2009-18

Performance-Engineered Network Overlays for High Quality Interaction in Virtual Worlds

Authors: Mart Haitjema, Ritun Patney, Jon Turner, Charlie Wiseman, John DeHart

June 8, 2009

Corresponding Author: mah5@arl.wustl.edu

Abstract: Overlay hosting systems such as PlanetLab, and cloud computing environments such as Amazon™s EC2, provide shared infrastructures within which new applications can be developed and deployed on a global scale. This paper ex-plores how systems of this sort can be used to enable ad-vanced network services and sophisticated applications that use those services to enhance performance and provide a high quality user experience. Specifically, we investigate how advanced overlay hosting environments can be used to provide network services that enable scalable virtual world applications and other large-scale distributed applications requiring consistent, real-time performance. We propose a novel network architecture called Forest built around per-session tree-structured communication channels that we call comtrees. Comtrees are provisioned and support both unicast and multicast packet delivery. The multicast mechanism is designed to be highly scalable and light-weight enough to support the rapid changes to multicast subscriptions needed for efficient support of state updates within virtual worlds. We evaluate performance using a combination of analysis and experimental measurement of a partial system prototype that supports fully functional distributed game sessions. Our results provide the data needed to enable accurate projections of performance for a variety of session and system configurations.

Performance-Engineered Network Overlays for High Quality Interaction in Virtual Worlds

Mart Haitjema
Washington University
+1-314-935-6124
mah5@arl.wustl.edu

Ritun Patney
Washington University
+1-314-935-4306
ritun@arl.wustl.edu

Jonathan Turner
Washington University
+1-314-935-8552
jon.turner@wustl.edu

Charlie Wiseman
Washington University
+1-314-935-4586
wiseman@wustl.edu

John DeHart
Washington University
+1-314-935-7329
jdd@arl.wustl.edu

ABSTRACT

Overlay hosting systems such as PlanetLab, and cloud computing environments such as Amazon's EC2, provide shared infrastructures within which new applications can be developed and deployed on a global scale. This paper explores how systems of this sort can be used to enable advanced network services and sophisticated applications that use those services to enhance performance and provide a high quality user experience. Specifically, we investigate how advanced overlay hosting environments can be used to provide network services that enable scalable virtual world applications and other large-scale distributed applications requiring consistent, real-time performance. We propose a novel network architecture called *Forest* built around per-session tree-structured communication channels that we call *comtrees*. Comtrees are provisioned and support both unicast and multicast packet delivery. The multicast mechanism is designed to be highly scalable and lightweight enough to support the rapid changes to multicast subscriptions needed for efficient support of state updates within virtual worlds. We evaluate performance using a combination of analysis and experimental measurement of a partial system prototype that supports fully functional distributed game sessions. Our results provide the data needed to enable accurate projections of performance for a variety of session and system configurations.

Keywords. network games, overlay networks, network processors, virtual worlds, cloud computing

1. INTRODUCTION

Network overlays have become an important tool for implementing Internet applications that require advanced services not available in the public Internet. While content-delivery networks provide the most prominent example of the commercial application of overlays [DI02, KO04], systems researchers have developed a variety of experimental overlay applications, demonstrating that the overlay approach can be an effective method for deploying a broad range of innovative systems [FR04, RH05, ST02]. Rising traffic volumes in overlay networks, and growing interest in the use of overlays for applications requiring consistent quality of service, make the performance of overlay nodes and overlay hosting services an issue of growing importance. Research testbeds such as Emulab [WH02] and PlanetLab [PE02] enable the development of experimental systems using overlay techniques but have been ineffective as service delivery vehicles, leading to efforts to create overlay hosting platforms that can support "internet-scale traffic volumes with router-like performance" [TU07]. NSF's GENI initiative [GENI] seeks to create a large-scale overlay hosting service that can support "at-scale" deployment of new network services and applications.

While the academic research community has been working to develop virtualized network testbeds capable of supporting multiple overlay networks, industry has been developing large-scale *cloud computing* infrastructures for similar purposes. While cloud computing is oriented more towards the delivery of scalable web services than advanced network services, it is built on much of the same technology base as the network testbeds. The scale and low cost of these cloud-computing infrastructures makes them a promising venue for the development of new applications based on overlay methods, potentially leading to more rapid innovation in advanced network services and applications. Services, such as Amazon's EC2 give developers a high degree of control over their "in-cloud" computing infrastructure, enabling developers to engineer systems that deliver complex services effectively, while allowing them to match the deployed resources to user demand on an hour-by-hour basis.

This paper is part of a larger research agenda centering on the use of shared infrastructures such as those provided by overlay hosting and cloud computing services. We are particularly concerned with applications for which a high quality user-experience depends on non-stop delivery of potentially complex, multimedia data streams. Such applications must be engineered to deliver consistent performance using a combination of dynamic provisioning mechanisms that respond to changing traffic loads and session-level resource-allocation mechanisms. Here, we explore the application of performance-engineered overlays to support high quality interaction in virtual worlds. We focus on overlays for highly interactive games, such as the first-person shooter genre, as these provide a readily accessible application testbed that exhibits very demanding performance requirements. However, we are also interested in the use of virtual worlds to support real-world collaboration, and this has led us to structure the underlying network services in a more general way, than we might, if we were concerned only with first-person shooters.

The rest of this paper is structured as follows. In Section 2, we describe the characteristics of virtual world applications as well as that of the overlay environment needed to support these applications. Section 3 describes the Forest overlay network architecture, and the services it provides. In Section 4, we describe a prototype implementation of the system with a distributed first-person shooter game that we adapted to use Forest. We evaluate the performance of the prototype in section 5 and evaluate the inherent scalability of the network architecture. Section 6 contains a discussion of related research and we close with a few remarks about the implications of our work and some future directions in Section 7.

2. DESIGN CONSIDERATIONS

2.1. Application Characteristics

We are primarily concerned with the network level services needed to support interactive virtual environments. However, we need some understanding of the application in order to make informed choices for the network services.

Virtual worlds are used in a variety of applications, from fast-paced first-person shooter games to role-playing games and socially-oriented worlds such as Second Life [RO03]. One important distinction among the different types of virtual worlds is the degree of interactivity and the degree to which consistent performance is essential to user satisfaction. The first-person shooters (FPS) are arguably the most demanding in this respect. Even small delays in the reactions of avatars to user input can make games difficult to play, causing users to lose interest. [CL06] quantified these delay requirements and found that the threshold latency for FPS games was about 100 ms, while it was about 500 ms for role-playing games and as much as 1000 ms for real-time strategy games. While some of these other classes of virtual worlds are relatively forgiving, more consistent performance could also significantly improve their users' satisfaction. As audio starts playing a larger role in such virtual worlds, consistent performance can be expected to become even more important.

First-person shooter games are typically implemented using a single server to support client machines for a few tens of users. Client machines accept user input, render the graphics for the virtual world and interact with the servers. The single-server approach is even used for online games with large user populations. These systems typically divide users among distinct copies of the virtual world with a single server supporting the users in each copy.

In systems where multiple servers cooperate to implement a single virtual world, the servers must interact with each other to share state information. While the use of multiple servers enables single sessions to have large numbers of users, it does bring with it significant scaling challenges. One of the primary issues facing the designer of a virtual world that uses multiple servers is how to divide the workload among the servers and keep the load on different servers balanced. The most commonly used approach is to divide the virtual world into regions and assign each region to a server [DE06, RO03]. Each server is responsible for maintaining the state of the users within its region. Since users mostly interact with other users in the same region, this approach reduces the amount of communication required among servers. On the other hand, as users move from region to region in the virtual world, the responsibility for maintaining their state must also move, and since users are free to move anywhere in the virtual world, servers can easily become overloaded if too many users crowd into the same region.

Another way to distribute the load is to make a fixed assignment of users to servers [BH06]. This approach is well-suited to first-person shooters, as it gives the system more control over the per-server load, and if the servers are distributed geographically, it allows users to be assigned to servers that are physically close to them. Since users' perception of system performance is determined primarily by the responsiveness of their own avatars to their input, the assignment of users to nearby servers can significantly improve performance from a user perspective. At the same time, it does increase the amount of interaction required among servers, as users on different servers may be close to one another in the virtual world, requiring their servers to exchange state updates to enable their interaction.

An important consideration in many virtual world applications is the provisioning of environmentally accurate audio. Today, this is of primary importance for virtual worlds oriented towards social interaction, but it can be expected to play a larger role in other types of virtual worlds in the future. High quality audio can enable much more natural interaction among users and can significantly

enhance their experience. However, delivering high quality audio presents additional challenges, as users must be able to receive unique *audio mixes* based on the audio produced by users (or other sources) in their immediate vicinity within the virtual world.

An overlay network supporting virtual worlds should support multiple approaches to managing system state, in order to avoid constraining the higher level application design, and to enable different kinds of virtual worlds to share a common set of network services. At the same time, it's useful to focus on specific usage scenarios, to enable informed choices among design alternatives. Since the assignment of users to servers based on physical proximity places the greatest demands on the overlay network services, we focus our attention on that approach. At the same time, we have taken care to avoid making the network services directly tied to any one approach.

In general, regardless of the higher level application design, each virtual world will be implemented by a set of core components: *clients*, *servers,* and *overlay routers*. Clients are individual user machines responsible for accepting user input and rendering the virtual world on the user's display. Each client interacts with one of a number of servers. The servers' job is to interact with their assigned clients, maintain their clients' state information and to share that information with other servers. Servers may also provide clients with information about the virtual world, although in cases where the virtual world is static, that information may be pre-loaded on the clients. Overlay routers provide network services in support of the clients and servers and these services are our primary focus.

2.2. Overlay Network Services

Since we are interested in supporting virtual worlds that are highly interactive and require consistent performance, it makes sense for the overlay network to support resource provisioning, so that each session has the network resources needed to ensure that its users have a satisfying experience. This means that each session must have an assigned amount of network bandwidth and processing resources on the overlay routers. Its real-time access to these provisioned resources must be guaranteed using traffic isolation mechanisms, such as weighted fair-queueing with per session queues, or something similar. Session resources are assigned based on the number of users, so in the absence of sufficient system resources, new users attempting to join a session in progress can be denied access if necessary, to ensure a high quality user experience for those users in the session.

Since the delivery of state update information is a major part of the overlay network's role, it's important to make the delivery of state updates as efficient as possible. Since many servers may require updates for a particular user, the overlay network should provide an efficient multicast mechanism for distributing updates among interested servers. Since servers' needs for specific information can change frequently, as users move around the virtual world, it is also important to support efficient subscription to multicast groups and to enable servers to subscribe to many different groups at the same time. The precise way that servers use multicast groups may vary among specific high level application designs, but the provision of a flexible, rapidly configurable multicast service can be broadly useful.

It's worth noting that overlay-based multicast, while useful, is not essential. Distributed game systems can be, and have been built, using only unicast packet delivery, so it's worth considering the question of whether multicast provides sufficient benefit to justify its inclusion as a core overlay network service. Multicast is useful primarily in two ways. First, it reduces the number of packets that a server must send. If a typical user is in view of an average of k other users, then a session involving n users will require the delivery of kn state updates during each update interval. Since k is typically fairly small (4-8), the advantage provided by multicast is limited, and since each server must receive an average of k updates per user in any case, the reduction in packet processing load at a server is at most a factor of two. However, in some virtual world environments, there can be individual users whose state updates are required by an unusually large number of others. This can make the peak load on a server substantially larger than the average, and in order to deliver consistent performance, sessions must be provisioned based on the expected peak load. This can significantly reduce the number of clients that a server can support, raising overall system costs. Indeed, some peer-to-peer game systems implement a form of application-layer multicast in order to cope with this peak loading effect [BH08]. Of course this also raises the question of users that must *receive* updates from an unusually large number of other users. Reference [BH08] also shows how to handle such situations by taking advantage of users' inability to focus on more than a few other users at a time. Their system delivers full-rate state updates for only the few "most important" users, while providing reduced update rates for those that are less important. They show that this technique effectively restrains the peak load on servers with only a limited impact on user-perceived performance.

The second way in which multicast is useful is that it reduces network bandwidth. There are two aspects to this, the average bandwidth used and the bandwidth that must be provisioned to ensure consistent performance. We examine this in section 3.5, where we find that for representative configurations, multicast distribution of state updates can reduce the average cost by a factor of two or more and the cost of the required provisioned capacity by a factor of five or more. We note in the next section that network bandwidth accounts for a significant fraction of the cost of these systems, so savings of this magnitude can be worthwhile.

2.3. Cost Factors

When designing any system, it's helpful to have an understanding of how different system resources contribute to the overall cost. This is particularly important when considering how design choices may affect the relative quantities of different types of resources that may be required. For overlay applications, there are three types of resources that are of primary concern: the servers, the overlay routers and the network bandwidth. In this section, we make some rough estimates of the costs of different components in order to get a sense of their relative contributions. We emphasize that these are rough estimates only, and the absolute values should not be taken too seriously. Our purpose in making these estimates is to develop an understanding of the *relative magnitude* of different cost factors, so we that can make more informed design trade-offs.

We start by considering the servers. Experience with single server game systems tells us that in highly interactive games, a single server can be expected to support a few tens of users. Let us assume that a commodity server can support 50 users and that the cost of acquiring and installing the server is about $2,000 and that servers are replaced every 24 months. This leads to a monthly cost of $1.67 per user. As power is a significant cost factor in modern data centers, we also include it in our estimate of the monthly cost of maintaining a server. [KO07] studied the power consumption of servers in the United States in 2007 and found that the average volume server uses about 187 Watts and when the power consumed by auxiliary equipment and cooling is included this number roughly doubles. If we use a more conservative estimate of 400 Watts with an average price of industrial power at about 6.9 cents per kWh [EIA], then we arrive at an electricity cost of approximately $20.15 per month, or about 40 cents per user. Adding this to the hardware cost we get a monthly cost per user of $2.07.

To evaluate the cost of the overlay routers, we assume that they are implemented using comparable commodity server hardware, but with an efficient kernel-resident networking software subsystem such as Click [KO00]. Previously reported results show that IP routers implemented with Click are capable of forwarding several hundred thousand packets per second, even on single-core processors. Recent work has also shown that when these systems are re-engineered to take full advantage of modern multicore servers, packet-forwarding rates in the millions of packets per second can be achieved [EG08]. If overlay routers forward packets at a conservative rate of 200 thousand per second, and the system sends 20 packets per second for each user, and these packets pass through an average of 10 overlay routers, then we need one router for every 1,000 users. This results in a monthly cost contribution of about 10 cents per user (including the cost of power).

	unit cost	monthly cost	users supported	cost per user
server	$2,000	$103	50	$2.07
router	$2,000	$103	1,000	$0.10
bandwidth (1 Mb/s)		$10	5	$2.00

Figure 1: Rough cost estimates

The difference in the cost contribution of these two components is striking. There are two factors at work here. First, the servers have a heavier computational load, since they must perform the physics simulation needed to determine the interactions among objects in the virtual world. In addition, they must exchange packets with clients and other servers. The second factor is significant in that their use of user-space processing in a general-purpose operating system makes it more difficult for them to deliver consistent performance, which in turn means that their average utilization cannot be very high. The overlay routers, on the other hand, need only forward packets and because they have a single function, can operate in the kernel and monopolize the processing resources.

We note that routers can be implemented using Network Processors (NP) systems, in place of conventional processors. While NPs are generally more expensive, they are engineered for packet processing, allowing them to achieve significantly higher performance than conventional processors. This can lead to improved overall cost-performance. However, since it's clear that the server cost plays a much larger role than the router cost in the virtual world application context, we don't consider this alternative in detail.

The third system resource that should be considered is network bandwidth, particularly wide-area network bandwidth. It is more difficult to quantify this with precision, but we note that ISPs such as Cogent offer leased wide-area connections for approximately $10 per month per Mb/s [TELE]. If the system sends 20 packets per second per user, with an average packet length of 250 bytes, we consume an average of 40 Kb/s per user. If each user's packets are sent over an equivalent of five wide area connections, each user consumes 200 Kb/s of wide area bandwidth, resulting in a monthly cost per user of $2.

These cost estimates while crude, do make it clear that the largest contributors to the system cost are the servers and the network bandwidth. This underscores the value of multicast as a core overlay network service, since it reduces the usage of wide area network bandwidth. Making the use of multicast as efficient as possible is also clearly worthwhile, so long as we can do so without conflicting with the objective of providing consistent performance to users.

The results also suggest that there may be opportunities for the overlay network to provide additional services that allow servers to support more users. This opportunity is inherently limited, since the servers' major task of phys-

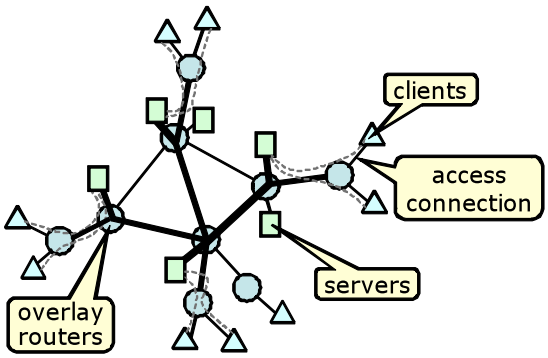

Figure 2: Overlay Components

ics calculations cannot be reduced. However, to the extent that communications overhead and processing of state updates limit servers' ability to support users, there may be some potential to reduce server load. It's also possible that overlay routers could provide services that reduce the peak load on servers, allowing them to operate at higher average utilization levels.

3. FOREST ARCHITECTURE

Based on the considerations discussed above, we have chosen to structure the overlay network around a core network service that uses tree-structured communications channels to support all types of communication. We refer to these channels as *comtrees*. Comtrees are configured for individual virtual world sessions and provide the framework for distributing state updates among servers, as well as for communication between servers and clients. Resources are explicitly allocated to comtrees based on the number of users and session-specific resource requirements. Forest also provides isolation mechanisms to ensure that comtrees are always able to access the resources they have been assigned. Separate comtrees are used for distributing control information not associated with individual sessions, and are provisioned to ensure that the control traffic is never blocked by contention from other traffic sources.

3.1. COMTREES

Comtrees are the central primitive in Forest. While the overlay network's links will typically form a general graph, a comtree uses a subset of the links that forms a tree. Each application session using Forest is assigned its own comtree and all communication for the session takes place within this tree-structured channel (of course, applications may use more than one comtree if appropriate). Comtrees support both unicast and multicast packet forwarding and operate as independent logical networks. Unicast routing information is acquired dynamically as a by-product of packet forwarding, in a way that is similar to the learning mechanisms used by Ethernet LANs. In the absence of routing information needed to forward a packet, a Forest router can forward the packet to all of a comtree's incident links (except of course, for the link on which the packet was received). Packets forwarded in this way are marked with a flag requesting routing information for the addressed destination, which triggers a response containing the required information.

Since all multicast forwarding also occurs over the tree, comtrees follow the shared tree approach to multicast routing where all members of a multicast group use the same shared tree to route multicast traffic. The alternative approach is known as source-based trees, in which each sender to a multicast group constructs its own shortest path tree to all the other members of the multicast group. With respect to multicast, a comtree represents a single shared tree used for all multicast groups within the session. The advantage of this approach is that it is straightforward to support highly dynamic multicast groups as there is no need to select routes for different multicast groups or for different users in a group. Of course, the configuration of a comtree for a session does require the selection of a tree that can support the session, but the configuration (and reconfiguration) of the session's comtree can occur on a much longer time-scale than the configuration of multicast groups within a session, which is driven by the movement of user avatars within the virtual world.

Figure 2 shows an example comtree used to support a session. The heavy-weight links define a tree connecting all the overlay nodes involved in the session. Servers share state updates over the comtree using multicast, while clients communicate to their assigned servers via unicast, as indicated by the dashed connections. More precisely, packets from clients enter the overlay from the public Internet at an overlay access point. The access point extracts a Forest packet from the IP packet it is contained in, and checks the Forest header information. These checks include a verification that the Forest source address is consistent with the source IP address and port number, and that the endpoint with that source address is allowed to send packets on the comtree specified in the packet header. The system can optionally restrict a given client, to a single unicast destination address. This is useful to ensure that clients interact only through their assigned servers. Client connect to the overlay at the nearest available overlay router, in order to minimize the reliance on public Internet connections. Servers may be located anywhere in the overlay infrastructure, although for highly interactive sessions, are preferably located close to their clients' access points.

Comtrees are also used for distributing information that is not associated with individual user sessions. For example, a link-state style routing protocol for distributing information about overlay network resources in Forest can be efficiently implemented on top of a comtree. Here multicast groups can be used to support aggregation of routing information, so that nodes can subscribe to detailed link-state information for nearby overlay nodes, while receiving coarser-grained information for more distant parts of the

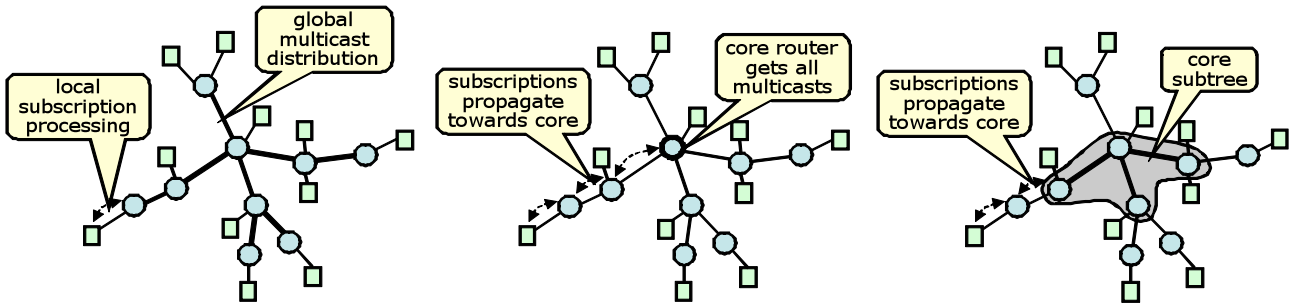

Figure 3: Scalable multicast routing in tree-structured channels. (*a*) global distribution, (*b*) distribution to/from single core router, (*c*) distribution to/from a core subtree

network. Multiple comtrees can be configured to balance traffic and provide protection against link and node failures.

3.2. Naming and Addressing

Users, sessions, and system components such as servers and overlay routers are identified in the system by globally unique, human-readable names. Comtrees are identified by a unique 32 bit numerical identifier that is included in the header of every packet sent on the comtree. Comtree ids are flat global identifiers and imply no semantic information. Endpoints may send packets using only comtree identifiers for which they have been configured, and Forest routers discard packets received from endpoints not configured to use them.

Network endpoints and routers are each assigned a unicast address for use within the comtree. These addresses implement a two level hierarchy to improve the scalability of routing information. Specifically, each unicast address has a "site" part that identifies a geographic location or region and an "endpoint" part that identifies a particular component within the site. A Forest router uses the site part of the address to reach routers in other sites and uses the endpoint part to reach components within its own site. We require that all nodes in a comtree with the same site number form a subtree within the comtree topology. This allows Forest routers to limit the amount unicast routing information they must maintain per comtree. Since addresses are local to a comtree, the number of unicast addresses needed to support a comtree used by a virtual world is determined primarily by the number of clients in that world. In this context, 32 bits provides an ample supply of addresses, while making a simple two level hierarchy sufficient for routing scalability.

Multicast groups require their own addresses. In the next section, we discuss how multicast packets are routed in a scalable way. Here, we simply note that no location information is required for multicast groups, so multicast addresses are simply flat numerical identifiers. This leads to a simple 32 bit address structure in which the high bit is used to distinguish between unicast and multicast addresses. Unicast addresses divide the remaining 31 bits between the site part (15 bits) and the endpoint part (16 bits). Multicast addresses use all 31 of the remaining bits to identify a comtree-wide multicast group.

3.3. Scalable Multicast Routing

Before discussing the specifics of multicast routing, it's useful to consider a specific usage scenario. One way in which servers can use multicast sessions to manage the delivery of state updates is to associate a separate multicast group with each region of the virtual world. A server sends a state update for a given user with the multicast address of the region currently occupied by the user's avatar. Servers can then subscribe to the multicast addresses for regions that are "visible" to their users. As users move, servers continuously update their subscriptions. Regions may have a fixed size or may vary in size to match the structure of the virtual world. The ratio of the number of regions to users can vary, depending on exactly how regions are defined and used, but we note that there is little value in having more regions than users and that there are reasonable designs in which the number of regions is comparable to the number of users. We also note that subscriptions may change rapidly. A server hosting 50 client machines might maintain subscriptions for a few hundred regions, and may add and remove a few tens of subscriptions per second. An overlay router supporting 100 servers could be required to process thousands of subscription requests per second, making it essential that subscription processing be very lightweight.

Since each session communicates over its own comtree, one way to implement multicast is simply to broadcast every multicast packet to every overlay router in the comtree and let the routers deliver packets to their directly attached endpoints based on local subscriptions (see left panel of Figure 3). This has the advantage that each router need only keep track of the subscriptions for its attached endpoints, minimizing the required multicast routing state, minimizing the subscription processing overhead and ensuring rapid response to subscription requests. On the other hand, it does require that multicast packets be distributed to Forest routers whose servers have no interest in them, needlessly consuming network bandwidth in these cases.

An alternate approach is to define a central "core" router in the session tree and configure each router with a "pointer" telling it which of its incident links leads to the core (see center panel of Figure 3). With this approach, all multicast packets are sent to the core router, and subscription requests are also forwarded towards the core router, while adding multicast routing state at each router along the path to the core. If a subscription request finds an overlay node along this path that is already subscribed to the given multicast, then the subscription is not propagated the rest of the way to the core. The use of a core router for routing in a shared multicast tree is not new and was first explored in Core-Based Trees [BA93], although there are some differences in the way that Forest uses the basic idea of a multicast core. In section 3.5 we show that using a core in comtrees largely eliminates the excessive transmission of unwanted multicast packets, while still allowing efficient subscription processing. On the other hand, it can slow down the response to subscription requests and places a larger burden for handling multicast routing state on the core router.

We have chosen a more general approach that can be used to implement either of the above options, as well as various intermediate points. In particular, we allow each comtree to define a "core subtree" consisting of a subset of its overlay routers (see right panel of Figure 3). Each router outside the core has a pointer telling it how to reach the core, and all multicast packets are sent towards the core and distributed to all the routers in the core. Note that this can be done without any multicast-specific routing state. Subscriptions also flow towards the core, as described in the previous paragraph and need never propagate any further than the first core router. We note that a small core provides the most efficient use of bandwidth at the cost of higher subscription processing overhead and slower response to subscription requests.

There are a variety of ways one might select which routers to include in the core. Perhaps the simplest approach is based on a specified maximum "distance" between an endpoint and its nearest core node; the distance metric can be a function of both hop count and link delay. The core can then be made as small as possible, consistent with this constraint, providing a bound on the response time to subscription requests. Alternatively, the core can be adjusted dynamically, based on the subscription volume at a node. We leave the detailed examination of these issues to future work.

3.4. Resource Allocation

Resources are allocated to sessions, which grow and shrink dynamically as users come and go. Some sessions may involve a fairly small number of users and be of modest duration. Others can become very large and last for days, months or even years (e.g. Second Life).

3.4.1. Allocating Server Resources

Servers can typically support a few tens of users, although the actual number will vary based on server capacity and the specific application. Ideally, we would like to have each server support just one session, as this maximizes the opportunity for sharing state among the users on a server and reduces the performance penalties associated with time-sharing a single server among multiple sessions. At the same time, we would like to map users to servers that are physically close to them. These two preferences have the potential to conflict with each other, particularly as users join and leave sessions that are in progress. We don't address the issue in detail here, but we note that the time-sharing penalties can be substantially reduced by implementing real-time scheduling mechanisms in the OS. So long as the total server load is limited, good performance can be achieved in virtual world applications if each virtual world process is guaranteed an opportunity to execute at least once every 20 ms. If the number of virtual world applications running on a single processor is small (say ten or less), this condition can be met, even using conventional operating systems.

3.4.2. Capacity Provisioning of Comtrees

The allocation of network bandwidth to sessions can be broken into two main parts. First, we have the traffic between clients and servers. This traffic is constrained to a specific (and typically short) path within the session's comtree and is predictable and continuous. This makes it straightforward to allocate the appropriate bandwidth as users are added and removed.

The provisioning of multicast bandwidth is somewhat more complicated and depends both on the number of users and the set of nodes that are assigned to the comtree's multicast core. As a basis for this provisioning, we require that each endpoint u specify a sending limit, $\alpha(u)$, and a receiving limit, $\omega(u)$. These limits will be specific to the virtual world application but we note that generally a server will have a sending limit proportional to the number of users it hosts. When determining its receiving limit, a server may need to assume an upper bound on the number of users it will receive state updates from concurrently. In fact, to avoid overloading servers, the application must be designed to limit the rate of arriving state updates to an amount that is consistent with its processing capacity. So, the receive limit arises naturally from the application's need to ensure real-time perfomance.

Given such limits, we can provision all the links in a comtree so that they have the capacity to support any traffic pattern that does not exceed the limits. It is up to the endpoints to ensure these limits are respected, which is reasonable given that virtual world servers are resources in the network designed to cooperate with one another. The problem of provisioning tree-structured communication channels with specified send/receive limits was studied in

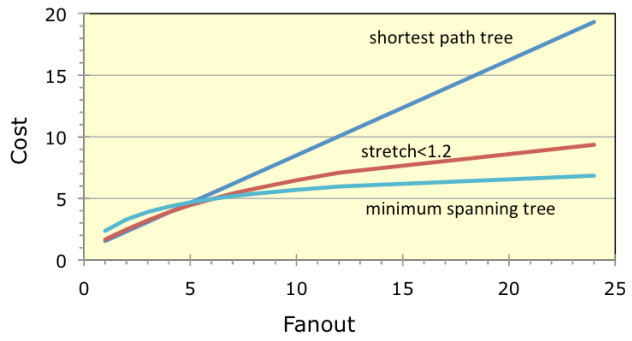

Figure 4: Alternate comtree topologies

another context by Fingerhut in [FI94, FI97]. He showed that one can provision the bandwidth on a link from router x to router y as follows. First, let X be the set of endpoints on x's side of the link and let $\alpha(X)$ be the sum of the send limits for the endpoints in X. Similarly, let Y be the set of endpoints on y's side of the link and let $\omega(Y)$ be the sum of the receive limits for the endpoints in Y. The bandwidth required from x to y is then just the smaller of $\alpha(X)$ and $\omega(Y)$. Moreover, one can compute the required link capacities for all links in the tree, using a single tree traversal requiring $O(n)$ time, for a tree with n nodes. To account for the use of a multicast core that receives copies of all multicast packets, we need to make a small modification to this procedure. Specifically, if there are any core routers on y's side of the link, the required bandwidth is $\alpha(X)$. Otherwise, the required bandwidth is $\min\{\alpha(X), \omega(Y)\}$. If the links are provisioned in this way, then the comtree is guaranteed to have the capacity needed for any traffic pattern that does not exceed the specified send and receive limits. It is worth noting that the addition of a new user often affects only a subset of the links in the comtree. In particular, if the core consists of a single central node, the addition of a new user affects links leading from the server assigned to the user to the core and perhaps a few more beyond the core.

3.5. Selecting a Comtree Topology

As there are many ways that virtual worlds can be distributed, different applications using different approaches may produce vastly different communication patterns. Therefore, configuring a comtree for a session requires selecting a subtree of the overlay network infrastructure that has enough capacity to support arbitrary communication patterns among network endpoints. This is a special case of the constraint-based network design problem also studied in [FI94, FI97, DU99]. It has been shown that in general, this problem is NP-hard, using a reduction from the Steiner tree problem. However, when the solutions are constrained to be trees, we can find optimal or near-optimal solutions in the cases most relevant to comtree configuration [FI94]. In particular, if A is the sum of all the $\alpha()$ values and Z is the sum of all the $\omega()$ values, then for $A=Z$, the optimal solution is a shortest path tree from some "central" vertex in the overlay network to all endpoints that are to be included in the comtree. Such a tree can be constructed by computing a shortest path tree for the entire overlay network and then pruning links not used to reach endpoints required for the comtree. By trying all possible center vertices, we can find the optimal solution in $O(mn + n \log n)$ time, where m is the number of links in the overlay network infrastructure, and n is the number of nodes. If $A<Z$, this shortest path tree is not optimal, but is guaranteed to have a cost no more than $(1+Z/A)/2$ times that of the least-cost tree.

While the prior work provides a solid basis for comtree configuration, it leaves several issues to be addressed. First, while reference [FI94] shows that shortest path trees are within a constant factor of optimal when $A<Z$, it provides no information about how to obtain better trees in this case. This is the case we would expect to find in most distributed virtual environments, as servers that share state using multicast will typically send far less than they receive. We find that in cases where A is much smaller than Z, other trees can substantially out-perform shortest path trees. We illustrate this with results from a simple experiment, shown in Figure 4. For this experiment, we generated random trees over n (=25) points distributed uniformly over a 2x2 square centered at the origin. Trees were constructed, starting from the most central vertex (that is, the one closest to the origin) and provisioned to determine the cost. For each point we assumed that there were n users transmitting state updates to users at *fanout* other (randomly selected) points, where the fanout was varied from 1 to 24 and each user had a send limit of 1. The cost of each provisioned link was taken to be its provisioned capacity times its length. Each data point in the figure shows normalized average results from 50 independently generated random trees. We do not show error bars, but standard deviations were computed and were typically less than 10% of the mean values. Results for three different trees are shown: shortest path trees, minimum spanning trees and an intermediate tree constructed using a variant of Prim's minimum spanning tree algorithm, with a bound on the maximum allowed "stretch" with respect to distances from the tree root; we show the results when the stretch is limited to 1.2 (note that constraining stretch to 1, yields shortest path trees, while allowing it to be unbounded, yields minimum spanning trees). The shortest path tree cost grows linearly with the fanout, and is very close to the analytical bound. The minimum spanning tree provides the best results for large fanout, and the bounded stretch trees perform nearly as well. We conjecture that a hybrid strategy, which mimics the minimum spanning tree algorithm in the early stages, and the shortest path tree algorithm in later stages, will outperform the "pure" strategies considered here.

The earlier work also does not address the use of a core subtree for multicast packets. Core subtrees are useful,

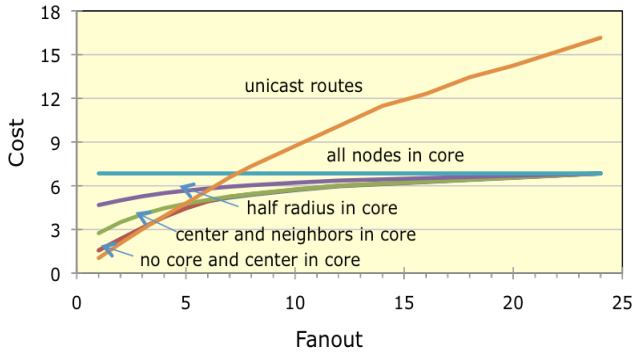

Figure 5: Alternate comtree topologies

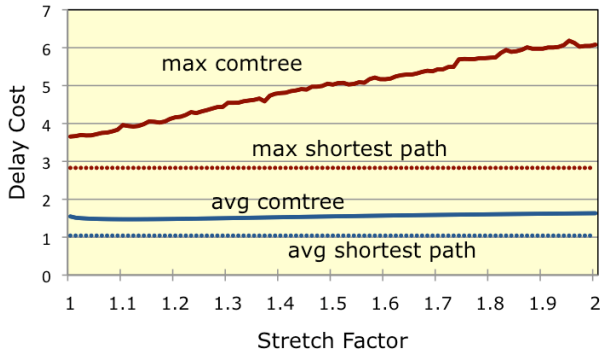

Figure 6: Delay cost in comtrees

because they can significantly reduce the amount of routing state needed to "locate" a multicast group. This can be particularly important for applications that use many small, dynamic, multicast groups, such as distributed virtual environments. On the other hand, the use of a core does impose a network bandwidth cost. We have examined how this cost changes with the size of the core, and compared this to the cost of implementing multicast without a core. We again generated random trees over n points distributed uniformly over a 2x2 square centered at the origin. Trees were constructed using the variant of Prim's algorithm mentioned earlier; for each case, several values of stretch were evaluated and the one that produced the least expensive tree for the given provisioning method was selected. The results appear in figure 5. First, we note that when the core consists of just the "center" node of the comtree, the cost is essentially indistinguishable from the case where no core is used. When the neighbors of the center node are added to the core, there is some increase, but the difference becomes negligible for larger fanouts. Larger cores lead to higher cost, but the cost difference shrinks rapidly as the ratio of receive limits to send limits grows. The curve labeled "unicast routes" shows the cost of routing traffic from senders to receivers using direct paths (that is, the cost was taken to be the Euclidean distance between sender and receiver). This is actually slightly more efficient than multicast when the fanout is 2, but is significantly less efficient for larger fanouts.

The prior work must also be extended to account for capacity limits in the underlying substrate. One way to incorporate capacity limits is to modify the tree construction algorithm to check capacity constraints as each new link is added to the tree; if adding a link causes a constraint to be exceeded (either for the given link or other links already in the tree), the link is marked as *excluded* and the algorithm proceeds to consider alternate choices. In the absence of capacity constraints, this produces trees that are provably optimal or close to optimal. In the presence of capacity constraints, there is no guarantee that this method will produce a solution at all, even when a solution is known to exist. However, it is a natural starting point for algorithmic study of the capacity-constrained case, which we plan to investigate further in future work.

Our strategy for provisioning comtree bandwidth can be overly conservative in systems where there is a strong locality to the communication patterns. This can cause it to allocate more bandwidth than the application requires, needlessly increasing cost. The constraint-based network design framework is general enough to accommodate situations like this. For each endpoint, u, we define a neighborhood N_u and specify a constraint $\alpha(u,N_u)$ on the amount of traffic that can go from u to nodes outside N_u. Constraints of the form $\omega(u,N_u)$ are defined similarly. With these added constraints, the objective for comtree selection is to find a subtree of the overlay network infrastructure that can support any traffic pattern that satisfies both the original send/receive constraints and these additional constraints. We expect that these *neighborhood constraints* will often be associated with clusters of nodes that are geographically close to one another, leading to a natural hierarchy that matches well with tree topologies. In future work, we will study how comtree selection algorithms can be designed to produce high quality solutions for cases like this.

Given that virtual world applications are highly sensitive to network delay, it is worthwhile considering the cost, in terms of delay, of routing traffic through the comtree. While the use of a shared multicast tree allows servers to join and leave many multicast groups very efficiently, source-based multicast routing would have the minimal possible delay between nodes since each node routes traffic over its own shortest-path tree. However, since comtrees are provisioned and isolated from one another, the primary source of network delay is expected to be propagation delay. Thus the difference in delay costs is roughly proportional to the difference in path lengths. The comtree selection algorithm described in this section attempts to constrain path lengths by including a stretch factor that bounds the distance from any node to the root of the tree. To verify that this approach gives an acceptable level of delay, we use the same experiment where we constructed comtrees in a 2x2 grid with 25 nodes. We produced a different set of comtree configurations for each value of

stretch and for each of the comtrees we recorded the path lengths between all pairs of nodes. We also took the cost of using shortest path trees as the Euclidian distance between the nodes. Figure 6 shows that the average delay cost of routing through the comtree in our 2x2 grid is about 1.5 (regardless of stretch) whereas the shortest path between the nodes is approximately 1.04. The maximum distance between any pair of nodes is $2\sqrt{2}$, and the maximum delay in the comtree is fairly close to this when the stretch factor is small. As noted earlier a stretch factor of 1.2 produces low provisioning costs, suggesting that one can limit the maximum delay, while still keeping the provisioning cost low. We note that while routing traffic within the comtree may cause some nodes that are physically close to each other to experience longer delays than they might otherwise, the maximum delay is really the critical consideration. We note that [VI08] provide a variety of strategies for selecting shared multicast trees that minimize delay.

Since users may join and leave a virtual world session over time, this implies that comtrees may need to be dynamically reconfigured to accommodate changes in the set of endpoints. Most often, it will be possible to add an endpoint, through adjustments to the provisioned capacity of a subset of the comtree links. In other cases, comtrees may need to be restructured in order to accommodate new endpoints. In this case, the running application will need to migrate from one comtree to another while minimizing the impact on running applications. We plan to address this issue carefully in future work.

4. APPLICATION TO AN FPS GAME

To obtain a deeper understanding of virtual world applications and how they can be effectively supported using advanced overlay network services, we have adapted an existing distributed implementation of the popular first person shooter game, Quake. We have chosen to focus on FPS games for two reasons: (1) their fast-paced nature means that they have demanding performance characteristics that push the boundaries further than less interactive virtual world applications, and (2) because there are available open-source software implementations that can be adapted to our purposes. In this section we describe some of the specific tradeoffs that have influenced our design, and provide details of a prototype implementation of the key overlay network services.

4.1. Distributed FPS Design

In Section 2.1 we described two approaches to distributing load among servers in a distributed system for virtual worlds. We have chosen to focus on the approach where players are statically assigned to servers that are physically nearby. This choice was made to help insure that servers respond rapidly and consistently to user input. Because this approach leads to higher server-to-server communication, it also represents the more challenging scenario from a networking perspective. We have adapted software developed for the Colyseus system [BH06] as our initial codebase, as Colyseus follows a similar approach to distributing server load, allowing us to use large parts of the Colyseus software without modification.

Before describing our modifications, we present a brief overview of Colyseus. In a typical FPS game, the terrain of the virtual world, or 'map', is generally static for the duration of the game session. Therefore the game state can be expressed as the state of all the mutable objects in the virtual world, e.g. player avatars, missiles, health packs, etc. In the Colyseus architecture, each server hosts a subset of these objects, which are known as the server's *primary* objects. The assignment of objects to servers does not have to be static, but the Colyseus designers note that object migration can be very disruptive, making a static allocation preferable. A Colsyeus server maintains the state and executes the game logic for each of its primary objects. It is also responsible for communicating with the clients whose player avatars it hosts.

Since objects hosted on different servers are part of the same virtual world, a Colyseus server keeps 'replicas' for the objects hosted on other servers that its primary objects may interact with. These replicas are weakly consistent copies of the primary. If a server needs to change the state of a replica, it must send a 'remote update' message to the server hosting the primary to request the change. The server hosting the primary keeps replicas loosely synchronized by sending out state updates whenever the state of the primary changes. Given the fast-paced nature of FPS games, objects tend to change state rapidly causing these state update messages to dominate the traffic among servers.

One issue raised by this approach is the need for an "object discovery" mechanism, that is, a mechanism by which a server can determine which objects, hosted on other servers, it must keep replicas for. Generally the rules of FPS games dictate that objects can only interact with other objects that are in the same visible region of the game world. Additionally, only "dynamic" objects such as player avatars and missiles may interact with other objects. Objects such as health packs and ammunition are more static and their game logic generally does not depend on nearby objects. Therefore, Colyseus determines the "area-of-interest" for its primaries by calculating the areas of the map that are visible to its dynamic objects. Servers learn of the objects they need regular updates for by periodically publishing the locations of their own objects and subscribing to their objects' areas-of-interest. In Colyseus, this publish/subscribe system is implemented using a distributed hash table.

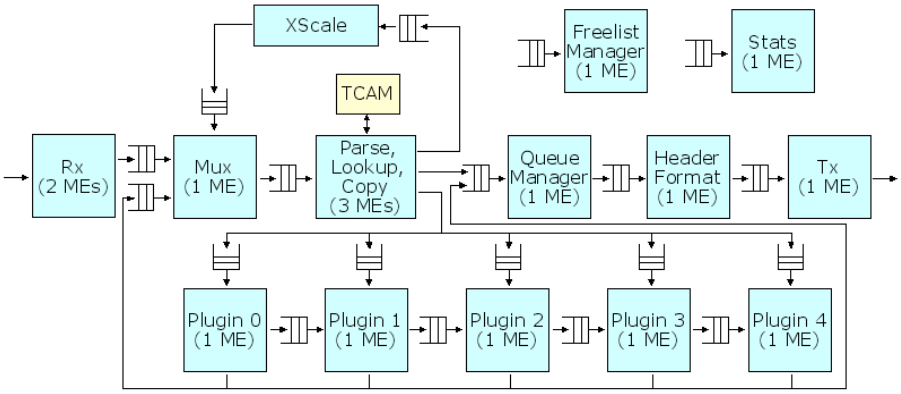

Figure 7: Data plane of the ONL router [Reproduced with permission from WI08]

Because Colyseus was designed to work over the commodity Internet, it relies on the unicast packet delivery service that the Internet provides. Since we are operating within in an overlay environment, we can exploit multicast for more efficient distribution of state updates. Moreover, by associating distinct multicast addresses with regions of the virtual world, we can eliminate the need for Colyseus' DHT-based object discovery mechanism. Servers simply subscribe to the multicasts for the regions of interest to their dynamic objects. It's worth noting that multicasts need not be used in this way. For example, one could assign a multicast address to each dynamic object in the virtual world, allowing servers to subscribe to the multicasts for the objects of interest to them. However, one would need to augment this with an object discovery mechanism (possibly using multicast); associating multicasts to regions allows us to avoid this.

Our approach raises a number of immediate issues, however, which results in several new tradeoffs. First, state updates do not contain the full state of an object, but rather are delta-encoded for bandwidth efficiency. This means a server will need to acquire the full state of the object before it can maintain a replica. Secondly, state updates are only sent when the state of an object changes and some more static objects, such as health packs, may not change state for long periods of time. Finally, Colyseus ensures replicas remain consistent by explicitly acknowledging every state update received. This last issue is problematical in a multicast context, as it requires a scalable reliable multicast service, which is considerably more complex than a simple best-effort multicast. We have chosen to address these issues by transmitting the full state of each object periodically, allowing a server to instantiate a replica by simply waiting for the full update to arrive. Periodic full updates also enable recovery from lost updates.

Since transmitting the full state of the object is relatively expensive, the period between full state updates represents a tradeoff. Retransmitting the full state more frequently consumes more bandwidth but allows servers to acquire replicas or recover from lost packets more quickly. We have chosen to send full state updates for each object once per second. In terms of object discovery latency, this ensures that servers will have to wait an average of 0.5 seconds after subscribing to a multicast address before receiving a copy of the object. This is comparable to the latency seen by Colyseus using its object discovery mechanism. Since the underlying network service supports bandwidth reservation, congestion-induced packet loss can be made very rare, minimizing the impact of delayed recovery from packet loss.

Another design issue raised is how the game world should be partitioned into regions. Ideally the map's terrain would be used to define a partitioning that minimizes visible boundaries among regions. This would reduce the number of regions needed to express an object's area of interest, thus reducing the overhead caused by multicast subscriptions. In this paper we have taken the much simpler approach of defining regions using a 2D rectangular grid. While this is less than ideal, it is worth evaluating since if such a simple approach proves satisfactory then there is not much point in pursuing more sophisticated methods. With this approach, the granularity of the grid, i.e. the number of regions used, represents a second tradeoff in the application design. Finer-grained partitioning means that servers can more accurately express the interest of their primary objects thus reducing the number of 'uninteresting' state updates received (due to objects that are not visible to a given object, but whose regions are partially visible). On the other hand, subscribing to more regions increases the multicast control overhead and has the potential to make area-of-interest calculation for objects more expensive.

4.2. Experimental Prototype

Our ultimate objective is to implement Forest within a high performance overlay hosting environment, such as the one being developed for NSF's GENI initiative [GE06]. We are also exploring the possibility of deployment within commercial cloud computing infrastructures [EC2]. As a first step, we are using Washington University's Open Network Lab [ONL] as a prototyping environment. ONL has recently been expanded to include network processor (NP) based routers with a flexible plugin subsystem for experimental extensions. This makes it a natural testbed for GENI

applications, since it is likely that GENI will support overlays using similar NP-based components.

The Open Network Lab is an Internet-accessible network testbed that is built around extensible gigabit routers that can be "wired" to each other to form arbitrary network topologies. It also provides a large number of PCs that can be connected to the routers and can host applications that communicate over the configured experimental network. The routers can be modified through the insertion of user-supplied plugins and we use this facility to prototype the core features of the comtree such as the multicast distribution of state updates and the associated dynamic subscription mechanism.

In this initial prototype, we have deferred the network control needed to create and reconfigure comtrees. This allows us to focus on the aspects of the system design that most directly affect the performance of the data path. In our experiments, we have also chosen to configure the multicast core to include all overlay routers, in order to minimize subscription processing overheads. This allows us to simplify the prototype implementation since subscriptions need not be propagated beyond the "first-hop" router.

Before describing the implementation of our plugin we provide a brief overview of ONL's NP-based routers (NPRs) but we refer the reader to [WI08] for a full discussion. The NPRs are constructed using Radisys Network Processor blades that host a pair of Intel IXP 2800 NPs. Each NP subsystem contains three banks of SDRAM, four banks of QDR SRAM, and they share a Ternary Content Addressable Memory (TCAM). The blade also has ten 1-gigabit data interfaces, which are divided between the two NPs, allowing them each to be used as a five port routers.

The IXP 2800 has one xScale management processor and 16 multi-threaded *Micro-Engines* (ME), which do the bulk of the packet processing. The micro-engines support efficient pipeline operations, but can be used to support arbitrary software structures. The data path of the router is shown in figure 7. As packets come in, they are stored in DRAM and a packet reference, which includes the metadata needed for a route lookup, is passed through the pipeline for processing. The TCAM is used primarily for route lookups. The user can also install filters in the TCAM to direct packets to specific queues, outgoing ports, or to plugins. A filter can be used to match a specific protocol (TCP, UDP, or ICMP), a specific source or destination port associated with the protocol, or any prefix of the incoming packet's source or destination IP address. The SRAM is used primarily for lookup tables, linked list queues, and as 'scratch' memory for user plugins.

In this ONL routers, five MEs have been set-aside as 'plugin' micro-engines that run user code. Each plugin ME can be loaded with code separately so that a user can have up to five different plugins. In addition, there are five ring buffers, implemented in SRAM, that feed packets into the plugins. As mentioned above, filters can be installed to direct traffic to the plugins by delivering packets to any one of the ring buffers. Once the plugin is done processing a packet, it may direct the packet to a specific output queue or it may defer the routing decision to the router and let the router match the packet to the TCAM a second time.

With this background, we briefly describe the implementation of our Forest plugin. The plugin implements the essential data path functions of a Forest router, including the forwarding of unicast and multicast packets, and the processing of multicast subscription packets. Multiple copies of the plugin can be installed to work in tandem, reading packets from the same ring buffer. The current prototype uses a relatively simplistic approach to managing multicast subscription state. Specifically, it uses the memory available for multicast state as a two-dimensional matrix indexed by the comtree id and the multicast destination address. Each entry in this matrix is a bit vector specifying the outputs that matching packets should be forwarded to. The range of comtree ids and multicast destination addresses is constrained to allow the entire matrix to fit in the available memory space. A more general approach would be to use a hash table, but we have taken the simpler approach in this initial evaluation.

While the NPR provides efficient support for IP multicast, we do not use these mechanisms, as we are prototyping an overlay environment in which multicast is provided as an overlay service. Since the plugin must direct each copy of a multicast packet to distinct destination addresses, it must copy the payloads explicitly, rather than simply copying a packet reference. As a result, our plugin replicates the packet payload, assigns each copy the appropriate destination address, and has the copies reclassified by the router to direct them to the correct output queue. This means that the "Parse, Lookup, and Copy" (PLC) block, which performs the classification step, must process each outgoing copy of each state update packet, in addition to the arriving packet.

5. EVALUATION

In this section, we evaluate the performance of the prototype Forest implementation and an FPS game application that uses its services to support large game sessions.

5.1. Router Microbenchmarks

We start by considering the raw packet processing performance of the Forest routers. As discussed above, the ONL implementation of the Forest router uses up to five micro-engines to implement the processing required for forwarding state update packets and for subscription processing. We start with results for a single micro-engine forwarding state updates. We considered two cases. In the first case we measured throughput for multicast traffic with a fanout of 1. Here traffic from five input ports is merged and forwarded out a single port. In this case the router is able to

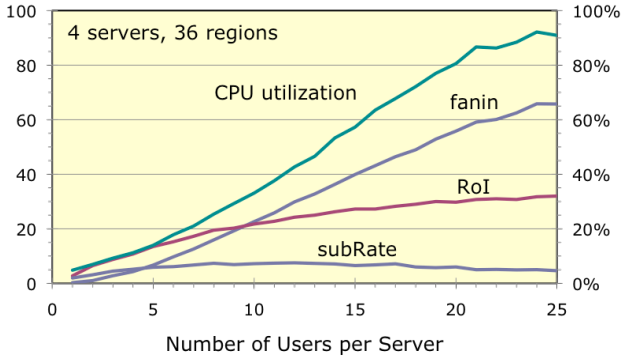

Figure 8: Impact of Number of Users on FPS Performance

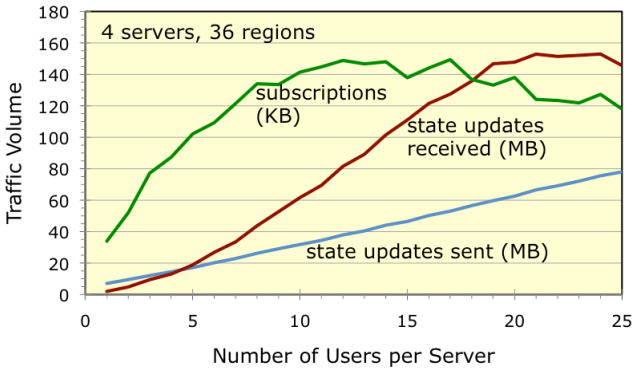

Figure 9: Traffic Volumes

forward packets at a maximum rate of 1.95 million per second for a packet payload size of 150 bytes. Our Forest protocol header adds another 32 bytes and the UDP-IP and Ethernet headers add roughly another 66 bytes. The resulting output data rate is about 3.9 Gb/s or 77% of the output link capacity. As the payload size increases, the packet processing rate drops, while the data rate increases, with the output links saturating for payload lengths above 250 bytes. For the second case we adjusted the fanout to 4 (the maximum for our 5 port router) by having the traffic received at each port be forwarded out all four of the other ports. We found that the output rate in this case was essentially the same as for the fanout 1 case, suggesting that the extra work required to copy multicast packets is balanced by the reduced input rate required to produce a given output rate. We also note that one of the factors limiting the router's performance is the requirement that the outgoing multicast packets have to be reclassified because of the change in destination IP address. For this reason, when we go from using a single micro-engine to using all five, the maximum packet-processing rate increases by less than 20%. This suggests that the router could likely accommodate the more complex packet processing that would be required in a realistic implementation that uses a hash table lookup in place of the simple direct lookup used here.

We evaluated the router's ability to process subscription messages by subjecting it to a load that consisted entirely of subscription packets, arriving on all input ports. We varied the number of subscription changes in each packet from 1 to 350 and found that the peak packet processing rate went from 3.15 million packets per second down to 70 thousand, while the resulting subscription processing rate went from 3.15 million up to 24.5 million. In the next section, we find that the subscription rate per user in actual game sessions is generally less than five per second, so a router can process the subscription requests for more than 50 thousand players, while using less than 10% of the its subscription processing capacity. However, it must be noted that this is for a simplified subscription processing mechanism, which neither propagates subscriptions, nor forwards acknowledgments to servers.

5.2. Performance of FPS Game Sessions

In this section, we study the performance of FPS game sessions that use Forest services to distribute state updates. We are interested in understanding how various application metrics are affected by the number of users in a session, the number of users per server and the number of regions used to partition the virtual world.

We start by considering a configuration using a single router with four servers, and study how various metrics change as we increase the number of players per server from 1 to 25. For this experiment, we divided the game world into 36 uniform regions. The results are summarized in Figure 8. We show four performance metrics (1) the server *CPU utilization* (as reported by the operating system), (2) the *fanin* per server (that is, the number of users for which a server receives state updates), (3) the number of *regions of interest* to a server (that is, the number of regions it is subscribed to) and (4) the *subscription rate* per server (the number of subscription changes per second). The values on the chart are averages over a five minute game session using simulated users (bots).

Let's focus first on fanin. We note that when there is just one user per server, users spend much of their time in isolated parts of the game world and have no interaction. This leads to a fanin less than one. As the number of users increases, the fanin grows for two reasons: first, because the fanin per user increases as there are more users to interact with and second, because the number of users per server increases. Consequently, the fanin grows super-linearly, for small numbers of users. However, as the fanin starts to approach the total number of users in the session, the growth rate becomes linear and then sub-linear, with a maximum of about 65, when the total number of players in the session is 100. The regions of interest metric also grows with the number of players per server, growing more rapidly than the fanin for small numbers of users and then more slowly, as the number of subscribed regions starts approaching the total of 36. The subscription rate reaches its maximum value of about five changes per second when there are 10-12 users per server. Note that for this number

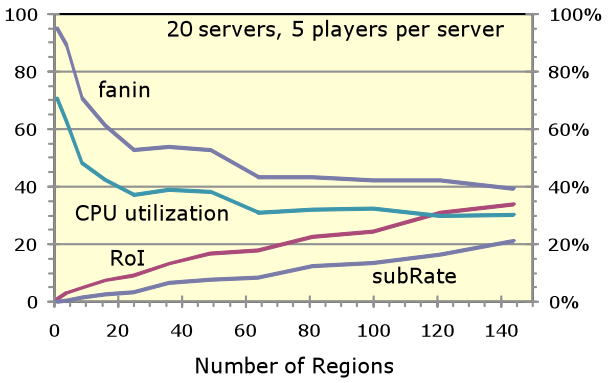

Figure 10: Impact of number of regions on FPS performance

of users, the typical server is subscribed to more than half the regions of the game world, so changes to the set of subscribed regions stabilizes and starts to decline at this point.

Finally, let's consider CPU utilization. We note that for a single user per server, the CPU utilization is 5% and that the addition of three more users per server increases the CPU utilization to 11%, suggesting that there is an initial overhead of about 3% and then a cost of about 2% per player for doing the game physics calculations. As the number of players grows further, the processing of state update messages starts to have a significant impact, causing a more rapid increase. We observe that if the only thing the CPU had to do was perform the game physics calculations, it could handle 25 users with just 50% of the CPU capacity. For the larger sessions, the CPU utilization is about double what we would expect for the game physics alone, providing a measure of the cost of distributing the game over multiple servers. We also note that improvements in handling of state updates can be expected to improve the server performance by no more than a factor of two.

We also recorded maximum values for the various metrics. When the number users is small, the maximum fanin can be four times the average, but as the number of users grows, the ratio of the max to the average drops to less than 1.2. The subscription rate is the most variable metric with a maximum that can be 4 to 7 times larger than the average.

Figure 9 shows how the traffic volumes vary with the number of players per server. We show results for the multicast state updates (in MB) and for the subscription traffic (in KB). The numbers reported are the total traffic volume over all servers for a five minute game session. First, note that the state update traffic dominates by a factor of 100 or more. For the state update traffic, the sending volume increases linearly with the number of servers, while the received volume tapers off as the number of users gets large. The received traffic is typically twice as large as the sent traffic. The state update packets have a typical payload size of about 290 byte, while the subscription packets have a typical payload size of about 12 bytes.

We now turn to a configuration with 100 users distributed across 20 servers linked by eight routers. In this case, we focus on how the various performance metrics change as the number of regions in the game is increased from 1 to 144, as shown in figure 10. Starting again with fanin, we note that for a single region, each server receives state updates from all 95 users on the other servers, and as the number of regions grows, the fanin drops sharply before leveling off at about 40. The CPU utilization drops along with the fanin, leveling off at a utilization of about 30% when the number of regions is large. The regions of interest metric increases roughly linearly with the number of regions and at 144 regions, we note that the average server is subscribed to roughly 25% of the regions of the game world. The subscription rate grows with the number of regions, topping out at about 20 subscription changes per second. We note again that the subscription rate is the most variable metric and for 144 regions, the maximum subscription rate is about 90 per second.

The computers used to implement the servers in these experiments are 2 GHz AMD Opterons running Linux version 2.6.21. Each is equipped with 512 MB of RAM and has a 1 Gb/s Ethernet interface. We used Quake's built-in bots to simulate players using the default difficulty/intelligence setting. We also used a fairly large custom map that consists largely of corridors and small rooms.

We should note that while our overlay network could have accommodated substantially large game sessions, we found that limitations in the Quake 3 and Colyseus code base made it difficult to scale to sessions with much more than 100 users. While we made some efforts to address these limitations, we concluded that the required effort was not justified, given that our principal interest is in the scaling characteristics of the overlay network services, rather than this particular FPS game.

5.3. Scalability of Overlay Forwarding

Next, we discuss how some of our basic mechanisms scale, as the number of users in a session grows. We start by noting that our choice of a comtree, which is a tree-structured communication channel, leads to some intrinsic limitations, as the router at the root of the comtree must have the capacity to forward state update packets from all senders. If Forest routers are implemented using conventional servers, we can expect packet forwarding rates of a few hundred thousand packets per second for servers with a single processor core and rates above one million for servers with eight or more cores. Given a state update frequency of 20 packets per second, a root router should be able to forward the state update packets for between 10 and 100 thousand users. Larger-scale sessions are possible, either using multiple comtrees for a single session or using multiple servers connected by high performance switches to implement high capacity routers serving a single session. We don't explore these options in detail here, instead limit-

ing ourselves to session sizes up to about 100 thousand users.

The scalability of packet forwarding is limited by the required routing state, in addition to the forwarding capacity of the routers. Unicast and multicast routes can be stored in a single hash table, where the hash is a function of the comtree id and destination address. The table can be stored in inexpensive DRAM, allowing millions of routes to be supported at a reasonable cost. The use of two level unicast addresses and tree-structured comtrees reduces the number of unicast routes that are needed for each comtree. Essentially, each router requires a route for each "foreign site" and for each endpoint in the "local site". For large sessions, we expect the number of required unicast routes to grow as the square root of the number of endpoints, ensuring that the amount of unicast routing state remains manageable. Routes are obtained dynamically by learning addresses. Most unicast routes will be associated with client/server traffic and routes will be established on the path joining a client to its server the first time they communicate with each other. Hence, the cost of acquiring the route is relatively small, compared to the normal communication that must take place between clients and servers.

The amount of multicast routing state required by a session depends on the size of the core. The worst-case is a single node core, since this requires the core node to maintain a multicast route for every multicast address. If we associate a separate multicast address with every user in the session (the option that uses the most multicast addresses), the number of multicast routes the core node must support is bounded by the number of users whose packets it forwards. Given that a router can economically support millions of routes, we expect the data forwarding requirements to limit the router long before the memory required for multicast routes becomes constraining. By a similar argument, the processing of subscription packets is unlikely to limit scalability, since the volume of subscription traffic is generally far smaller than the volume of data traffic.

6. RELATED WORK

This paper focuses primarily on overlay network services tailored to support distributed virtual environments. Our discussion, however, touches on a number of other aspects related to the support of distributed virtual environments. In particular, we have already described several methods for load balancing in a distributed virtual environment and in our game system we applied a region-based multicast technique to manage the interests of servers.

The use of a region-based multicast scheme has been explored previously. Macedonia et al. [MA95] separates objects in DIS simulations into separated spatial, temporal, and categorical groups and associates these groups with IP multicast addresses. For their spatial partitioning, they used a similar region-based multicast approach except they used hexagonal regions and calculated the regions in the object's area-of-interest by defining a simple fixed-size radius. Kantawala et. al. [KA96] described a similar region-based approach for DIS using a square grid of regions and ATM multipoint connections.

A number of more sophisticated region-based interest management techniques have been investigated such as [AB98, FE02, HU04]. These approaches all offer more complex methods for region partitioning that are intended to make interest management more precise and minimize the number of multicast addresses used. In our context, minimizing the use of multicast addresses is a lesser concern, as we have per session address spaces and lightweight mechanisms for joining and leaving multicast groups.

Network services designed to support distributed virtual environments have been explored in the active networking context. The SANDS system [ZA02] uses active networks to support interest management in the network infrastructure. In their approach, which they call "active interest filtering", the application uses a signaling protocol to install interest filters in the active routers that describes the content the application is interested in (e.g. regions in the game world). Packet payloads are then tagged with content descriptors that the router uses to match against the subscriptions of end hosts. Rajappan et al [RA03] augmented this work to provide reliable multicast for distributed simulations that are loss-sensitive. ATOM [GR00] describes an approach to using active networking to provide a scalable totally ordered multicast service and they cite games and virtual environments as a motivation. Their network architecture is structured around using a sequencer node to provide a totally ordered multicast service. While ordered multicast may be a useful service to applications that require a high degree of consistency, we did not include this as a core network service in Forest.

We also note that the use of multiple core routers in shared multicast trees has been proposed before but typically to address issues relating to routing in IP which is a slightly different context. For instance, Distributed Core Multicast [BL99], assumes a two-level network hierarchy where there is a backbone network that connects multiple area networks together. A "distributed core router" (DCR) is assigned for each multicast group within a given area. The DCR acts as the area's local core for a multicast group and it communicate with DCRs in other areas to determine which areas have members in the multicast group. This approach reduces the amount of multicast routing state needed in backbone routers but it is also intended to avoid the triangular routing problem and for limiting traffic across expensive backbone links. While our approach using a simple core subtree also reduces routing state for the core routers, it does not involve signaling between the core routers as our goal is to provide fast and efficient subscriptions to many groups by limiting the amount of subscription processing in the tree.

The notion of overlay hosting services and networks that are engineered to provide a consistent level of performance has only recently received significant attention, most notably in the context of NSF's GENI initiative [AN05, GE06]. The VINI system [BA06] has extended PlanetLab, enabling users to reserve a specified fraction of nodes and network bandwidth in a distributed overlay environment. The SPP platform described in [TU07] seeks to support higher performance provisioned overlays through a scalable system architecture that incorporates multiple servers and network processors. Amazon's EC2 service [EC2] makes some of the capabilities developed in PlanetLab and these more recent systems available in a commercial setting.

7. CLOSING REMARKS

We have presented the design of Forest, a performance-engineered network architecture to support distributed virtual environments that require consistently high performance. The network is based on tree-structured communication channels called comtrees that support both unicast and multicast packet delivery. The system is designed to support large-scale use of highly dynamic multicast groups for efficient distribution of state updates. To demonstrate the feasibility of the design and to assess its scalability we implemented a partial prototype of our system using NP-based routers and evaluated the performance of a distributed first-person shooter game which we modified to use the provided overlay network services. Our results indicate that there is no reason why systems based on this architecture could not support sessions with tens of thousands of users even in demanding virtual environments such as first-person shooters.

While our initial results are encouraging, we have deferred much of the network control and we intend to evaluate some of the control issues in future work. In particular, we need to develop a number of additional control protocols such as the routing protocol used to distribute information about the available capacity of links and overlay routers. A signaling protocol also needs to be defined to allow endpoints to create and modify comtrees as well as an access protocol to allow registered users to connect new endpoints to the Forest overlay.

There is also a rich set of open problems related to the configuration and reconfiguration of comtrees. In section 3.5 we briefly discussed how the provisioning mechanisms could be designed to accommodate capacity constraints, and how they could use knowledge of traffic locality to reduce the amount of bandwidth that must be provisioned. Perhaps most importantly, we need to address the issue of reconfiguring comtrees as endpoints are added or removed and as the demands of the virtual world session changes over time. We would also like to evaluate the benefits of centralized vs. distributed comtree configuration as well as investigate the use of comtrees to distribute control information and provide fault tolerance.

We are also considering several possible extensions to the core services that Forest provides. These might include end-to-end support for reliable multicast, to make it easier to implement applications with strong consistency requirements. Additionally we could offer high quality synchronization and synchronized packet delivery by having timing information piggy-backed on all packets exchanged between neighboring routers. This can be useful to ensure event ordering in distributed applications. Ideally we would also like to evaluate a more complete deployment of Forest in GENI and perhaps within Amazon's EC2 computing cloud.

Finally we note that the work presented here represents part of a broader research agenda investigating the potential of developing network architectures on shared infrastructures to support demanding applications. This work focuses on supporting virtual environments, but the use of comtrees provisioned to support individual users may have broader applications. In particular we envision the use of Forest to support audio and or video conferencing perhaps in conjunction with virtual environments.

REFERENCES

[AB98] Abrams, H., K. Watson, M. Zyda. "Three-tiered interest management for large-scale virtual environments," In *Proc. Proceedings of the ACM symposium on Virtual reality software and technology* (VRST), 1998.

[BA93] Ballardie, T., P. Francis, J. Crowcroft. "Core Based Trees," In *Proc. of SIGCOMM*, 1993.

[BH06] Bharambe, A., J. Pang, S. Seshan. "Colyseus: A Distributed Architecture for Online Multiplayer Games," In *Proc. Symposium on Networked Systems Design and Implementation* (NSDI), 3/06.

[BH08] Bharambe, A., J. Douceur, J. Lorch, T. Moscibroda, J. Pang, S. Seshan, X. Zhuang. " Donnybrook: Enabling Large-Scale, High-Speed, Peer-to-Peer Games," in *Proceedings of ACM SIGCOMM*, 2008.

[BL99] Blazevic, L., and J. Le Boudec. " Distributed Core Multicast (DCM): a multicast routing protocol for many groups with few receivers," ACM SIGCOMM Computer Communications Review, vol. 29, no. 5, 10/99.

[CH03] Chun, B., D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. "PlanetLab: An Overlay Testbed for Broad-Coverage Services," *ACM Computer Communications Review*, vol. 33, no. 3, 7/03.

[CL06] Claypool, M., and K. Claypool. "Latency and Player Actions in Online Games," *Communications of the ACM*, vol. 49, no. 11, 11/06.

[DE06] Deen, G., M. Hammer, J. Bethencourt, I. Eiron, J. Thomas and J. H. Kaufman. "Running Quake II on a Grid," *IBM Systems Journal*, 2006.

[DI02] Dilley, J., B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. "Globally Distributed Content Delivery," *IEEE Internet Computing*, September/October 2002.

[DU99] Duffield, N., P. Goyal, and A. Greenberg. "A flexible model for resource management in virtual private networks," in *ACM SIGCOMM,* 1999.

[EC2] Amazon Elastic Computing Cloud. `http://aws.amazon.com/ec2/`, 2009.

[EIA] Energy Information Administration. Average Retail Price of Electricity to Ultimate Customers by End-Use Sector. `http://www.eia.doe.gov/cneaf/electricity/epm/table5_6_a.html`

[EG08] Egi, Norbert, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy. "Towards High Performance Virtual Routers on Commodity Hardware," *Proceedings of ACM CoNEXT*, 10/08.

[FI94] Fingerhut, J. A. "Approximation Algorithms for Configuring Nonblocking Communication Networks," Washington University doctoral dissertation, 5/1994. Available at `www.arl.wustl.edu/~jst/`

[FI97] Fingerhut, J. A., S. Suri, and J. Turner. "Designing Least-Cost Nonblocking Broadband Networks," Journal of Algorithms 1997, pp. 287-309.

[GR00] Graves, R., and I. Wakeman. "ATOM – Active Totally Ordered Multicast," In *Proceedings of the Second International Working Conference on Active Networks* (IWAN), 2000.

[FR04] Freedman, M., E. Freudenthal and D. Mazières. "Democratizing Content Publication with Coral," In *Proc. 1st USENIX/ACM Symposium on Networked Systems Design and Implementation*, 3/04.

[GE06] Global Environment for Network Innovations. `http://www.geni.net`, 2009.

[FE02] Fiedler, S.,M. Wallner, and M. Weber. "A communication architecture for massive multiplayer games," In *Proc. of the 1st workshop on Network and system support for games* (NetGames), 2002.

[HU04] Hu, S. and Guan-Ming Liao. "Scalable peer-to-peer networked virtual environment," In *Proc. of 3rd ACM SIGCOMM workshop on Network and system support for games* (NetGames), 2004.

[KA96] Kantawala, Anshul, Gur Parulkar, John DeHart and Ted Marz. "Supporting DIS Applications Using ATM Multipoint Connection Caching," *Proc. of Infocom*, 1996.

[KO00] Kohler, E., R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. "The Click Modular Router," *Proc. of IEEE, Special Issue on Evolution of Internet Technologies*, 9/04.

[KO04] Kontothanassis, L. R. Sitaraman, J. Wein, D. Hong, R. Kleinberg, B. Mancuso, D. Shaw and D. Stodolsky. "A Transport Layer for Live Streaming in a Content Delivery Network," *Proc. of the IEEE, Special Issue on Evolution of Internet Technologies*, 9/04.

[KO07] Koomey, J. G. "Estimating Total Power Consumption by Servers in the U.S. and the World," in `http://enterprise.amd.com/Downloads/svrpwrusecompletefinal.pdf`, 2007.

[MA95] Macedonia, M., M. Zyda, D. Pratt, D. Brutzman, P. Barham. "Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments," in *VRAIS*, 1995.

[ONL] Open Network Lab. `www.onl.wustl.edu`, 2008.

[PE02] Peterson, L., T. Anderson, D. Culler and T. Roscoe. "A Blueprint for Introducing Disruptive Technology into the Internet", *Proc. of ACM HotNets-I Workshop*, 10/2002.

[RA03] Rajappan, G. and M. Dalal. "Reliable Multicast with Active Filtering for Distributed Simulations," *Military Communications Conference Proceedings (MilCom),* 2003.

[RA05] Radisys Corporation. "Promentum™ ATCA-7010 Data Sheet," product brief, available at www. radisys.com/files /ATCA-7010_07-1283-01_0505_datasheet.pdf.

[RH05] Rhea, S., B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica and H. Yu. "OpenDHT: A Public DHT Service and Its Uses," *Proceedings of ACM SIGCOMM*, 9/2005.

[RO03] Rosedale, P., C. Ondrejka, "Player-Created Online Worlds with Grid Computing and Streaming," *Gamasutra,* 9/03.

[ST02] Stoica, I., D. Adkins, S. Zhuang, S. Shenker, S. Surana, "Internet Indirection Infrastructure," *Proc. of ACM SIGCOMM*, 8/02.

[TELE] "Cogent throws down pricing gauntlet," www. `telephony onine.com/mag/telecom_cogent_throws_down/`, 2008.

[TU07] Turner, J., P. Crowley, J. DeHart, A. Freestone, B. Heller, F. Kuhns, S. Kumar, J. Lockwood, J. Lu, M. Wilson, C. Wiseman and D. Zar. "Supercharging PlanetLab – a High Performance, Multi-Application, Overlay Network Platform," *Proc. of SIGCOMM*, 2007.

[VI08] Vik, K., P. Halvorsen, and C. Griwodz. "Multicast Tree Diameter For Dynamic Distributed Interactive Applications," *Proc. of INFOCOM*, 2008.

[WH02] White, B., J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler,C. Barb, and A. Joglekar. "An Integrated Experimental Environment for Distributed Systems and Networks", *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, 12/2002.

[WI08] Wiseman, C., J. Turner, M. Becchi, P. Crowley, J. DeHart, M. Haitjema, S. James, F. Kuhns, J. Lu, J. Parwatikar, R. Patney, M. Wilson, K. Wong, D. Zar, "A Remotely Accesible Network Processor-Based Router for Network Experimentation," *Proc. of ANCS*, 2008

[ZA02] Zabele, S. M. Dorsch, Z. Ge, P. Ji, M. Keaton, J. Kurose, J. Shapiro, D. Towsley. "SANDS: Specialized Active Networking for Distributed Simulation", *Proc. of DARPA Active Networks Conference and Exposition*, 2002