Oasis: An Overlay-Aware Network Stack

Harsha V. Madhyastha, Arun Venkataramani, Arvind Krishnamurthy, and Thomas Anderson
University of Washington and University of Massachusetts Amherst

ABSTRACT

Overlays have enabled several new and popular distributed applications such as Akamai, Kazaa, and Bittorrent. However, the lack of an overlay-aware network stack has hindered the widespread use of general purpose overlay packet delivery services [16, 29, 26]. In this paper, we describe the design and implementation of Oasis, a system and toolkit that enables legacy operating systems to access overlay-based packet delivery services. Oasis combines a set of ideas – network address translation, name resolution, packet capture, dynamic code execution – to provide greater user choice. We are in the process of making the Oasis toolkit available for public use, specifically, to ease the development of PlanetLab-based packet delivery services.

1. INTRODUCTION

In recent years, overlay networks have enabled several real-world applications such as Web content distribution [1], file sharing [15], and popular downloads [7] that have seen widespread deployment. Concurrently, research in overlay networks has demonstrated the potential of providing improved or new network functionality on top of existing networks; examples include general purpose packet delivery services such as multicast [9], detour routing [24], reliable routing [3], quality of service [29], security [16], anycast [6], indirection-based routing [26], multipath routing [37], etc.. More recently, overlays have even been proposed as a building block in the design of an evolvable network architecture to combat the ossification of the Internet [4].

Unfortunately, to date, overlays proposed for general purpose packet delivery[1] [3, 29, 8] services have seen little deployment. For arbitrary end users to obtain end-to-end benefits by employing such overlays, an important practical concern is *ease of accessibility*, which is either ignored or addressed using a custom overlay protocol. Consequently, there is a lack of understanding of a cohesive overlay *architecture* and an accompanying set of design principles (such as the end-to-end principle for the Internet). Prematurely, some recent studies appear pessimistic towards the ability of overlays to provide qualitatively better packet-delivery services [2, 29, 16, 36, 22] on top of a best-effort imperfect underlying network.

In this paper, we adopt a more optimistic stance towards using overlays for general purpose packet delivery. We argue that the lack of an overlay-aware network stack is a critical handicap to the development and widespread deployment of packet delivery overlays. Our main contribution is Oasis, an Overlay Access System for Internetworked Services, that allows users of the commodity operating systems, Linux, Windows XP and Mac OS, to route their traffic via an overlay of their choice. A user may specify routing preferences, such as " route all traffic on port 80 via the Web optimizing overlay" or performance goals, such as "use an overlay that limits loss rate to at most 1%". Oasis dynamically selects an appropriate overlay to meet the user's preferences and falls back on underlay routing when no overlay can satisfy the performance goals.

Oasis' key design goals are to – (1) provide fine-grained control to a user to specify routing preferences on an opt-in basis, (2) provide extensible interfaces to accommodate diverse overlays, but remain agnostic to their implementation, (3) do no harm, *i.e.*, ensure that performance or fault-tolerance is no worse than the underlying Internet, and (4) interoperate with legacy systems for easy deployability.

A guiding design principle in Oasis is minimalism much like the thin-waisted IP in today's Internet. The minimalist design allows Oasis to be interoperable with generic overlay services and yet be agnostic of how the overlay provides said services. Nevertheless, Oasis is powerful enough to facilitate overlay-based complex network and transport services that cut across layers of a traditional network stack. Oasis achieves this combination of overlay-agnosticism and extensibility through a careful division of labor between the end host and an overlay service provider (OSP); in particular, through Oasis's ability to safely execute code supplied by an OSP to harness end host support. Thus, Oasis allows legacy applications, overlay-aware applications, overlay packet delivery services, and the underlay to evolve independently.

More broadly, today's Internet also evolved on top of phone networks as an overlay, and few could have predicted the diversity of applications it has enabled since then. Similarly, we believe that the full potential of overlays as an integral architectural component of an evolvable Internet can be judged fairly only by making it easy to access and develop overlay-based services. Oasis and the associated toolkit are but a first step in this direction.

The rest of the paper is organized as follows. Sections 2 and 3 present, respectively, the design and implementation of Oasis. In Section 4, we outline the additional modules we make available in our toolkit. Section 5 presents the performance evaluation of Oasis. We present the related work in Section 6 and conclude with a discussion of ongoing and future work in Section 7.

2. DESIGN

Oasis is an easily configurable system that enables an end user to route his traffic via heterogeneous overlay networks instead of the default Internet. This section outlines the design of the core architecture of Oasis.

[1]In this paper, the term *packet delivery* refers to services traditionally thought of as part of the network or transport layers, eg, routing, forwarding, congestion control etc.

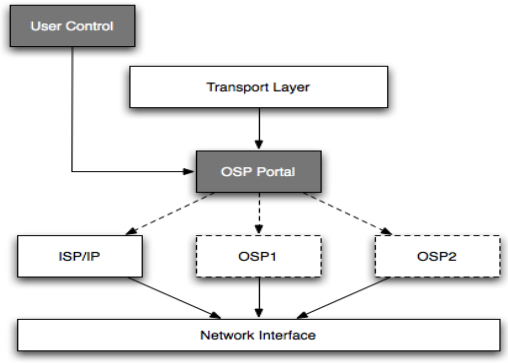


Figure 1: Basic architecture of Oasis

2.1 Design Goals

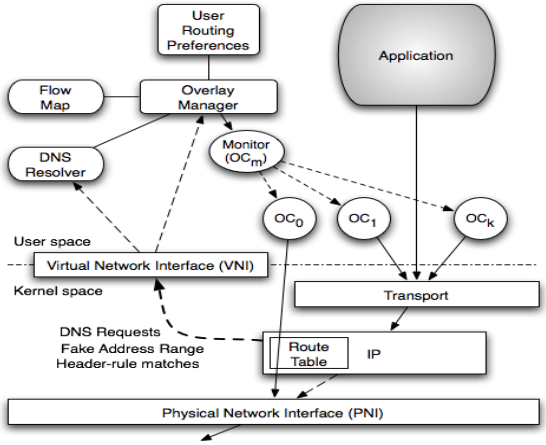
The following objectives guide the design of Oasis.

- *Fine-grained control*: Oasis should enable the user to exercise fine-grained control in determining preferred Overlay Service Providers (OSPs) or performance goals for different portions of traffic. Note that, with the existing ISP model, user-choice is undermined because of the high overhead of switching carriers or employing multiple carriers simultaneously.
- *Do no harm*: Oasis should not worsen performance or fault-tolerance compared to the existing Internet, *i.e.*, an overlay should be employed only when it optimizes user-specified metrics beyond the underlay[2] Further, the computational overhead that Oasis itself imposes upon all traffic should be minimal.
- *Extensibility*: Oasis should also expose a generic interface to overlays to support diverse overlay services and protocols. To allow for Oasis, overlay services, and the division of labor between them to evolve independently, Oasis should be equipped to safely execute untrusted code supplied by OSPs.
- *Deployability*: Oasis should not require any modification to legacy applications and operating systems.

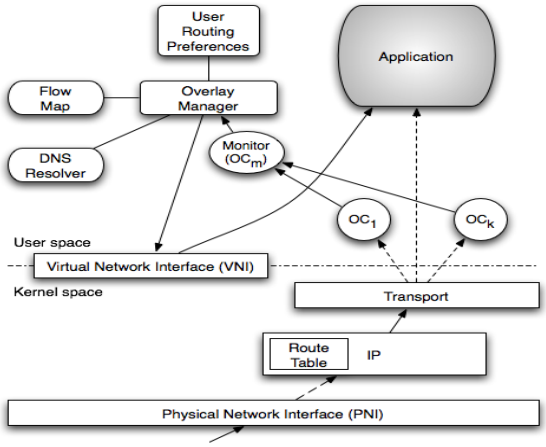
2.2 Architecture

Oasis has three main tasks: (i) to provide a facility to the user to specify routing goals, (ii) to intercept relevant packets and determine the optimal OSP dynamically, (iii) to interface with overlays to send and receive encapsulated packets. Figure 1 shows the core architecture of Oasis that we choose so as to fulfil these goals while conforming to the above design guidelines. The components in white with bold boundaries are those present in legacy operating systems. The shaded pieces, *User Control* and *OSP Portal* are the basic components that Oasis introduces. The components in dotted boxes are overlay-specific modules that are dynamically employed based on the user's preference. This decoupling between the OSP portal and the individual overlay modules has two benefits. First, Oasis is extensible to new overlay services as it does not restricting overlays to abide by a fixed API. Second, the native underlay (denoted by ISP/IP) can be used when no overlay can improve user-specified metrics beyond the underlay.

[2]The underlay refers to the existing Internet. We treat the underlay as a special case of an overlay and sometimes abuse the terms *overlay* and *OSP* to be inclusive of the underlay and the ISP.



(a) Outgoing packet path



(b) Incoming packet path

Figure 2: Components of Oasis. The key packet divert mechanism is a *virtual network interface*. Straight arrows represent fixed forwarding paths and dotted arrows represent one of multiple forwarding choices for a packet. Undirected edges represent two-way information exchange. Some edges have been omitted for simplicity, e.g., an overlay code module OC_i may again forward the packet to the underlay OC_0.

Our realization of this architecture of Oasis consists of the following components as illustrated in Figure 2. VNI is a virtual network interface used to intercept, modify, and resend network packets from user-space. DNS Resolver maintains mappings needed to implement name-based routing preferences. Flow Map maintains per-flow state for address translation and routing. Monitor is a brokering agent that maintains a database of end-to-end path metrics for different OSPs. Overlay Manager manages the list of available OSPs. Route Table is an operating system-specific module for inserting routes into the routing table on the host machine, $OC_1, OC_2, \ldots$ are custom overlay code modules supplied by the respective OSPs, and OC_0 simply routes the packet to the underlay by default.

The User Routing Preferences component implements *User Control* in Figure 1, and the components, Overlay Manager and Monitor, implement *OSP Portal*. The components VNI, DNS Resolver, and Route Table implement the packet interception mechanism (described in greater detail in section 2.2.4) that enables OSP Por-

tal to logically interpose itself between the transport and network layer. Next, we describe in detail how these components interact to accomplish Oasis' objectives, while adhering to the design goals listed above.

2.2.1 Joining an Overlay

Oasis maintains a list of overlays, including the underlay, from which it dynamically chooses an OSP for a connection. To add a new overlay as a potential carrier of packets, the user specifies a URL and a name of local-scope, which she uses to refer to this overlay henceforth. Oasis downloads an XML specification document from the URL. This document contains overlay-specific information that includes the list of entry overlay nodes and the respective port numbers, and may specify the length of the shim header to be padded to enable certain special overlay services (refer section 2.2.3) or to avoid unnecessary copying overhead.

The XML document may also specify an URL from where Oasis downloads and executes an overlay-specific class file as a separate process. For controlling privileges of the overlay code, Oasis uses Java's security mechanisms that restrict I/O and network communication of foreign code.

2.2.2 Addition of Filters

The user specifies rules that Oasis uses to direct traffic to the best-suited overlay. A rule is in the format <*subset, overlay, [metric]*>, where *subset* identifies a portion of the user's traffic. There are two ways to identify such traffic (i) by specifying regular expressions over DNS names, e.g. all traffic destined for *www.cnn.com* or *.edu, and (ii) specifying the values that fields in the packet header should match, e.g., all traffic destined to port 80, or all traffic on port 6510 (used by Bittorrent) destined to select (say, university) address prefixes. The second field, *overlay*, is a string identifying the OSP. The third field, *metric*, is an optional string that specifies user goals, e.g, optimize total transfer time, or minimize loss rate and so on.

When the *metric* field is absent, *overlay* specifies a particular overlay network that is to be always used for that subset. The overlay referred to here is expected to be one that routes the traffic forwarded to it out onto the network. However, when *metric* is specified, *overlay* refers to a specialized externally-supplied code module, which we refer to as a *Monitor*. Monitor is an agent that chooses the overlay best suited to satisfy the user-specified goal of optimizing *metric*. The Monitor, being stacked in between Oasis and OSP modules, appears just like another overlay to Oasis. It forwards packets back and forth between Oasis and OSP modules and monitors end-to-end performance of different overlays. The idea of delegating the responsibility of overlay selection also to an "overlay" borrows from Active Names [33] and enables Oasis to incorporate new performance metrics and techniques to monitor such metrics.

2.2.3 Interfacing with Overlays

When forwarding packets to overlays, by default, an explicit overlay header is absent, *i.e.*, Oasis simply hands a raw IP packet to an OSP module. The custom OSP module is only required to receive these encapsulated UDP packets at the port the OSP specified in its XML document. Subsequently, it is expected to hand packets received along the reverse path back to Oasis at its standard port. In addition to the packet that needs to be forwarded, Monitor also needs to know the metric that is to be optimized and the overlays it can choose from. This is accomplished by including the string, specified as the metric by the user, in the shim header and granting Monitor access to Oasis' configuration file that contains all the overlays the user has joined. We emphasize that Oasis is agnostic to the contents or semantics of the shim. Furthermore, the shim itself is used only for the special Monitor modules when user routing preferences are name-based and name-to-address mappings are non-invertible. An alternate to such overlay-agnostic shims could be for Monitor to provide a dynamic name resolution service similar to DNS Resolver that selects the OSP at name-resolution time on a per-connection basis.

Table 1: API exported by Oasis to users and overlays

User Interface	• Add overlays by <*name, URL*> • Add filters by <*subset, overlay, [metric]*>
Overlay Interface	• Provide XML specification file with port number and URL for custom overlay module • Overlay module · receives IP packets from Oasis · sends IP packets to Oasis at standard port · sends IP packets to Oasis to send on underlay

OSP modules can also send packets that are to be routed on the underlay back to Oasis at another standard port. A typical OSP module uses this channel when it cannot improve performance for a given flow. Monitor does the same if it deems none of the overlays as superior to the underlay.

Table 1 summarizes the restrictions imposed on Oasis' interface with users and overlays by the above design choices. The overlay interface is different only for the Monitor OSP module, which sends and receives encapsulated packets with a shim containing *metric* information.

2.2.4 Packet Interception

On an Oasis-equipped client, a packet can traverse one of three paths.

1. The packet does not match any rule specified by the user and is forwarded to the underlay without mediation by Oasis.
2. The packet matches one of the user-specified rules and is intercepted by Oasis, which then forwards it to an OSP module that handles further processing.
3. The packet matches a user-specified rule and is intercepted and forwarded to an OSP module that hands it back to Oasis to send via the underlay.

The core of Oasis' packet interception mechanism that enables this is a virtual network interface (VNI). Traffic identified by the users through their filters is diverted to the VNI. Oasis intercepts packets based on the name of the destination or fields in the TCP/IP headers. We briefly explain both modes of interception below.

Name-Based interception Oasis usurps the name-to-address resolution service on the host machine by diverting all DNS (destination port 53) requests to the VNI. If the user has requested optimized carrier selection for that name, Oasis forwards the request to the default nameserver to obtain an address IP_{real}, say, but returns to the application a *fake* address, IP_{fake}, that is chosen from a fake IP address range - a set of addresses that are not valid Internet-routable addresses, eg, 10.0.*.*. Further, it stores a mapping between IP_{real} and IP_{fake} in DNSResolver. Oasis modifies the IP route table on the host machine so as to divert all traffic destined to an address in the fake range to the VNI.

Now, when the legacy application, unaware of Oasis' sleight, attempts to send the first IP packet (eg, a SYN in case of TCP) to

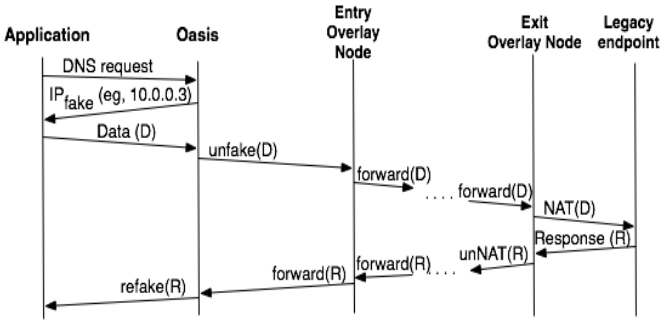


Figure 3: Name-faking based packet timeline

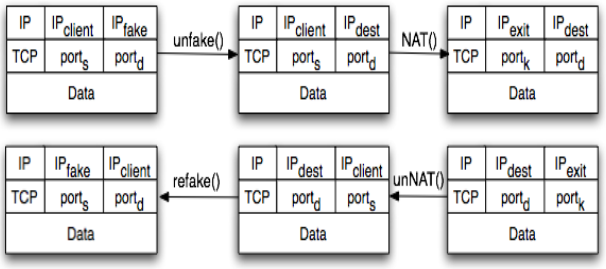


Figure 4: Packet transformations when [IP_{client}, $port_s$] tries to connect to [IP_{dest}, $port_d$]. The client and the destination perceive the other end to be [IP_{fake}, $port_d$] and [IP_{exit}, $port_k$] resp.

IP_{fake}, it gets diverted to the VNI. Oasis reads the packet from the VNI, replaces IP_{fake} with IP_{real} in the packet by consulting DNSResolver, and then transmits the packet via the appropriate carrier as shown in Figures 3 and 4.

If the chosen carrier is the underlay, Oasis simply directs the raw IP packet to the physical network interface (PNI). If the chosen carrier is an overlay, Oasis hands the packet to the Overlay Manager that delivers it via UDP to an entry node of the chosen overlay, which could be running locally. Note that the custom overlay module may again dynamically deduce the underlay to be a superior carrier after all and choose to direct the packet back to Overlay Manager, to forward to the PNI.

For compatibility with legacy destinations, the choice of the carrier remains fixed throughout the lifetime of a connection. However, if the destination is also equipped with Oasis, carrier selection can take place dynamically for every packet independently.

Header-Based interception Header-based packet interception directly diverts matching packets to the VNI. As with name-based interception, matching packets are diverted to the VNI from where Oasis transmits the packet via the appropriate OSP module.

Header-based interception avoids Oasis' computational overhead for traffic that the user does not wish to route via an overlay. Note that fields in a packet's header are determined only when the first packet of the connection is generated. With only name-based interception, every packet would go through Oasis even if it is destined with certainty for the underlay.

2.3 Summary

The core of Oasis itself is designed simply for carrier brokerage based on end-to-end performance monitoring. Oasis is agnostic to how overlays provide a specific packet delivery service. All overlay-specific functionality is implemented by the OSP and the corresponding code downloaded dynamically. This division of labor has two benefits - (i) Oasis is simple and consequently robust and easy to maintain, (ii) the minimality of the Oasis-overlay interface encourages diverse overlay services to independently evolve. A similar argument for evolution of transport protocols was used by Patel et al [20].

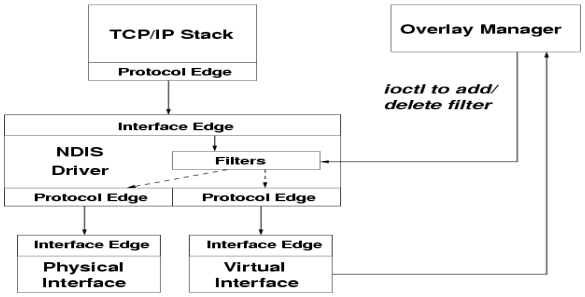


Figure 5: Windows Protocol Driver

3. IMPLEMENTATION

Since the main focus of this paper is to introduce the API exported by Oasis to users and overlay developers, we outline details about the implementation only briefly. Oasis is largely implemented in Java and works on Linux, Windows, and MacOS and is about 6K lines of code.

The OS-specific code in Oasis mainly concerns enabling capture and redirection of packets. The code for this is mostly implemented in native C. Oasis uses the TUN/TAP virtual network interface (corresponding to VNI in the design) for packet redirection on all OS's. Oasis redirects DNS requests and packets destined to the fake IP address range (*10.0.x.x*) to TUN. The other OS-specific component of Oasis is the ability to dynamically route intercepted packets via the underlay. For uniformity, this component is also implemented as the special case of an overlay and is associated with the code module OC_0 that simply decapsulates and sends out the packet. Hence, on both Windows and Linux, the underlay simply appears like another overlay with the local address and the reserved port acting as the entry node specification. Local code of an overlay can choose to send out a packet via the underlay by forwarding the packet to this reserved port at the local address.

3.1 Implementation on Linux

On Linux, redirection of packets to TUN is accomplished using *iptables* rules. For each class of packets that are to be redirected to TUN, an *iptables* rule is added to mark the corresponding packets. An entry is added to the local routing table to route all marked packets to TUN. *iptables* rules are employed in this manner to redirect DNS requests, packets destined to a fake address and packets that match any of the packet header-based rules specified by the user. The component that sends packets out on the underlay does so on Linux using raw sockets.

3.2 Implementation on Windows

On Windows, DNS requests and packets destined to a fake address are redirected to TUN by setting TUN as the default nameserver and mangling the local routing table, respectively. However, on Windows XP, there is no means by which packets can be redirected to TUN based on fields in their header, and support for

raw sockets is currently disabled [3] due to security concerns. To circumvent these restrictions, we implemented an NDIS protocol driver [18] to intercept and route packets based on values of header fields. As shown in Fig. 5, this driver sits in between the TCP/IP stack and the network interfaces.

One edge of the NDIS driver appears as a network interface to the TCP/IP stack. The other edge appears as a protocol to the network interfaces on the system. So, all packets going from the network stack to any network interface, or in the opposite direction, go through this driver. The driver also receives *ioctl* calls from user-space to add and delete filters that determine which traffic should be redirected to TUN. The Overlay Manager component (as depicted in Figure 2) uses these *ioctl* calls to interact with the driver. Each packet received by the driver is passed through the filters that it has received. If a match is found with any of the user-specified filters, the packet is redirected to TUN by the driver. For underlay routing, a special port on the host machine is reserved, and all packets destined to that port are decapsulated and forwarded to the physical network interface by the driver.

3.3 User and Overlay Interface

Currently, a perl script lets the user specify routing preferences. We plan to replace this script with a graphical user interface soon. When invoked, this script encodes the user's input and passes it on to Oasis in a UDP packet. If the command issued by the user is to join a new overlay, Oasis fetches the specification document for that overlay. The specification document is assumed to be in XML with a common standard for the tags used. This document is expected to contain the URL from where Oasis can download code specific to the overlay. The code that is downloaded is required to be a collection of Java class files. Safe execution of this code is ensured using Java's security mechanisms, by taking away all permissions except to open and listen on UDP sockets. Performance-monitoring overlay modules are also granted permission to read Oasis' config file that lists the set of overlays that the user has added.

4. OVERLAY TOOLKIT

In addition to Oasis, we have developed a few additional modules that would help in the development and deployment of overlay-services, especially on PlanetLab.

Interfacing with legacy destinations We have implemented a generic exit overlay node module that runs within a PlanetLab slice. This receives packets from clients, other overlay nodes and from legacy destinations that it has forwarded packets to. Its core utility is in the NAT functionality it implements. It appropriately mangles encapsulated packets that it receives from Oasis-equipped clients and other overlay nodes, before forwarding them to legacy destinations via raw sockets. Sufficient state is maintained to invert this mangling on packets received back from the destinations. The exit-node module implements queueing mechanisms to service traffic received from different nodes with differential forwarding priorities. This feature is of essence in an overlay that involves nodes residing in different administrative entities, which is the case in PlanetLab. Each node would presumably attach higher priority to traffic that belongs to clients in the local institution as compared to traffic forwarded from other nodes in the overlay.

TCP connection proxy As a sample OSP module that would run on an Oasis-equipped client, we have put together a basic TCP connection proxy. The expected mode of operation of this module is for it to be run on both the Oasis-equipped client as well as on the exit overlay node. This module has a sender-side and a receiver-side component. The sender-side component at the client receives TCP packets sent out by the application and the sender-side component on the exit node receives TCP packets sent by the legacy destination. The sender-side component at either end would pass on packets to the receiver-side component at the other end. Thus, by sitting in between the TCP stacks of the legacy client and destination, this pair of TCP connection proxies enables a new transport protocol to be employed through the overlay.

[3]Service Pack 2 onwards

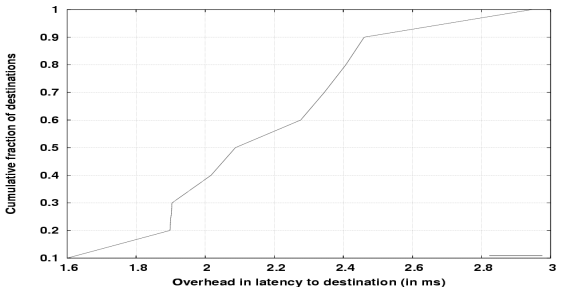


Figure 6: Overhead due to Oasis on individual packets

PlanetLab client Though Oasis is primarily intended to be run on legacy end-hosts, we have also developed a port of Oasis that runs from within a PlanetLab slice. The main purpose of this port is to aid in the process of evaluating overlays, by enabling the emulation of end-clients from PlanetLab hosts. From the perspective of Oasis, the main points of distinction between a PlanetLab slice and a normal Linux box is that *iptables* rules cannot be employed to redirect traffic and the routing table cannot be mangled. Our port to PlanetLab works only for Web traffic. The Web client should use the local address at a particular reserved port as a Web proxy. Oasis captures packets by opening a raw socket at this reserved port. Oasis is required to send back a fake SYN-ACK to the application and look into the HTTP GET request to determine the destination. But, once the connection is setup, it only needs to mangle sequence numbers in either direction.

These modules enable an overlay-developer to assemble a basic setup, wherein routing of packets from a legacy client to a legacy destination is done over a dummy overlay. By adding intelligence to the TCP connection proxy and by interposing overlay nodes that do intelligent routing between the Oasis-equipped client and the exit node, novel routing and transport protocols can be evaluated.

5. EVALUATION

In this section, we present the results of the experiments we conducted to micro-benchmark Oasis. The two classes of experiments we conducted were to evaluate the overhead incurred by routing packets through Oasis and the overhead of running Oasis in terms of CPU utilization. All our experiments were conducted with a Linux box at the University of Washington as the client.

5.1 Latency Overhead

Our first set of experiments were to determine the latency overhead imposed by Oasis in 3 different scenarios - (i) for individual packets, and for transfer times of (ii) short, and (iii) long flows. Our results indicate that the overhead attributable to Oasis is relatively low in each of these cases.

Each of the experiments in this category was first performed without any modification on the client. The experiment was then repeated with Oasis running on the client and all flows to *.edu* names redirected to go out via the underlay, but with every packet mediated by Oasis.

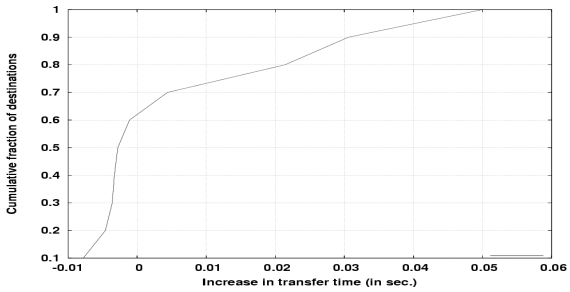


Figure 7: Overhead due to Oasis on short transfers

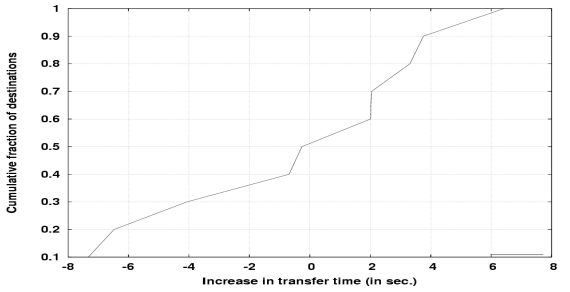
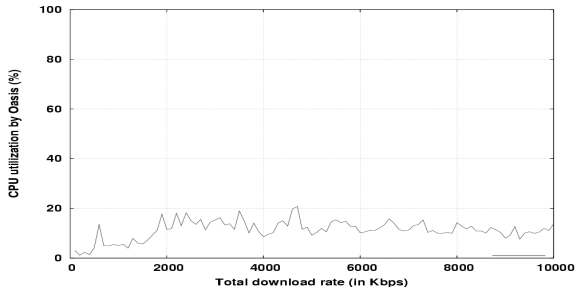
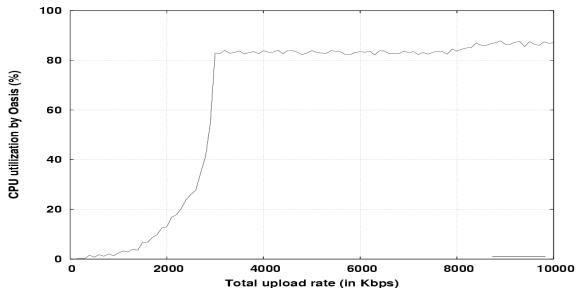


Figure 8: Overhead due to Oasis on long transfers



(a)



(b)

Figure 9: CPU utilization of Oasis with respect to (a) total download rate, and (b) total upload rate

First, we examined what the additional latency borne by each individual packet is because of it having to go through Oasis. For this, we identified a random set of 10 *.edu* domains distributed across the US and pinged each of them with 100 ICMP ECHO probes from our client. Considering the time taken by a probe packet to each destination to be the average of the 100 probes sent to it, Fig. 6 shows the distribution of the overhead incurred for each destination. The median overhead is $2ms$, which accounts for just around 4% of the average latency of $55ms$ to the 10 destinations we measured latencies to. So, the extra latency incurred by each individual probe packet is relatively low.

We next determined how this overhead affects the total transfer time for a TCP flow. The overhead experienced by each packet would dilate the Round Trip Time (RTT) between the two end-hosts. And, the increase in RTT would also affect the throughput of the flow due to TCP's RTT bias. To examine the effect of this, we fetched the *index.html* object from all of the 10 *.edu* domains chosen previously, 10 times each. Considering the download time from each domain to be the average of the 10 downloads performed from it, Fig. 7 plots the distribution of the increase in transfer time. The median increase in the download time is 0, which demonstrates that the overhead incurred by each packet does not have any significant effect on the total transfer time for a flow.

The *index.html* object fetched from each of the 10 domains was just a few KiloBytes large. To evaluate the overhead on larger transfers, which would involve a larger number of packets, we identified 10 mirrors of the Fedora Core 4 Linux distribution in *.edu* domains across the US. We fetched the first 20MB of a particular *iso* file in the distribution from each of these 10 mirrors. The increase in total transfer time for each of these sites, comparing the transfer performed with and without Oasis, is shown in Fig. 8. Even in this scenario, the median increase in transfer time is 0. So, the low overhead that each packet incurs while going through Oasis does not have any significant effect on the total download time even in the case of large transfers.

Since Oasis also catches DNS requests and sends back fake responses, we also determined the overhead incurred in making a DNS lookup. Name-based rules in Oasis are in the form of regular expressions. Checking whether a string matches any among a set of regular expressions can be optimized by building an appropriate finite state machine. However, we ran our experiment with the simple implementation of checking against each rule in sequence. We observed that the overhead incurred by a DNS lookup, even when Oasis is executed with as many as 100 name-based rules, is of the same order as that incurred by normal packets.

5.2 CPU Utilization

The next set of experiments we conducted were to determine the overhead of using Oasis in terms of its CPU utilization. We evaluated this overhead with respect to the total rate of (i) download, and (ii) upload traffic being sent through Oasis. For both the experiments, we generated traffic at a desired rate using a simple tool that sourced UDP traffic at a constant rate.

First, we ran our tool on a machine in the same subnet as our client to send out traffic to our client at a constant rate. We ran the tool for 60 seconds and determined the CPU utilization of Oasis once every 5 seconds. Ignoring the first and last sample, the CPU utilization of Oasis for a given download rate was computed as the mean of the other 10 samples. This experiment was repeated varying the download rate from 100Kbps to 10Mbps in increments of 100Kbps. Fig. 9(a) shows that Oasis' CPU utilization does not exceed 20% even at a download rate of 10Mbps.

Next, we ran the tool on the client so as to send out traffic at a constant rate. As before, we determined the CPU utilization of Oasis for upload rates varying from 100Kbps to 10Mbps. Fig. 9(b) shows that the CPU utilization of Oasis remains reasonable until an upload rate of around 2.5Mbps and then rises pretty steeply, making Oasis unusable at upload rates above 3Mbps. The significantly lower bottleneck on upload rate is due to the overhead involved in intercepting packets.

6. RELATED WORK

Oasis primarily enables end-users to access overlays that provide improved packet delivery services. Examples include overlays that offer better reliability [3, 11] and better loss rates [29].

Overlays such as RON [3], HIP [21], ROAM [38] and some services to support end-host mobility [28, 31, 35] enable access for legacy applications. Some network architectural proposals [34, 5] outline techniques to interoperate with legacy applications. Nakao et al. [17] describe mechanisms to enable access for users that are not part of the overlay. Unfortunately, the above schemes are customized for a specific overlay service and route all or a fixed subset of a user's traffic through the overlay. In contrast, Oasis is designed to enable greater user choice in an environment consisting of multiple overlays. Oasis allows a user to specify her routing preferences in a fine-grained manner and dynamically selects the overlay best suited to carry a packet in accordance with those preferences. Further, a user can easily add a new candidate overlay by downloading the corresponding overlay code module.

Oasis combines several existing techniques, none of which are novel by themselves, to interoperate with legacy applications. DNS rewriting, employed by Oasis to virtualize addresses, has been used before in [10], [19], [23] and [35]. Oasis virtualizes addresses into the *10.0.x.x* address range that is marked for private use [13]. This idea of exploiting local-scope addresses has been employed in earlier work that enables mobility [28, 31, 35], redirection [12], process migration [27, 28] and availability [27]. Yalagandula et al. [35] outline the limitations of local-scope addresses and work-arounds for the same.

OCALA[14], an overlay-generic proxy for the i3 [26] overlay, is closely related to Oasis. Key differences are as follows - (i) Oasis enables captured traffic to be sent out via the underlay if none of the overlays help improve performance, (ii) Oasis supports header-based (in addition to name-based) redirection of packets and avoids imposing unnecessary overhead on traffic destined to the underlay, (iii) Oasis does not impose a carrier-Oasis API (like OCALA's OC-D layer) and therefore has a "thinner waist", (iv) Oasis can safely execute sandboxed code supplied by an OSP, and (v) Oasis does not explicitly mandate an inter-overlay bridging API and delegates that responsibility to the peering overlays. We note that an important advantage of overlays over today's multidomain Internet is that a single overlay can have a global span; thus, bridging multiple overlays appears to be an overkill from an architectural perspective. In comparison to OCALA, we believe that Oasis' design choices ensure that performance or fault-tolerance is no worse than the underlying Internet ("do no harm"), enable finer-grained control over routing preferences, and provide greater extensibility. Finally, Oasis' main focus is on making packet delivery overlays services usable, thus the associated toolkit provides reusable modules for building overlay transport and routing services.

Architecturally, the do-no-harm design and greater extensibility are the two features that distinguish Oasis from other overlay-brokerage systems [21, 5, 14, 32] as well. First, with existing systems, when a user assigns a subset of her traffic to an overlay, that traffic always goes through the overlay. In contrast, Oasis allows a user to specify the delivery metric of her choice and dynamically selects the overlay best suited to carry that traffic. Additionally, Oasis or the overlay itself may choose to direct a packet to the underlay if it deems the latter as the best carrier. Second, existing systems require a shim layer in a fixed format in the encapsulated overlay packet. This requirement not only restricts the class of interoperable overlays, but also fixes this set of overlays at compile-time. Oasis can interoperate with a generic overlay as it forwards packets with an empty shim header to the overlay-specific code module responsible for constructing the shim. Oasis enables a user to join a new overlay at runtime by downloading the corresponding code module.

Broadly, Oasis can be viewed as an "end-to-end-compliant" instantiation of the general idea of Active Names [33] and Active Networks [30]. Oasis allows dynamic name translation based on performance goals. In particular, delegating the task of monitoring end-to-end performance across overlays also to an "overlay" borrows the idea of stackable name resolution from Active Names. Unlike Active Networks, the (overlay) network hands code to the end host on a demand basis when the overlay is initiated or upgraded. This design provides flexibility while upholding the end-to-end principle.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we described the design and implementation of Oasis an instantiation of an overlay-aware network stack. Oasis's primary task is to perform overlay-carrier brokerage in accordance with a user's preferences. Thus, it acts as the first hop router residing on the local machine for all network traffic.

Oasis's ability to execute overlay-specific code allows sophisticated transport and routing overlay services to be built using it. As an example, we have built a multipath routing and congestion control system using the Oasis toolkit. The multipath controller itself is an overlay module that hides multiple paths from legacy applications, but is capable of inducing an application to send data faster than TCP [25]. An associated exit overlay node module in the toolkit performs a similar connection proxying to be compatible with legacy destinations. Other services we have built using Oasis include detour routing for short end-to-end paths, reliability etc. Currently, we are investigating the use of Oasis in designing incentive-compatible mechanisms for cooperative detour routing via managed as well as peer-to-peer overlays. A detailed description of these services will appear in a future paper.

We are currently in the process of making the Oasis system and toolkit available for public use. A preliminary version may be obtained at `http://www.cs.umass.edu/~arun/Oasis/`.

8. REFERENCES

[1] Akamai, inc. home page. www.akamai.com.

[2] A. Akella, J. Pang, B. Maggs, S. Seshan, and A. Shaikh. A comparison of overlay routing and multihoming route control. In *ACM SIGCOMM*, pages 93–106, New York, NY, USA, 2004.

[3] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. Resilient overlay networks. In *SOSP*, pages 131–145. ACM Press, 2001.

[4] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet impasse through virtualization. *Computer*, 38(4):34–41, 2005.

[5] K. Argyraki and D. Cheriton. Loose source routing as a mechanism for traffic policies. In *Proc. of FDNA*, 2004.

[6] H. Ballani and P. Francis. Towards a global ip anycast service. *SIGCOMM Comput. Commun. Rev.*, 35(4):301–312, 2005.

[7] Bittorrent home page. www.bittorrent.com.

[8] Y. Chen, D. Bindel, H. Song, and R. H. Katz. An algebraic approach to practical and scalable overlay network monitoring. *SIGCOMM Comput. Commun. Rev.*, 34(4):55–66, 2004.

[9] H. Eriksson. MBone: The Multicast Backbone. *Communications of the ACM*, 37(8):54–60, Aug. 1994.

[10] M. Freedman, E. Freudenthal, and D. Mazi. Democratizing content publication with coral, 2004.

[11] 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 *USENIX OSDI*, pages 183–198, 2004.

[12] S. Gupta and A. L. M. Reddy. A client oriented, ip level redirection mechanism. In *Proceedings of IEEE Infocom*, 1999.

[13] Internet protocol v4 address space.

[14] D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle. Ocala: An architecture for supporting legacy applications over overlays. Technical Report UCB/CSD-005-1397, UC Berkeley, EECS, 2005.

[15] Kazaa home page. www.kazaa.com.

[16] A. Keromytis, V. Misra, and D. Rubenstein. Sos: Secure overlay services, 2002.

[17] A. Nakao, L. Peterson, and M. Wawrzoniak. A divert mechanism for service overlays. Technical Report TR-668-03, Computer Science Department, Princeton, Feb. 2003.

[18] Pcusa ndis tutorial. www.pcausa.com/pcasim/packetredir.htm.

[19] T. S. E. Ng, I. Stoica, and H. Zhang. A waypoint service approach to connect heterogeneous internet address spaces. In *USENIX Technical Conf.*, 2001.

[20] P. Patel, A. Whitaker, D. Wetherall, J. Lepreau, and T. Stack. Upgrading transport protocols using untrusted mobile code. In *SOSP*, October 2003.

[21] P. J. R. Moskowitz, P. Nikander and T. Henderson. Host identity protocol, 2003.

[22] S. Ratnasamy, S. Shenker, and S. McCanne. Towards an evolvable internet architecture. In *In ACM SIGCOMM*, 2005.

[23] P. Rodriguez, S. Mukherjee, and S. Rangarajan. Session level techniques for improving web browsing performance on wireless links. In *WWW*, 2004.

[24] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: a case for informed Internet routing and transport. *IEEE Micro*, 19, No. 1:50–59, January 1999.

[25] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP congestion contorl with a misbehaving receiver. *Computer Communication Review*, 29(5), 1999.

[26] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. *IEEE/ACM Trans. Netw.*, 12(2):205–218, 2004.

[27] G. Su. *MOVE: Mobility with Persistent Network Connections*. PhD thesis, Columbia University, Oct. 2004.

[28] G. Su and J. Nieh. Mobile communication with virtual network address translation. Technical Report CUCS-003-02, Columbia University, Feb. 2002.

[29] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz. OverQoS: Offering Internet QoS using overlays. In *NSDI*, 2004.

[30] D. Tennenhouse and D. Wetherall. Towards an Active Network Architecture. In *Computer Communication Review*, 1996.

[31] F. Teraoka, Y. Yokote, and M. Tokoro. A network architecture providing host migration transparency. In *Proceedings of the ACM SIGCOMM '88 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 1991.

[32] J. Touch, Y. Wang, L. Eggert, and G. Finn. Virtual internet architecture. Technical Report ISI-TR-2003-570, Information Sciences Institute (ISI), March 2003.

[33] A. Vahdat, T. Anderson, and M. Dahlin. Active Naming: Programmable Location and Transport of Wide-area Resources, Aug. 1999.

[34] M. Walfish, J. Stribling, and M. Krohn. Middleboxes no longer considered harmful. In *OSDI*, 2004.

[35] P. Yalagandula, A. Garg, M. Dahlin, L. Alvisi, and H. Vin. Transparent mobility with minimal infrastructure. Technical Report TR-01-30, UT Austin, June 2001.

[36] X. Yang, D. Wetherall, and T. E. Anderson. A dos-limiting network architecture. In *SIGCOMM*, 2005.

[37] M. Zhang, J. Lai, A. Krishnamurthy, L. Peterson, and R. Wang. A transport layer approach for improving end-to-end performance and robustness using redundant paths. In *USENIX Annual Technical Conference*, June 2004.

[38] S. Zhuang, K. Lai, I. Stoica, R. Katz, and S. Shenker. Host mobility using an internet indirection infrastructure. In *Proc. of Mobisys*, 2003.