Opus: an Overlay Peer Utility Service

Rebecca Braynard, Dejan Kostić, Adolfo Rodriguez, Jeff Chase and Amin Vahdat*
Department of Computer Science
Duke University
{*rebecca,dkostic,razor,chase,vahdat*}*@cs.duke.edu*

Abstract

Today, an increasing number of important network services, such as content distribution, replicated services, and storage systems, are deploying overlays across multiple Internet sites to deliver better performance, reliability and adaptability. Currently however, such network services must individually reimplement substantially similar functionality. For example, applications must configure the overlay to meet their specific demands for scale, service quality and reliability. Further, they must dynamically map data and functions onto network resources—including servers, storage, and network paths—to adapt to changes in load or network conditions.

In this paper, we present Opus, a large-scale *overlay utility service* that provides a common platform and the necessary abstractions for simultaneously hosting multiple distributed applications. In our utility model, wide-area resource mapping is guided by an application's specification of performance and availability targets. Opus then allocates available nodes to meet the requirements of competing applications based on dynamically changing system characteristics. Specifically, we describe issues and initial results associated with: i) developing a general architecture that enables a broad range of applications to push their functionality across the network, ii) constructing overlays that match both the performance and reliability characteristics of individual applications and scale to thousands of participating nodes, iii) using Service Level Agreements to dynamically allocate utility resources among competing applications, and iv) developing decentralized techniques for tracking global system characteristics through the use of hierarchy, aggregation, and approximation.

1 Introduction

A key insight from the Active Networking approach is that distributed services can benefit tremendously from pushing their functionality to intermediate points in the network. The extreme stance of associating code with every packet that travels across the network has not seen wide deployment. However, instances of Active Networking abound in the current Internet architecture in the form of L4/L7 switches, overlay networks, transparent proxy caches, firewalls, and network address translation. Most of these network services provide point solutions to point problems. Overlay networks [3, 18, 20], however, are emerging as a fundamental technique for enhancing wide-area service scalability, performance, and availability. Consider the following applications:

- *Replicated Services:* To deliver target levels of performance and availability, application developers are increasingly replicating their service at multiple wide-area sites. Of course, replication is not a panacea. Important questions here include where to place replicas, how many replicas are required, how to route client requests to the proper replica, and how to propagate updates to maintain consistency across replicas.

- *Application Layer Multicast (ALM):* Despite over a decade of research, native IP multicast has not enjoyed wide-spread deployment. However, multicast-style applications, such as stock quotes, event notification, audio, and video, are continuing to grow in popularity. Thus, many industrial and research efforts are investigating and deploying ALM, where nodes across the Internet act as intermediate routers to efficiently distribute data along a pre-defined mesh or tree. Initial evaluations [18, 20] indicate that ALM can perform within a small factor of the cost and latency of native IP multicast by routing through strategic intermediate points in the network and by building an overlay that matches the characteristics of the underlying network.

*This research is supported in part by the National Science Foundation (EIA-9972879, ITR-0082912), Hewlett-Packard, IBM, Intel, and Microsoft. Braynard is supported by an NSF graduate fellowship and Vahdat is also supported by an NSF CAREER award (CCR-9984328).

- *QoS Guarantees:* Finally, more traditional unicast applications are also using distributed network resources to achieve better performance and reliability than would be delivered by the underlying IP network. One initial study [31] used network traces to determine that, in many cases, default IP routing results in paths with inferior reliability and performance relative to indirect routing through one of a set of intermediate nodes. Recent work on Resilient Overlay Networks [3] quantifies such improvements in a 13-node testbed running across the Internet. Another recent study [32] advocates using multiple intermediate points in an overlay to redundantly transmit the same data from source to destination, reducing end-to-end loss rates and performance variability.

Today, all of these services must redundantly acquire and administer nodes across the Internet to provide the requisite functionality. This approach forces services to reimplement substantially similar functionality, such as tracking changing network characteristics, building appropriate topologies, failure detection, or IP topology matching. Further, given the highly bursty nature of Internet traffic and service access patterns, individual services must overprovision for peak levels of demand that are often a factor of 3-10 higher than the average case.

We believe that a common system infrastructure to support the requirements of a broad range of applications will improve the performance of existing applications by exporting common best practices and will also effect a qualitative shift in the ease with which novel distributed applications can be deployed. Thus, we propose Opus, an Overlay Peer Utility Service, to automatically configure server network overlays with the goal of dynamically meeting the performance and availability requirements of a broad range of competing applications. By observing changing network conditions and application access patterns, Opus will: i) allocate a portion of global resources to each application, ii) place these replicas at appropriate points in the network, and iii) create overlays that satisfy the requirements of individual distributed applications. Key to our approach is building scalable structures to track changing system characteristics and developing a common abstraction for prioritizing among competing applications.

The rest of this paper is organized as follows. Section 2 describes the Opus system architecture. Next, Section 3 presents individual challenges we are addressing to realize this model and some initial results. Section 4 compares our work to related efforts and Section 5 presents our conclusions.

2 Architecture

The Opus service allocates overlays of server and network resources from a shared pool, as needed to meet service quality goals efficiently in the face of dynamically changing global characteristics. Service workloads are streams of requests originating from clients throughout the network, and requiring varying amounts of computation, shared data access, and data transfer to and from each client. We assume that service applications show stable average per-request behavior; loads are defined by offered request rates that may vary continuously through time.

Figure 1 depicts the high-level Opus system model. We envision a collection of server sites (e.g., small clusters or data centers) colocated with switching centers at the interior of the Internet "cloud." Opus manages these Points-of-Presence (PoPs) in a coordinated fashion as a shared physical infrastructure for distributed Internet applications. Applications consist of components running on selected Opus nodes. Opus configures these nodes to run application software and organizes them as an application-specific overlay network.

Opus resource allocators cooperate to assign resources to each overlay application. These resources include "slices" of the server and network capacity on some subset of the Opus PoPs—the pie charts in Figure 1 represent per-region application demand levels that ideally correspond to resource allocation levels in nearby Opus PoPs—together with an overlay topology configured for that application. Our approach is to describe applications abstractly in terms of their service quality goals, then generate candidate allotments and overlay topologies that balance service quality with network performance and cost. A *request routing* infrastructure directs external traffic (e.g., client requests) destined for each application service to selected nodes assigned to that application. Request routing may leverage DNS redirection [10], anycast [5, 21, 38], or an Opus naming interface.

The service overlay. Each Opus PoP runs an instance of the Opus *site manager*, which coordinates resource usage at that site and exchanges status summaries with other Opus sites. Opus uses its own overlay services internally to disseminate status information and related metadata through a *service overlay* that interconnects all active nodes. The service overlay forms the "backbone" for coordinated, decentralized resource allocation and resource control, as described below. Thus the service overlay must be dynamic and self-healing: if a network path is lost or degraded, then the service overlay must reconfigure to reroute traffic through a different path.

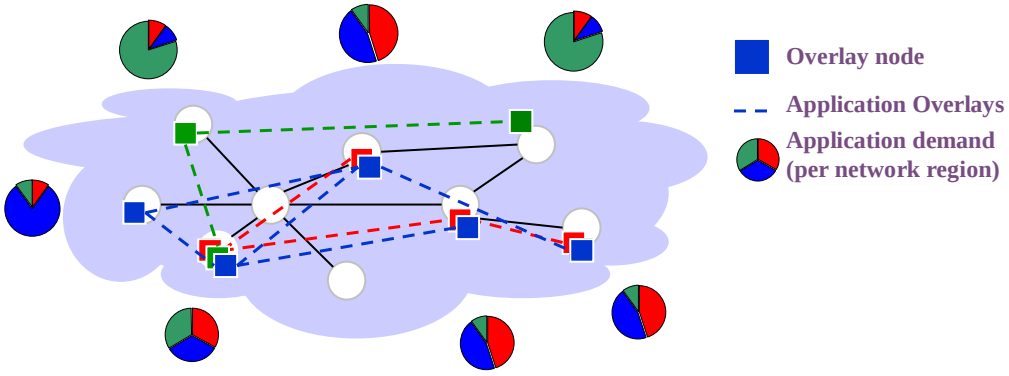

Figure 1: Opus system model.

Scalability is a key concern in the design of the service overlay, as we expect Opus to scale to 10,000 or more nodes across the wide area. We take a decentralized approach in which local site managers "think globally but act locally," making local resource allocation choices to converge on desirable global outcomes based on information disseminated through the service overlay. This status information includes summaries of resource availability, network conditions, load, and delivered performance at each site. The schemes for configuring overlays also require estimates of link capacities, delays, and failure probabilities for the physical network interconnecting the Opus sites. A key premise of the Opus architecture is that effective resource allocation and control requires only *approximate* information about global conditions. Section 3.3 presents some results from our initial approach to scalable dissemination of system metadata through the service overlay, called *dicast*.

Adaptive per-application overlays. A primary task of the service overlay is to assist in the construction and maintenance of application overlays. Individual applications use these overlays to route internal application traffic, disseminate content, and/or synchronize their state information efficiently. For example, a video delivery system would use its overlay to disseminate its content to participating sites, which in turn transmit the data to end clients. A replicated database system would use its overlay to maintain replica consistency by propagating updates among active replica sites.

At the core of the Opus architecture are algorithms to select the number and placement of site locations for each application, allocate global resource shares, and configure overlays to link the selected sites. These inter-related aspects of the overlay construction problem interact in a complex way to determine end-to-end application behavior. As an example, consider an Internet service using dynamic replication for scalability and availability. For a replicated service, the need to propagate updates across replicas imposes new network load and may compromise availability. Previous work in the TACT project has shown that the availability of a replica configuration depends on the application's consistency demands, as well as the number and placement of replicas and the reliability of their interconnections [39]. While adding server sites can improve performance and availability, more is not necessarily better: we find that in some cases additional sites can actually *reduce* overall performance and availability depending on application consistency requirements. In fact, a small set of well-placed and well-provisioned replica sites generally outperforms a larger set of poorly-placed replicas. The techniques developed for TACT allow the Opus allocators to predict performance and quantify availability as a function of the candidate overlay characteristics and the application's consistency targets.

After instantiating an overlay for an application, the Opus resource allocators dynamically adapt the overlay topology and site allotments to respond to observed load and network conditions. For example, if many accesses are observed for an application in a given network region, the system may reallocate additional resources at a PoP close to that location, possibly adding a new site presence to the application overlay. The system continuously monitors local and global conditions through the service overlay, and uses feedback control as a basis for incremental, adaptive resource provisioning. In addition to enabling dynamic adaptation, the feedback loop enables the system to continuously refine resource allotments so that the quality of an initial solution is less critical.

Resource allocation and service quality. Opus strives for resource allotments that are both *effective* and *efficient*. An effective allotment meets service quality goals; an efficient allotment conserves resources. One approach is to

strive for least-cost allotments that satisfy fixed application service quality bounds under existing traffic conditions. We take a more general approach to enable the system to prioritize applications under resource constraint. Although we expect that the utility is adequately provisioned and employs admission control to avoid overcommitting its resources, the Internet environment is adversarial, and large-scale services should design in "fallback" positions for extreme scenarios involving site or link failures, flash crowds, or attacks on the system or its physical infrastructure. For this reason, our approach emphasizes dynamic tradeoffs of service quality and cost. This can enable the system to match resource demand with dynamically varying levels of resource supply, in order to maximize the global good under the full range of conditions and constraints that it might encounter in operation. Indeed, a key benefit of the utility approach is that it can reallocate shared infrastructure to respond to adverse conditions. Such reallocation may take place based on economic considerations (e.g., who is willing to pay the most?) or based on relative application priority (e.g., which services must absolutely stay up and running during a denial of service attack?).

A major technical challenge for flexible resource allocation in an overlay utility service is to generate candidate overlay configurations with varying tradeoffs of cost and service quality (benefit). Section 3.1 presents two overlay structures we are investigating to support this objective. Dynamic cost/benefit tradeoffs also depend on models to predict and quantify the benefit of each candidate configuration along multiple dimensions of service quality. The models must consider non-traditional quality measures such as availability or consistency, as well as more traditional performance measures such as response time, fairness, or throughput. The units to quantify different dimensions of service quality and cost are arbitrary: the system may scale these measures arbitrarily before comparing or combining them to balance competing objectives.

Given measures for service quality and cost for candidate overlay configurations, the system needs flexible criteria to establish customer priority. Our initial approach is to present service quality goals to the resource allocators as *value-based* Service Level Agreements (SLAs) representing a continuum of service quality tradeoffs. The utility functions may represent arbitrary criteria for establishing customer priority. Opus SLAs specify service quality goals as continuous *utility functions* specifying values associated with various levels of service volume and quality for each customer. In our previous work on server provisioning in individual data centers, we found that generalized utility functions are a flexible means to guide dynamic tradeoffs of service quality and cost [8]. Modest constraints on the form of the utility functions enable the resource allocators to identify utility-maximizing allocations efficiently, and refine them incrementally in a feedback loop. Section 3.2 outlines our approach in more detail.

Security and isolation. Security is an important consideration for any general-purpose utility. Opus allocates resources to applications at the granularity of individual nodes, eliminating a subset of the security and isolation issues associated with simultaneously hosting multiple applications. In the future, we plan to investigate the use of virtual machine technology to isolate services running on the same physical host [37]. On the network side, we must still isolate traffic on the wire from different applications running at the same Opus site. VLANs, supported in most modern switches, support such functionality and it should be straightforward to automate the requisite isolation in response to dynamic reallocation of local site resources. Finally, policy-based sharing of physical resources depends on accurate measures of application resource demand. In some cases it may be useful for the customer itself to provide the load and QoS measures. If so, Opus relies on simple economics to encourage customers to deploy efficient software and accurately represent their resource needs for a given demand level: customers conserve resources when they are asked to pay for their usage.

3 System Components

This section describes four of the principal challenges that we must address to deploy a general-purpose and large-scale service utility: constructing overlays, allocating resources, propagating status, and improving reliability through multi-path routing. A common theme running across all system components is that local decisions must be made to approximate the global good based on partial and uncertain information.

3.1 Overlay Topology Construction

As discussed above, Opus must build and maintain two separate types of overlays. The service overlay maintains and distributes overall service metadata among participating sites. The service overlay also facilitates the construction of smaller-scale application overlays designed to meet the performance and reliability requirements of a broad range of network services. A number of efforts have investigated techniques for building proper overlay topologies to match a particular application's requirements [3, 18, 20]. Further, the scalability of current techniques require global

knowledge and do not scale beyond a few tens of nodes.. We must devise solutions that scale to thousands of nodes for application overlays and tens of thousands of nodes for the service overlay.

Our initial work focuses on developing a general overlay topology that enables dynamic tradeoffs between network performance/reliability and cost. Note that a cost of an overlay link can be assigned arbitrarily, but is likely to depend upon the cost of the individual physical links that compose an overlay link. This cost may reflect current congestion levels on a link, the price paid to an ISP to use a link, etc. The actual assignment of cost to individual links is beyond the scope of this paper, though we do assume that no individual Opus nodes are aware of this global cost metric and that the metric changes dynamically over time.

One of the key initial goals in our work is to build application overlays to enable flexible and dynamic tradeoffs between overlay cost—logically a measure of total network resources consumed in transmitting data across the overlay—and the associated performance and reliability characteristics of the overlay. To quantify the benefits of competing structures, we need a set of metrics to compare the quality of candidate overlay topologies. Initially, we focus on network cost and relative delay penalty (RDP) to characterize overlay topologies. RDP measures the relative increase of delay incurred from using a particular overlay relative to direct transmission in the underlying IP network. Network cost is the sum of all the link weights associated with a given overlay topology.

We have identified two candidate overlay topologies that enable such flexible tradeoffs [23]. A *k-spanner* [7] ensures that all paths in an overlay have an RDP no worse than k. Lower values of k result in higher cost for building the overlay. Because k-spanners attempt to guarantee low-latency paths between all pairs of hosts, it is more appropriate for multi-sender applications. A second structure, *LAST* [22] (lightweight approximate shortest-path tree), enables similar tradeoffs for single-sender applications. With a LAST, a configuration parameter, α, bounds the RDP of all paths from a designated source to all destinations have an RDP no worse than α. For instance, a LAST with $\alpha = 1.5$ ensures that all destinations receive data with delay at most 50% higher than transmission through IP.

We now present the results of some initial experiments to quantify the benefits of k-spanners and LASTs. The principal goal here is to enable Opus to use overlay-specific tuning parameters to match application requirements. For example Opus can adapt to changing conditions by turning a knob (such as the k or the α value) to reallocate resources to adjust the balance of cost and performance. To quantify the benefits of dynamically trading network cost for performance in overlays, we ran some simulations of both k-spanners and LASTs. For our experiments, we constructed a 200-node overlay randomly distributed among a 600-node GT-ITM generated topology [6]. Edge delay was assigned based on default GT-ITM parameters. For these experiments, we equate edge cost with delay though we are currently investigating techniques to allow simultaneous, bi-criteria network optimization [25]. In the case of a LAST, Figure 2(a) shows how the α parameter affects the cost of the resulting overlay, relative to both a shortest path tree (RDP=1.0) and a minimum cost spanning tree (with an unbounded RDP). At $\alpha = 1$, the overlay cost is high, comparable to a shortest path tree. However, as demonstrated in Figure 2(b), this same point corresponds to the best performance (comparable to shortest path routing packets in the underlying network). As α increases, the network cost of the LAST overlay decreases, eventually matching the cost of a minimum cost spanning tree at $\alpha = 3$. Of course, Figure 2(b) also shows that such a low-cost overlay also results in relatively poor performance. One nice quality of the tradeoffs expressed above is that it is possible to build distribution structures that balance cost and RDP. For example, with $\alpha = 1.5$, we are able to obtain a cost within 15% of an MST and an RDP within 15% of an SPT for our target topology and edge weights. This result shows promise for our ability to build overlays that match application requirements with relatively low cost overhead (for all but the most demanding applications).

A key next challenge is to develop scalable distributed algorithms for building and maintaining k-spanners or LASTs. To support our goal of scalability, we must avoid the necessity of global knowledge, excessive network probing, and distributed locking to build and maintain such topologies. Our approach is to use probabilistic techniques and hierarchy to selectively probe the characteristics of various network regions. Key to our approach is having each node gradually migrate to its (approximately) "proper" location in the overlay. This is relatively straightforward assuming the presence of global group membership and pairwise probing. However, this requires unscalable $O(n^2)$ memory and network overhead respectively). Recent proposals in peer-to-peer networking address scalability concerns by building randomized overlays [28, 30, 33] requiring only $O(\lg n)$ per-node state. In contrast, our goal is to investigate the practicality of constructing overlays with specific performance characteristics using partial, approximate and probabilistic knowledge of network information.

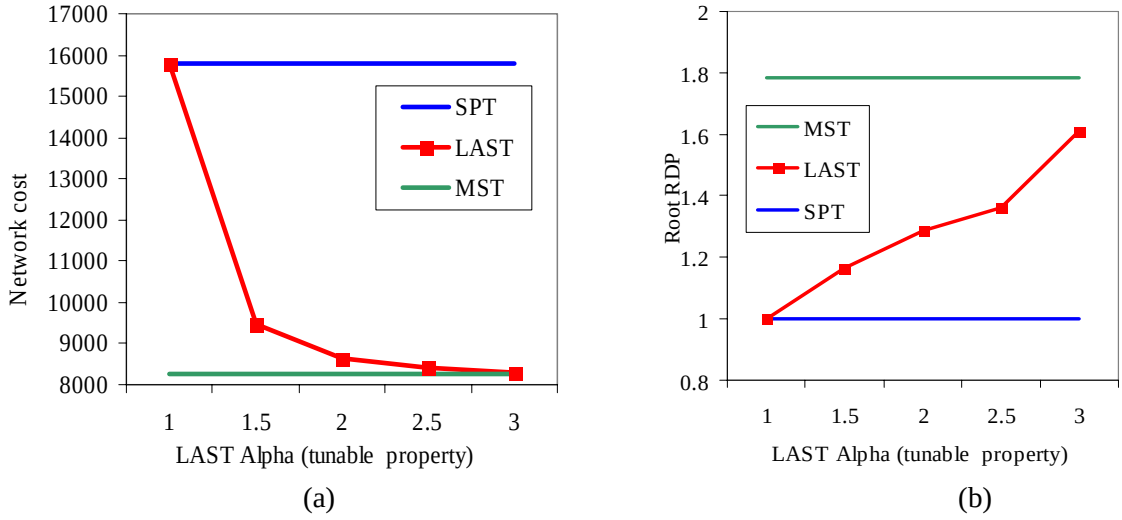

Figure 2: Dynamically trading network cost for relative delay product using a lightweight approximate shortest path tree (LAST).

3.2 Resource Allocation

One of the key components of Opus is resource allocation among competing applications. This principally requires determining the relative priority for competing applications and the proportion of global resources that should be allocated based on current system conditions. We will use SLAs as the basis for economic prioritization, building on our initial success with using an economic model for prioritization and resource allocation in a cluster setting [8].

The basic resource mapping challenge is to establish a matrix of allotments from j system resources across i customers (applications). The system resources include servers in the Opus PoPs and network links interconnecting them. The system strives to balance the service quality of the selected allotments with their costs. In Opus, our challenge is to allocate these resources in a decentralized manner, based on partial information about resource supply and demand collected through the service overlay.

Opus uses a generalized measure of benefit or utility as a basis for flexible SLAs representing dynamic tradeoffs between service quality and value. Customers are associated with *utility functions* specifying the value of any given level of service volume and service quality predicted to result from a candidate allotment. Opus makes resource allocation decisions by comparing the expected utility of a set of candidate configurations, with the goal of maximizing global utility. The system uses models to predict the effects of candidate resource allotments on service quality, then evaluates the SLA functions to determine the expected value of the predicted behavior. Informally, the domains of these composite functions are continuous measures of the cost of resources assigned, e.g., the aggregate amount of server resources assigned to the application at the Opus PoPs, or the network cost of a LAST tree with a given α parameter. The units of value are arbitrary, as long as the system can combine values assigned to multiple measures of service quality, and compare the total values of candidate configurations to determine which of the alternatives is preferable.

The resulting optimization problems fall into a classical economic framework for resource allocation. Computing optimal resource allocations from sets of utility functions and service quality estimates is a linearly constrained non-linear optimization problem. To make the problem tractable, we constrain the composite utility functions to be *concave*. This means that the marginal benefit of assigning additional resources, e.g., servers or network links, to a configuration declines steadily and approaches zero: adding resources beyond some point does not result in meaningful improvement of service quality. More formally, the utility gradient (the derivative if the function is differentiable over a real-valued domain) is non-negative and monotonically non-increasing. This reduces the optimization problem to a simple convex programming problem with an efficient solution based on gradient climbing[19]. If there are sufficient resources to avoid starving any customer, then there exists a unique optimal solution with the property that the marginal utility of an additional resource unit is in equilibrium across all cus-

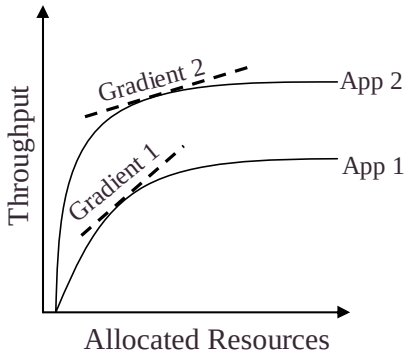

Figure 3: Example of gradient climbing to determine resource allocation.

tomers. This equilibrium marginal utility is equivalent to the *equilibrium price* that matches resource demand with available supply in an economic market for allocating resources.

A simple example helps to illustrate this point. Suppose that an Opus system hosts two application services with a constant level of offered load. Figure 3 shows concave curves that qualitatively represent the throughput of the two hypothetical applications as a function of the resources allocated to each. If the SLA functions for these customers define value as linear with delivered throughput, and they have equal priority (their utility functions have the same slope), then Opus will seek a resource allotment that maximizes global throughput. Note that while we use throughput in this simple example, the y-axis could just as easily represent availability, reliability, latency, or some other service quality metric, e.g., a composite metric representing expected customer revenue.

The curves show that adding resources significantly improves throughput when allotments are low and the customers are starved. However, as more resources are added, the marginal gain in throughput declines and approaches zero (trivially, for an offered load of 100 small file requests per second, changing allocation from 10 to 11 machines is not likely to measurably improve throughput). The marginal benefit of an additional resource unit can be measured by the first derivative or gradient of each application's "utility curve." In this example, the gradients at particular points on the x-axis represent the current allocation of resources to each application and the expected benefit of allocating an additional unit of resource to application 1 versus application 2. Here, application 1 would enjoy a greater estimated boost in throughput from an additional unit of resource because it has a larger gradient than application 2. Opus gives preference to application 1 until its marginal gain equalizes.

Thus, the Opus resource allocators strive to maximize global value across all applications. In the general case, the SLA functions may specify utility as a combination of service quality metrics in a "common currency" of value. The utility functions may also incorporate priority by valuing service quality for some applications higher than others. For example, the value metrics would prioritize, say, dissemination of tactical information over distribution of training videos, enabling the system to provision resources rationally if faced with an unexpected crisis and resource shortage. The value of the allotments changes dynamically with changing conditions and offered load. Our challenge then is to estimate the changing shape and gradient for these curves to respond to dynamic changes, based on partial knowledge propagated through the service overlay.

Overall, the concavity constraint allows the system to adjust equilibrium allotments incrementally to adapt to changing conditions. The system continuously monitors load and resource status, and propagates status information through the service overlay. This status information constitutes a feedback signal to trigger adaptive resource reallocation. Rather than computing a new allocation from scratch, the system responds to changes by incrementally adapting an existing configuration to restore equilibrium. This can be done using an efficient greedy algorithm whose cost scales with the magnitude of shifts in load or resource availability from one interval to the next [8].

Economic resource allocation scales naturally using a decentralized federation of autonomous local "markets" exchanging information to converge toward a global equilibrium. Our initial design centers around a hierarchical structure to aggregate related resources into *cells* capable of planning their internal allocations locally. A cell might be an entire Opus PoP or a portion of a large PoP, e.g., an array of generic servers sharing a redirecting switch node in close proximity. Cells cooperate to trade load or resources in order to balance resource usage across the systems. To derive the magnitude of resource shifts, cells exchange information about the supply and demand for resources in each cell. This can be captured compactly as the marginal utility gained by adding resources to that cell or shifting load away from that cell to free up resources for some other use; this marginal utility is equivalent to the "price" for resources in that cell. We believe that this cellular structure is the key to scalable resource provisioning in large data centers and networks of server sites.

A key tenet of this work is that service quality must be measured in an application-specific manner. Thus, one important question involves incorporating multiple dimen-

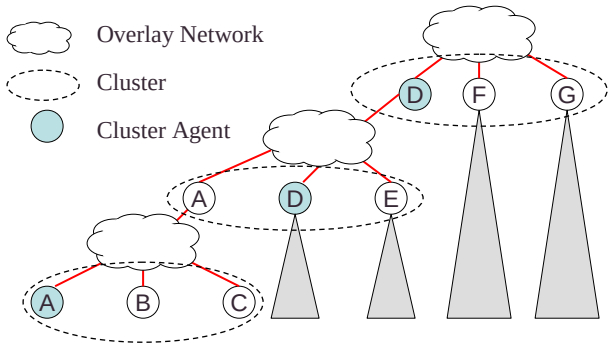

Figure 4: Hierarchical data dissemination in dicast.

sions of service quality, including reliability, performance, and data consistency, into a single utility function. One option is to define a unified *performability* measure incorporating all aspects of service quality, with a single utility function for each customer. An alternative is to define each dimension of service quality as a separate utility function, and represent tradeoffs optimizing the sum of the individual value measures.

3.3 Scalable Tracking of System Characteristics

As discussed above, a primary challenge to building and maintaining large scale utilities involves maintaining distributed state about global system characteristics. Consider the requirements of the following Opus tasks. First, for request routing, clients spread across the network must choose the replica most likely to deliver the best performance, reliability, security, etc. To achieve such functionality, the request routing infrastructure must track dynamically changing replica characteristics, for instance, available bandwidth and load information. Second, building and maintaining overlays requires probing the network characteristics among all participating replicas. In general, nodes "near" one another in the underlying topology (i.e., displaying strong pair-wise performance and reliability characteristics) should peer together in the overlay. Finally, the system must track dynamic group membership information to retire nodes that fail or fall behind long-lasting network partitions. Thus, an Opus node requires an abstraction to communicate its local state and local observations (e.g., network probes) to other system nodes. Similarly, Opus nodes must receive updates about global system characteristics from remote sites. In a large-scale utility, it is impractical to maintain accurate global system characteristics. Our challenge then is to balance communication costs with data accuracy as a function of system size and global characteristics.

To develop a communication abstraction able to scale to large numbers of nodes, we draw inspiration from Internet routing protocols [16, 26, 29], perhaps the best example of distributed protocols that scale to global proportions. The fundamental lesson we draw is that *aggregation*, *hierarchy* and *approximation* are fundamental to wide-area scalability. We apply these design ideas to a generic communication library within Opus, called dicast, designed to distribute approximate data in a scalable fashion. Thus, not all updates originating at a given node will be (or even need be) delivered to all participants. Further, individual updates may be aggregated together to increasing degrees as data moves through the network.

The use of aggregation in dicast naturally leads to the construction of a tree-based structure, as depicted in Figure 4. Nodes are partitioned into clusters of size d, where d determines the height of the tree (for n nodes, a cluster size of d implies a tree height of approximately $\lg_d n$). Each cluster elects an *agent*, a speaker responsible for disseminating local cluster information to the rest of the dicast tree. Agents from d adjacent clusters form second-level clusters. This process is repeated until an h-th level cluster is formed, where h is the height of tree. Note that all physical nodes in the dicast tree are at the leaves (first-level clusters) and intermediate nodes in the tree are elected members from the leaf set who serve multiple responsibilities. Deriving good performance from such an approach requires assigning nodes to clusters with other topologically "nearby" nodes. We plan to leverage existing work on clustering [24] to aid in this process where possible.

In dicast, data travels up the tree, potentially being aggregated with data from other dicast nodes. At each level of the tree, an overlay (as discussed in Section 3.1 above) propagates the data among all participating cluster members. Associated with each level of the tree is a target (application-specific) level of accuracy for either aggregated or individual node information. Once a particular update reaches a level of the tree where aggregate accuracy requirements are not violated, it will be buffered awaiting the arrival of further updates that will eventually force the propagation of an aggregate update to nodes higher up the tree. As data spreads to higher-level clusters, it is in turn transmitted back toward the leaves because each agent is a member of at least two adjacent levels in the tree.

One example use of dicast is to propagate per-region resource consumption information to influence local resource allocation decisions. Thus, a local node may have exact information about per-application resource consumption for "nearby" nodes (in the same cluster). However, it may only have aggregate (and somewhat inaccurate) in-

formation about resource consumption in remote clusters. However, such approximate and aggregate data is likely to be sufficient to set local allocation levels to meet global allocation targets. Similarly, information on per-cluster load imbalances may be used to make a decision to reallocate a given replica from one application to another to better meet target SLAs or to maximize global system throughput.

3.4 Reliability QoS Guarantees

For many emerging Internet services, reliability and availability are more important metrics than raw service performance. There are a number of potential definitions for service availability; we define availability to be the percentage of requests that can be satisfied within individual client performance requirements. Many existing metrics for availability consider a service "available" if it is currently satisfying client requests with availability reducing to a simple measure of uptime, or the amount of time without hardware/software failures. For our approach, availability is measured by integrating across all client requests, with those requests that return too slowly (e.g., based on an expected distribution or even on per-client performance expectations) marked as "unavailable."

In the context of a replicated utility, an individual hosted service may be considered unavailable for a number of reasons, including failures in the request routing infrastructure, in network links, or, in our more general model, because insufficient resources were allocated to meet target performance characteristics. One approach we are pursuing for addressing failures at the network level, called *restricted flooding*, is to build overlay topologies that redundantly transmit the same data over multiple logical paths [32]. However, we use a variant of anti-entropy [34] to minimize the overhead associated with redundant transmission for certain application classes. Here particular overlay nodes may choose to forward an application-layer frame redundantly along multiple paths to a single destination, especially if any given path does not meet aggregate reliability requirements. As the data travels toward its destination, certain downstream nodes may receive multiple copies of the same frame (as identified by a unique identifier). In this case, the downstream node will re-evaluate the estimated reliability of the remainder of the path and suppress duplicate frames if reliability targets are likely to be achieved with the propagation of a single frame. This manner of restricted flooding provides two principal advantages. First, restricted flooding means that the overlay does not have to necessarily prevent "loops," simplifying overlay construction. Next, multiple independent paths to the destinations mean that individual delays, failures, or packet drops will not necessarily prevent the timely delivery of data given available redundancy in the distribution graph.

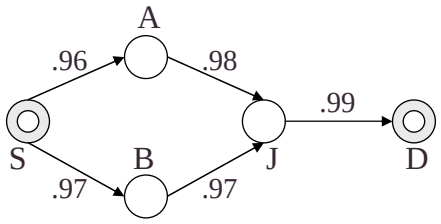

Figure 5: Using Restricted Flooding to control cost versus reliability tradeoffs.

A primary challenge to developing such an approach is ensuring that the overlay topology matches the failure characteristics of the underlying network. For instance, if separate logical links in an overlay correspond to a common failure-prone link in the underlying physical network, a failed physical link can result in failures in multiple logical overlay links. Thus, it is important to construct overlays with *disjoint paths*, where the failure correlation among logical overlay links is low. We determine the loss correlation among multiple potential links by collecting statistical information about loss correlations and by using network topology information where available. Our use of multiple redundant paths enables immediate failover rather than relying on the underlying network to converge to new routes in the face of failure. Further, we hope that the combination of restricted flooding and careful construction of overlay topologies will result in only minimal traffic overhead relative to single-path routing.

Consider the simple example depicted in Figure 5 where a source S wishes to transmit data to a destination, D, with an end-to-end reliability of 98% and where all links are disjoint. Omitting the details of the simple calculations, transmitting the data through either A or B toward D results in reliability of 93.1%. However, by transmitting data through both A and B means that at least one copy of the data arrives at the join point, J, 99.6% of the time, with two copies of the data arriving with an 88.5% probability. When node J forwards one copy of the data (suppressing the second should it arrive later), resulting end-to-end reliability is 98.7%, which meets the target yield. Forwarding both copies results in 99.8% reliability.

The goal of our work in restricted flooding is to provide each node with enough information to determine how

many simultaneous routes to maintain for a given communication stream to achieve a given level of reliability. Intermediate nodes must then determine if it is feasible to suppress subsequent transmission of the same data and still maintain target reliability. In this example, restricted flooding must determine if an approximately 88% increase in the utilization of the overlay edge JD for this particular communication stream is worth the potential 1.1% improvement in end-to-end reliability. Of course, this evaluation must be made in response to changing network conditions and application demands.

Finally, in Section 3.1, we discussed techniques for allowing application developers to dynamically trade "cost" for performance. Our approach to providing high reliability through redundant transmission and disjoint paths adds another dimension to this tradeoff: it allows applications to specify both performance and reliability targets. Opus then strives to build the lowest cost (or lowest overhead) overlay to meet the specified goals.

4 Related Work

Our work on Opus is inspired by related efforts in a number of different fields. Research into Active Networks [1, 17, 27, 36] proposes moving computation into the network on a per-packet level. We view our utility model as a logical culmination of the Active Network philosophy. That is, overlays push application-level functionality to specific intermediate nodes in the network. However, the granularity of computation in overlays is coarser grained than in Active Networks, operating on application-layer frames [9] rather than individual network packets. In designing the abstractions for our utility environment, we will build on the work already performed in the context of Active Networks.

Work into Active Services [2] investigates a similar intermediate point of pushing application functionality into the network. Relative to this effort, we focus on the wide-area issues associated with simultaneously deploying and allocating resources among competing applications in a scalable utility. Where possible, we intend to leverage the set of abstractions developed for active services running within a cluster environment (analogous to our individual Opus sites).

A number of efforts are investigating a utility model for wide-area computing. Akamai [10] hosts a large number of servers across the Internet. Globus [13] and Legion [14] investigate resource allocation in the context of a wide-area computational grid. WebOS [35] investigates system support for wide-area services. Within a single machine room, Cluster Reserves enforces a global allocation of resources among multiple "resource principals" [4]. Relative to these efforts, our goal is to simultaneously investigate issues of resource allocation, replica placement, and overlay construction based on an economic model to determine per-application priority levels. We believe that our work in Opus will be complementary to these existing efforts.

Our utility model investigates techniques for allocating network resources to competing applications. We leverage overlay networks both to track the characteristics of the utility as a whole, as well as to propagate updates among individual application nodes. The idea of an overlay network is not new, having been leveraged to ease the deployment of both multicast in the Mbone [12] and IPv6 in the 6bone [15]. Until recently, overlays were viewed as a transition technology. However, recent academic and commercial efforts are advocating the use of overlays as a fundamental approach for both deploying new network functionality (e.g., multicast [18, 20]) and for improving the performance and availability of existing applications (e.g., improved application-layer routing [3, 31, 32]). Relative to existing approaches, our work is a general utility infrastructure to allocate nodes among competing applications. Further, we investigate fundamental techniques for scaling overlay networks to thousands of nodes and for designing, implementing, and evaluating distributed algorithms for building and maintaining overlays capable of matching application performance and availability requirements.

Recently, there has been tremendous interest in scalable peer-to-peer lookup services [11, 28, 30]. At a high level, these systems hash an object name to a key within some address space and randomly assign cooperating peers to be responsible for some region of this address space. An end client wishing to lookup a particular object performs the hash and uses the lookup infrastructure to route its request to the appropriate peer in $O(\lg n)$ application-level hops. The system is scalable in that peers maintain no more than $O(\lg n)$ state in facilitating this lookup. These elegant designs provide significant scalability benefits at the cost of loss of control over exactly how nodes are interconnected, the cost of resulting overlays, etc. Our work on resource allocation and managing inexact information across the wide area is orthogonal to these efforts. However, one explicit goal of this work is to determine the relative performance benefits and computational/communication of explicit versus implicit overlay construction and maintenance in large scale distributed systems.

5 Conclusions

This paper presents a novel model for wide-area computing where a collection of server sites distributed across the Internet simultaneously support the requirements of a broad range of decentralized Internet applications. Rather than forcing individual applications to reimplement significant functionality and to redundantly administer distributed service resources, an *overlay peer utility service*, Opus, dynamically allocates resources among competing applications. This paper describes our approach to realizing this vision and some of the specific research issues we are addressing. In particular, we present: i) the system architecture and abstractions necessary for diverse applications to push functionality to intermediate nodes, ii) models for resource allocation and replica placement for competing applications based on dynamically changing system characteristics, iii) constructing dynamic per-application scalable overlays that both match application performance/availability requirements and that make efficient use of underlying network resources, and iv) decentralized and scalable techniques for tracking global system characteristics through aggressive use of hierarchy, aggregation, and approximation.

References

[1] D. Scott Alexander, William A. Arbaugh, Michael W. Hicks, Pankaj Kakkar, Angelos D. Keromytis, Jonathan T. Moore, Carl A. Gunter, Scott M. Nettles, and Jonathan M. Smith. The SwitchWare Active Network Architecture. *IEEE Network*, 12(3):29–36, May/June 1998.

[2] Elan Amir, Steven McCanne, and Randy Katz. An Active Service Framework and its Application to Real-Time Multimedia Transcoding. In *Proceedings of SIGCOMM*, September 1998.

[3] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient Overlay Networks. In *Proceedings of SOSP 2001*, October 2001.

[4] Mohit Aron, Peter Druschel, and Willy Zwaenepoel. Cluster Reserves: A Mechanism for Resource Management in Cluster-based Network Servers. In *Proceedings of the ACM Sigmetrics 2000 International Conference on Measurement and Modeling of Computer Systems*, June 2000.

[5] S. Bhattarcharjee, M. Ammar, E. Zegura, V. Sha, and Z. Fei. Application-Layer Anycasting. In *Proceedings of IEEE Infocom*, April 1997.

[6] Ken Calvert, Matt Doar, and Ellen W. Zegura. Modeling Internet Topology. *IEEE Communications Magazine*, June 1997.

[7] Barun Chandra, Gautam Das, Giri Narasimhan, and Jose Soares. New Sparseness Results on Graph Spanners. In *Symposium on Computational Geometry*, pages 192–201, 1992.

[8] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP)*, October 2001.

[9] David D. Clark and David L. Tennenhouse. Architectural Considerations for a New Generation Protocols. In *Proceedings of SIGCOMM*, September 1990.

[10] Akamai Corporation, 1999. `www.akamai.com`.

[11] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area Cooperative Storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, October 2001.

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

[13] Ian Foster and Carl Kesselman. Globus: A Metacomputing Infrastructure Toolkit. In *International Journal of Supercomputer Applications*, volume 11(2), pages 115–128, 1997.

[14] Andrew S. Grimshaw, William A. Wulf, and the Legion team. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1), January 1997.

[15] I. Guardini, P. Fasano, and G. Girardi. IPv6 Operational Experience within the 6bone. In *Proceedings of the Internet Society Conference*, July 2000.

[16] Roch Guerin and Ariel Orda. QoS-based Routing in Networks with Inaccurate Information. In *Proceedings of IEEE INFOCOM*, 1997.

[17] Michael Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles. PLAN: A Packet Language for Active Networks. In *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming Languages*, pages 86–93, 1998.

[18] Yang hua Chu, Sanjay Rao, and Hui Zhang. A Case For End System Multicast. In *Proceedings of ACM Sigmetrics*, June 2000.

[19] Toshihide Ibaraki and Naoki Katoh, editors. *Resource Allocation Problems: Algorithmic Approaches*. MIT Press, Cambridge, MA, 1988.

[20] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and Jr. James W. O'Toole. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of Operating Systems Design and Implementation (OSDI)*, October 2000.

[21] Dina Katabi and John Wroclawski. A Framework for Scalable Global IP-Anycast. In *Proceedings of Sigcomm*, August 2000.

[22] S. Khuller, B. Raghavachari, and N. Young. Balancing Minimum Spanning and Shortest Path Trees. In *Proc. ACM/SIAM Symp. on Discrete Algorithms*, January 1993.

[23] Dejan Kostić and Amin Vahdat. Latency versus Cost Optimizations in Hierarchical Overlay Networks. Technical Report CS-2001-04, Duke University, January 2002.

[24] Balachander Krishnamurthy and Jia Wang. On Network-Aware Clustering of Web Clients. In *Proceedings of ACM SIGCOMM 2000*, August 2000.

[25] Adam Meyerson, Kamesh Munagala, and Serge Plotkin. Cost-Distance: Two Metric Network Design. In *Proceedings of the Symposium on the Foundations of Computer Science (FOCS)*, November 2000.

[26] J. Moy. OSPF Version 2. Technical Report RFC 2178, Internet Engineering Task Force, Network Working Group, July 1997.

[27] Erik L. Nygren, Stephen Garland, and M. Frans Kaashoek. PAN: A High-Performance Active Network Node Supporting Multiple Mobile Code Systems. In *Proceedings IEEE OpenArch 1999*, March 1999.

[28] Sylvia Ratnasamy, Paul Francis Mark Handley, Richard Karp, and Scott Shenker. A Content Addressable Network. In *Proceedings of SIGCOMM 2001*, August 2001.

[29] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). Technical Report RFC 1771, Internet Engineering Task Force, Network Working Group, March 1995.

[30] Antony Rowstron and Peter Druschel. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, October 2001.

[31] Stefan Savage, Thomas Anderson, Amit Aggarwal, David Becker, Neal Cardwell, Andy Collins, Eric Hoffman, John Snell, Amin Vahdat, Geoff Voelker, and John Zahorjan. Detour: A Case for Informed Internet Routing and Transport. *IEEE Micro*, 19(1), January 1999.

[32] Alex C. Snoeren, Kenneth Conley, and David K. Gifford. Mesh-Based Content Routing Using XML. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, October 2001.

[33] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer to Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 SIGCOMM*, August 2001.

[34] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, December 1995.

[35] Amin Vahdat, Thomas Anderson, Michael Dahlin, Eshwar Belani, David Culler, Paul Eastham, and Chad Yoshikawa. WebOS: Operating System Services for Wide-Area Applications. In *Proceedings of the Seventh IEEE Symposium on High Performance Distributed Systems*, Chicago, Illinois, July 1998.

[36] David Wetherall. Active Network Vision and Reality: Lessons From a Capsule-based System. In *Proceedings of the 17th Symposium on Operating Systems Principles (SOSP)*, December 1999.

[37] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Denali: Lightweight Virtual Machines for Distributed and Networked Applications. Technical Report 02-02-01, University of Washington, 2002.

[38] Chad Yoshikawa, Brent Chun, Paul Eastham, Amin Vahdat, Thomas Anderson, and David Culler. Using Smart Clients to Build Scalable Services. In *Proceedings of the USENIX Technical Conference*, January 1997.

[39] Haifeng Yu and Amin Vahdat. Design and Evaluation of a Continuous Consistency Model for Replicated Services. In *Proceedings of Operating Systems Design and Implementation (OSDI)*, October 2000.