Wide-Area Route Control for Distributed Services

Vytautas Valancius*, Nick Feamster*, Jennifer Rexford†, Akihiro Nakao‡
*Georgia Tech †Princeton University ‡The University of Tokyo

ABSTRACT

Many distributed services would benefit from control over the flow of traffic to and from their users, to offer better performance and higher reliability at a reasonable cost. Unfortunately, although today's cloud-computing platforms offer elastic computing and bandwidth resources, they do not give services control over wide-area routing. We propose replacing the data center's border router with a *Transit Portal* (TP) that gives each service the illusion of direct connectivity to upstream ISPs, without requiring each service to deploy hardware, acquire IP address space, or negotiate contracts with ISPs. Our TP prototype supports many layer-two connectivity mechanisms, amortizes memory and message overhead over multiple services, and protects the rest of the Internet from misconfigured and malicious applications. Our implementation extends and synthesizes open-source software components such as the Linux kernel and the Quagga routing daemon. We also implement a management plane based on the GENI control framework and couple this with our four-site TP deployment and Amazon EC2 facilities. Experiments with an anycast DNS application demonstrate the benefits the TP offers to distributed services.

1. Introduction

Cloud-based hosting platforms make computational resources a basic utility that can be expanded and contracted as needed [10, 26]. However, some distributed services need more than just computing and bandwidth resources—they need control over the *network*, particularly over the wide-area routes to and from their users. More flexible route control helps improve performance [7, 8, 12] and reduce operating costs [17]. For example, interactive applications like online gaming want to select low-latency paths to users, even if cheaper or higher-bandwidth paths are available. As another example, a service replicated in multiple locations may want to use IP anycast to receive traffic from clients and adjust where the address block is announced in response to server failures or shifts in load.

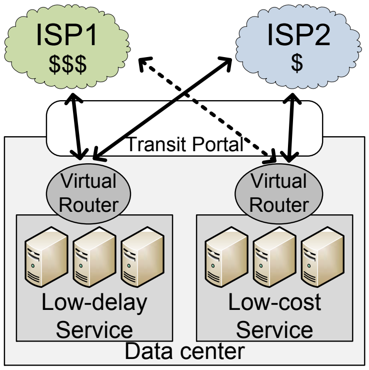


Figure 1: Connecting services though the Transit Portal.

Although flexible route control is commonplace for both content providers and transit networks, today's cloud-based services do not enjoy the same level of control over routing. Today, the people offering these kinds of distributed services have two equally unappealing options. On the one hand, they could build their own network footprint, including acquiring address space, negotiating contracts with ISPs, and installing and configuring routers. That is, they could essentially become network operators, at great expense and with little ability to expand their footprint on demand. On the other hand, they could contract with a hosting company and settle for whatever "one size fits all" routing decisions this company's routers make.

This missed opportunity is not for a lack of routing diversity at the data centers: for example, RouteViews shows that Amazon's Elastic Cloud Computing (EC2) has at least 58 upstream BGP peers for its Virginia data center and at least 17 peers at its Seattle data center [20]. Rather, cloud services are stuck with a "one size fits all" model because cloud providers select a single best route for all services, preventing cloud-based applications from having any control over wide-area routing.

To give hosted services control over wide-area routing, we propose the *Transit Portal (TP)*, as shown in Figure 1. Each data center has a TP that gives each service the appearance of direct connectivity to the ISPs of its choice. Each service has a dedicated *virtual router* that acts as a gateway for the traffic flowing to and from its servers. The service configures its virtual router with its own policies for *selecting paths* to its clients and *announcing routes* that influence inbound traffic from its clients. By offering the abstraction of BGP sessions with each upstream ISP, the TP allows each service to capitalize on existing open-source software routers (including simple lightweight BGP daemons) without modifying its application software. That said, we believe extending TP to offer new, programmatic interfaces to distributed services is a promising avenue for future work.

Using the TP to control routing provides a hosted service significantly more control over the flow of its traffic than in today's data centers. In addition, the services enjoy these benefits without building their own network footprint, acquiring address space and AS numbers, and negotiating with ISPs. These are hurdles that we ourselves faced in deploying TPs at several locations; the TP obviates the need for the services that it hosts to do the same. In addition, the TP simplifies operations for the ISPs by offloading the separate connections and relationships with each application and by applying packet and route filters to protect them (and the rest of the Internet) from misconfigured or malicious services.

The design and implementation of the TP introduces several challenges. In the control plane, the TP must provide each virtual router the appearance of direct BGP sessions to its upstream ISPs. The TP must also forward outgoing packets to the right ISP and demultiplex incoming packets to the right virtual router. Our solutions to these problems must scale as the number of services increases. To solve these problems, we introduce a variety of techniques for providing layer-two connectivity, amortizing memory and message overhead, and filtering packets and routes. Our prototype implementation composes and extends open-source routing software—the Quagga software router for the control plane and the Linux kernel for the data plane—resulting in a system that is easy to deploy, maintain, and evolve. We also built a management system based on the GENI control framework [16] that automates the provisioning of new customers. The TP is deployed and operational at several locations.

This paper makes the following contributions:

- We explain how flexible wide-area route control can extend the capabilities of existing hosting platforms.
- We present the design and implementation of Transit Portal and demonstrate that the system scales to many ISPs and clients.
- We quantify the benefits of TP by evaluating a DNS service that uses IP anycast and inbound traffic engineering on our existing TP deployment.
- We present the design and implementation of a management framework that allows hosted services to dynamically establish wide-area connectivity.
- We describe how to extend the TP to provide better forwarding performance and support a wider variety of applications (*e.g.*, virtual machine migration).

The remainder of the paper is organized as follows. Section 2 explains how distributed services can make use of wide-area route control. Section 3 presents the design and implementation of the Transit Portal. Section 4 evaluates our three-site deployment supporting an example service, and Section 5 evaluates the scalability and performance of our prototype. Section 6 presents our management framework, and Section 7 describes possible extensions to the TP. Section 8 compares TP to related work; we conclude in Section 9.

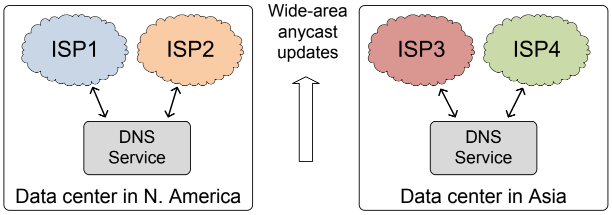


Figure 2: ***Reliable, low-latency distributed services:*** **A service provider that hosts authoritative DNS for some domain may wish to provision both hosting and anycast connectivity in locations that are close to the clients for that domain.**

2. A Case for Wide-Area Route Control

We aim to give each hosted service the same level of routing control that existing networks have. Each service has its own virtual router that connects to the Internet through the *Transit Portal*, as shown in Figure 1. The Transit Portal allows each service to make a different decision about the best way to exchange traffic with its users. The Transit Portal also allows each service to announce prefixes selectively to different ISPs or send different announcements for the same IP prefix to control how inbound traffic reaches downstream networks.

2.1 How Route Control Helps Applications

This section describes three services that can benefit from having more control over wide-area routing and the ability to rapidly provision connectivity. Section 4 evaluates the first service we discuss—improving the reliability, latency, and load balacing traffic for distributed services—through a real deployment on Amazon's EC2. We do not evaluate the remaining applications with a deployment, but we explain how they might be deployed in practice.

Reliable, low-latency distributed services. The Domain Name System (DNS) directs users to wide-area services by mapping a domain name to the appropriate IP address for that service. Service providers often use DNS for tasks like load balancing. Previous studies have shown that DNS lookup latency is a significant contributor to the overall latency for short sessions (*e.g.*, short HTTP requests). Thus, achieving reliability and low latency for DNS lookups is important. One approach to reducing DNS lookup latency is to move the authoritative DNS servers for a domain closer to clients using anycast. Anycast is a method where multiple distinct networks advertise the same IP prefix; client traffic then goes to one of these networks. Hosting authoritative name servers on an anycasted IP prefix can reduce the round-trip time to an authoritative name server for a domain, thus reducing overall name lookup time.

Although anycast is a common practice for DNS root servers, setting up anycast is a tall order for an individ-

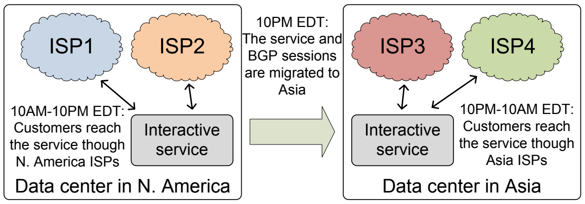

ISP1
ISP2
10PM EDT: The service and BGP sessions are migrated to Asia
ISP3
ISP4
10AM-10PM EDT: Customers reach the service though N. America ISPs
Interactive service
Data center in N. America
Interactive service
10PM-10AM EDT: Customers reach the service though Asia ISPs
Data center in Asia

Figure 3: *Using routing to migrate services:* A service provider migrates a service from a data center in North America to one in Asia, to cope with fluctuations in demand. Today, service providers must use DNS for such migration, which can hurt user performance and does not permit the migration of a running service. A provider can use route control to migrate a service and re-route traffic on the fly, taking DNS out of the loop and enable migration of running services.

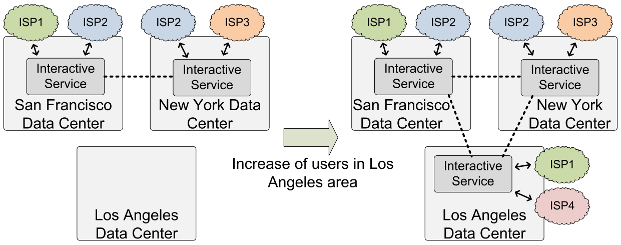


Figure 4: *Flexible peering and hosting for interactive applications:* Direct control over routing allows services to expand hosting *and upstream connectivity* in response to changing demands. In this example, a service experiences an increase in users in a single geographic area. In response, it adds hosting and peering at that location to allow customers at that location to easily reach the service.

ual domain: each domain that wants to host its own authoritative servers would need to establish colocation and BGP peering at multiple sites and make arrangements. Although a DNS hosting provider (*e.g.*, GoDaddy) could host the authoritative DNS for many domains and anycast prefixes for those servers, the domains would still not be able directly control their own DNS load-balancing and replication. Wide-area route control allows a domain to establish DNS-server replicas and peering in multiple locations and to change those locations and peering arrangements when load changes. Figure 2 shows such a deployment. We have deployed this service [24] and will evaluate it in Section 4.

Using routing to migrate services. Service providers such as Google commonly balance client requests across multiple locations and data centers to keep the latency for their services as low as possible. To do so, they commonly use the DNS to re-map a service name to a new IP address. Unfortunately, relying on DNS to migrate client requests requires the service provider to set low time-to-live (TTL) values on DNS replies. These low TTL values help a service provider quickly re-map a DNS name to a new IP address, but they also prevent the client from caching these records and can introduce significant additional latency; this latency is especially troublesome for short-lived sessions like Web search, where the DNS lookup comprises a large fraction of overall response time. Second, DNS-based re-mapping cannot migrate ongoing connections, which is important for certain services that maintain long-lived connections with clients (*e.g.*, VPN-based services). Direct wide-area route control allows the application provider to instead migrate services using routing: providers can migrate their services without changing server IP addresses by dynamically acquiring wide-area connections and announcing the associated IP prefix at the new data center while withdrawing it at the old one. Figure 3 shows how this type of migration can be implemented. This approach improves user-perceived performance by allowing the use of larger DNS TTL values and supporting live migration of long-lived connections.

Flexible peering & hosting for interactive applications. To minimize round-trip times, providers of interactive applications like gaming [2] and video conferencing [3, 4] aim to place servers close to their customers to users and, when possible, selecting the route corresponding to the lowest-latency path. When traffic patterns change due to flash-crowds, diurnal fluctuations, or other effects, the application provider may need to rapidly reprovision both the locations of servers *and* the connectivity between those servers and its clients. Figure 4 shows an example of an interactive service that suddenly experiences a surge in users in a particular region. In this case, the hosting facility will not only need to provision additional servers for the interactive service provider, but it will also need to provision additional connectivity at that location to ensure that traffic to local clients enter and leave at that facility.

2.2 Deployment Scenarios

Cloud providers can provide direct control over routing and traffic to hosted applications. A cloud service such as Amazon's EC2 can use direct wide-area route control to allow each application provider to control inbound and outbound traffic according to its specific requirements. Suppose that two applications are hosted in the same data center. One application may be focused on maintaining low-cost connectivity, while the other may want to achieve low latency and good performance at any cost. Today's cloud services offer only "one size fits all" transit and do not provide routing control to each hosted service or application; the Transit Portal provides this additional control.

An enterprise can re-provision resources and peering as demands change. Web service providers such as Google and Microsoft share a common infrastructure across multiple applications (*e.g.*, search, calendar, mail, video) and continually re-provision these resources as client demand shifts. Today, making application-specific routing decisions in a data center (as shown in Figure 1) is challenging, and re-routing clients to services in different data centers when demands change is even more difficult.

The Transit Portal can provide each application in a data center control over routing and peering, allowing it to establish connectivity and select paths independently of the other properties. This function also makes service migration easier, as we describe in further detail below.

A researcher can perform experiments using wide-area routing. Although existing testbeds such as Emulab [14] allow researchers to operate their own wide-area networks, they generally do not offer flexible control over connectivity to the rest of the Internet. Different experiments will, of course, have different requirements for the nature of their connectivity and routing, and researchers may even want to experiment with the effects of different peering arrangements on experimental services. As part of the GENI project, we are building facilities for this level of route control by connecting Transit Portal to downstream virtual networks to allow researchers to design and evaluate networked services that require greater control over wide-area routing.

3. Design and Implementation

This section describes the design and implementation of a Transit Portal (TP); Section 6 completes the picture by describing the management framework for a network of TPs. The TP extends and synthesizes existing software systems—specifically, the *Linux* kernel for the data plane and the *Quagga* routing protocol suite for the control plane. The rest of this section describes how our design helps the TP achieve three goals: (1) *transparent connectivity* between hosted services and upstream ISPs; (2) *scalability* to many hosted services and upstream ISPs; and (3) the ability to *protect* the rest of the Internet from accidental or malicious disruptions. Table 1 summarizes our design and implementation decisions and how they allow us to achieve the goals of transparent connectivity, scalability, and protection.

3.1 Transparent Connectivity

The TP gives client networks the appearance of direct data- and control-plane connectivity to one or more upstream ISPs. This transparency requires each client network to have a layer-two link and a BGP session for each upstream ISP that it connects to, even though the link and session for that client network actually terminate at the TP. The client's virtual routers are configured exactly as they would be if they connected directly to the upstream ISPs without traversing the Transit Portal. The TP has one layer-two connection and BGP session to each upstream ISP; this connection multiplexes both data packets and BGP messages for the client networks.

Different layer-two connections for different clients. Connecting to an upstream ISP normally requires the client to have a direct layer-two link to the ISP for carrying both BGP messages and data traffic. To support this abstraction, the TP forms a separate layer-two connection to the client for each upstream ISP. Our implementation uses the Linux 2.6 kernel support for IP-IP tunnels, GRE tunnels, EGRE tunnels, and VLANs, as well as UDP tunnels through a user-space OpenVPN daemon.

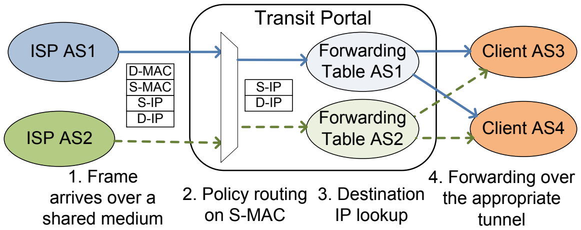


Figure 5: ***Forwarding incoming traffic:*** **When a packet arrives at the Transit Portal (Step 1), the TP uses the source MAC address (S-MAC) to demultiplex the packet to the appropriate IP forwarding table (Step 2). The lookup in that table (Step 3) determines the appropriate tunnel to the client network (Step 4).**

```
# arp -a
router1.isp.com (1.1.1.1) at 0:0:0:0:0:11 on
     eth0
# iptables -A PREROUTING -t mangle -i eth0 -m
     mac --mac-source 0:0:0:0:0:11 -j MARK
     --set-mark 1
# ip rule add fwmark 1 table 1
```

Figure 6: Linux policy routing de-multiplexes traffic into the appropriate forwarding table based on the packet's source MAC address. In this example, source address `0:0:0:0:0:11` de-multiplexes the packet into forwarding table `1`.

Transparent forwarding between clients and ISPs. The TP can use simple policy routing to direct traffic from each client tunnel to the corresponding ISP. Forwarding traffic from ISPs to clients, however, is more challenging. A conventional router with a single forwarding table would direct the traffic to the client prefix over a single link (or use several links in a round robin fashion if multipath routing is enabled.) As shown in Figure 5, though, the TP must ensure the packets are directed to the appropriate layer-two link—the one the client's virtual router associates with the upstream ISP. To allow this, the TP maintains a *virtual forwarding table* for each upstream ISP. Our implementation uses the Linux 2.6 kernel's support for up to 252 such tables, allowing the TP to support up to 252 upstream ISPs.

The TP can connect to an upstream ISP over a point-to-point link using a variety of physical media or tunneling technologies. We also intend to support deployment of Transit Portals at exchange points, where the TP may connect with multiple ISPs over a local area network via a single interface. Each ISP in such shared media setup sends layer-two frames using a distinct source MAC address; the TP can use this address to correctly identify the sending ISP. Figure 6 shows how such traffic de-multiplexing is configured using policy routing rules. The ISP router has an IP address `1.1.1.1` with a MAC address `0:0:0:0:0:11` and a dedicated forwarding ta-

Requirement	Decision	Implementation
Transparent Connectivity (Section 3.1)		
Different layer-two connections	Tunnels between TP and virtual router	Tunneling technologies supported by the Linux kernel
Transparent traffic forwarding	Isolated forwarding tables for ISPs	Virtual forwarding tables and policy routing in Linux
Scalability (Section 3.2)		
Scalable routing with the # of ISPs	Isolated routing tables for ISPs	BGP views feature in Quagga `bgpd` daemon
Scalable updates with # of clients	Shared route update computation	Peer-group feature in Quagga `bgpd` daemon
Scalable forwarding with # of ISPs	Policy/default routing	Modifications to the Quagga `bgpd` daemon
Protection (Section 3.3)		
Preventing IP address spoofing	Packet filtering on source IP address	Linux `iptables`
Preventing prefix hijacking	Route filtering on IP prefix	Quagga prefix filters
Limiting routing instability	Rate-limiting of BGP update messages	Route-flap damping in Quagga `bgpd` daemon
Controlling bandwidth usage	Traffic shaping on virtual interfaces	Linux `tc`

Table 1: Design and implementation decisions.

ble, `1`. Line 1 shows the TP learning the MAC address of an upstream ISP when a new session is established. Then, lines 3–4 establish a policy-routing rule that redirects all the packets with this MAC address to a virtual forwarding table serving a new upstream ISP.

Transparency is another important goal for connectivity between client networks and the Transit Portal. In other words, a client network's connection to the TP should appear as though it were directly connected to the respective upstream networks. In the control plane, achieving this goal involves (1) removing the appearance of an extra AS hop along the AS path; and (2) passing BGP updates between client networks and upstreams as quickly as possible. The first task is achieved with the `remove-private-as rewrite` configuration (line 10 in Figure 7(a)), and the second task is achieved by setting the advertisement interval to a low value (line 18 in Figure 7(a)).

The Transit Portal supports clients regardless of whether they have a public or private AS number. To ensure transparency for the clients with a public AS number, the TP forwards the updates from such clients unmodified. Updates from clients with private AS numbers require rewriting.

3.2 Scalability

The TP maintains many BGP sessions, stores and disseminates many BGP routes, and forwards packets between many pairs of clients and ISPs. Scaling to a large number of ISPs and clients is challenging because each upstream ISP announces routes for many prefixes (*i.e.*, 300,000 routes); each client may receive routes from many (and possibly all) of these ISPs; and each client selects and uses routes independently. We describe three design decisions that we used to scale routing and forwarding at the TP: BGP views, peer groups, and default routing.

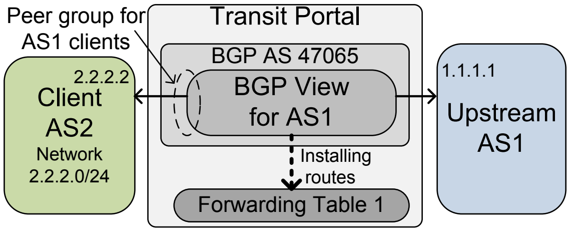
Scalable routing tables using BGP views. Rather than selecting a single best route for each destination prefix, the TP allows each service to select from the routes learned from all the upstream ISPs. This function requires the Transit Portal to disseminate routes from each ISP to the downstream clients, rather than selecting a single best route and could be achieved by having the TP run a separate instance of BGP for each upstream ISP, with BGP sessions with the ISP and each of the clients. Unfortunately, running multiple instances of BGP, each with its own process and associated state, would be expensive. Instead, the TP runs a single BGP instance with multiple "BGP views", each with its own routing table and decision process, for each upstream ISP. Using BGP views prevents the TP from comparing routes learned from different ISPs, while still capitalizing on opportunities to store redundant route information efficiently. Any downstream client that wants to receive routing messages from a specific upstream ISP need only establish a BGP session to the associated view. Our implementation uses the BGP `view` feature in Quagga; in particular, Figure 7(a) shows the configuration of a single "view" (starting in line 3) for upstream ISP 1. Section 5.2 shows that using BGP views in Quagga allows us to support approximately 30% more upstream ISPs with the same memory resources compared to the number of supported ISPs using conventional BGP processes.

Scalable updates to clients with BGP peer groups. Upon receiving a BGP update message from an upstream ISP, the TP must send an update message to each client that "connects" to that ISP. Rather than creating, storing, and sending that message separately for each client, the TP maintains a single BGP table and constructs a common message to send to all clients. Our implementation achieves this goal by using the `peer-group` feature in Quagga, as shown in Figure 7(a); in particular, line 14 associates `Client A` (CA) with the `peer-group` View1 for upstream ISP 1, as defined in lines 17–20. Note that although this example shows only one peer-group member, the real benefit of peer groups is achieved when multiple clients belong to the same group. Section 5.3 shows that peer-groups reduce CPU consumption threefold.

Smaller forwarding tables with default routes and policy routing. Despite storing and exchanging many BGP routes in the control plane, the Transit Portal should try to limit the amount of data-plane state for fast packet forwarding. To minimize data-plane state, the TP does *not* install all of the BGP routes from each BGP view in the kernel forwarding tables. Instead, the TP installs the smallest

```
 1 bgp multiple-instance
 2 !
 3 router bgp 47065 view Upstream1
 4   bgp router-id 47.0.0.65
 5   bgp forwarding-table 1
 6   bgp dampening
 7
 8   ! Connection to Upstream ISP
 9   neighbor 1.1.1.1 remote-as 1
10   neighbor 1.1.1.1 remove-private-AS rewrite
11   neighbor 1.1.1.1 attribute-unchanged as-path
       med
12
13   ! Connection to Downstream Client
14   neighbor 2.2.2.2 peer-group View1
15   neighbor 2.2.2.2 remote-as 2
16   neighbor 2.2.2.2 route-map CA-IN in
17   neighbor View1 peer-group
18   neighbor View1 advertisement-interval 2
19   neighbor View1 attribute-unchanged as-path med
20   neighbor View1 local-as 1 no-prepend
21 !
22 ip prefix-list CA seq 5 permit 2.2.2.0/24
23 !
24 route-map CA-IN permit 10
25   match ip address prefix-list CA
26 !
```

(a) Quagga configuration.



(b) Control-plane setup.

Figure 7: *Example control-plane configuration and setup:* The TP is a hacked version of Quagga that installs non-default routes into Forwarding Table 1.

amount of state necessary for custom packet forwarding to and from each client. On each virtual forwarding table, the TP stores only a default route to an upstream ISP associated with that table (to direct clients' outbound traffic through the ISP) and the BGP routes announced by the clients themselves (to direct inbound traffic from the ISP to the appropriate client). As shown in Section 5.2, this arrangement allows us to save about one gigabyte of memory for every 20 upstream ISPs. To selectively install only the routes learned from clients, rather than all routes in the BGP view, we make modifications to Quagga.

3.3 Protection

The TP must protect other networks on the Internet from misbehavior such as IP address spoofing, route hijacking or instability, or disproportionate bandwidth usage.

Preventing spoofing and hijacking with filters. The TP should prevent clients from sending IP packets or BGP route announcements for IP addresses they do not own. The TP performs ingress packet filtering on the source IP address and route filtering on the IP prefix, based on the client's address block(s). Our implementation filters packets using the standard `iptables` tool in Linux and filters routes using the `prefix-list` feature, as shown in lines 16 and 22-26 of Figure 7(a). In addition to filtering prefixes the clients do not own, the TP also prevents clients from announcing smaller subnets (*i.e.*, smaller than a /24) of their address blocks. Smaller subnets are also filtered by default by most of the Internet carriers. Section 7 describes how TP can overcome this limitation.

Limiting routing instability with route-flap damping. The TP should also protect the upstream ISPs and the Internet as a whole from unstable or poorly managed clients. These clients may frequently reset their BGP sessions with the TP, or repeatedly announce and withdraw their IP prefixes. The TP uses route-flap damping to prevent such instability from affecting other networks. Our implementation enables route-flap damping (as shown in line 6 of Figure 7(a)) with the following parameters: a half-life of 15 minutes, a 500-point penalty, a 750-point reuse threshold, and a maximum damping time of 60 minutes. These settings allow client to send the original update, followed by an extra withdrawal and an update, which will incur penalty of 500 points. Additional withdrawals or updates in a short timeframe will increase the penalty above reuse threshold and the route will be suppressed until the penalty shrinks to 750 points (the penalty halves every 15 minutes). There is no danger that one client's flapping will affect other clients, as route damping on the Internet operates separately for each announced route.

Controlling bandwidth usage with rate-limiting. The TP should prevent clients from consuming excessive bandwidth, to ensure that all clients have sufficient resources to exchange traffic with each upstream ISP. The TP prevents bandwidth hoarding by imposing rate limits on the traffic on each client connection. In our implementation, we use the standard `tc` (traffic control) features in Linux to impose a maximum bit rate on each client link.

4. Deployment

We have deployed Transit Portals in five locations. Four TPs are deployed in the United States, in Atlanta, Madison, Princeton, and Seattle. We also have one Transit Portal deployment in Tokyo, Japan. All Transit Portals are deployed in universities and research labs, whose networks act as a sole ISP in each location. Each ISP also provides full transit for our prefix and AS number. We are actively expanding this deployment: We are engaging with operators at two European research institutions and with one commercial operator in the U.S. to deploy more Transit Portals, and we are planning to expand our Seattle installation to connect to more ISPs.

The TPs advertise BGP routes using origin AS 47065 and IP prefix 168.62.16.0/21. Clients currently use a pri-

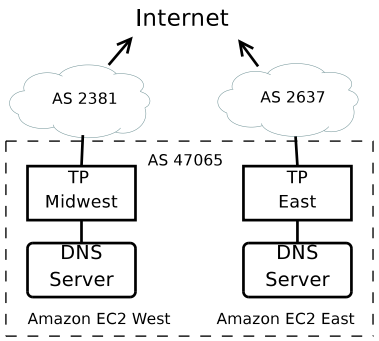


Figure 8: IP anycast experiment setup.

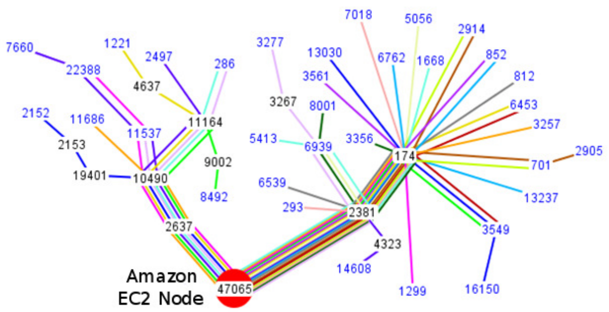


Figure 9: AS-level paths to an EC2 service sitting behind the Transit Portal (AS 47065), as seen in RouteViews.

vate AS number that the TP translates to the public AS number, 47065, before forwarding an update. Clients can also obtain their own AS number, in which case the TP re-advertises the updates without modification.

This section presents the deployment of a distributed, anycasted DNS service, as we described in Section 2, that uses the TP for traffic control, similar to the service we described in Section 2 (Figure 2). In our evaluation of this deployment, we demonstrate two distinct functions: (1) the ability to load balance inbound and outbound traffic to and from the DNS service (including the ability to control the number of clients that communicate with each replica); and (2) the ability to reduce latency for specific subsets of clients with direct route control.

4.1 DNS With IP Anycast

In this section, we show how the TP delivers control and performance improvements for applications that can support IP anycast, such as anycast DNS resolution. The TP allows an IP anycast service to: (1) react to failures more quickly than using DNS re-mapping mechanisms, (2) load-balance inbound traffic, and (3) reduce the service response time. We explain the experiment setup and the measurements that show that adding IP anycast to services running on Amazon EC2 servers can improve latency, failover, and load-balance.

We deploy two DNS servers in Amazon EC2 data centers: one in the US-East region (Virginia) and another in the US-West region (Northern California). The servers are connected to two different TP sites and announce the same IP prefix to enable IP anycast routing as shown in Figure 8. The US-East region is connected to AS 2637 as an upstream provider, while the US-West region is connected to AS 2381 as its upstream provider. We measure the reachability and delay to these DNS servers by observing the response time to the IP anycast address from approximately 600 clients on different PlanetLab [21] nodes. Because our goal is to evaluate the scenario where the TP is collocated with a cloud computing facility, we adjust the measurements to discount the round-trip delay between the TP and the Amazon EC2 data centers.

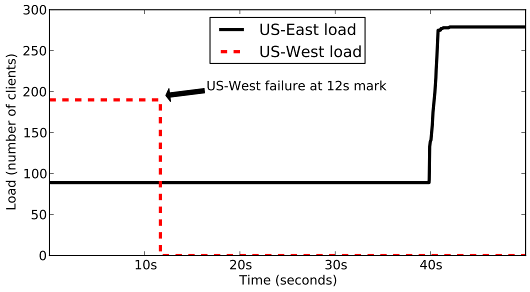


Figure 10: Failover behavior with two IP anycast servers.

The main provider of both upstreams is Cogent (AS 174), which by default prefers a downstream link to AS 2381. Cogent publishes a set of rules that allows Cogent's clients (*e.g.*, AS 2381, AS 2637, and their respective clients) to affect Cogent's routing choices [13]. The DNS service hosted on an Amazon host runs a virtual router and thus can apply these rules and control how incoming traffic ultimately reaches the service.

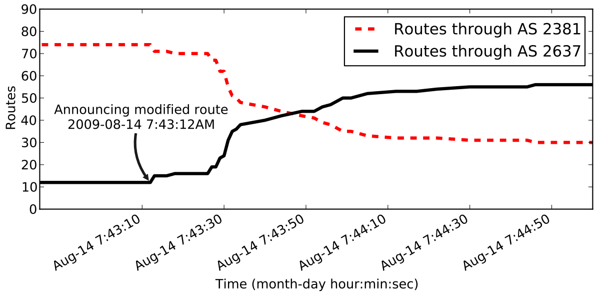
Figure 9 shows a capture from the BGPlay tool [1], which shows the initial routing state with the original BGP configuration. Most of the Internet paths to AS 47065 traverse Cogent (AS 174), which in turn prefers AS 2381 to forward the traffic to the client. Note that the client is configured with private AS number, but the TPs rewrite the updates before re-advertising them to the Internet. This rewriting causes the routers on the Internet to observe prefixes from as if they were announced by AS 47065.

Failover. Today's Internet applications use DNS name re-mapping to shift services to active data centers in the case of a data center or network failure. DNS name re-mapping is a relatively slow process because it requires the DNS entries to expire in DNS caches across the Internet. Applications that support IP anycast can rely on the routing infrastructure to route traffic to active data centers. In our experiment, we fail the server deployed in the US-West region and observe how quickly the clients converge to the US-East region.

Figure 10 shows how the load changes as we introduce failure. After twelve seconds, we fail the US-West deploy-

```
router bgp 65000
  neighbor 10.1.0.1 route-map OUT-2381 out
!
route-map OUT-2381 permit 10
  match ip address prefix-list OUR
  set community 174:10
```

(a) Route map.



(b) Route convergence after applying the route map.

Figure 11: Load balance: Applying a route map to outbound advertisements to affect incoming traffic.

ment and stop receiving requests at that site. After approximately 30 seconds, the routing infrastructure reroutes the traffic to the US-East site. The reaction to failure was automatic, requiring no monitoring or intervention from either the application or the TP operators.

Inbound traffic control. Assume that the DNS service would prefer most of its traffic to be served via AS 2637, rather than AS 2381. (The client network might prefer an alternate route as a result of cost, security, reliability, delay, or any other metric.) The Transit Portal clients can apply BGP communities to affect how upstream ISPs routes to its customers. On August 14, we changed the client configuration as shown in Figure 11(a) to add the BGP community `174:10` to a route, which indicates to one of the upstream providers, Cogent (AS 174), to prefer this route less than other routes to the client network.

To see how quickly the Internet converges to a new route, we analyze the route information provided by RouteViews. Figure 11(b) shows the convergence of the Internet routes to a new upstream. The dashed line shows the number of networks on the Internet that use the AS 2381 link, while the plain line shows the number of networks that use the AS 2637 link to reach the client hosted in the Amazon data center. (Note that the number of routes corresponds only to the routes that we collected from RouteViews.)

IP anycast application performance. We evaluate three DNS service scenarios: (1) US-East only, (2) US-West only, and (3) both servers using IP anycast routing. We measure the delay the PlanetLab clients observe to the IP anycast address in each of these scenarios. Using IP anycast should route each client to the closest active data center.

	Avg. Delay	US-East	US-West
US-East	102.09ms	100%	0%
US-West	98.62ms	0%	100%
Anycast	94.68ms	42%	58%

Table 2: DNS anycast deployment. Average round trip time to the service and fraction of the load to each of the sites.

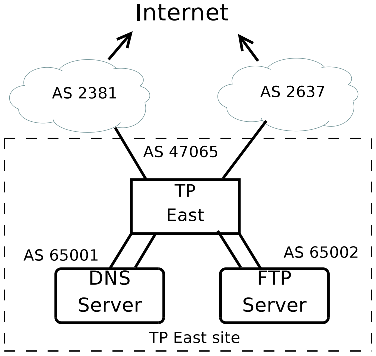


Figure 12: Outbound TE experiment setup.

Table 2 shows the results of these experiments. Serving DNS using IP anycast improves response time by 4-8 milliseconds compared to serving from either of the sites separately. The improvement is not significant in our setup, since the Midwest and East Coast TP deployments are not far from each other. We expect more improvement when IP anycast is used from more diverse locations.

4.2 Outbound Traffic Control

We now show how two different services in a single data center can apply different outbound routing policies to choose different exit paths from the data center. Figure 12 shows the demonstration setup. DNS and FTP services run virtual routers configured as AS 65001 and AS 65002 respectively; both services and the Transit Portal are hosted at the same site as the ISP with an AS 2637. The ISP with an AS 2381 is in a different location and, for the sake of this experiment, the TP routes connections to it via a tunnel.

Without a special configuration, the services would choose the upstream ISP based on the shortest AS path. In our setup, we use the `route-map` command combined with a `set local-preference` setting to configure AS 65001 (DNS service) to prefer AS 2637 as an upstream and AS 65002 (FTP service) to prefer AS 2381 as an upstream. Figure 13 shows how the traceroutes to a remote host differ because of this configuration. The first hop in AS 65001 is a local service provider and is less than one millisecond away. The AS 65002 tunnels are transparently switched through a local TP and terminated at the remote AS 2381, which introduces additional delay.

```
AS65001-node:~# traceroute -n -q 1 -A
     133.69.37.5
traceroute to 133.69.37.5
 1  10.0.0.1 [*]  0 ms
 2  143.215.254.25 [AS2637]  0 ms
[skipped]
 8  203.181.248.110 [AS7660]  187 ms
 9  133.69.37.5 [AS7660]  182 ms
```

(a) Traceroute from AS 65001 client.

```
AS65002-node:~# traceroute -n -q 1 -A
     133.69.37.5
traceroute to 133.69.37.5
 1  10.1.0.1 [*]  23 ms
 2  216.56.60.169 [AS2381] 23 ms
[skipped]
 9  192.203.116.146 [*]  200 ms
 10  133.69.37.5 [AS7660]  205 ms
```

(b) Traceroute from AS 65002 client.

Figure 13: Traceroute from services co-located with TP East and AS 2637.

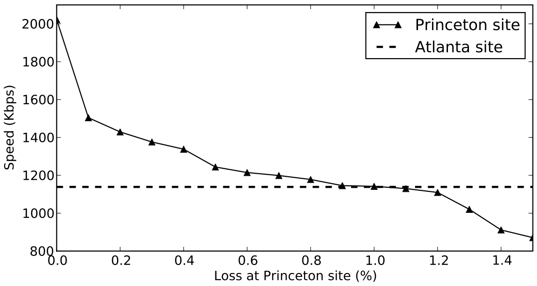


Figure 14: Average access speed from U.S. North East as packet loss is introduced at Princeton site.

4.3 Performance Optimization

The TP can be used to optimize the Internet service access performance. We simulate a video content provider with video streaming services running in cloud sites at the Princeton and the Atlanta locations. Bandwidth measurements show that the Princeton site offers better speed for clients in the northeast U.S., while the Atlanta site is preferred for the southeast U.S.

Assume that, due to periodic congestion, Princeton experiences intermittent packet loss every day around noon. Because the packet loss is intermittent, the application operator may be reluctant to use DNS to re-map the clients. Instead of DNS, the operator can use TP for rerouting when packet loss is detected. Figure 14 shows the average service access speed from the clients in the northeast as the loss at the Princeton site is increasing. As Princeton reaches a 1.1% packet loss rate, the Atlanta site, with its baseline speed of 1140 Kbps, becomes a better choice. Application operators then can use the methods described in Section 4.1 and 4.2 to reroute their applications when they observe losses in Princeton higher than 1.1%.

AS	Prefixes	Updates	Withdrawals
RCCN (1930)	291,996	267,207	15,917
Tinet (3257)	289,077	205,541	22,809
RIPE NCC (3333)	293,059	16,036,246	7,067
Global Crossing (3549)	288,096	883,290	68,185
APNIC NCC (4608)	298,508	589,234	9,147
APNIC NCC (4777)	294,387	127,240	12,233
NIX.CZ (6881)	290,480	150,304	11,247
AT&T (7018)	288,640	1,116,576	904,051
Hutchison (9304)	296,070	300,606	21,551
IP Group (16186)	288,384	273,410	47,776

Table 3: RIPE BGP data set for September 1, 2009.

5. Scalability Evaluation

This section performs micro-benchmarks to evaluate how the Transit Portal scales with the number of upstream ISPs and client networks. Our goal is to demonstrate the feasibility of our design by showing that a TP that is implemented in commodity hardware can support a large number of upstream ISPs and downstream clients. Our evaluation quantifies the number of upstream and client sessions that the TP can support and shows how various design decisions from Section 3 help improve scalability. We first describe our evaluation setup; we then explore how the number of upstream ISPs and downstream client networks affect the TP's performance for realistic routing workload. Our findings are unsurprising but comforting: A single TP node can support tens of upstream ISPs and hundreds of client networks using today's commodity hardware, and we do not observe any nonlinear scaling phenomena.

5.1 Setup

Data. To perform repeatable experiments, we constructed a BGP update dataset, which we used for all of our scenarios. We used BGP route information provided by RIPE Route Information Service (RIS) [23]. RIPE RIS provides two types of BGP update data: 1) BGP table dumps, and 2) BGP update traces. Each BGP *table dump* represents a full BGP route table snapshot. A BGP *update trace* represents a time-stamped arrival of BGP updates from BGP neighbors. We combine the dumps with the updates: Each combined trace starts with the stream of the updates that fill in the BGP routing table to reflect a BGP dump. The trace continues with the time-stamped updates as recorded by the BGP update trace. When we replayed this trace, we honored the inter-arrival intervals of the update trace.

Table 3 shows our dataset, which has BGP updates from 10 ISPs. The initial BGP table dump is taken on September 1, 2009. The updates are recorded in 24-hour period starting on midnight September 2 and ending at midnight September 3 (UTC). The average BGP table size is 291,869.1 prefixes. The average number of updates during a 24-hour period is 1,894,474.3, and the average number of withdrawals is 111,998.3. There are more announcements than withdrawals (a withdrawal occurs only if there is no alternate route to the prefix).

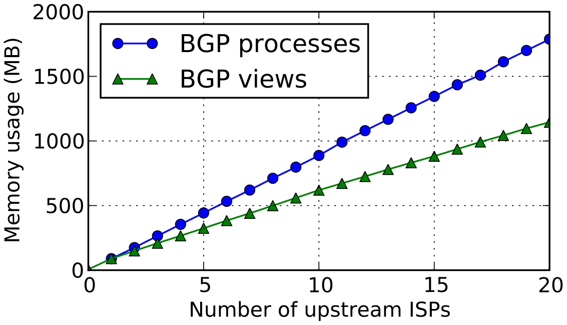


Figure 15: Control plane memory use.

The data set contains two upstream ISPs with an unusually high number of updates: AS 3333 with more than 16 million updates, and AS 7018 with more than 900,000 withdrawals. It is likely that AS 3333 or its clients use reactive BGP load-balancing. In AS 7018, the likely explanation for a large number of withdrawals is a misconfiguration, or a flapping link. In any case, these outliers can stress the Transit Portal against extreme load conditions.

Test environment. We replayed the BGP updates using the `bgp_simple` BGP player [11]. The `bgp_simple` player is implemented using Perl `Net::BGP` libraries. We modified the player to honor the time intervals between the updates.

Unless otherwise noted, the test setup consists of five nodes: Two nodes for emulating clients, two nodes for emulating upstream ISPs, and one node under test, running the Transit Portal. The test node has two 1 Gbps Ethernet interfaces, which are connected to two LANs: one LAN hosts client-emulating nodes, the other LAN hosts upstream-emulating nodes. Each node runs on a Dell PowerEdge 2850, with a 3 Ghz dual-core 64-bit Xeon CPU and 2 GB of RAM. The machines run Fedora 8 Linux.

When we measured CPU usage for a specific process, we used the statistics provided by `/proc`. Each process has a `jiffies` counter, which records the number of system ticks the process used so far. For each test, we collect jiffies at five-second intervals over three minutes and the compute average CPU usage in percent. The `vmstat` utility provides the overall memory and CPU usage.

5.2 Memory Usage

Upstream sessions. Using a commodity platform with 2 GB of memory, TP scales to a few dozen of upstream ISPs. Figure 15 shows how the memory increases as we add more upstream ISPs. When TP utilizes separate BGP processes, each upstream ISP utilizes approximately 90MB of memory; using BGP views each upstream ISP utilizes approximately 60MB of memory. Data plane memory usage, as shown in Figure 16, is insignificant when using our forwarding optimization.

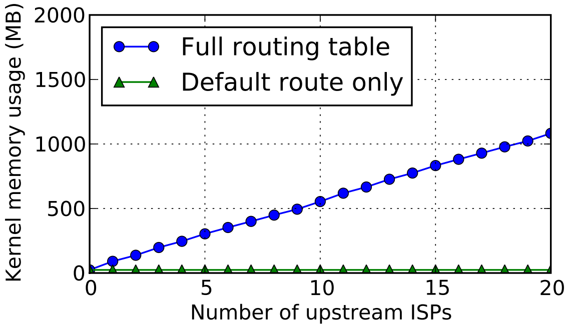


Figure 16: Data plane memory use.

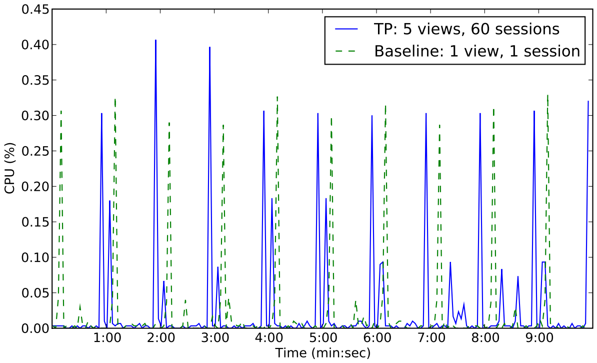


Figure 17: CPU usage over time (average taken every 3 seconds).

Downstream sessions. Each session to a downstream application consumes approximately 10MB of memory. For example, given 20 upstream ISPs, a client "subscribing" to all of them will consume 200MB. Upgrading the TP machine to 16GB of memory would easily support 20 upstream ISPs with more than a hundred clients subscribing to an average of 10 ISPs. The clients use only a small amount of memory in the data plane. The TP ensures forwarding only to the prefixes clients own or lease.

5.3 CPU Usage and Propagation Time

The main users of TP CPU are a BGP scan process, which scans the routes for the changes in reachability information, and BGP update parsing process, which parses the updates which arrive intermittently at a rate of approximately 2 million updates per day.

Figure 17 shows the timeseries of the CPU usage of two BGP processes as they perform routine tasks. Routing updates for both processes arrive from five different ISPs according to their traces. The baseline uses a default Quagga configuration with one routing table, and one client. The Transit Portal configuration terminates each ISP at a different virtual routing table and connects 12 clients (which amounts to a total of 60 client sessions). The TP configuration, on average consumes 20% more CPU than a baseline configuration; most of this load overhead comes from

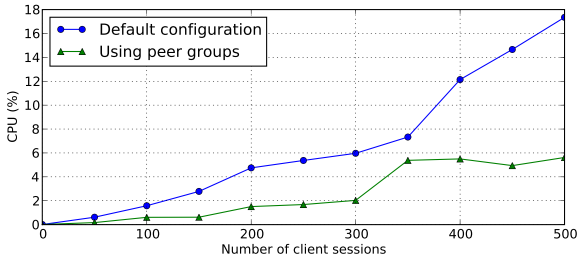


Figure 18: CPU usage while adding client sessions.

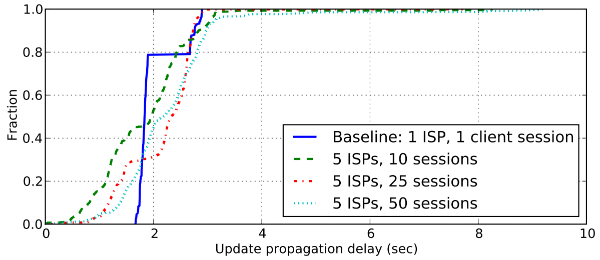


Figure 19: Update propagation delay.

maintaining more client sessions. The spikes in both plots correspond to a BGP scan that occurs every 60 seconds.

The TP can easily support hundreds of client BGP sessions. Figure 18 shows CPU usage as more client sessions are added. Two plots shows CPU usage using the default client configuration and CPU usage using a client configuration with a peer-group feature enabled. While conducting this experiment, we add 50 client sessions at a time and measure CPU load. We observe fluctuations in CPU use since at each measurement the number of updates passing the TP is slightly different. Nevertheless the trend is visible and each one hundred of client sessions increase CPU utilization by approximately a half of a percent.

Figure 19 shows the update propagation delays though the TP. The baseline configuration uses minimal configuration of Quagga with advertisement interval set to 2 seconds. Other configurations reflect the setup of five upstream providers with 10, 25, and 50 sessions. Approximately 40% of updates in the setup with 10 sessions are delivered within 1.6 seconds, while the baseline configuration seems to start deliver updates only at around 1.7 seconds due to the grouping of updates at the TP. A single upstream ISP sends updates in batches and each batch is subject to the configured two-second advertisement interval. When multiple upstream ISPs are configured, more updates arrive at the middle of advertisement interval and can be delivered as soon as it expires.

6. Framework for Provisioning Resources

The TP service provides an interface to the clients of existing hosting facilities to provision wide-area connectivity. In this section, we describe the design and implementation of this management plane. We first discuss the available resources and how they are specified. Next, we describe the process for clients to discover and request resources. Then, we discuss how we have implemented the management plane in the context of the GENI control framework [16]. In the future, the management plane will also control the hosting resources, and provide clients a single interface for resource provisioning.

```
1  <rspec type=”advertisement” >
2
3   <node virtual_id=”tp1”>
4     <node_type type_name=”tp”>
5       <field key=”encapsulation” value=”gre”/>
6       <field key=”encapsulation” value=”egre”/>
7       <field key=”upstream_as” value=”1”/>
8       <field key=”upstream_as” value=”2”/>
9       <field key=”prefix” count=”3” length=”24”/>
10    </node_type>
11  </node>
```

Figure 20: ***Resource advertisement:*** **In Step 0 of resource allocation (Figure 21), the TP's component manager advertises available resources. This example advertisement says that the TP supports both GRE and EGRE encapsulation, has connections to two upstream ASes, and has three /24 prefixes available to allocate.**

6.1 Resources and Their Specification

The current resource allocation framework tracks *numbered* and *network* resources. The numbered resources include the available IP address space, the IP prefixes assigned to each client network, the AS number (or numbers) that each client is using to connect to the TPs, and which IP prefix will be advertised from each location. The framework must also keep track of whether a client network has its own IP prefix or AS number. Network resources include the underlying physical bandwidth for connecting to clients, and bandwidth available to and from each upstream ISP. Management of hosting resources, at this stage, is left for the client networks to handle.

The available resources should be specified in a consistent, machine-readable format. Our implementation represents resources using XML. Figure 20 shows an example advertisement, which denotes the resources available at one TP that offers two upstream connections, as indicated by lines 7–8 in the advertisement. The advertisement also indicates that this TP has three prefixes available for clients (line 9) and can support both GRE and EGRE tunneling (lines 5–6).

6.2 Discovering and Requesting Resources

Each Transit Portal runs a *component manager* (CM) that tracks available resources on the node. To track available capacity between TPs, or on links between virtual hosting facilities, the service uses an *aggregate manager* (AM). The aggregate manager maintains inventory over global resources by aggregating the available resources reported by the component managers. It also

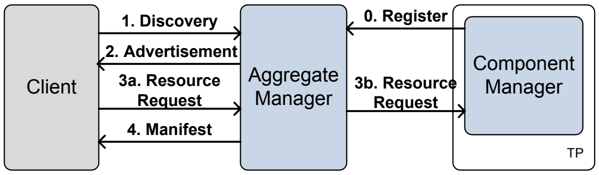


Figure 21: Resource allocation process.

brokers client requests by contacting individual CMs, as shown in Figure 21.

Clients can discover and request resources using a supplied command line tool `en-client.py`. The tool can issue resource discovery and resource reservation requests to a hard-coded AM address as shown in Section 4.2.

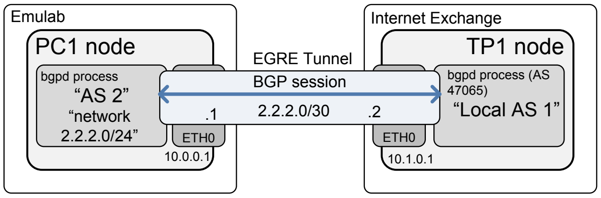
Before clients can request resources, the AM must know about resources in all TP installations. Each component manager *registers* with the AM and provides the list of the available resources, such as the list of upstream ISPs and available IP prefixes (Step 0 in Figure 21). To request resources, a client first issues a *discovery* call to an AM (Step 1). The AM replies with *advertisement*, which describes resources available for reservation (Step 1), such as the example in Figure 20. After receiving the resource advertisement, a client can issue a *resource request* (Step 3), such as the example in Figure 22(b). If the resources are available, the AM issues the *reservation request* to TP1 (Step 4) and responds with a *manifest* (Step 5), which is annotated version of the *request* providing the missing information necessary to establish the requested topology, as shown in Figure 22(c). The AM also provides sample client configuration excerpts with the manifest to streamline client configuration setup. The client uses the manifest to configure its end of the links and sessions, such as the configuration of PC1 in Figure 22(a).

6.3 Implementation

We implement the provisioning framework in the spirit of the Slice-based Facility Architecture (SFA) [5]. This management plane approach is actively developed by projects in the GENI [16] initiative, such as ProtoGENI [22]. SFA is a natural choice because our intermediate goal is to integrate the TP with testbeds like ProtoGENI [22] and VINI [25]. We use the *Twisted* event-driven engine libraries written in *Python* to implement the management plane components.

The primary components of the SFA are the Component Manager (CM) and Aggregate Manager (AM) as introduced before. The interface between the AM and CM is implemented using XML-RPC calls. The client interacts with the AM though a front-end, such as the Emulab or PlanetLab Web site, which in turn contacts the AM using XML-RPC or through the supplied client script.

Currently, access control to the AM is controlled with static access rules that authenticate clients and authorize or prevent a client from instantiating resources. To support more dynamic access control, we plan to expand the AM and CM to support security credentials, which will enable us to inter-operate with existing facilities (*e.g.*, PlanetLab, VINI, GENI) without using static access rules. We also plan to extend the resource management to include slice-based resource allocation and accounting.



(a) Topology resulting from the resource request.

```
<rspec type=”request” >
 <node virtual_id=”tp1”>
   <node_type type_name=”tp”>
     <field key=”upstream_as” value=”1” />
     <field key=”prefix” count=”1”>
   </node_type>
 </node>
 <link virtual_id=”link0”>
   <link_type name=”egre”>
     <field key=”ttl” value=”255”>
   </link_type>
   <interface_ref virtual_node_id=”tp1” />
   <interface_ref virtual_node_id=”pc1”
        tunnel_endpoint=”10.0.0.1” />
 </link>
</rspec>
```

(b) The *resource request* specifies the client's tunnel endpoint, `10.0.0.1`, and asks for an EGRE tunnel, as well as an IP prefix and upstream connectivity to AS 1.

```
 <rspec type=”manifest” >
   <node virtual_id=”tp1”>
     <node_type type_name=”tp”>
       <field key=”upstream_as” value=”1” />
       <field key=”prefix” count=”1” value=”
           2.2.2.0/24” />
     </node_type>
   </node>
   <link virtual_id=”link0”>
     <link_type name=”egre”>
       <field key=”ttl” value=”255” />
     </link_type>
     <interface_ref virtual_node_id=”tp1”\
       tunnel_endpoint=”10.1.0.1”\
       tunnel_ip=”2.2.2.2/30” />
     <interface_ref virtual_node_id=”pc1”\
       tunnel_endpoint=”10.0.0.1”\
       tunnel_ip=”2.2.2.1/30” />
   </link>
 </rspec>
```

(c) The *manifest* assigns an IP prefix to the network, `2.2.2.0/24`, and specifies the parameters for the tunnel between PC1 and TP1.

Figure 22: The *resource request* (Step 3) and *manifest* (Step 5) of the resource allocation process, for an example topology.

7. Extensions to the Transit Portal

The TP is extensible. In the future, we plan to add support for lightweight clients who don't want to run BGP, support for smaller IP prefixes, support for backhaul between different TP sites, extensions for better scalability using hardware platforms for the data plane, and extensions for better routing stability in the face of transient client networks.

Support for lightweight clients. Some client networks primarily need to control traffic but do not necessarily need to run BGP between their own networks and the transit portal. In these cases, a client could use the existence or absence of a tunnel to the TP to signal to the TP whether it wanted traffic to enter over a particular ingress point. When the client network brings up a tunnel, the TP could announce the prefix over the appropriate ISP. When the client brings the tunnel down, the TP withdraws the prefix. As long as the tunnels are up, the client is free to choose an outgoing tunnel to route its traffic.

Support for small IP prefixes. Many client networks may not need IP addresses for more than a few hosts; unfortunately, these client networks would not be able to advertise their own IP prefix on the network, as ISPs typically filter IP prefixes that are longer than a /24 (*i.e.*, subnetworks with less than 256 addresses). The TP could allow client networks with fewer hosts to have BGP-like route control without having to advertise a complete /24 network. Clients for such networks would have full control of outgoing route selection and limited control for influencing incoming routes.

Better scalability. The scalability of the TP data plane can be further improved in two ways: (1) running multiple TPs in an Internet exchange, each serving subset of upstream ISPs, and (2) running the data and control plane of a TP on separate platforms. The first approach is easier to implement. The second approach offers the convenience of a single IP address for BGP sessions from ISPs and follows the best practices of data plane and control plane separation in modern routers. A data plane running on a separate platform could be implemented using OpenFlow or NetFPGA technologies.

Better routing stability in the face of transient client networks. The Transit Portal can support transient client networks that need BGP-based route control but do not need to use network resources all of the time. For example, suppose that a client network is instantiated every day for three hours to facilitate a video conference, or bulk transfer for backups. In these cases, the TP can simply leave the BGP prefix advertised to the global network, even when the client network is "swapped out". In this way, TPs could support transient client networks without introducing global routing instability.

Backhaul between sites. Today's cloud applications in different data centers, performing tasks such as backup or synchronization, must traverse the public Internet. For instance, Amazon EC2 platform offers sites in U.S. East coast, U.S. West coast and in Ireland. Unfortunately, the platform offers little transparency or flexibility for application operators seeking to connect the applications in different sites into a common network. TP platform, on the other hand, is well suited to support sophisticated backhaul between applications in different sites. Each TP site can choose among multiple paths to other TP sites and application operator could exercise control on what path applications are routed to reach other sites. In addition, TP could facilitate private networking between the applications in different sites by using tunnels between TPs. The TP could also improve connectivity between the sites through overlay routing, two TP sites exchange traffic through a third intermediate TP site.

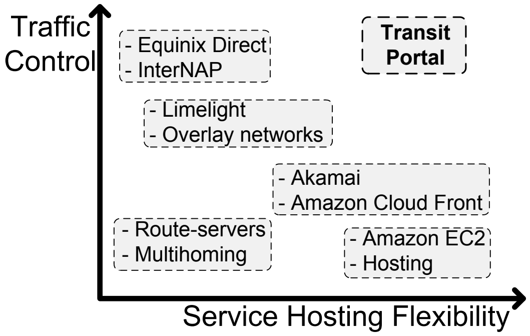


Figure 23: Transit Portals allow cloud providers to offer wide-area route control to hosted services

8. Related Work

The Transit Portal resembles several existing technologies, including content distribution networks, route servers, cloud hosting providers, and even exchange point operators. We describe how these existing technologies differ from TP, with respect to their support for the applications from Section 2.

Content distribution networks and cloud hosting providers do not provide control over inbound and outbound traffic. Content distribution networks like Akamai [6] and Amazon Cloud Front host content across a network of caching proxies, in an attempt to place content close to users to improve performance and save bandwidth. Each of these caching proxies may be located in some colocation facility with its own upstream connectivity. Some content providers may care more about throughput, others may care about low delay, others may care about reliability, and still others might care about minimizing costs. In a CDN, however, the content provider has no control over how traffic enters or leaves these colocation facilities; it is essentially at the mercy of the decisions that the CDN provider makes about upstream connectivity. The Transit Portal, on the other hand, allows each content provider to control traffic independently.

Exchange points do not provide flexible hosting. Providers like Equinix Direct [15] allow services to change their upstream connectivity on short timescales and connect on demand with ISPs in an Internet exchange. Equinix Direct operates only at the control plane and expects clients and ISPs to be able to share a common LAN. Unlike Equinix Direct, the Transit Portal allows services to establish connectivity to transit ISPs without renting rack space in the exchange point, acquiring numbered resources, or procuring dedicated routing equipment.

Route servers do not allow each downstream client network to make independent routing decisions. Route servers reduce the number of sessions between the peers in an exchange point: instead of maintaining a clique of connections, peers connect to a central route server. Route servers aggregate the routes and select only the best route to a destination to each of the peers [18]. This function differs from the TP, which provides transparent access to all of the routes from upstream ISPs.

DNS-based load balancing cannot migrate live connections. Hosting providers sometimes use DNS-based load balancing to redirect clients to different servers—for example, a content distribution network (*e.g.*, Akamai [6]) or service host (*e.g.*, Google) can use DNS to re-map clients to machines hosting the same service but which have a different IP address. DNS-based load balancing, however, does not allow the service provider to migrate a long-running connection, and it requires the service provider to use low DNS TTLs, which may also introduce longer lookup times. The Transit Portal, on the other hand, can move a service by re-routing the IP prefix or IP address associated with that service, thereby allowing for longer DNS TTLs or connection migration.

Overlay networks do not allow direct control over inbound or outbound traffic, and may not scale. Some control over traffic might be possible with an overlay network (*e.g.*, RON [9], SOSR [19]). Unfortunately, overlay networks can only indirectly control traffic, and they require traffic to be sent through overlay nodes, which can result in longer paths.

9. Conclusion

This paper has presented the design, implementation, evaluation, and deployment of the Transit Portal, which offers flexible wide-area route control to for distributed services. The TP prototype is operating at several geographically diverse locations with a /21 address block, AS number, and BGP sessions to upstream providers. We have used Transit Portal in both our research and in projects for a graduate course [**?**], and we plan to deploy and evaluate additional services, including offering our platform to other researchers, and to offer new, more lightweight interfaces to the Transit Portal.

Acknowledgments

We thank Alex Fabrikant, Steven Ko, Ellen Zegura, and our shepherd, Andrew Warfield, for their feedback. This work was funded by NSF award CNS-0626950, NSF CAREER award 0694974, and a grant from the GENI project office "Bringing Experimenters and External Connectivity to GENI".

REFERENCES

[1] BGPlay Route Views. `http://bgplay.routeviews.org/bgplay/`.

[2] Gaikai Demo. `http://www.dperry.com/archives/news/dp_blog/gaikai_-_video/`, 2010.

[3] ooVoo. `http://www.oovoo.com/`, 2010.

[4] Skype. `http://www.skype.com/`, 2010.

[5] Slice-based facility architecture. `http://www.cs.princeton.edu/~llp/arch_abridged.pdf`, 2010.

[6] Akamai. http://www.akamai.com, 2010.

[7] A. Akella, B. Maggs, S. Seshan, A. Shaikh, and R. Sitaraman. A measurement-based analysis of multihoming. In *Proc. ACM SIGCOMM*, Karlsruhe, Germany, Aug. 2003.

[8] A. Akella, J. Pang, B. Maggs, S. Seshan, and A. Shaikh. A comparison of overlay routing and multihoming route control. In *Proc. ACM SIGCOMM*, Portland, OR, Aug. 2004.

[9] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 131–145, Banff, Canada, Oct. 2001.

[10] M. Armbrust, A. Fox, R. Grifth, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical report, University of California at Berkeley, Feb. 2009.

[11] bgpsimple: simple BGP peering and route injection script . `http://code.google.com/p/bgpsimple/`.

[12] Cisco Optimized Edge Routing (OER). `http://www.cisco.com/en/US/products/ps6628/products_ios_protocol_option_home.html`, 2010.

[13] Cogent Communications BGP Communities. `http://www.onesc.net/communities/as174/`, 2010.

[14] Emulab. `http://www.emulab.net/`.

[15] Equinix Direct. `http://www.equinix.com/solutions/connectivity/equinixdirect/`, 2010.

[16] GENI: Global Environment for Network Innovations. `http://www.geni.net/`.

[17] D. K. Goldenberg, L. Qiu, H. Xie, Y. R. Yang, and Y. Zhang. Optimizing cost and performance for multihoming. In *Proc. ACM SIGCOMM*, pages 79–92, Portland, OR, Aug. 2004.

[18] R. Govindan, C. Alaettinoglu, K. Varadhan, and D. Estrin. Route Servers for Inter-Domain Routing. *Computer Networks and ISDN Systems*, 30:1157–1174, 1998.

[19] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the reliability of Internet paths with one-hop source routing. In *Proc. 6th USENIX OSDI*, San Francisco, CA, Dec. 2004.

[20] U. of Oregon. RouteViews. `http://www.routeviews.org/`.

[21] PlanetLab. `http://www.planet-lab.org/`, 2010.

[22] ProtoGENI. `http://www.protogeni.net/`, 2010.

[23] Réseaux IP Européens Next Section Routing Information Service (RIS). `http://www.ripe.net/ris/`.

[24] A. Schran, J. Rexford, and M. Freedman. Namecast: A Reliable, Flexible, Scalable DNS Hosting System. *Princeton University, Technical Report TR-850-09*, 2009.

[25] VINI: Virtual Network Infrastructure. `http://www.vini-veritas.net/`.

[26] A. Weiss. Computing in the clouds. *netWorker*, 11(4):16–25, 2007.