

CYBERNET

OpenSkies

Networking Engine

**Massive Multiplayer Online Gaming
(MMPORG) SDK**



www.openskies.net

MMPORG
MASSIVE
MULTIPLAYER
ONLINE GAMING

Openskies Massive Multiplayer Online Game SDK

Why You Need OpenSkies Massive Multiplayer Technology in Your Game

- (1) You have decided to make an online *massive multiplayer* game (MMPOG) because:
 - It fits your game concept
 - It provides a new persistent income stream
- (2) You know that networking is hard to implement and debug, especially if you have a very large number of players (>32) – which translates to a large, hard-to-control testing regime.
- (3) As game developers you have focused on how to develop and implement content – maybe you put off implementing the communications systems until last and it is turning out to be harder than you thought to get working.
- (4) Maybe you don't know how many players your server systems will ultimately have to handle – it depends on game popularity and marketing down stream.
- (5) You are worried about security against spoofing of your MMPOG – one player getting the drop on others through hacking.
- (6) You need proven, in-use massive multiplayer technology – OpenSkies technology is used in Taldren Star Fleet Command II – see Cybernet's press release attached.

Cybernet's OpenSkies MMPOG system of APIs, servers, lobby managers, and routing systems may be your answer.

What is OpenSkies and What Can it Do for You?

OpenSkies offers a versatile proven set of C++ APIs for massive multiplayer communications that easily integrates with your game architectures. For those familiar with the US Government HLA standard, the lowest level OpenSkies APIs offer compatibility for real time HLA compliant simulations. For PC and console game implementers, C++ APIs that offer superior ease of integration are built on top of the HLA level API. OpenSkies APIs allow both peer-to-peer and peer-to-server cluster networking and support both application push and automated object state-change push communications models.

OpenSkies developers can:

- (1) Use OpenSkies to allow clients to communicate directly to each other without using external servers (*Peer-to-peer*).
- (2) Use OpenSkies to allow clients to communicate to each other via external servers (federation hosts or *FedHosts*) to intelligently cull traffic (*FedHost Routed*).
- (3) Use OpenSkies to allow large numbers of clients to communicate to game servers via FedHost routing. (*Server Cloud*)
- (4) Support hundreds, thousands and even multiple thousands of simultaneous player connections. Why stop at 32? The aggregate throughput of the externalized federation hosts scale simply by adding

additional hardware. Each PC-based federation host services nominally 500 player connections¹ and as you add users you simply added additional FedHosts – scaling is linear.

- (5) Assume fault tolerance and 100% game server availability – when software or hardware fail any attached play clients are transparently moved to functional hosts. Your game will never be down with redundant server hardware in your game network. Redundant hardware can be externally contracted – it can be anywhere on the Internet.
- (6) Rest assured that OpenSkies is prepackaged and pre-tested in real game use – see attached press release. Proven technology – no prototype or untested software here.
- (7) Control message culling and message acceptance rules which operate on the FedHosts. Your security can be as tight as you want it to be and your routing can be as simple or as complex as you need it to be.
- (8) Network your game servers as easily as the game clients.

The OpenSkies team can provide you APIs and servers, support, and game development support. Typical pricing is an advance against royalties and a negotiated royalty rate over the minimum advance. We can set up and operate your OpenSkies servers at your discretion.

We want to work with you to make your MMPOG a great success. Call or e-mail us to let us know how we can help.

Cybernet Systems also implements other game engine components and content for terrain management, object management, physics models, training modules, and user interfaces on Windows or Unix/Linux systems on a contractual basis. Cybernet does both commercial and government simulation and software development on an RFP/RFQ basis.

Contact us for more information or an evaluation package which includes a full OpenSkies documentation set, “Hello World” code snippets, a sign on for Cybernet’s demo, Edge of Extinction MMP sample game system, and a complimentary Star Fleet Command (SFC) II CD patch with OpenSkies MMPOG upgrades:

sales@cybernet.com
734-668-2567

or

Cybernet Systems Corporation
Attn: Sales or Charles Cohen
727 Airport Blvd.
Ann Arbor, MI 48108

www.cybernet.com – Corporate site

www.edgeofextinction.com – Demo MMPOG Flight simulation site

www.OpenSkies.net – Information on the OpenSkies MMPOG communications system

www.taldren.com – The Star Fleet Command II MMPOG site

To order SFCII go to www.compusa.com and search for Star Fleet Command: Volume II.

¹ This 500 client estimate varies depending on host hardware speed and game requirements. The 500 estimate was verified on testing done for Cybernet’ EOE where each player produces 3D location/velocity state packets once every 1/10 of second. For games like Star Fleet Command II with lower bandwidth requirements substantially more clients can be supported per FedHost.

OpenSkies APIs Used in Your Game

There are actually three levels of OpenSkies API you can use in your communications. Each is a C++ API that is currently supported on Win32 platforms. They are the *HLA* interface, the *IET* interface, and the *P2PS* Interface. Government or military agencies or their contractors who adhere to the *HLA* standard would use that interface. Game developers are encouraged to use the *IET* interface, as it is the simplest to use and has been designed with game development in mind. The *P2PS* interface is a point-to-point messaging API that is built on top of the *IET* interface and can be used in conjunction with it.

Documentation for all of the Openskies APIs may be found at <http://www.openskies.net/download/download.html>.

High Level Architecture (HLA) Interface

The lowest level API is modeled after the Department of Defense's HLA. This API architecture provides calls that register simulation objects and can push object state change data. Simulation objects from other players in the network appear in your application as simulation objects whose behavior is controlled by the owner's message stream (as opposed to your application's physics models). See http://www.openskies.net/files/Openskies_HLA_Guide.pdf.

ImportExportTable (IET) Interface

A very simple and efficient interface, the IET development kit includes a simple application called *setup.exe* that allow the game developer to specify the format for network-enabled objects. *Setup.exe* then automatically generates the header (.h) and source (.cpp) files required for network communication. Updating your object on the net is now as easy as:

```
object->Update();
```

This interface is the preferred mechanism for game developers because it is designed to be as simple as possible without sacrificing capability. The IET interface is an object-oriented architecture that allows objects to inherit networking functionality in much the same way C++ methods and variables are inherited. See http://www.openskies.net/files/Openskies_IET_Guide.pdf.

Point-to-Point Switch (P2PS) Messaging Interface

This interface, which was build on top of the IET Interface allows the user to send messages from one object to another by name. Messages can be sent to objects that have not yet been registered. In this case the data is cached until an object with that name is (if ever) registered. Additionally, the relative location of the 2 objects (whether they are on the same or different machines) is transparent to the application. Such an interface greatly simplifies point-to-point communications between objects, allowing the developer to move and/or combine code modules without having to modify the communication code. See http://www.openskies.net/files/Openskies_P2PS_Guide.pdf.

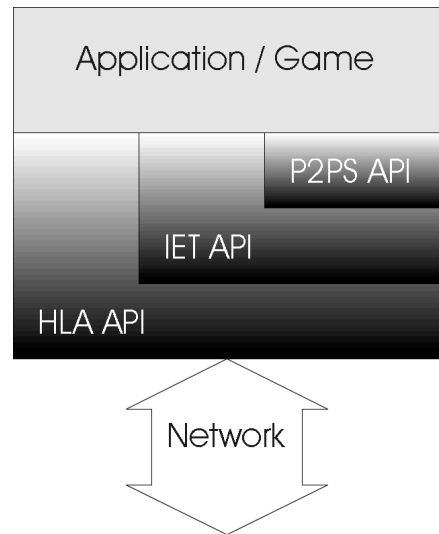


Figure 1: Openskies APIs

Authentication

The OpenSkies system provides you with sample routines for authenticating users to a game space. It allows you to design your own authentication scheme as well, without having to worry about all of the communications. It also provides example code for sending and retrieving permanent player and game data to/from centralized database servers, and a web site template, which allows game network state (coded in the central game database) to be viewed over the internet using your favorite browser. See http://www.openskies.net/files/Openskies_Security_Guide.pdf.

OpenSkies APIs are coded in C++ for Windows (95, 98, ME, NT, and 2000). To keep things simple, they do not reference MS Foundation Class Libraries (MFC) and stick to simple C++ and operating systems calls.

Why Windows? Because that is where many new games are developed and launched. We will support your development efforts targeted for Linux, X-Box, or Playstation 2 as part of a qualified development partnership agreement.

Minimum Requirements: Any Windows compatible platform that can run your game.

OpenSkies Server Architecture

The OpenSkies networking library can either be *Peer-to-Peer*, *FedHost-Routed*, or *Server Cloud*. Each of these modes is API compatible with each other. In other words, no code changes beyond setting a simple switch are required to transition between modes.

Peer-to-Peer

The *peer-to-peer* mode functions in an HLA or DirectPlay-like manner. This mode, essentially performing communications directly between players, is suitable for group play use over local LANs and a limited number of WAN hook-ups. On a local LAN up to several hundred players can inter-operate (dependent on station-to-station network performance). During play, each player/client, called a *federate*, owns a collection of simulation objects whose state changes must be sent to all other player stations. This results in mn^2 communications paths where n is the total number of players and m is the number of moving or changing objects per player. Clearly this mode does not scale well to a very large number of players, but it does allow you to quickly and easily set-up light multiplayer games.

FedHost Routed

FedHost routing is what allows your application to be truly Massively Multiplayer. The OpenSkies calls in your game would be identical to the calls for *Peer-to-Peer* mode, but instead of sending data directly to the other clients or players, the OpenSkies library will communicate with specialized servers located on your LAN or on the Internet (over WAN, DSL, and/or dial-up connections).

Two essential server modules implement ***FedHost Routing***. They are the ***FedHost*** and the ***LobbyManager***. See http://www.openskies.net/files/Openskies_Network_Architecture.pdf for general information about the OpenSkies network architecture and http://www.openskies.net/files/NetmaxOS_Guide.pdf for specific information about setting up OpenSkies servers.

The federation host or **FedHost** process is the core of the system. During multiplayer play each player/client is connected via a two-way socket connection to one FedHost. Each FedHost supports message input and output to a number of game clients set by the power of the hardware upon which the FedHost process runs (a typical number might be 500 clients² to FedHost connections per hardware platform). Each FedHost also is in communication with all other FedHosts, which support the entire massive multiplayer game (in HLA terms this collection of clients and FedHosts implementing a single integrated game zone is called a *federation*).

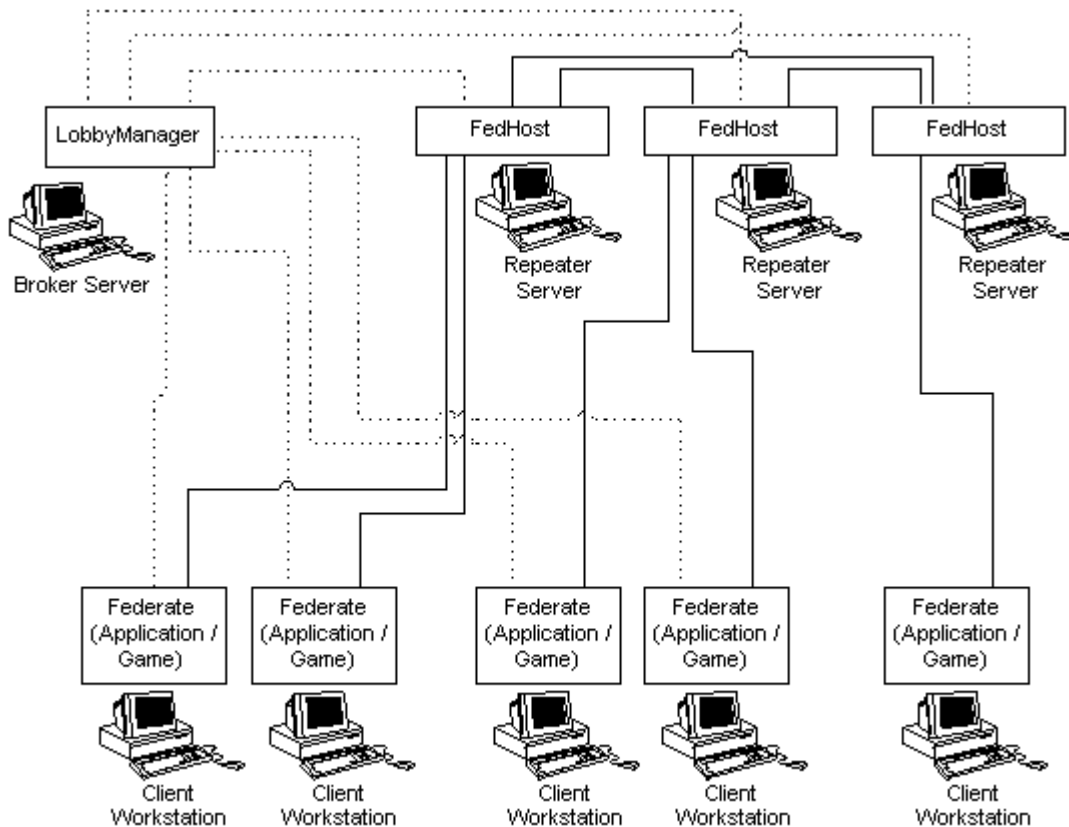


Figure 2: Topology of a Routed Federation

FedHosts are game-independent, but can apply federation-specific message routing and culling rules within each federation that is active. The game developer can implement culling rules (rules which validate player messages and use game-specific heuristics for reducing player to player communications) using the **FedHost API**. The developer creates a shared object (or dynamically linked library) that the FedHost accesses in order to reduce traffic. See http://www.openskies.net/files/Openskies_Culling_Guide.pdf.

FedHosts can maintain multiple federations and federation-specific routing and culling rules at the same time, thus supporting multiple game zones within the same gaming hardware resource pool. FedHosts are in constant communications so that all servers know what any server knows (after limited communications delays) – this enables smooth addition of new FedHosts (possibly as game traffic increases) and movement

² The 500 client estimate varies depending on host hardware speed and game requirements. The 500 estimate was verified on testing done for Cybernet' EOE where each player produces 3D location/velocity state packets once every 1/10 of second. For games like Star Fleet Command II with lower bandwidth requirements substantially more clients can be supported per FedHost.

of client connections from one FedHost to another (for instance if a FedHost fails or is taken offline) without any loss in game zone accessibility to the player.

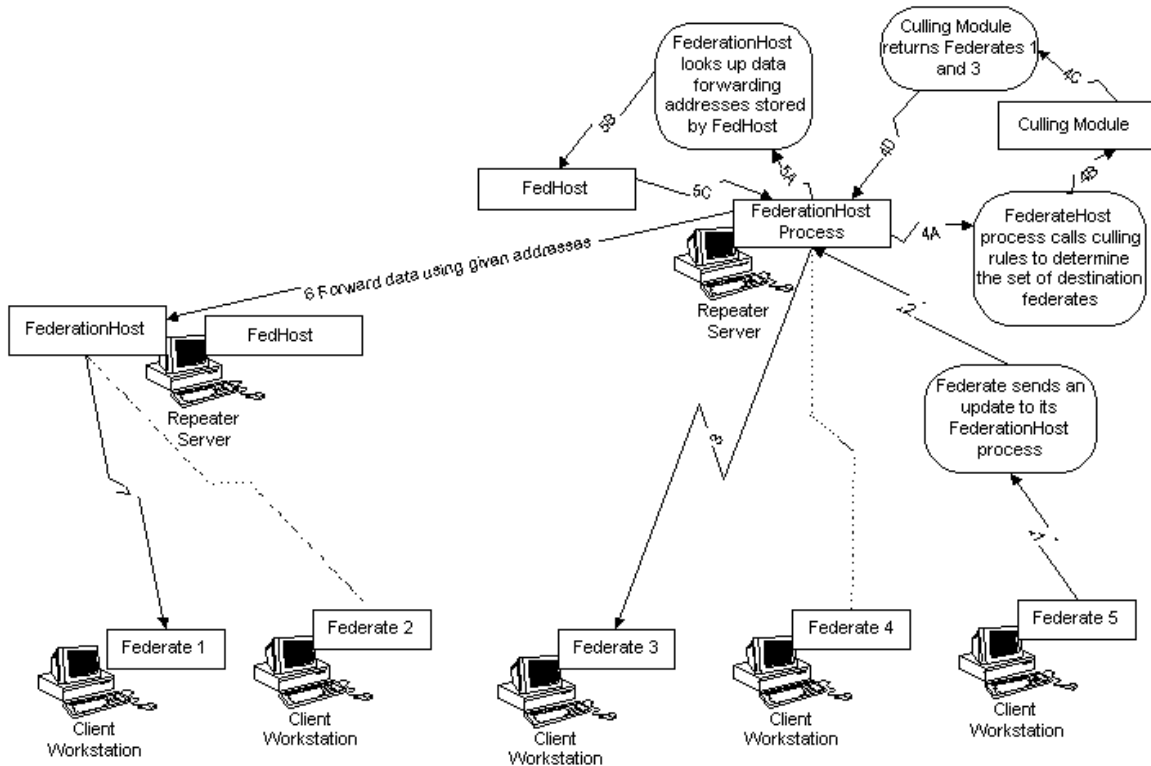


Figure 3: Data being routed by FedHosts

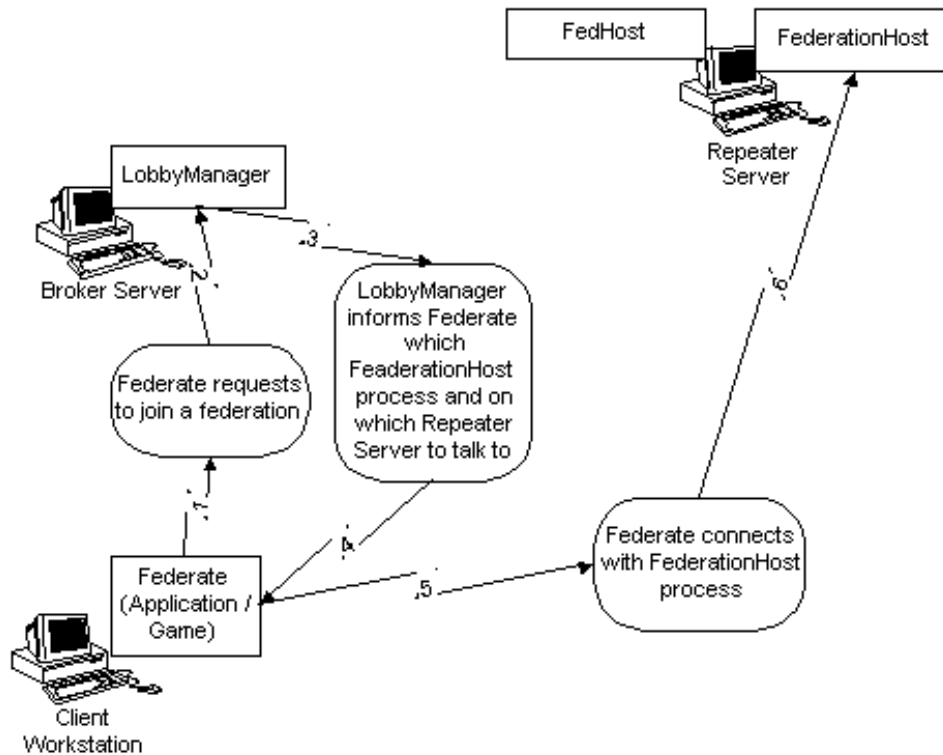


Figure 4: Connecting to Federation through LobbyManager

The **LobbyManager** process accepts and authenticates player connection requests, knows where all FedHost processes are located, and mediates player connection/disconnection to/from a designated FedHost. Lobby managers continuously keep track of every FedHost’s execution status so that a complete model of the game network traffic is always available. This information is sent to an SQL database and can be queried via a browser. The Lobby manager also implements basic connection and disconnection accounting functions.

Server Cloud

Games have historically been modeled as “Peer-to-Peer” or as “Server-Based”. Peer-to-Peer games typically involve small numbers of players that are all in constant communication with one another. This paradigm does not scale to massive multiplayer. Server-based games typically execute game logic on expensive specialized hardware that acts as a hub for thousands (perhaps all) of the participating clients. These “Server Farms” must contain machines that are powerful enough to perform the game logic and handle the network traffic required to support thousands of players.

The OpenSkies network architecture allows you to add one or more **GameServers**, central server processes that know about aspects of the multiplayer game logic and/or maintain the game universe. A game server in an OpenSkies network is an automated federate that acts the same as any player/client and uses the same API to gain access to the network for communication with other federates. Game servers implement game automation and/or game-specific access to permanent memory and do not normally require graphics. The way to differentiate these game server federates from regular player federates is to write culling rules that may preferentially route traffic to and from game servers.

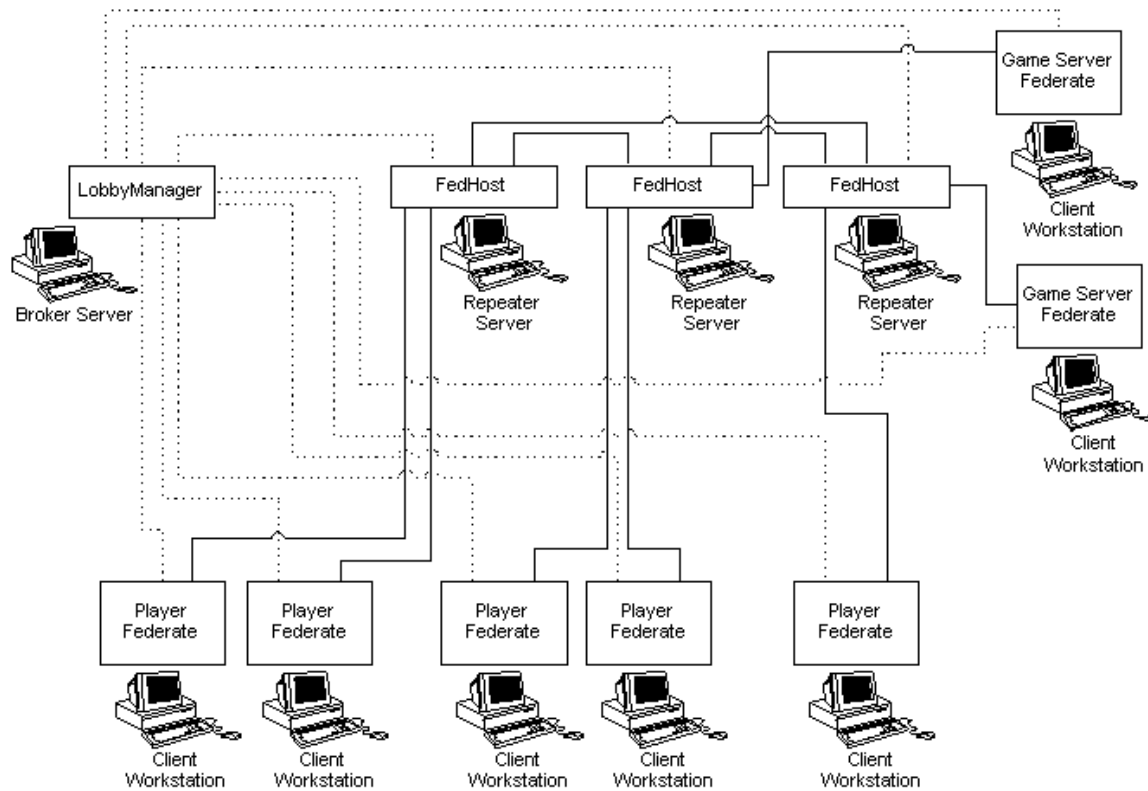


Figure 5: Federation with Game Servers

OpenSkies system does not implement your game server(s), but gives you the power to implement whatever configuration or function in game servers that you want. Here are some possible game server setups:

- One game server doing all game logic
- Multiple game servers, each of which handles an geographical/spatial segment in the virtual game universe
- Multiple game servers, each of which handles an aspect of play (e.g. time, score, mission assignment , AI,...)

You may choose to implement virtually all network game logic on the game servers, with the player's client providing only interaction and game space display functions. Many developers prefer this approach because it isolates game logic from would-be hackers, thereby reducing the vulnerability of the network game to malicious acts. At the other end of the security spectrum, each game server may simply be an automated player.

As indicated earlier, OpenSkies APIs are coded in C++ for Windows (95, 98, ME, NT, and 2000). To keep things simple, they do not reference MS Foundation Libraries and stick to simple C++ and operating systems calls. *This means your game server will be one or more Windows applications with the standard OpenSkies system.* We will support your development game servers hosted on other platforms like Linux as part of a qualified development partnership agreement.

Other Components

To complete a network game you will typically need a **Web Server** to post game specific marketing and community information, a **Database Server** to store permanent game and user information, and a **Network Console** to keep you current on your game network operations or status.

Any SQL compatible database can be used to implement your **Database Server**. The database server acts as the permanent and user data memory for the OpenSkies Database functions API within your game. We have integrated a Linux/MySQL for this purpose and deliver this as an example set-up with the OpenSkies server system and documentation so you can easily modify or extend it. We will support your choice as part of a qualified development partnership agreement. For information about the OpenSkies database demonstration implementation see http://www.openskies.net/files/Openskies_Database_Guide.pdf.

Most games have web sites that support down load of patches, launch, and provide community facilities from the web site (or sub-web site linked into multi-game web site). The **Web Server** holds up these sites. We have implemented sample game web sites on Cybernet's Linux-based **NetMAX Web/Internet Server Suite** products (Linux, Apache, Perl, PHP, MySQL – more information is available at <http://www.netmax.com/support/onlinedocs/onlinedoc.html>). A wide variety of alternative web platforms are also in use. We are familiar with most of the alternatives and will support your choice of web options as part of a qualified development partnership.

The **Network Console** allows you to monitor the health and status of the routing backbone (i.e. the FedHosts and Lobby Managers). The OpenSkies network can be monitored and observed from a secure webpage that is maintained on the database machine(s). Therefore any web browser-enabled or MS Access/ODBC station can implement your **Network Console**. See http://www.openskies.net/files/Openskies_Database_Guide.pdf and http://www.openskies.net/files/NetmaxOS_Guide.pdf for more information.

Why Linux for the Servers/Routers?

Your game server will probably be implemented in Windows to make debugging and implementation as easy for you as game client implementation.

The platform for database and web server will be your choice, but we implement the FedHosts and Lobby Manager applications on Linux (and our web and database systems are on Linux as well). We use our Linux-based NetMAX product as the starting point for network nodes and routers because:

- (1) We control the entire system from the hardware up – no unpredictable version change problems in midstream.
- (2) Our benchmarks on FedHost performance indicate that in this kind of network intensive application, Linux-based systems are about 3 times faster than Windows 2000. This equates to 1/3 the hardware cost.
- (3) Linux uptime reliability is superior to Windows. We really do not want to reboot servers any more than needed (even though we have built in player transparent fault tolerance).
- (4) Cost. Using the Linux-based NetMAX to perform network routing results in low routing node hardware cost.

Server and Game Security

We run our Linux server components on top of NetMAX (see www.netmax.com). NetMAX products support firewalls, VPN, SSL web services, and proven Unix-style messaging and authentication methods. They are as secure as the best on the net.

Security for your game server can be handled at two levels:

1. You can do it yourself:
 - By putting game logic into your game clients so they only respond to valid requests.
 - By implementing all game logic on your own, using OpenSkies technology to scale up your messaging capabilities for massive multiplayer use.
2. You can leverage OpenSkies architecture:
 - By using the OpenSkies FedHost culling rule system to implement game network message sniffers which only allow message forwarding of valid player actions. This can be done even after a game has been released by changing rule code on the FedHosts.
 - By modifying FedHost and Lobby Manager authentication methods (without changing game logic).

See http://www.openskies.net/files/Openskies_Security_Guide.pdf.

Minimum Requirements

All Linux-based server components (FedHost, Lobby Manager, Database, and Web Server) can be run on one machine, but we typically functionally divide them into one or more FedHosts (distributed as close to a cluster of client connections as possible), the Lobby Manager, and the Database/Web Server. For a heavily hit web site, the Web Server is replicated with request redirection based on web server loading.

For the Linux-based server components any Intel server capable platform with a network card and connection will suffice. CPU, RAM, and HD storage requirements depend on:

- Maximum number of players per FedHost (scales linearly)
- Requirements of the application dependant culling/routing algorithms
- Application-dependant DB server requirements

The Linux-based server components require no graphics or audio hardware (minimum VGA may be useful for some low level systems diagnostic functions).

Contact us for more information or an evaluation at:

sales@cybernet.com

734-668-2567

or

Cybernet Systems Corporation

Attn: Sales or Charles Cohen

727 Airport Blvd.

Ann Arbor, MI 48108

www.cybernet.com – Corporate site

www.edgeofextinction.com – Demo MMPOG Flight simulation site

www.OpenSkies.net – Information on the OpenSkies MMPOG communications system