Service Oriented Internet*

Jaideep Chandrashekar[1], Zhi-Li Zhang[1], Zhenhai Duan[1], and Y. Thomas Hou[2]

[1] University of Minnesota, Minneapolis, MN 55455, USA,
{jaideepc,zhzhang,duan}@cs.umn.edu
[2] Virginia Tech, Blacksburg, VA 24061, USA,
thou@vt.edu

Abstract. Effective service delivery capabilities are critical to the transformation of the Internet into a viable commercial infrastructure. At the present time, the architecture of the Internet is inadequately equipped to provide these capabilities. Traditionally, overlay networks have been proposed as a means of providing rich functionality at the upper layers. However, they suffer from their own drawbacks and do not constitute a perfect solution. In this paper, we propose a novel, overlay based *Service Oriented Internet* architecture that is meant to serve as a *flexible, unifying and scalable* platform for delivering services over the Internet. As part of this architecture, we introduce a new two-level addressing scheme and an associated *service layer*. We also describe the functionality of the new network elements that are introduced, namely *service gateway* and *service point-of-presence*, and subsequently discuss algorithms that are responsible for distributing *service reachability* across the overlay framework. We also present a few examples of application services that benefit significantly when deployed on our architecture.

1 Introduction

Over the last decade, the unanticipated popularity of applications such as the World Wide Web and E-mail has transformed the Internet into the *de facto* global information infrastructure that underlies much of today's commercial, social and cultural activities. People rely on the Internet for a variety of services essential to their daily lives, ranging from communications and information access to e-commerce and entertainment. Because of this, many new requirements of Internet services are more critical and urgent than ever before. These requirements include service *availability* and *reliability* (i.e., "always-on" services), *quality* and *security*. In spite of this, the Internet is still essentially a "best-effort" entity with end-to-end connectivity being its only service offering. In its present form, this architecture cannot adequately support the requirements of emerging services. Various *ad-hoc* mechanisms have been proposed and deployed to address these different issues. Examples include the deployment of CDNs (content distribution networks) and the widespread use of Network Address Translation. However, it is important to realize

* This work was supported in part by the National Science Foundation under the grants ANI-0073819, ITR-0085824, and CAREER Award NCR-9734428. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation

M.E. Orlowska et al. (Eds.): ICSOC 2003, LNCS 2910, pp. 543–558, 2003.

that *ad-hoc* solutions are by nature temporary short term measures – they do not address the underlying problem, and in some instances simply shift the bottlenecks elsewhere.

In this paper, we describe a new architecture — the "Service Oriented Internet" or SOI, which can be described as an *efficient, generic and unifying framework* for enabling the deployment of new services. In the design of the SOI architecture, we introduce three key abstractions: (1) the notion of a *service cloud*, which is 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 for forwarding packets to the appropriate service endpoints.

The main contributions of this paper are two-fold. We first present a critique of the current Internet architecture, highlighting obstacles in supporting future requirements of applications and services. Secondly, we outline an architectural framework that addresses these obstacles. We would like to think of our architecture as an evolutionary guideline that would enable the Internet to become a viable platform for the delivery of services. Specifically, the proposed architecture provides support for newer applications with service requirements beyond what can be currently realized. The architecture also introduces an economic framework which could be used to provide QoS support for applications that require it.

2 Service Oriented Internet

In recent times, *overlay networks* have emerged as an effective way to implement functionality which otherwise would require significant change at the IP layer. Such networks are appealing because they can be realized with very little infrastructure overhead. A set of end-nodes can decide to form an overlay, and cooperatively construct it, without any additional support from the network or the ISP's. However, this transparency comes at some cost. First, 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. Secondly, it might be that a particular overlay provides some service that is mandated on a well behaved underlying network. In the present case, ISP's do not differentiate between packets that will be routed to an overlay (or packets being forwarded on an overlay network) and other packets. This could be to the detriment of the application. For example, in the instance of a multicast overlay used for conferencing, it is reasonable to expect that the overall experience would be benefited by some prioritization of packets that are being exchanged on this overlay. If the packets meant for the overlay are similar to all the other transiting packets, there is no way to provide this service differentiation. Now, if we can imagine a situation where the ISP was in some sense "overlay-aware" and also that packets heading to the overlay could be identified, then it might be possible to ensure that a certain degree of service differentiation is provided, leading to a better experience on the overlay. For this to happen, in a realistic setting, the ISP would need some economic incentive to actually do this. Third, if we were to imagine a number of overlay networks in operation (over the same underlying network), each of the overlays would be replicating 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. Since both overlays are dealing with real-time traffic, they probably involve some active measurement component, running independent of each other. A far more efficient architecture would decouple the active measurement component from the overlay operation and allow the different overlays to share the the measurement infrastructure. A similar idea has been discussed in [1], where the authors advocate a *routing underlay* that takes over the common tasks.

The primary argument that we make through 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 shortcomings listed above.

In the rest of the paper, we focus on the details of the infrastructure and how it can be realized. To make the description complete, we also use examples to show how specific services can be realized and supported over the infrastructure.

2.1 Overview

In our architecture, we distinguish between the *data transport networks*, which roughly correspond to the existing autonomous systems (and the IP networks), and the *service overlay networks* (SON), each of which can be thought of as providing a well defined service. The role of the data transport networks is to provide bit-pipes as a service to the service overlay networks.

The service networks, on the other hand, are designed to provide specific value-added services to subscribers. These networks are operated by service providers and can be visualized as clouds which interface at multiple points with the data networks. Client requests are routed over the data network to the nearest (or most appropriate) point of entry into a particular service cloud. The client's request is then served from some host inside the cloud. This high level description is depicted in Figure 1, with the data networks shown towards the bottom of the figure and the service clouds near the top.

The *logical decoupling* between the data network domains and the service networks allows the independent evolution of each, allowing for flexible deployment of future Internet services while still supporting existing services. 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 the mechanisms independently, 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)*. Each of these will be described in this section.

2.2 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, as follows.

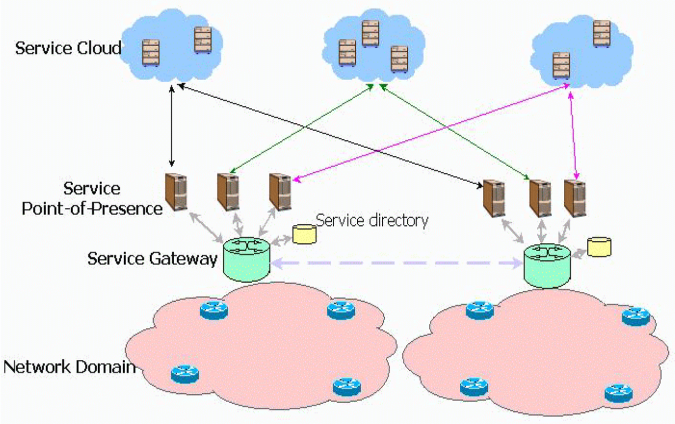

Fig. 1. Illustration of the SOI architecture.

Service Cloud abstraction: A service cloud is a collection of service entities (e.g., servers, proxies, caches and content switches) that are deployed over the Internet (typically at the network edges) to collectively and collaboratively provide a set of application/information services to users. It 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[1]. Each service cloud has one or more points interfacing with the Internet, referred to as the *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** is *globally defined* and *centrally administered*, just like today's IP addresses (or rather network prefixes); whereas the syntax and semantics of **oid** are defined by each individual service cloud, and thus are *service-specific*. Moreover, they are never interpreted outside the service cloud.

Service (routing/delivery) layer: Underlying the SOI architecture is a new *service layer* that resides above the IP network layer. Corresponding to the two-level $\langle \mathbf{sid}, \mathbf{oid} \rangle$ addressing scheme, the service layer comprises 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 who own and manage them, and are typically deployed at the edge of a network domain. They examine only the **sid** part of the two-level address and are responsible for routing and service delivery across

[1] Note that the separation between data transport domains and service clouds is purely logical. It may well be that the nodes in the service cloud use the IP networks to move data between them.

network domains. 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.

All data destined for a service cloud passes through a particular service gateway, which is owned and managed by the network domain. This provides a way for the network domain to accurately identify and track traffic meant for the overlay networks. Also, since each service cloud has a distinct identifier, it is possible for a service gateway to perform service differentiation. This provides a framework in which economic incentives can be applied — and is one of the key features of our architecture. A concrete example of how this can be used is presented in Section 4.3.

3 SOI Architecture

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

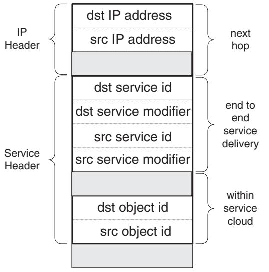

Fig. 2. Service object header format.

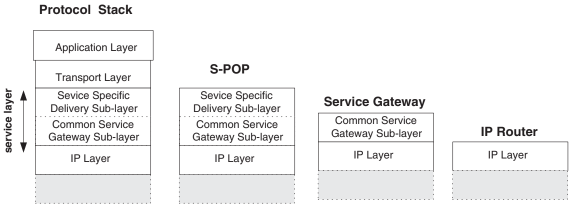
Protocol Stack
Application Layer
Transport Layer
Sevice Specific Delivery Sub-layer
Common Service Gateway Sub-layer
IP Layer
service layer
S-POP
Sevice Specific Delivery Sub-layer
Common Service Gateway Sub-layer
IP Layer
Service Gateway
Common Service Gateway Sub-layer
IP Layer
IP Router
IP Layer

Fig. 3. Service layer and the SOI protocol stack.

3.1 Addressing and Name Resolution

The name resolution should return a two level address corresponding to the $\langle \mathbf{sid}, \mathbf{oid} \rangle$ addressing scheme. One of the key features of our architecture is that the resolution separates these two address components. At a very high level, we could say that the **sid** mapping is performed external to the service cloud, while the **oid** mapping is performed inside the cloud. The advantages of this should 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 (just like IP addresses). 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[2]. 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 suffices to say that there are a number of existing approaches that can be readily adapted for our purpose. It is important to note that the mappings can be cached locally (with a reasonable sized cache), since the number of service names is significantly smaller than the number of domain names in the current DNS system, and service-name-to-**sid** mappings are essentially *static*. Hence, under the SOI architecture, 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 maximum flexibility and efficiency for defining its own *object naming and addressing* system to meet its business needs. It also off-loads many service-specific functions (e.g., object resolution, internal routing, load balancing, etc.) to individual service clouds. This mechanism promotes a socially optimal solution. Providers that want more complicated mechanisms to perform the name resolution are forced to hide the complexity inside the service clouds. 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 re-mapped.*

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 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 Figure 8 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 service delivery needs[3]. 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. One possible use of these attributes is explained in Section 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). However, to ensure security, we require that *the originating service cloud must verify the source* **sid** *of the object*

[2] This is not mandated by our architecture, and is just a suggestion that reflects the belief that most current service providers fall into these categories.

[3] It must be pointed out that 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 as appropriate.

before it passes outside the service cloud. In fact, we can even allow for in-flight **sid** resolution, i.e. the end-host sends a service object to the service gateway, the service gateway initiates the service name resolution, fills in the **sid** appropriately, and forwards the packet to the corresponding next-hop. These mechanisms enforce ingress filtering [2], and prevent address spoofing.

Figure 3 shows the relative position of the *service layer* in the protocol stack. Also shown in the figure are the layers of the stack that are interpreted by the different entities along the path. This should serve to clarify that the service layer lies above the IP layer and is completely independent of it.

The service layer consists of two sub-layers: the *common service gateway* layer where only **sid**'s are used to determine how to forward objects among service clouds; and the *service-specific delivery* layer where **oid**'s are used to decide how objects are delivered within 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 (this could either be an adjacent S-PoP, or another SG), using the **dst sid** (or both **dst sid** and **src sid**) and the associated service modifier(s). For this purpose, each SG maintains a *service routing* table (similar to an IP routing table), which is built by running 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 an infrastructure 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 a brief discussion of some of the existing possibilities are listed in Section 5.

To begin receiving service objects from the data networks, an S-PoP participates in SGRP, advertising its presence and properties to neighboring SGs. It builds a (partial) service routing table which maps the service id space to appropriate neighboring SGs, and uses it to forward service objects out of its service cloud.

3.2 Service Gateway Routing Protocol

This protocol is mainly responsible for constructing the forwarding tables on all of the Service Gateways. At a very high level, SGRP involves two distinct functions, each of which is described in turn. The first component involves the individual *S-PoPs* registering themselves with the local *SG*. This has the effect of making them eligible to receive traffic for the cloud they represent. The second component, which is somewhat similar to the BGP protocol, distributes this reachability information to all the other *SGs*.

S-PoP registration and advertisement: When a new S-PoP of a service cloud is deployed in the Internet, it must announce its availability to the rest of world. This is done by the S-PoP advertising its presence to and registering itself with the SGs it is (logically) adjacent to[4]. In the registration process, the S-PoP sends the nearby SGs the **sid** of the service cloud it represents and *a set of service modifiers* it supports.

The set of service modifiers essentially describes the capabilities of the S-PoP and tells the SGs exactly what kind of traffic it can handle (as a subset of the traffic that is destined for its service cloud)[5]. Only service objects with a service modifier matching one of the service modifiers advertised should be forwarded to the S-PoP. A *null* set means that the S-PoP can handle all traffic destined to its service cloud.One operational possibility is that the set of service modifiers is represented as an ordered list of *bit-pattern matching rules* (e.g., using regular expressions).

Note that there is *no* need for an SG to understand the syntax or semantics of the advertised service modifiers in order to perform the bit-pattern matching. This is important, since the syntax and semantics of the advertised service modifiers are defined by each individual service cloud, and thus are *service-specific*.

Propagating Service Reachability: This component of *SGRP* uses a path vector protocol model similar to that used in BGP [3]. However, we use a novel mechanism to limit *path exploration* which is the fundamental cause of the slow convergence in path vector protocols. Due to a lack of space, we are forced to make this discussion brief, so we present a very high level overview of the *Service Reachability Propogation* mechanism of *SGRP*. A more detailed account of this idea is presented in [4].

SGs perform a function similar to that of BGP routers, albeit propagating *service reachability* (as opposed to *network reachability*). Each *SG* "learns" of a number of paths from neighboring *SGs*, from which it selects a single *best path*. This best path is announced to all its neighbors (with the exception of the neighbor that the best path was learnt from). The key departure from BGP is that in our architecture, distinct *S-PoPs* could be associated with the same **sid**, with the effect that multiple *SGs* announce reachability for the same **sid**. In the following, we describe some key concepts that are central to our scheme.

First, we introduce the notion of a *fesn* (or forward edge sequence number), which is an integer value associated with an *SG-SG* edge[6] (and a specific **sid**). Consider two *SGs*, A and B, which are adjacent. Suppose that A is announcing reachability for **sid** d to *SG* B for the first time. It initializes an *fesn* that is directly associated with the neighbor B and the service identifier d. The adjective *forward* signifies that these numbers are in the context of outgoing edges (i.e. the direction that messages are sent). The value of the *fesn* is modified only by the "owner" (which in the current case is *SG* A). In particular, the value of the *fesn* is incremented when the edge between two *SGs* comes back up after a failure event, or when an *SG* re-advertises reachability for a **sid** (following a previous withdrawal) — *these are the only scenarios that could cause an* fesn *to change.* Note that an *fesn* is associated with a unique *SG-SG*, and essentially tracks the status of this edge over time. The monotonicity of the *fesn* allows an SG to determine if an advertisement is *new*.

[4] By this, we mean that the two *SGs* can talk to each other directly (at a layer above the network layer.)

[5] As an example, consider a content distribution cloud; here a particular S-PoP can declare via the service modifiers, that it can handle dynamic content.

[6] To simplify the description, we assume that there is at most one *SG* within an AS.

At any *SG* (say A), given an AS path P and a neighboring *SG* node, B, we associate with it a *fesnList*, which is the ordered list of *fesn*'s corresponding to *SG-SG* edges along the path from B to the destination. Note that for a given AS Path, the *fesnList* is unique. To make the distinction between a *path* and the *fesnList*, consider a simple example. Let SG A be adjacent to *SG*s B and C, to whom it is about to announce the same *selected best* path. The paths announced to B and C would be the same, but the associated *fesnList*s are distinct (since the *fesn*s that correspond to the edges $A - B$ and $A - C$ are distinct).

When an *SG* sends a service reachability announcement, it includes the **sid**, the AS Path, the associated *fesnList*, along with any service modifiers or additional attributes. If the *SG* is originating the reachability, as would happen if the **sid** was announced by a local *S-PoP*, the *fesnList* will only contain the single *fesn* (i.e. that of the outgoing edge.) On the other hand, if the announcement is being forwarded, i.e. the reachability was learnt from another *SG*, then the *fesn* (corresponding to the recipient of the announcement) is appended to the *fesnList* before being sent.

In the event that the edge between *SGs* A and B fails, or if A decides to stop accepting traffic from B for a **sid** d, then A generates a withdrawal message and sends it to B. The contents of this message include the **sid** being withdrawn, the path being withdrawn, and the *fesnList* associated with this path.

All service announcements and withdrawals carry the path being announced (or withdrawn) along with the associated *fesnList*. This is in contrast with the behavior in BGP, where withdrawal messages do not carry any path information.

The *fesnList* is used for two different purposes. First, it allows an *SG* to determine whether a message received corresponds to new information. By the monotonicity of the individual *fesn*s, only repair (or re-advertisement) events would cause the *fesnList* to change. Thus, an *SG* only needs to process the first message received with a different *fesnList* (from the one that it already knows about). Secondly, it allows an *SG* to correctly determine which paths to invalidate upon receiving a withdrawal. The *fesnList* for any two *SG* level paths are distinct even though the AS level paths might be the same. Thus, upon receiving a withdrawal, an *SG* can correctly invalidate *all paths* that it knows about which contain (as a prefix) the withdrawn path. This eliminates the problem of *path exploration* which causes the slow convergence in traditional path vector protocols.

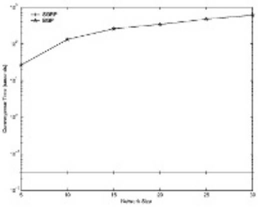

Fig. 4. Time complexity for Clique topologies

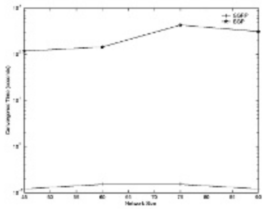

Fig. 5. Time Complexity for Power Law Random Graphs

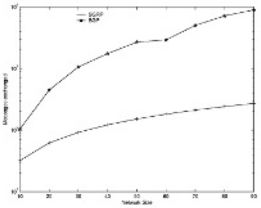

Fig. 6. Communication complexity for Random Network

When a withdrawal is received at $SG\ B$ that causes Bs *preferred* path to change, the (same) withdrawal is sent along with the announcement of the new path *in a single message*. Correspondingly, at the receiving node, the advertisement for a new path could have an attached withdrawal. In such a case, the withdrawal is processed first and subsequently the announced path is included in the path-set. On the other hand, if the received withdrawal causes all the available paths to be invalidated, indicating that there is no path to the destination, the withdrawal is propagated immediately. However, suppose only an announcement is received (without an accompanying withdrawal). Then the newly received path is included in the pathset and the best path recomputed. If the best path changes, then it is subsequently announced to the neighbors (along with the associated *fesnList*). Note that this announcement is subject to the hold timer constraint used to delay advertisements. Table 1 describes the step-by-step operations that are carried out when a withdrawal (or an announcement) is received at an *SG*.

In Figures. 4,5 and 6 we show the results of some simulations, carried out using the SSFNet simulation suite, comparing the performance of BGP and SGRP. In order to these protocols, we used only a single (unique) **sid** for each node. Each topology had nodes numbered from 0 through n. A special node, d (which originated some **sid**) was attached to node 0. At some time, d was disconnected from the network and two metrics were measured — convergence time and communication complexity. Figures 4,5 plot the size of the network (clique and power law topology, respectively) against the time taken (after d is disconnected from the network) for all the nodes to converge. In Figure 6, the x axis represents the size of the network (which is a random graph), while the y axis represents the number of messages that were exchanged in the network during the period of convergence. As can be seen in all the graphs, our scheme dramatically outperforms BGP for both of these metrics, and clearly demonstrates the advantage of limiting path exploration. We also carried out simulations with different topologies and failure scenarios, which we had to omit for lack of space. These results are presented in a related paper [4].

Table 1. Service Reachability Propogation

	`Service Withdrawal`	`Service Announcement`
1	validate the *fesnList*	validate the *fesnList*
2	update the pathset by marking dependent paths invalid	if withdrawal is defined, update the pathset by marking dependent paths invalid.
3		Include P into the pathset.
4	select best path	select best path
5	if best path is empty, then send withdrawal to all neighbors. Otherwise, (if best path changed) package best path and received withdrawal and send to all the neighbors	if best path changed, package best path and received withdrawal (if any), and send to all the neighbors.

4 Examples

In this section we present three generic situations in which our architecture provides a tangible benefit. In the first example, we detail how a multimedia content delivery service would be supported in our architecture. This example demonstrates how a service cloud could, by associating some consistent semantics with the service modifiers, dictate the forwarding behavior at the Service Gateways. The second application presented is that of an integrated communications service. This examples illustrates the powerful mechanisms that can be supported inside a service cloud, without any of the complexity being visible externally. The last example presented is that of a large scale VoIP service. This was chosen to illustrate the utility of the economic framework that is enabled by our architecture, and which can be used to provide QoS support to applications that require it.

4.1 Multimedia Content Delivery

Consider a service cloud that provides multimedia content delivery services. To support its services effectively, the service cloud deploys a collection of S-PoPs organized in a 3-level hierarchy as depicted in Figure 7. 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 Figure 8), 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.

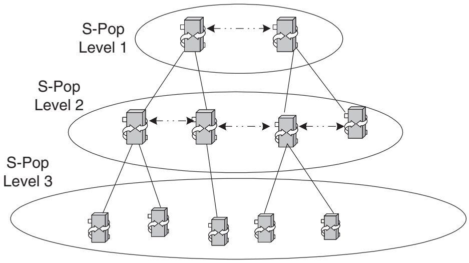

Fig. 7. A three-level S-PoP hierarchy.

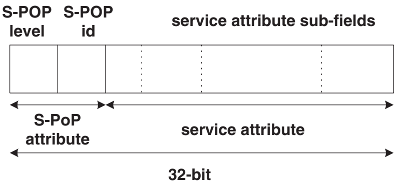

Fig. 8. Service modifier.

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. *Regular expressions*, for instance, can be used for specifying such bit-pattern matching rules.

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-pattern matching rules which specify the service modifiers it can handle). SGs formulate service reachability advertisements (SRAs) for the service cloud and propagate them after performing appropriate filtering and aggregation processing. From SRAs received, each SG can build corresponding entries in its service routing table. We want to emphasize that SGs do *not* need to understand the *syntax* and *semantics* of the service modifiers defined by the service clouds[7]. All that is required is *the ability to manipulate regular expressions and perform table look-ups*. This is a key feature of our SOI architecture, as it allows for simple and efficient SG implementation, while providing maximum flexibility in supporting diverse Internet services.

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. 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 *light-loaded* level-2 S-PoP by specifying its S-PoP id (instead of the default value 0) in the S-PoP id sub-field.

4.2 Integrated Communications

We use this example to illustrate the advantages of the location-independent two-level addressing scheme in supporting user mobility and a variety of other value-added services such as personalized communications, anycasting and multicasting. The focus will be on the functions of S-PoPs.

Consider a service cloud offering integrated communication services to its users (subscribers of the services). It deploys a collection of S-PoPs organized in a flat, “peer-to-peer” architecture. It allows users to define a number of communication modes such

[7] In the present example, the SG is oblivious of the internal S-PoP hierarchy and content classification i.e. cacheability.

as *urgent*, *normal*, and *batch* modes. In the *urgent* mode a subscriber wants to be notified via *all possible* communication means such as cell phone, office/home phones at various locations, email and instant messaging, etc. In the normal mode, communication requests will be directed to *one of* its communication means, e.g., the one the user has turned on at a given moment, or if more than one have been turned on, the preferred one (say the cell phone.) If it is not picked up immediately, an “alert” message is generated afterwards and sent to all communication means within a pre-specified time (for example, 5 minutes). In the batch mode, communication requests are directed to the user’s multimedia email-box, which the user may check at leisure.

Corresponding to these communication modes, separate user *object ids* (*oids*) will be created for each user. When logging on to the service cloud (e.g., by turning on one or more communication devices), a user registers with a nearby S-PoP that tracks and records the locations of his/her communication devices. The user can (pre-)specify the set of users he/she is willing to communicate in the different modes, making the corresponding user **oid**’s only visible to the specified users. For instance, the urgent mode is only for close family members and friends; normal mode for colleagues and acquaintances, etc. The personalized communication modes can be implemented by performing the appropriate mapping between the logical **oid** and the physical devices. For example, an S-PoP receiving an urgent call from a close family member will forward the call to the S-PoPs to which the user’s communication devices are connected. The S-PoPs then direct the call to these devices, transforming the message formats if necessary, e.g., a voice call to the cell phone, an instant text message to the PDA, a voice email to the email account, etc. As the user moves around, the S-PoPs will update various mappings between the logical **oid**’s and physical devices.

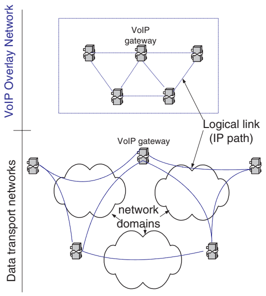

Fig. 9. An overlay network providing a VoIP service.

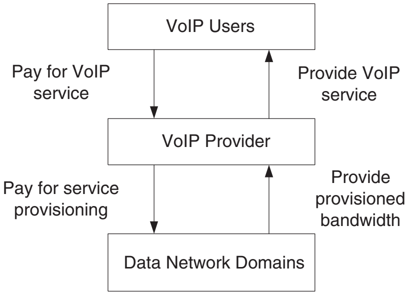

Fig. 10. Revenue flows and business relationships in the VoIP overlay network.

4.3 Voice over IP Service

This example particularly demonstrates how a service with QoS requirements can be deployed using our architecture. A provider wishing to deploy the VoIP service would first enter into agreements with the underlying data networks, and by means of SLA’s or some such enforcing mechanism, obtain some network pipes between the regions where the service would be introduced. One possible service identifier space allocation could map users (or the particular device) to a unique **sid**. With the infrastructure in place, users of the service would place calls by identifying the person that will receive the call. This request is routed to the service provider, and the lookup service resolves this to a particular **sid**. Once the end points are mapped to the particular **sid**'s, the flow of packets corresponding to the call are routed to the closest S-PoP (in both directions). Once inside the service cloud, the packets are forwarded along the appropriate (pre-provisioned path) towards the S-PoP that is closest to the other end-point. A high level illustration of this is provided in Figure 9. In fact, the S-PoPs described in our architecture are very similar in functionality (in the context of a VoIP service) with the *gatekeeper* entities in the established **H.323** standard.

By means of purchasing pre-provisioned bandwidth pipes, the provider can get around the problem of transiting peering points, which have been anecdotally identified as the bottleneck areas. In the absence of the provider entering into agreements with both network domains (on two sides of a network boundary), there would have been no incentive for the network providers to provide a better service for packets being transited for the VoIP provider. [5] presents an architecture for providing Internet QoS using overlay networks, which can be used by the VoIP provider to construct the VoIP infrastructure. In fact, we argue that one of the benefits of our architecture is that it puts in place a framework over which bilateral agreements can be implemented. The economic relationships are illustrated in Figure 10.

The above examples help illustrate the *versatility* and *flexibility* afforded by our proposed SOI architecture. We also see that, with the help of the new location-independent two-level $\langle \mathbf{sid}, \mathbf{oid} \rangle$ addressing scheme as well as the SGRP protocol, this architecture provides better support for service availability, reliability and mobility. Furthermore, the SOI architecture facilitates the creation and deployment of new value-added services such as QoS-enhanced services, enabling rich business models and economic relations to be established among service clouds and network domains. Note in particular that, since service modifiers are defined by each individual service cloud, there is no need for a "global standard" such as IntServ [6] or DiffServ [7] – how the service is supported is decided by the *bilateral* service agreement between a service cloud and an underlying network domain. In addition, because of the location-independent addressing scheme, the "opaqueness" of service clouds, and the ease and efficacy of object verification and filtering based on **sid** and service modifier fields, our SOI architecture also greatly improves *security*, effectively vitiating such threats as DDoS attacks and IP spoofing.

5 Related Work

In this section, we contrast our own architecture with similar research in the area. We introduce the abstraction of a *service layer* that takes care of the service delivery from end to end. A somewhat similar abstraction has been mentioned in [8] where the authors advance the notion of a "content layer". The key idea in this work is that packets carry the resource name rather than IP address, and the corresponding forwarding is performed based on the carried name. However, given the unconstrained sizes of names, it is not realistic to think that packet forwarding will ever be based on names.

There has been much effort put into supporting more advanced service primitives. Overlay networks have been the most dominant mechanism to provide support for these services. Recent proposals advocate the use of overlays to provide an extremely diverse range of applications such as multicast [9,10], multimedia broadcast distribution [11], resilient routing [12] and content distribution. However, these proposals suffer from scalability and performance issues native to the current overlay paradigm. Our architecture provides a way to address these issues by means of a underlying substrate that would allow these applications to scale.

Within the domain of overlay networks, one of the interesting research issues has to do with locating objects in the network. Most peer to peer (*P2P*) applications use either a centralized directory, such as Napster or specify a flooding algorithm to propogate the search through the overlay (as in the Gnutella network). These approaches do not scale well with the size of the overlay, and there have been a number of recent proposals that address this. The most prominent are the algorithms based on *Distributed Hash Tables* [13,14]. Though originally intended as a way to address the scaling problems in existing *P2P* networks, they are complimentary to our architecture, since the algorithms can be used inside a service cloud to locate objects.

The idea of supporting QoS over the Internet by means of overlays is discussed in [15]. 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* [16]. 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 slightly different set of problems.

6 Conclusions and Future Work

In this paper, we explored the inadequacies of the current Internet architecture — and its inability to satisfy the requirements of emerging applications, and also briefly described some capabilities that are currently lacking. It is our thesis that these capabilities are critical to the transformation of the Internet into a viable platform on which services can be delivered. The *SOI* architecture that we articulate in this paper has the capability to transform the Internet into a viable framework for the flexible deployment of new services. The architecture presented has the following properties.
1) Provides an infrastructure that can enable new Internet services that have requirements which cannot be currently satisfied —- such as reliability, security, service availability and QoS.
2) Leverages the existing IP infrastructure. It essentially introduces a service layer that is laid over the IP network layer, and which uses the IP network layer to transport the actual bits.
3) Extends the overlay network paradigm to be more efficient and scalable.

In our description, given the breadth of the architecture, we were forced to sacrifice detail when describing certain components of our architecture so as to present a coherent description in the limited space. At the same time, there are several details that were glossed over in the design of our architecture, such as the exact specifics of the packet forwarding, details of the service name resolution, and so on. In the future, we intend to flesh out the details of these components and construct a prototype of our architecture to evaluate its real world effectiveness.

References

1. Akihiro Nakao, Larry Peterson, A.B.: Routing underlay for overlay networks. In: Proc. ACM SIGCOMM, ACM Press (2003)
2. Ferguson, P., Senie, D.: RFC 2827: Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing. http://www.ietf.org/rfc/rfc2827.txt (2000) Best Current Practices.
3. Rekhter, Y., Li, T.: A Border Gateway Protocol 4 (BGP-4) (1995) RFC 1771.
4. Chandrashekar, J., Duan, Z., Zhang, Z.L., Krasky, J.: Limiting Path Exploration in Path Vector Protocols. Technical report, University of Minnesota (2003)
5. Subramanian, L., Stoica, I., Balakrishnan, H., Katz, R.H.: OverQoS: Offering internet QoS using overlays. In: First HotNets Workshop, Princeton, NJ (2002)
6. Braden, R., Clark, D., Shenker, S.: RFC 1633: integrated services in the internet architecture (1994)
7. Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W.: RFC 2475: an architecture for differentiated services (1998)
8. Gritter, M., Cheriton, D.R.: ‘an Architecture for Content Routing support in the Internet. In: USITS. (2001)
9. Chu, Y.H., Rao, S.G., Zhang, H.: A case for End System Multicast. In: Proc. ACM SIGMETRICS, ACM (2000)
10. Banerjee, S., Bhattacharjee, B., Kommareddy, C.: Scalable application layer multicast. In: SIGCOMM, ACM Press (2002)
11. Chawathe, Y.: Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service. PhD thesis, University of California, Berkeley (2000)
12. D. Andersen, H. Balakrishnan, M.K., Morris, R.: The case for resilient overlay networks. In: Proc. 8th Annual Workshop in Operating Systems. (2001)
13. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proc. ACM SIGCOMM, ACM Press (2001) 149–160
14. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. In: Proc. ACM SIGCOMM, ACM Press (2001)
15. Duan, Z., Zhang, Z.L., Hou, Y.T.: Service overlay networks: Slas, qos and bandwidth provisioning. In: Proc. International Conference on Network Protocols. (2002)
16. Stoica, I., Adkins, D., Zhuang, S., Shenker, S., Surana, S.: Internet indirection infrastructure. In: Proc. ACM SIGCOMM, ACM Press (2002) 73–86