way to address these shortcomings by means of a underlying substrate that will allow these applications to scale.

The idea of supporting QoS over the Internet by means of overlays is discussed in [9], [10]. Such an idea fits very well into our framework, and suggest possible ways of deploying overlays that require QoS support such as multimedia delivery, VoIP etc.

Perhaps the idea that comes closest to ours is that of

i3 [11]. In this work, the overlay paradigm is taken further to provide a common "indirection infrastructure" that is interposed between the two parties in a transaction. This indirection decouples the sender and receiver—which enables essential service primitives such as multicast, anycast, host mobility etc. Our own work (in comparison) is broader in scope and addresses a different set of problems.

Our work does not address the issue of how routing and forwarding are performed inside individual service clouds. In fact, this should be seen as a feature of our design: individual clouds are free to design and deploy their own mechanisms internally, completely unfettered by how packets are forwarded externally. We do note however, that there exist several well studied methods that may be adopted for this purpose [12]–[14].

6. Conclusion

In this paper, we highlighted the inadequacies of the current Internet design to satisfy the requirements of emerging Internet applications. The traditional way to satisfy these requirements by using overlay networks. However, as discussed, overlay networks fail to address the design shortcomings of the Internet. While it is important for the future evolution of the Internet to facilitate the deployment of these applications, it is impractical to do away with the current Internet design and start over. The *SOI* architecture that we describe in this paper provides a compromise between these two choices. It reuses the existing IP infrastructure, but at the same time provides the required abstractions that allow requirements such as availability, robustness, mobility and quality of service to be supported. The framework enables the easy deployment of new applications and services that cannot be supported within the confines of the current Internet design. A significant development since we embarked upon this work is the maturity of the PlanetLab infrastructure, which is intended to be an open research platform for deploying and testing internet-scale services [15]. At the present time, we are investigating the possibility of deploying our framework on the PlanetLab network.

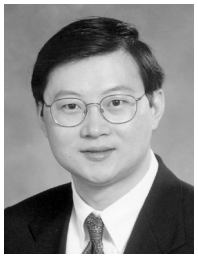
References

[1] A. Nakao, L. Peterson, and A. Bavier, "A routing underlay for overlay networks," Proc. ACM SIGCOMM 2003, pp.11–18, ACM Press, New York, NY, USA, 2003.
[2] Y. Rekhter and T. Li, "A Border Gateway Protocol 4 (BGP-4)," RFC 1771, March 1995.
[3] L. Subramanian, M. Caesar, C.T. Ee, M. Handley, M. Mao, S. Shenker, and I. Stoica, "HLP: A next generation inter-domain routing protocol," Proc. ACM SIGCOMM 2005, pp.13–24, ACM Press, New York, NY, USA, 2005.
[4] M. Gritter and D.R. Cheriton, "An architecture for content routing support in the Internet," Proc. 3rd USENIX Symposium on Internet Technologies and Systems, pp.37–48, March 2001.
[5] Y.H. Chu, S.G. Rao, and H. Zhang, "A case for end system multicast," Proc. ACM SIGMETRICS 2000, pp.1–12, ACM Press, New York, NY, USA, 2000.
[6] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," Proc. ACM SIGCOMM 2002, pp.205–217, ACM Press, New York, NY, USA, 2002.
[7] Y. Chawathe, Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service, Ph.D. Thesis, University of California, Berkeley, 2000.
[8] D.G. Andersen, H. Balakrishnan, M.F. Kaashoek, and R. Morris, "Resilient overlay networks," SOSP, pp.131–145, Oct. 2001.
[9] Z. Duan, Z.L. Zhang, and Y.T. Hou, "Service overlay networks: SLAs, QoS and bandwidth provisioning," ICNP, pp.334–343, IEEE Computer Society, 2002.
[10] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz, "OverQoS: Offering Internet QoS using overlays," 1st Hot-Nets Workshop, Princeton, NJ, Oct. 2002.
[11] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," Proc. ACM SIGCOMM 2002, pp.73–86, ACM Press, New York, NY, USA, 2002.
[12] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," Proc. ACM SIGCOMM 2001, pp.149–160, ACM Press, New York, NY, USA, 2001.
[13] S. Ratnaswamy, A Scalable Content-Addressable Network, Ph.D. Thesis, University of California, Berkeley, Oct. 2002.
[14] B. Wong, A. Slivkins, and E.G. Sirer, "Meridian: A lightweight network location service without virtual coordinates," Proc. ACM SIGCOMM 2005, pp.85–96, ACM Press, New York, NY, USA, 2005.
[15] "PlanetLab." http://www.planet-lab.org/

Jaideep Chandrashekar received a B.E. degree from Bangalore University, India, in 1997, and a Ph.D. from the University of Minnesota in December 2005. He is currently with Intel Research in Santa Clara, CA. His research interests include computer networks and distributed systems, especially Internet technologies, network routing and computer security. He is a member of IEEE and ACM.

Zhi-Li Zhang received a B.S. degree in computer science from Nanjing University, China, in 1986 and his M.S. and Ph.D. degrees in computer science from the University of Massachusetts in 1992 and 1997. In 1997 he joined the Computer Science and Engineering faculty at the University of Minnesota, where he is currently an Associate Professor. His research interests include computer communication and networks, especially the QoS issues in high-speed networks, multimedia and real-time systems, and modeling and performance evaluation of computer and communication systems. He is co-chair of IEEE/IFIP International Workshop on QoS 2004, and is co-chair of IEEE INFOCOM 2006 in Barcelona Spain.

Zhenhai Duan received the B.S. degree from Shandong University, China, in 1994, the M.S. degree from Beijing University, China, in 1997, and the Ph.D. degree from the University of Minnesota, in 2003, all in Computer Science. He is currently an Assistant Professor in the Computer Science Department at the Florida State University. His research interests include computer networks and multimedia communications, especially the scalable network resource control and management in the Internet, Internet routing protocols and service architectures, and networking security. He is a co-recipient of the 2002 IEEE ICNP Best Paper Award.



Y. Thomas Hou obtained his B.E. degree from the City College of New York in 1991, the M.S. degree from Columbia University in 1993, and the Ph.D. degree from Polytechnic University, Brooklyn, New York, in 1998, all in Electrical Engineering. From 1997 to 2002, He was a researcher at Fujitsu Laboratories of America, IP Networking Research Department, Sunnyvale, California. Since Fall 2002, he has been an Assistant Professor at Virginia Tech, the Bradley Department of Electrical and Computer Engineering, Blacksburg, Virginia. His current research interests include resource (spectrum) management and networking issues for SDR-enabled wireless networks, optimization and algorithm design for wireless ad hoc and sensor networks, and video communications over dynamic ad hoc networks. In recent past, he also had work on scalable architectures, protocols, and implementations for differentiated services Internet, service overlay networking, video streaming, and network bandwidth allocation policies and distributed flow control algorithms. He has published over 100 journal and conference papers in the above areas. Dr. Hou is a member of ACM and a senior member of IEEE.