On demand platform for online games

A. Shaikh
S. Sahu
M.-C. Rosu
M. Shea
D. Saha

A shared infrastructure, based on emerging on demand computing models, that supports multiple games offers an attractive option for large-scale multiplayer online game providers who want to avoid the risk of investing in dedicated resources. In this paper, we describe a prototype implementation of a service platform for online games. The platform design follows the on demand computing paradigm. It offers integration using open standards and off-the-shelf software and embraces virtualization and simplification to enable sharing resources across games. We describe our experience with identifying appropriate performance metrics for provisioning game servers and with implementing reusable platform components that provide useful functionality for a variety of games.

INTRODUCTION

The traditional approach taken by most publishers and providers of large-scale multiplayer online games is to install a dedicated infrastructure for each game. This approach has many drawbacks. It involves high risk and investment with little knowledge of how successful a new game will be. For example, an examination of subscriber populations of massively multiplayer online role-playing games (MMORPGs)[1] shows that they all follow a similar life cycle (***Figure 1***), but predicting at launch how different titles will perform or how long their subscriber populations will continue growing remains challenging—player populations can experience sharp increases or drops in a period of just a few weeks.

Game publishers and developers face several "pain points" related to this problem:

- *Sharing existing infrastructure across game titles*—Repurposing servers, changing software stacks, and reconfiguring the network to accommodate a new game or function is often a cumbersome, manual process.
- *Scaling the infrastructure in response to player demand*—Adding and removing servers, support functions, or other resources is not automated.
- *Managing a large, heterogeneous game server infrastructure*—Server and network management is typically well outside the core competency of game providers.

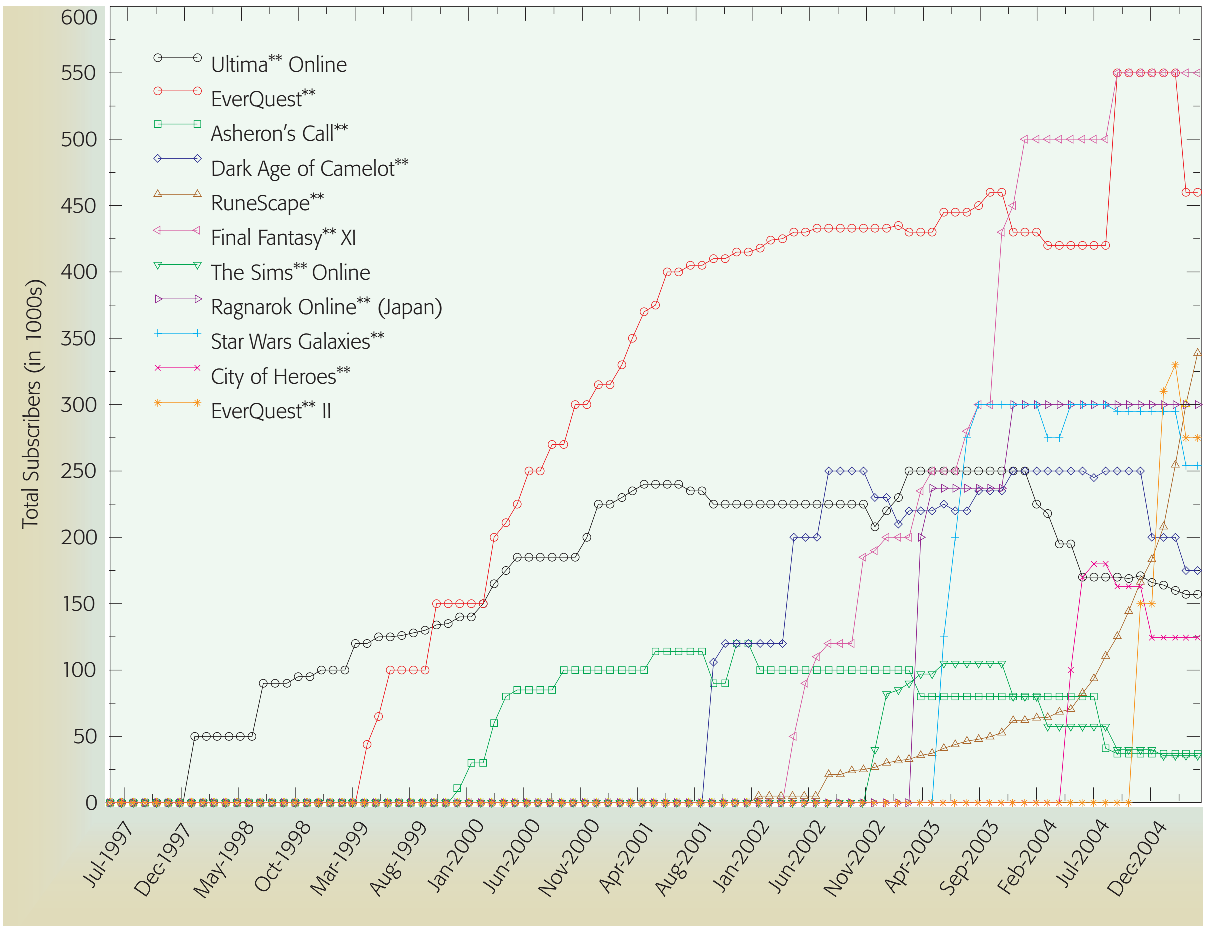


Figure 1
Player subscriptions for popular MMPORPG games (100,000 to 600,000 players)[1]

The on demand business model proposed by IBM[2] addresses similar problems with business applications, where the issue of infrastructure cost strongly motivates new models for utility computing offerings. These models provide the flexibility to scale an application or service in response to user demand by rapidly adding or removing resources (e.g., servers, storage, databases, network bandwidth, etc.) from a pool that may be shared among multiple applications or customers. With an on demand infrastructure, online game providers could enjoy similar benefits by reducing initial investment, scaling rapidly according to demand, and adding new services. For example, an on demand infrastructure based on open standard grid technology[3] was proposed for hosting online games.[4]

In this paper, we describe our work to realize some of the major components of an on demand service platform for games. Our work is based on the premise that an on demand computing architecture can benefit game applications with some modifications and key additional game-specific services. We present a design and implementation that we believe serves the need of a number of classes of online games. Hence, this paper intends to provide a proof of concept that can be used as a starting point for designing and deploying an on demand gaming infrastructure.

We demonstrate the feasibility and operation of the platform by provisioning multiple instances of id Software Quake II**, a popular action game. Though Quake II falls in the category of server-based first-person shooter (FPS) games, the platform is applicable to a variety of game genres (e.g., distributed and single-server FPS, cluster-based massively multiplayer games, Web-based games, and game support services, such as lobbies, database servers, etc.). For example, traditional FPS games are played by a group of players on a single server, but if the game becomes very popular, additional (disconnected) copies of the game "map" can be deployed by provisioning additional servers on demand. Newer game architectures distribute the game map across multiple servers so that resources can be added to support a larger number of players in a seamless world.[5,6] The "shard" (or realm) model used in most MMPORGs similarly can benefit from this approach by adding or removing servers in a cluster managing a single shard automatically as the shard population grows or shrinks.

GAMES SERVICE PLATFORM ARCHITECTURE

In this section we discuss the design of the service platform and follow with an overview of the current prototype architecture.

Design objectives

The service platform design follows several basic principles. First, the platform and associated services should be minimally intrusive to the game applications and, at the same time, still provide value to game providers by relieving them from managing the system infrastructure. A platform must be general enough to support many types of games (so that it can be shared), yet still be able to be tailored when necessary by an individual game publisher. A second possibility that follows from this is a modular platform architecture which allows game publishers to take advantage of functions that address their needs and forego others that may not be as relevant. Finally, to ensure the flexibility and extensibility of the platform, open standards and open-source tools should be used wherever possible, consistent with the principles of the On Demand Operating Environment (ODOE).[7]

The objectives just described are somewhat idealized, and our current prototype does not meet all of them. Nonetheless, our implementation represents a first step toward realizing an ODOE for games.

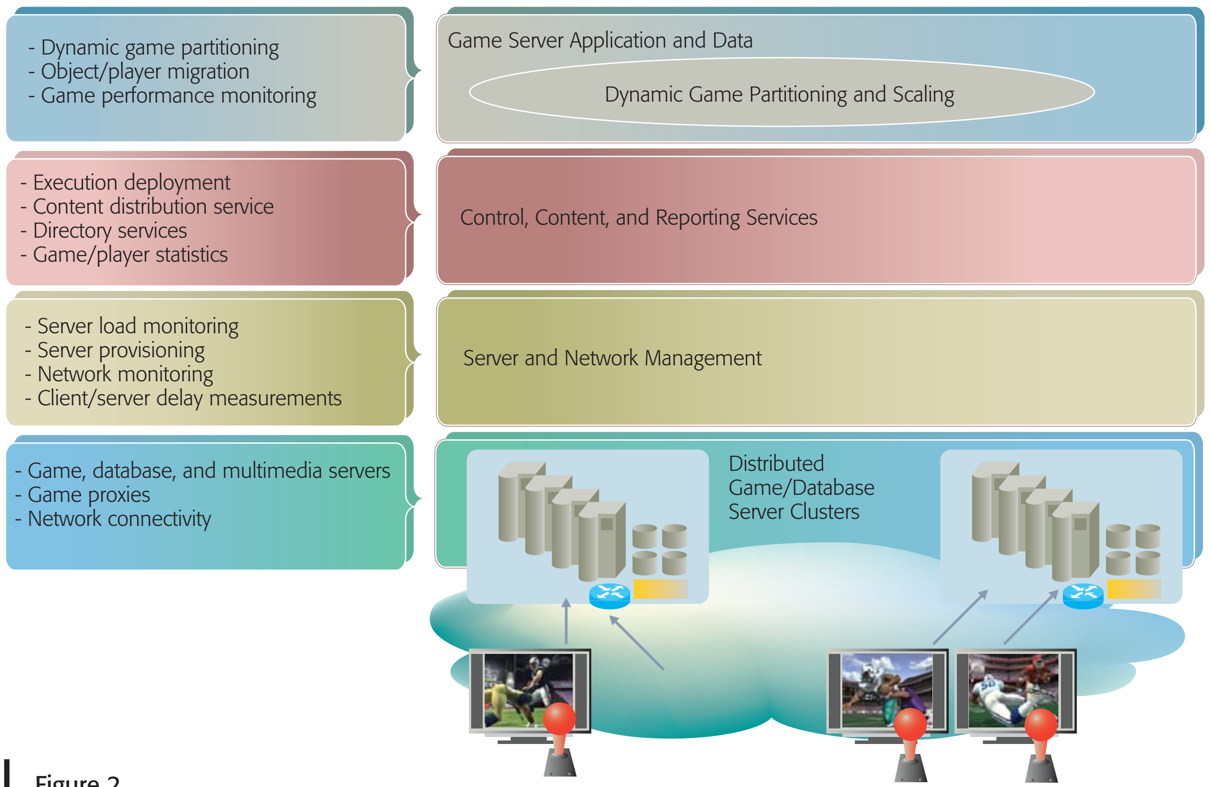
Logical platform architecture

Figure 2 shows the logical relationships of each of the architecture components. Conceptually, the platform may be thought of as a layered architecture, with upper layers comprising *application-level* services and lower layers acting as *system-level* services. At the bottom is the hardware and networking infrastructure, consisting of shared clusters of game servers, database servers, proxies, content servers, and wide area network (WAN) connectivity. We show two sets of server clusters to emphasize that the platform is not limited to residing in a single data center. As the number and distribution of players and games grows, the platform may be deployed in multiple hosting locations.

Above the infrastructure layer are the main system-level services, which consist of non-game-specific functions, such as server and network monitoring and server provisioning. These functions are generally useful for any game and for many auxiliary game functions. Server provisioning will be customized to install specific application code, but the basic provisioning operation—for example, on demand installation of a software stack on a target server—is common to most game applications. The first two layers (i.e., distributed server clusters and server and network management) are also applicable to other networked applications deployed using ODOE concepts. This similarity stems from our conscious decision to design the system so that it leverages components and services that may be used already to manage business applications in an on demand fashion.

Control, content, and reporting are more sophisticated functions characterized by higher complexity and functionality or requiring more substantial modifications to work with each game application. For example, the executable deployment function must interpret the requirements and policies as specified by the game provider and transform these into provisioning operations and orchestration policies for the provisioning manager (PM). Similarly, the game and player statistics reporting function requires interfacing to the game to extract such information.

Atop this logical functional stack is the game application itself. At this level, the primary platform services can dynamically partition and distribute the game over multiple servers. A number of distributed game server architectures and middleware solutions are available that provide this service for games instrumented to use them.[5,6,8] Hence, the partitioning and scaling services are shown embedded in the game application. In addition, an important function at this level is the game-specific performance monitoring that is collected directly for use by the



- Dynamic game partitioning
- Object/player migration
- Game performance monitoring
Game Server Application and Data
Dynamic Game Partitioning and Scaling
- Execution deployment
- Content distribution service
- Directory services
- Game/player statistics
Control, Content, and Reporting Services
- Server load monitoring
- Server provisioning
- Network monitoring
- Client/server delay measurements
Server and Network Management
- Game, database, and multimedia servers
- Game proxies
- Network connectivity
Distributed Game/Database Server Clusters

Figure 2
Logical platform architecture

PM or by the distributed game middleware to make its resource management decisions. In the first version of our prototype implementation, we demonstrate the platform components by using a server-based game that is not instrumented for distribution across multiple servers. However, we are working to integrate our ODOE for games with the IBM OptimalGrid approach reported by G. Deen et al.[5] to demonstrate this feature.

Note that the two top layers set the game service platform apart from a general utility-computing platform for other applications. These layers represent the key set of services and components we have identified that are needed to adapt the on demand computing model to online games.

Prototype architecture

Figure 3 shows the platform architecture of our prototype, including the users of the platform—players, game publishers, and system administrators—who access the platform over the WAN (as depicted by the dashed lines). The game servers reside in a shared resource pool that can be used across different game applications from multiple publishers. In addition to operating the games themselves, the servers may perform additional related functions—such as login management, lobby services, or matchmaking—for a particular game. The PM manages and automatically provisions the game servers. Its main function is to collect performance and availability metrics from the server and network infrastructure (as shown in the figure) and respond to changing resource requirements by adding or removing server and network resources. The PM implements a set of data and performance models that describe the information available from each game along with its implication for game performance. This implies that, in general, the PM must be somewhat tailored to each game, because these metrics and models may be unique to a particular game application. The PM also collects generic system statistics about the server platform, such as CPU utilization, memory usage, and bandwidth consumption.

In addition to game server monitoring and provisioning, the platform also provides a number of other services for users. These functions are implemented by a separate group of auxiliary

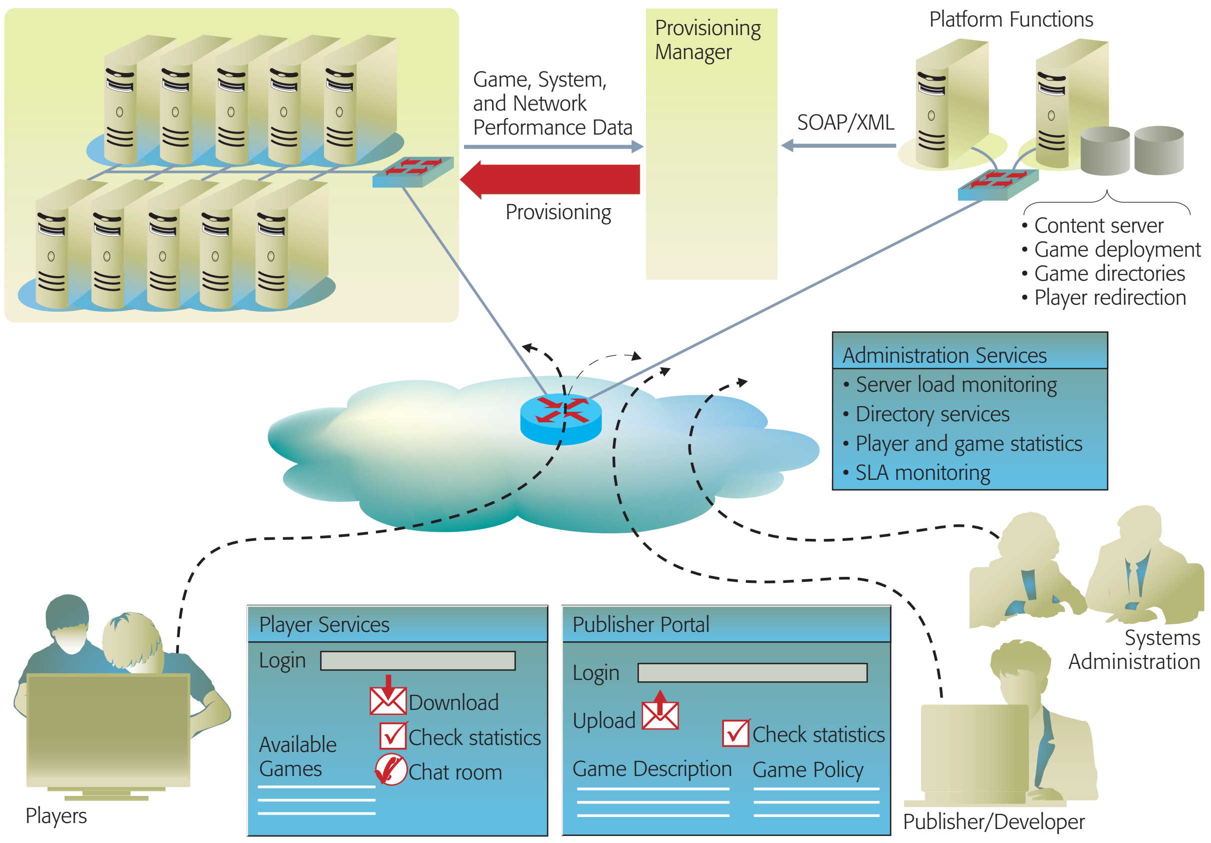


Figure 3
Prototype platform architecture

platform servers that provide an interface to the platform, apart from actual game play. For example, a publisher wishing to deploy a new game sends a request to these auxiliary servers, which process it and contact the PM to provision servers as necessary. Platform servers can also provide directories of games and players, redirection of players to appropriate game servers, and access to metadata for the content distribution service. Details of these services and their current implementation are described later in the section "Auxiliary platform services."

Note that game players continue to access game servers much as before, directly by means of the game client code located, for example, on their game consoles or PCs. Other player services on the platform are accessed by using the Web-based player portal. Some minor modifications to game client software will likely be required to make full use of the platform. If the publisher wishes to take advantage of the content distribution service, for example, the game client software must be able to access the service to obtain patches and new game content. Also, because the servers running a particular game may change dynamically as servers are added and removed, the players must access the redirection service upon initial connection to the game.

Game publishers and developers have their own portal through which they interact with the game platform. The functionality of this portal is implemented on the platform servers. For example, requests to deploy a new game with specific policies (e.g., minimum server requirements) and queries for information about running games are sent by the portal to the platform servers. Similarly, system administrators contact the platform servers to get information about platform conditions, such as overall server and network utilization, number of available servers in the shared pool, and per-game statistics. Requests from each of the portals may also trigger commands to the PM server to query or

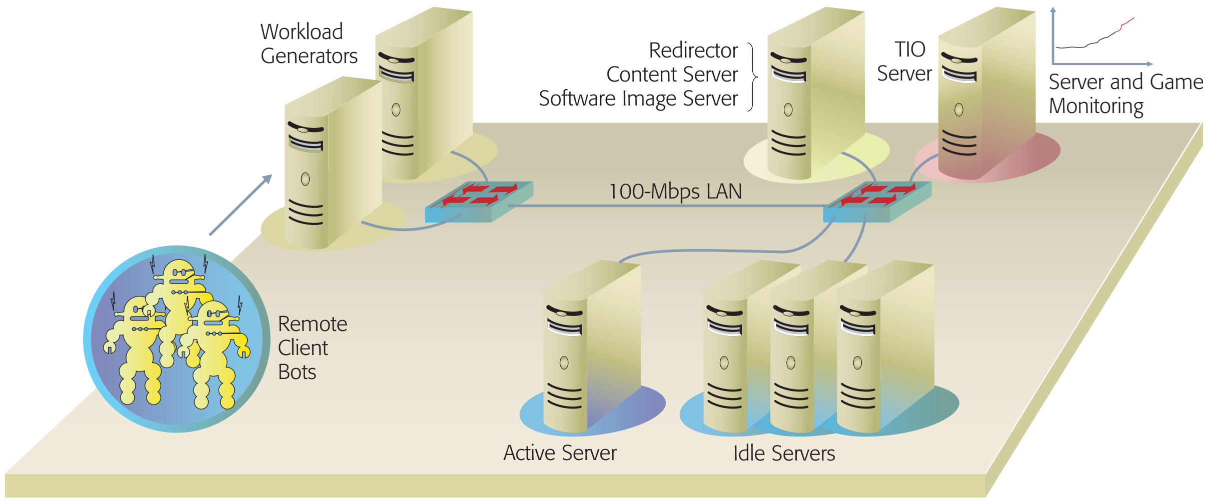


Figure 4
Prototype testbed configuration

provision game servers. As Figure 3 shows, the PM has a SOAP/XML (Simple Object Access Protocol/Extensible Markup Language) interface (in addition to the native command line interface) to handle these commands.

PLATFORM IMPLEMENTATION

Together, the provisioning manager, server resource pool, platform server, and workload generators constitute the major components of our prototype implementation. For our experiments with different provisioning metrics, we used a simplified test bed with two game servers (*Figure 4*). One server in the resource pool is marked active while the other server is assigned to the shared free pool of idle servers. The platform actually supports a large number of servers grouped into pools of similar machines. To evaluate the platform without having to recruit a large number of live players, we implemented remote bots (described later) that generate synthetic player traffic from a small number of workload machines. The prototype uses a single auxiliary platform server to handle several of the functions described in the previous section and to serve as a software repository containing, for example, game server software and operating system images. The machines in our prototype implementation are dual-processor 3.0-GHz Intel Xeon**-based servers with 2 GB of random access memory (RAM). The game servers are 2.4-GHz Intel Pentium 4** PCs with 512 MB of RAM. All of the servers in our test bed run Linux** 2.4 and are interconnected over a 100-Mbps switched Ethernet local area network (LAN).

Provisioning and data collection

Our provisioning manager is based on the IBM Tivoli* Intelligent Orchestrator (TIO), an off-the-shelf software product that automatically deploys and configures servers, software applications, and network devices in a data center or enterprise environment.[9] TIO is designed to manage the performance of multitiered Web-based business applications by deciding how resources should be allocated to different clusters in an on demand fashion. Actual provisioning tasks, such as machine configuration or software installation, are performed using workflows that execute low-level operations in a consistent manner.

The primary function of TIO in our platform is to collect performance and availability metrics from game servers and decide how to adjust server allocations accordingly. Game performance metrics and system utilization information are collected continuously from all of the servers in the shared pool. These metrics are then evaluated against a performance model that determines the current resource requirements for each game. TIO has some built-in performance models, but also allows development of plug-ins (called *objective analyzers*) that can be fully customized to compute resource requirements with a variety of metrics and algorithms. The per-application resource requirements are conveyed to the global resource manager (GRM), which serves as an arbitrator among requests from different applications (i.e., games and related applications). If resource allocations need to be adjusted, the GRM activates the deployment engine that does the actual work of provisioning (or deprovisioning) servers and other resources for different applications.

In our prototype, for example, the TIO server periodically collects the CPU utilization on the active game server using Simple Network Management Protocol (SNMP) queries. If the utilization crosses a threshold, TIO installs the appropriate game server software on a new idle server and adds it to the active set. TIO then executes a custom workflow that notifies the redirection server that a new game server is available for joining players. Similarly, when players leave a game server and it becomes idle, TIO removes it from the active set and returns it to the idle pool after all players have disconnected. The redirection server is an instance of a game server with a slight modification to the communication protocol to allow clients to first issue a server assignment query and receive a redirect response to connect to the appropriate server, rather than connecting directly.

Although we have developed some custom workflows (e.g., to install game server software and update the redirection server) and configuration (e.g., to increase the metric sampling rate), we are largely using TIO "out of the box" (that is, without modification, accepting the defaults). Much of our current effort is aimed at adapting it to use different performance metrics and more flexible models to decide when to adjust resources (i.e., more sophisticated than a fixed threshold on CPU utilization). We present a summary of our examination of the suitability of different metrics in the section "Automatic provisioning metrics."

Game metrics and workload generation

We demonstrate our gaming platform by using the open-source Quake II multiplayer action game with a few minor modifications to the server and client. For example, on the server, we removed the fixed limit on the number of players so that the relatively powerful game servers can be loaded with a large enough number of players to cause performance degradation. We also instrumented the server to export performance metrics beyond those available from the standard Quake II server interface when queried by public tools like QStat.[10] One of these metrics is *slack time*—the time remaining in the server fixed 100-ms state processing cycle. During each cycle, the game server computes the full state of the game based on player and object updates received from clients and then transmits a relevant view of the new state to each player. Thus, slack time indicates how close the server is to exceeding its time budget, after which the game state is updated more slowly (as cycles are "skipped"), noticeably degrading game play at client terminals. Another metric we introduced is the total network traffic transmitted by the server, which reflects the volume of player and object updates that must be sent to players. We also implemented a fine-grained CPU load metric from within the server software that is updated every five computation cycles (i.e., every 500 ms). We refer to this as the *system CPU load*. Finally, we also evaluate the TIO fixed CPU utilization metric, which is collected according to a defined polling interval, along with a smoothing process similar to a weighted moving average.

On the client side, it is important to recreate actual client traffic in order to demonstrate the service platform. Hence, while many server-side bots are available that emulate multiple players on the Quake II server itself, we require a client-side game traffic generator. Unfortunately, we were unable to find a suitable open implementation of a Quake II client-side bot. As a result, we made some simple modifications to the Quake II client itself to act as a synthetic workload generator. Our bot operates externally to the client software, sending it native movement and game play commands (e.g., fire weapon and turn left) via `stdin`. We also modified the client to sleep periodically to reduce the load on the workload generator machines. This allowed us to instantiate more bots while still emulating player behavior. Note that the goal of our bots was not to accurately re-create real player movements and actions; rather, they were meant to create load on the server by using plausible player actions. Real players might impose an equivalent load at a different rate or population level than the bots.

Auxiliary platform services

As mentioned previously, the service platform includes a collection of reusable auxiliary services that can be accessed through either the player or publisher portals. Publisher services help game providers manage their game titles, control the way software is deployed onto the platform, and initiate distribution of game patches or new content. Player services include common tasks such as authenticating to the game platform (thus providing a hook for services like accounting and billing), querying the presence of other players, and tracking the availability of new games. Our implementation focuses on two of the key publisher services, as described next.

Game deployment service

Based on a data center model (DCM),[9] TIO represents the logical and physical assets under its management. These objects can be enumerated in an XML document and stored in an IBM DB2* database during operation. The model includes information technology objects (e.g., servers, network switches, and software applications) and logical entities (e.g., customers and application clusters). In our implementation, we represent a new game publisher as a customer with an associated application cluster (i.e., game servers running the game application). The cluster has an associated software stack that includes the software necessary to run the game server, including the operating system, support libraries, and game software. For games consisting of components running on separate servers (e.g., shared database, game physics computation, etc.), the multitiered application support of TIO can be leveraged. Each component (e.g., database, game, authentication) is represented as a separate cluster belonging to the same overall application, and each cluster has an associated software stack. Note that the DCM is quite general in its treatment of applications and therefore can be used to represent nearly any type of game server that consists of a group of software packages.

■ A platform must be general enough to support many types of games (so that it can be shared), yet still be able to be tailored when necessary by an individual game publisher. ■

Deploying a game on our platform thus requires creation in the DCM of a new application object together with the corresponding clusters and software stacks. In the current implementation, we have defined a single game publisher operating two game applications with their own individual (but similar) software stacks. We have developed a Web-based publisher portal that allows game publishers to upload the game software and an installation script that is packaged by the portal application into a suitable software stack object for TIO. The deployment service also accepts some simple policy information, such as system requirements, and the minimum and maximum number of servers that should be used for the game. This allows a publisher to control, for example, how much to spend on infrastructure for each game. Once the publisher or developer submits this information, an XML specification file is created that describes the new DCM objects. The XML file is read by TIO, which then makes the corresponding incremental additions to the DCM. The new game software stack is also transferred to the image server, from which it can be installed on game servers belonging to the active pool. In this way, publishers or developers do not need to know any underlying details of the service platform implementation; they interact only with a simple portal from which they can deploy their games.

Game content distribution

A major cost factor in operating online games is the significant bandwidth cost associated with game traffic and downloads of patches and new game content. Downloads generate significant load on the publisher servers and access network due to the files being large, the potentially high frequency of patches, and, most important, the flash-crowd nature of the downloads as soon as new content is released. Soon after new content or a patch for a popular game is made available, a large number of players attempt to download it within a very short time interval. In one example, the Steam distributed-content delivery network from Value Software, Inc.—used for in-game authentication and distributing software patches for a number of games[11]—reported that 70–80 percent of the patch content is downloaded in the first two days after a patch is released.[12] Providing enough download servers and bandwidth is prohibitively expensive and failure to do so results in customer dissatisfaction and potential revenue loss.

In light of these issues, the content distribution service uses a peer-to-peer architecture to deliver content to users quickly while conserving bandwidth at the publisher's content servers. Our implementation is based on the BitTorrent** peer-to-peer distribution system.[13] We have integrated the Quake II client with a modified BitTorrent client. The integrated game client is able to initiate downloads through the peer-to-peer distribution service. Some game publishers have recently started employing similar approaches to mitigate their bandwidth costs.[14]

Our modifications to BitTorrent focus on improving its availability and making it network-aware. Our version of the content distribution service provides multiple BitTorrent trackers, thereby eliminating the single point of failure in the standard system. In addition, we enhanced the tracker to provide a list of a client's nearby peer nodes, which are more likely to offer better download performance (as opposed to the current random list). The key challenge is to determine in a scalable manner which peers are nearest. In our initial prototype, we propose to map peers based on their network prefixes and estimate network distances using Internet coordinate systems.

AUTOMATIC PROVISIONING METRICS

In this section, we briefly discuss automatic provisioning of game servers with a focus on evaluating different choices of performance metrics used to trigger resource allocations. Specifically, we consider three key issues: which candidate metric best reflects the server load, whether provisioning should be based on raw or on smoothed measurements, and whether different player arrival processes affect the provisioning decision differently.

In our initial implementation, we use two thresholds to trigger game server provisioning: one for deciding when resources should be added and the other for deciding when excess resources should be released. The two thresholds overlap to avoid frequent reallocations during transient fluctuation in game performance or server utilization. We adopted the TIO default threshold settings and used our experiments to suggest parameter changes that enable better performance for games.

Candidate metrics

We considered several system-level and game-specific performance metrics to better understand the variation of resource requirements as a function of player population dynamics. We identified three candidates: raw CPU utilization of game servers, TIO-computed CPU utilization, and Quake II slack time.

Raw CPU utilization is collected every 500 ms from within the game application. The TIO metric is a smoothed average of CPU utilization collected roughly every 30 seconds using SNMP. The smoothing is based on a weighted moving average that acts as a low-pass filter. Slack time is specific to the Quake II engine, though it may apply to other game engines and the games built using them.

Figure 5A plots the first three metrics as a function of time during the experiment. Each metric is normalized by the maximum value so that a metric value of 1.0 represents the maximum, rather than 100 percent (e.g., in Figure 5A, 1.0 on the graph represents a maximum system CPU utilization of 88 percent, while in *Figure 5B* 1.0 represents 100-percent CPU load). Note that this allows relative comparison between different metric ranges on the same graph, but does not show the absolute values of the metrics. The workload is generated by using bots to simulate remote players; bots connect to the game server according to a Poisson process with a mean interarrival time (duration between the arrival of two consecutive bots) of $1/\lambda = 1$ second. The number of players connected to the server as the experiment progresses is shown at the bottom of the graph using a separate y-axis. After the number of players is sufficient to saturate the server (approximately 200 seconds into the experiment), they are removed until the load decreases to roughly 50–60 percent CPU utilization. Then players are added to increase load again during the 300–400 second time period, after which they begin departing.

We observe that the raw CPU utilization metric tracks the workload on the game server quite accurately as the workload changes. The slack time metric offers similarly accurate tracking. However, for this rapid interarrival rate, we observe a lag with the smoothed TIO CPU utilization metric. For example, most of the second increase in the workload is missed by the TIO metric, although the load reaches roughly 90 percent of the maximum.

This simple experiment shows that using a smoothed performance metric based on past measurement

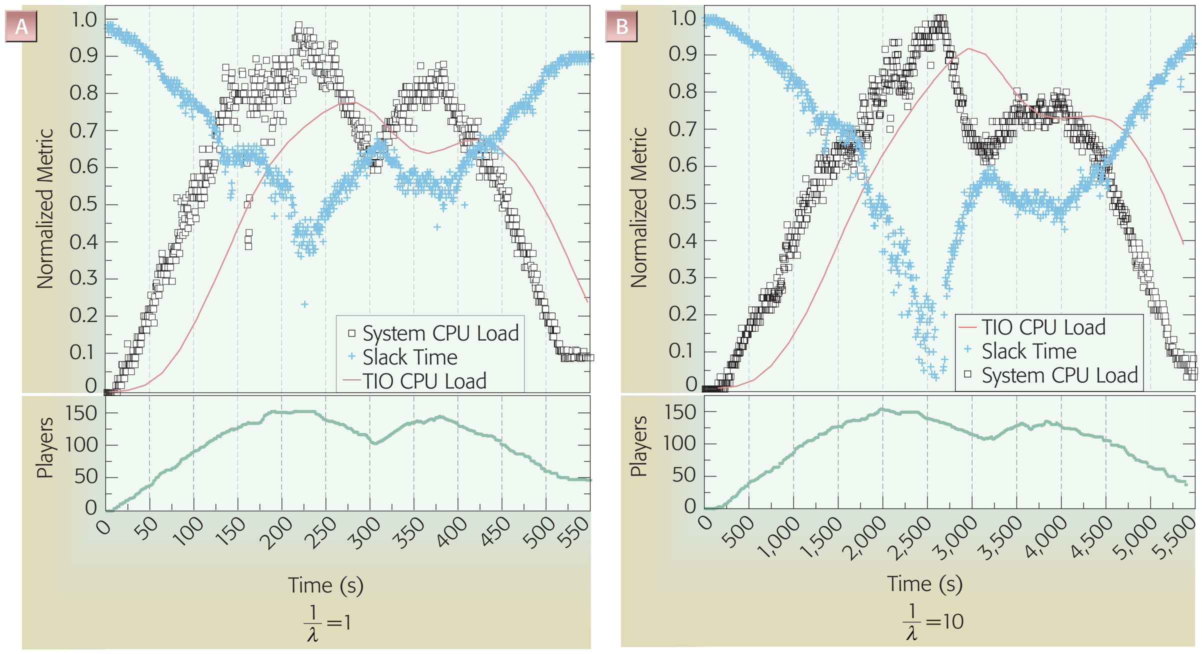


Figure 5
CPU utilization and slack time for different mean interarrival times

samples is not a good estimator of the real workload, particularly when game sessions arrive at a high rate over a short interval. Most important, the smoothed metric can lead to inefficient resource utilization. For example, suppose the desired allocation threshold is set at a CPU utilization of 0.7. Using the TIO-computed metric then requires that the threshold be set at approximately 0.3 to achieve the same allocation and ensure a similar game experience. This would likely result in overprovisioning (almost twice the number of servers), even after the actual load has decreased on the game servers.

Impact of workload

With regard to timescale, we repeated the above experiment for mean player interarrival times of 10 and 30 seconds, again according to a Poisson process. Figure 5B, which plots the same metrics as Figure 5A but for 10-second interarrivals, shows that with a more slowly varying workload, the smoothed CPU utilization metric is better able to follow the instantaneous CPU utilization metric. Here, the smoothed metric may be preferred as it is less likely to provision too quickly in response to abrupt but small changes in the workload.

Next, we examined whether the statistical arrival process has an impact on the performance of different metrics. Specifically, we compared the instantaneous and smoothed CPU utilization metrics for Poisson and deterministic game session arrival processes. Our experiments showed that with a deterministic arrival process, the smoothed metric tracks the actual utilization extremely well, even with a very small interarrival time between sessions. The Poisson arrivals drive the server to peak utilization earlier than the deterministic arrivals, likely due to occasional bursts of arrivals inherent in the stochastic process.

In general, the nature of session arrivals does have a clear impact on which metrics are suitable for game server provisioning. Additional results and a more detailed discussion can be found in A. Shaikh et al.[15]

DISCUSSION

In this section, we discuss some of the additional issues that arise in implementing a shared service platform for games.

Multiplexing games with business applications

The player population in an online game exhibits a fairly predictable daily usage pattern, with the peak time in the evening and on weekends. Therefore, sharing a server resource pool across games alone offers little opportunity for statistical multiplexing because the peak and idle periods for all of the games are expected to occur at roughly the same time.

If we expand the set of applications using the shared pool to include business applications, however, the opportunities to multiplex become clear. For example, we compared the load over a one-week period of the popular multiplayer action game Half-Life** (collected from GameSpy**) and the normalized aggregate request volume for a large consumer credit card Web site. We found that these applications have similar usage peaks during the week, but a considerably different profile on Friday evening and Saturday, which could be exploited for sharing. Other business applications (e.g., enterprise applications that are not consumer-facing) may provide further opportunities for multiplexing. Our adoption of off-the-shelf provisioning software designed for enterprises and data centers was strongly motivated by the potential to integrate our gaming service platform with utility offerings for business applications.

The ODOE for games can provide additional efficiencies, even when managing only gaming applications. For example, the ability to manage the server infrastructure and software deployment from a single point and automatically add or remove servers based on resource demands simplifies management tasks considerably.

Designing on demand games

We demonstrated our current implementation by managing server-based games like FPSes. The applicability of the platform to such games is clear, as well as for other gaming functions that scale up simply by adding additional independent server resources without requiring coordination or synchronization among servers (e.g., lobby servers for console games). MMORPGs however, are usually designed as a cluster of servers with each server hosting its own copy of the game world (or *shard*) for a large, but isolated group of players. In such games, scaling is on the granularity of shards. In order to scale such games on a finer scale (e.g., by adding or removing a single server), the game application must be written to take advantage of additional resources in small increments. In this way, the shard architecture can be replaced with a single game world in which all players reside. Some recently proposed distributed game architectures offer communications support to enable a single large game world to be partitioned across multiple servers, allowing all players to interact seamlessly (e.g., see References 5, 6, 8, and 16). In such architectures, an on demand infrastructure can dynamically adjust server resources as the player population in different areas of the game changes.

Usage scenarios for the ODOE for games

The design and features of the on demand platform for games enable a number of interesting usage models. Our original intent was to provide a game hosting infrastructure that could be deployed by major hosting service providers (e.g., IBM Global Services). A shared infrastructure that is further multiplexed with other business applications has the potential to lower the cost of hosting games, particularly for smaller game publishers who could not otherwise afford to build their own infrastructures. An advantage of lowering the barrier to entry is to encourage risk and innovation in games that might not be supported by major game publishers. The gaming platform can also enable large game publishers to offer a platform to third-party game developers (i.e., as opposed to in-house game studios). For example, console game providers could use the platform to offer a relatively low-cost hosting solution to encourage third-party developers to enable their games for online play. Finally, the automatic provisioning and software deployment services of the platform can be applied in next-generation gaming services. One example is an "on demand gaming party" service in which customers select games to play with friends on well-provisioned private servers. The platform automatically provisions powerful game servers with very good network connectivity to the corresponding game server software for the duration of the party (assuming that the requisite licensing agreements can be worked out with game publishers). The customer could invite players by distributing the game server address and necessary authentication information.

■ If we expand the set of applications using the shared pool to include business applications, the opportunities to multiplex become clear. ■

Security issues

The fact that provisioning in our current implementation is done on a server-by-server basis simplifies the security issues arising from more fine-grained resource sharing. In practice, however, this policy can lead to unnecessary overprovisioning, for example for games with modest resource requirements. In such cases, it may be desirable to host multiple games on a single server, but this clearly requires sufficient protection between the games running on the same server (e.g., to prevent one game from corrupting another or causing it to crash). Additionally, while the peer-to-peer distribution model is appealing from the standpoint of lowering bandwidth costs, it also introduces added liability for game providers who use it to distribute content. Players must allow unknown and untrusted machines to connect to their own machines, thus increasing the risk of exposure to malicious users.

Future work

We plan additional enhancements to the service platform for use with other types of games and with new services. For example, we plan to demonstrate automatic provisioning for persistent world multiplayer role-playing games and some of the new distributed game architectures. To realize player services, we plan a number of directory services based on Lightweight Directory Access Protocol (LDAP) that account for player preferences when connecting to game servers. Although we mention that network resources can also be managed in an on demand fashion, we have not implemented this in the first version of our prototype, choosing instead to focus on server and software provisioning. Related research on measuring the network profile (e.g., in terms of bandwidth consumption) for various games provides guidance for managing network bandwidth.[17–19] We plan to add support for monitoring and tuning bandwidth usage for the in-game network based on these profiles. Finally, we are investigating the suitability of different game metrics (i.e., at the application level) for other types of games.

SUMMARY

In this paper, we described the development of a shared, on demand infrastructure for hosting large-scale multiplayer games. Our work is inspired by the on demand computing environment model and embraces several of its main concepts. For example, the platform supports integration through its modular, service-oriented design in which components can be used as needed and uses open standards including Linux, XML, and SOAP. Infrastructure simplification is achieved in the platform through the use of automation in the form of provisioning and workflows and through virtualization, which allows servers to be used across multiple games, applications, and customers.

In our prototype, we used commercial off-the-shelf provisioning software and adapted it for game applications. We also implemented a number of services that make it easy for publishers to use and demonstrate the platform with the Quake II game.

Our experiences illustrate the feasibility of applying on demand computing concepts to an infrastructure for hosting multiplayer online games. Through our ongoing implementation efforts, we have been able to identify where existing utility computing technology and open standards may be leveraged and where game-specific customizations are required.

*Trademark, service mark, or registered trademark of International Business Machines Corporation.

**Trademark, service mark, or registered trademark of id Software, Inc., Intel Corporation, Linus Torvalds, Origins Systems, Inc., Sony Computer Entertainment America Inc., Turbine, Inc., Mythic Entertainment, Inc., Jagex Software, Square Enix Co., Ltd., Electronic Arts Inc., Gravity Interactive LLC, Lucasfilm Entertainment Company Ltd., Cryptic Studios Inc., NCSoft Corp., BitTorrent, Inc., Valve Corporation, or IGN Entertainment in the United States, other countries, or both.

CITED REFERENCES

1. B. S. Woodcock, "An Analysis of MMOG Subscription Growth," Version 16, April 2005, http://www.mmogchart.com.
2. *The On Demand Operating Environment*, IBM Corporation, http://www.ibm.com/ebusiness/ondemand/us/overview/operating_environment.shtml.
3. I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich, "The Open Grid Services Architecture, Version 1.0," Global Grid Forum OGSA Working Group GFD-I.031 (January 2005), http://www.gridforum.org/documents/GFD.30.pdf.
4. D. Saha, S. Sahu, and A. Shaikh, "A Service Platform for On-Line Games," *Proceedings of 2nd Workshop on Network and System Support for Games (May 2003),* pp. 180–184.
5. G. Deen, M. Hammer, J. Bethencourt, I. Eiron, J. Thomas, and J. Kaufman, "Running Quake II on a Grid," *IBM Systems Journal* **45**, No. 1, 21–44 (2006, this issue).

6. A. Bharambe, J. Pang, and S. Seshan, *A Distributed Architecture for Interactive Multiplayer Games*, Technical Report CMU-CS-05-112, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 (2005).
7. B. Jacob, S. Mui, J. Pannu, S. Park, H. Raguet, J. Schneider, and L. Vanel, *On Demand Operating Environment: Creating Business Flexibility*, IBM Redbook (2004), http://www.redbooks.ibm.com/abstracts/sg246633.html.
8. BigWorld Server, BigWorld Pty., Ltd., http://www.bigworldtech.com/server.php.
9. E. Manoel, S. C. Brumfield, K. Converse, M. DuMont, L. Hand, G. Lilly, M. Moeller, A. Nemati, and A. Waisanen, *Provisioning On Demand: Introducing IBM Tivoli Intelligent ThinkDynamic Orchestrator*, IBM Redbooks (2003), http://www.redbooks.ibm.com/redbooks/pdfs/sg248888.pdf.
10. S. Jankowski, *QStat: Real-time Game Server Status,* http://www.qstat.org.
11. Steam, Valve Corporation, Inc., http://www.steampowered.com.
12. C. Chambers, W.-C. Feng, S. Sahu, and D. Saha, "Measurement-Based Characterization of a Collection of On-line Games," *Proceedings of the Internet Measurement Conference* (October 2005), http://www.usenix.org/events/imc05/tech/full_papers/chambers/chambers.pdf.
13. BitTorrent, Inc., http://www.bittorrent.com, 2005.
14. Blizzard Downloader FAQ, Blizzard Entertainment, http://www.worldofwarcraft.com/info/faq/blizzarddownloader.html.
15. A. Shaikh, S. Sahu, M. Rosu, M. Shea, and D. Saha, "Implementation of a Service Platform for Online Games," *Proceedings of ACM SIGCOMM Workshops on NetGames* (August 2004), pp. 106–110.
16. P. Rosedale and C. Ondrejka, *Enabling Player-Created Online Worlds with Grid Computing and Streaming*, Gamasutra, GMP Media Inc. (2003), http://www.cs.ubc.ca/~krasic/cpsc538a-2005/papers/rosedale.pdf.
17. Wu-chang Feng, F. Chang, Wu-chi Feng, and J. Walpole, "A Traffic Analysis of Popular On-line Games," *IEEE/ACM Transactions on Networking* **13**, No. 3, 488–500 (June 2005).
18. K. Chen, P. Huang, C. Huang, and C. Le, "Game Traffic Analysis: An MMORPG Perspective," *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video* (June 2005), pp. 19–24.
19. S. Zander and G. Armitage, "A Traffic Model for the Xbox Game Halo 2," *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video* (June 2005), pp. 13–18.

Accepted for publication July 29, 2005.
Published online January 11, 2006.

Anees Shaikh
IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (aashaikh@watson.ibm.com). Dr. Shaikh is a research staff member in the Networking Software and Services Group. He received B.S.E.E. and M.S.E.E. degrees from the University of Virginia, and a Ph.D. degree in computer science and engineering from the University of Michigan in 1999. His research at IBM has focused on Internet service infrastructure, particularly in the areas of content distribution, Web and network performance, and Internet measurement. Dr. Shaikh has also published a number of papers on load-sensitive routing, middleware for real-time communication, and multicast routing.

Sambit Sahu
IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (sambits@us.ibm.com). Dr. Sahu is a research staff member in the Networking Software and Services Group. He received a Ph.D. degree in computer science from the University of Massachusetts at Amherst in 2001. He joined IBM in 2001. His research has focused on overlay-based communication, network configuration management, content distribution architecture, and design and analysis of high-performance network protocols. Dr. Sahu has published a number of papers in the areas of differentiated services, multimedia, overlay-based communication, and network management and has over 20 patents filed in these areas.

Marcel-Catalin Rosu
IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (rosu@us.ibm.com). Dr. Rosu is a research staff member in the Ubiquitous and Wearable Computing Department at IBM Research. He received M.S. degrees from the Polytechnic University of Bucharest and Cornell University in 1987 and 1995, respectively, and a Ph.D. degree from Georgia Institute of Technology in 1999, all in computer science. His research interests include operating systems, distributed systems, multimedia, Web applications and services, and mobile computing.

Michael Shea
IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (mike.shea@gmail.com). Mr. Shea received a B.Math. degree in computer science from the University of Waterloo, Canada, in 2003. He worked as a co-op at the IBM Thomas J. Watson Research Center in the Network Software and Services Group in 2004. Mr. Shea is currently a developer at Tira Wireless, working on a system for automating the porting of cell phone games.

Debanjan Saha
IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (dsaha@us.ibm.com). Dr. Saha is with the Security, Privacy, and Networking Department at IBM Research. He leads a group of researchers working on different aspects of network software and services. He received a B.Tech. degree in computer science and engineering from the Indian Institute of Technology in 1990, and M.S. and Ph.D. degrees in computer science from the University of Maryland at College Park in 1992 and 1995, respectively. Dr. Saha has authored more than 50 papers and standards contributions and is a frequent speaker at academic and industry events. He serves regularly on the executive and technical committees of major networking conferences. ■