

*CYBERNET*

**OpenSkies**

Networking Engine

**Network Architecture**



[www.openskies.net](http://www.openskies.net)

**MMPG**  
**MASSIVE**  
**MULTIPLAYER**  
**ONLINE GAMING**

# Openskies Network Architecture

## System Components

### *Lobby Manager*

The following applies to installations that make use of servers outside of your local LAN. These servers are provided by third party service providers such as Cybernet Systems Corporation.

LobbyManager can be placed on a "broker server" computer to manage a large network of federations over the Internet. The first feature for such purpose is user authentication. An application can use the "Login" member function of HLA\_RTI::CLobbyManager class to login to LobbyManager, and the "Logoff" member function to log off. HLA\_RTI::CLobbyManager is declared in LobbyManager.h.

LobbyManager can keep track of a list of "repeater server" machines. Each "repeater server" machine will be running a copy of FedHost to be discussed later.

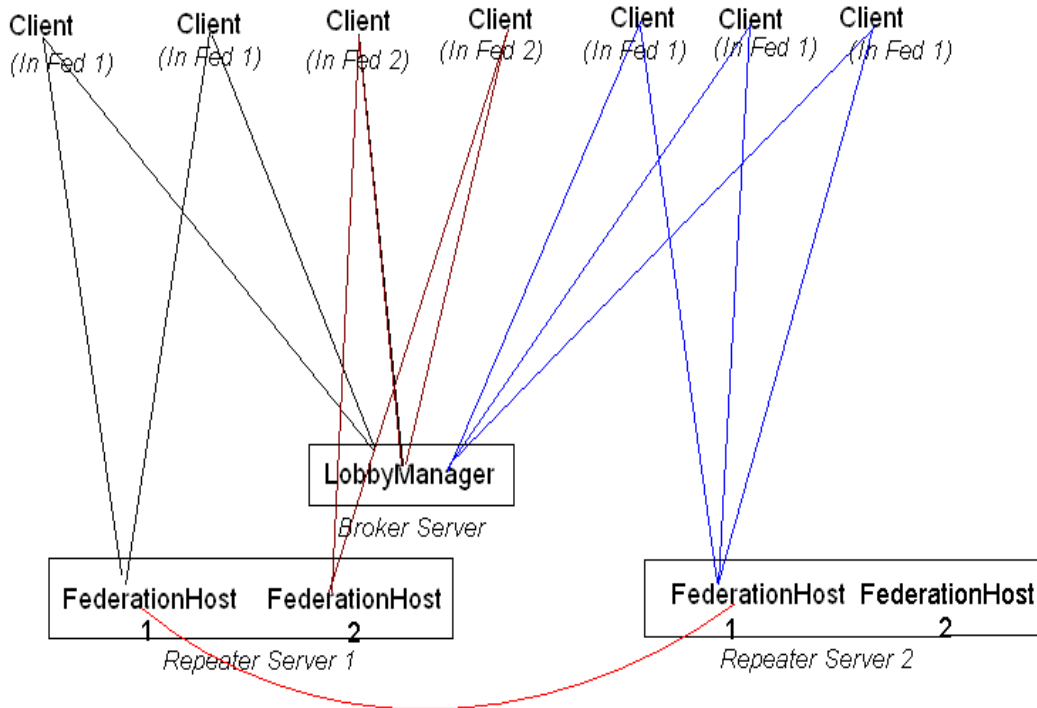
If a user requests to create a new federation, a new federation host process will be launched on each "repeater server" machine. If a user wants to join an existing federation, a least busy repeater server with the least ping time will be assigned to him. All FederationHost processes for the same federation on various repeater server machines will be able to communicate with each other, and each will have information about the entire federation.

For example, if a client requests LobbyManager to create a federation called "Fed1", LobbyManager makes sure that there is no federation called "Fed1" out there, and then create a hosting process for it on each repeater server, namely 1 and 2, called "FederationHost1". When another client requests to join "Fed1", it will be assigned to the least busy FederationHost1 on repeater server2. Based on the load percentage = number of clients / maximum number of allowed clients of each repeater server, the next client can be assigned to either repeater server 1 or repeater server 2. repeater server 1 or repeater server 2 share the same client list, the same object list, etc.

The reason that we do not start new federation host processes as needed is that when the number of federates is large ( $\geq 100,000$ ), starting new federation hosts can be very costly. You must update information for all federates on this new host. Thus adding new host processes can be used for system recovery after a partial crash, but not for dynamic client connections. However, new federation hosts can be added after a federation starts up when needed. They become available to users when all necessary updates are completed.

Suppose that a client requests to create another federation called "Fed2", LobbyManager will make sure that there is no federation called "Fed2" out there, and create new hosting processes

called FederationHost2 on repeater server1 and repeater server 2, respectively. When another client requests to join "Fed2", it will be assigned to FederationHost2 on repeater server2, etc., just like for "Fed1".



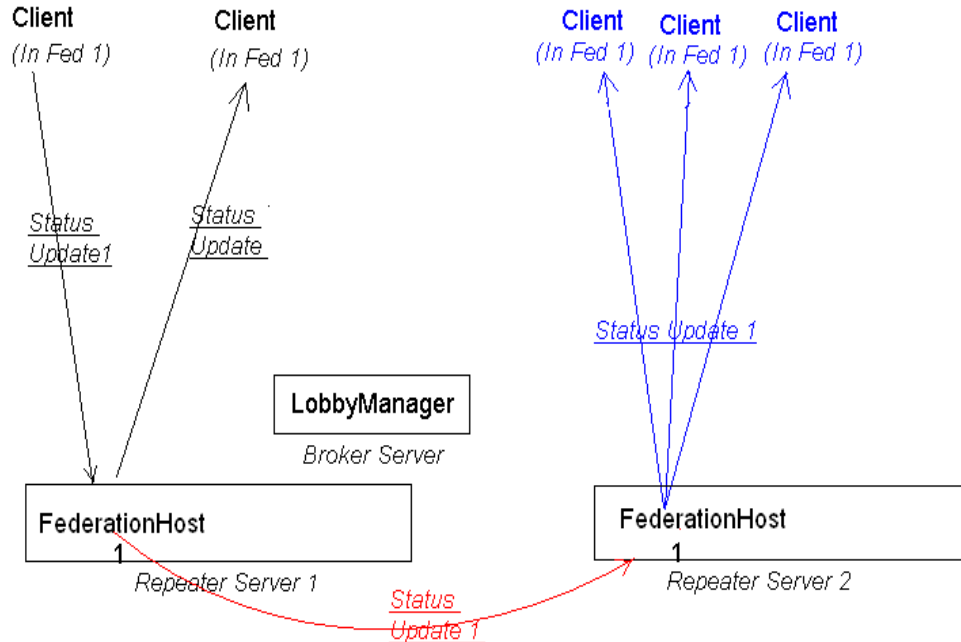
LobbyManager run-time configuration is stored in a file named LobbyManager.ini. It must be stored in the same directory as LobbyManager itself. It is an ASCII file, and can be edited with a text editor. It contains 3 sections. The first section specifies NIC address and port number it uses. The second section specifies multicast address and port it uses. The third section lists all repeater servers that it is supposed to manage. An actual copy of a LobbyManager.ini file can be found in Appendix A.

### ***FedHost***

As mentioned in the previous section, when managing a large network of federations over the Internet, LobbyManager will spawn FederationHost processes to host a federation. There is an executable called FedHost running on each repeater server at all times. When LobbyManager needs to spawn new FederationHost processes, it sends a command to each FedHost, and each FedHost will in turn spawn an actual FederationHost process on the repeater server that it resides on. *The actual code for FederationHost is contained in the FedHost executable.*

FederationHost will perform all the host functions in the existing RTI code. They communicate with each other via TCP/IP connections, IP multicast, and UDP datagrams. They will each maintain a complete list of federates, but each will communicate directly with a limited number of these clients.

LobbyManager determines exactly which clients will communicate with a given FederationHost. The client code in the Openskies client libraries receives the IP address of a repeater server, and a port address of a FederationHost Process from the LobbyManager after it logs in. Then the client code will establish a TCP connection with the FederationHost, as well as UDP communication.



Because clients do not communicate directly with each other, network traffic is greatly reduced for the client. For example, if FederationHost 1 on repeater server 1 is hosting 10 users, and FederationHost 1 on repeater server 2 is hosting 10 users, for the update of the status of a single client connected to repeater server 1, there is going to be one and only one transmission of data from repeater server 1 to repeater server 2. The status update does not need to be done 10 times for each client connected to repeater server 2 between the repeater servers.

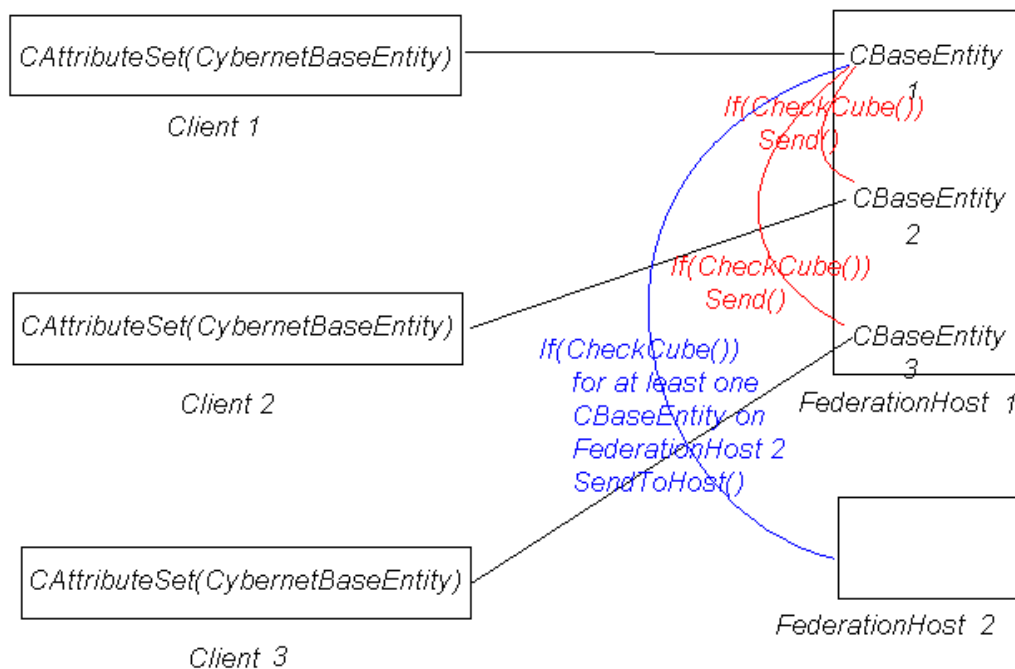
FederationHost will also perform the functions of culling to further reduce network traffic. For a given application, the data objects defining the packetization of data are set in a configuration file called a FED file, which resides on each repeater servers and accessed by the FedHost on that server. The Culling functions, which also reside on the repeater server, are defined as "member functions" for the attribute sets in the FED file. These attribute set definitions will be provided by Cybernet in a FOM (federation object model) library. Application developers can also develop their own culling routines in DLL's, or "so" files under LINUX. Each function can be turned on and off at run-time. Because one can derive new attribute sets from existing ones, just like C++ class derivation with single inheritance, we can create other attribute sets, and not limit ourselves to what are in the FOM library. We will have a complete list of attribute set definitions in the next section.

For example, if we start with "CybernetBaseEntity", which consists of double Altitude, double Latitude, and double Longitude, we can override the "Cull" member function, and define it in such a way that it returns

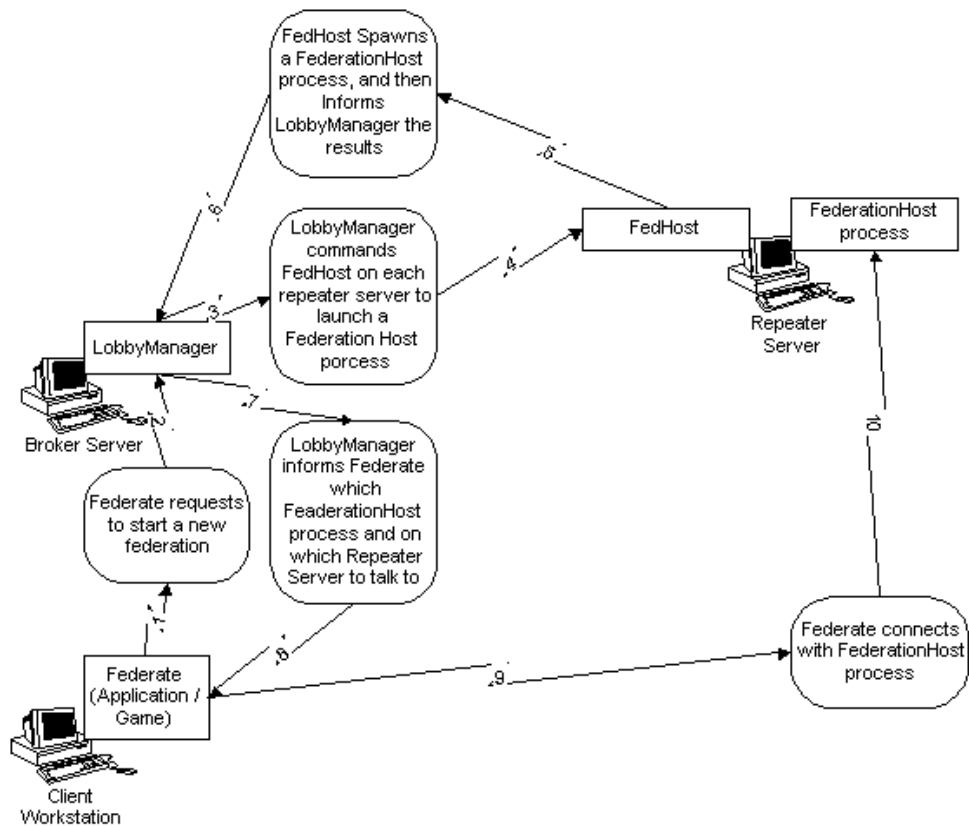
TRUE if (a2 >= Altitude-Altitude0 >= a1) and (b2 >= Latitude-Latitude0 >= b1) and (c2 >= Longitude-Longitude0 >= c1)

FALSE if otherwise.

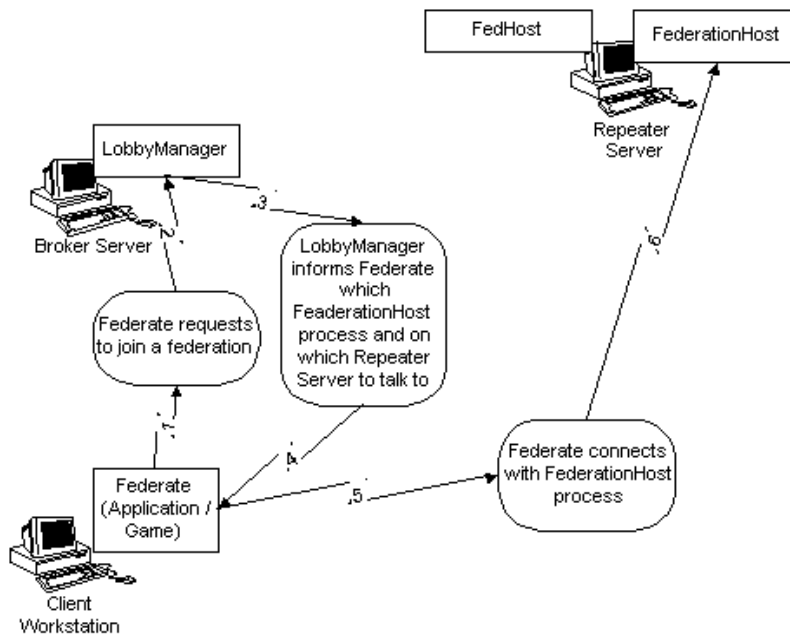
(Altitude0, Latitude0, Longitude0) is a "CybernetBaseEntity" that belongs to the potentially receiving federate.



FedHost is an executable running on a repeater server. Each repeater server will have one or more FedHost executables launched. When LobbyManager needs to launch a new FederationHost process on a repeater server, it connects to FedHost on that repeater server using TCP/IP, and sends the request. FedHost will launch the requested new FederationHost process and return the status of the new process to LobbyManager, so that a client application such as Openskies can connect with the FederationHost process (please see the diagrams below). *The code for each FederationHost process is inside the FedHost executable.*



**Figure 1 Launching a new Federation**

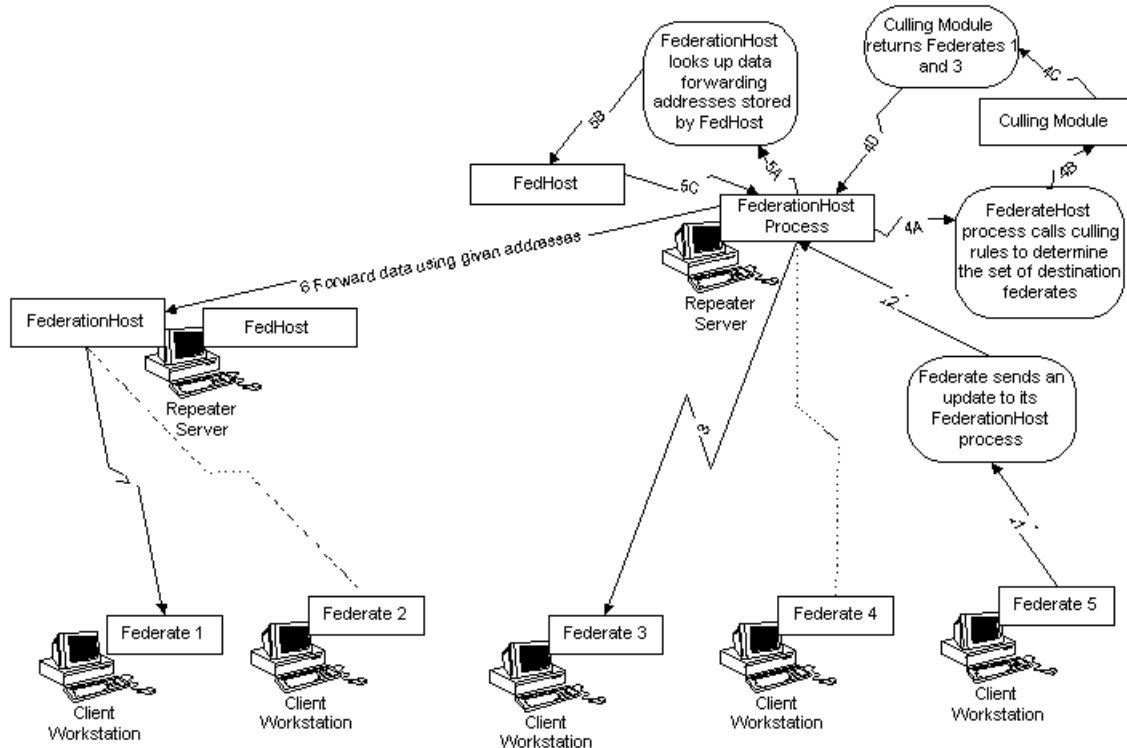


**Figure 2 Join a Federation**

Run-time configuration of FedHost is stored in an ASCII file named FedHost.ini. It must be stored in the same directory as FedHost itself. It contains 3 sections. The first section

contains the NIC address and port number it uses. It must match what is listed in the LobbyManager.ini file. The second section specifies socket options. The last section is for standard culling rules in the Cybernet FOM library. An actual copy of a FedHost.ini file can be found in Appendix B.

Besides acting as a process launcher, FedHost will also establish IP multicast traffic forwarding route from one repeater server to another when an IP multicast connection is not available amongst some Repeater Servers. It will select routes of least travel for all forwarded data.



**Figure 3. Data Traffic Flow**

By relaying data through repeater servers, we can reduce network traffic through the Internet backbone considerably. If the topology for repeater server connectivity is such that a datagram can reach every node with a single pass on every connection segment, the amount of data sent across the Internet is simply proportional to the number of clients. Consider the example of a flight simulator. An aircraft needs to transmit its altitude, latitude, longitude in doubles, heading, pitch, bank in floats, and ID in 32 bit integer for a total of 76 bytes at 10 Hz, i.e., 760 bytes/sec, in UDP datagrams. This number can be further reduced by not sending the high 32 word of each double every frame, for example. So the number becomes 64 bytes at 10 Hz, i.e., 640 bytes/sec. If there are 100 players, the amount of data sent across the Internet backbone is  $100 * 640 \text{ bytes/sec} = 64 \text{ kilobytes/sec}$ . For 500 users, it is about 320 KB/sec. And so on. Local traffic at each repeater server would still be  $n(=100)\text{-squared}$  times 640 bytes/sec with the absence of culling.

Due to the limited bandwidth that is available to each end user, we must rely on culling to significantly reduce the amount of data sent to each user. Assume that the user is using a 56kb/sec modem, the number of aircraft it can handle is about 10-15. Since we must leave room for infrequently transmitted data, plus things such as missiles and other projectiles, we should limit the number of planes to 10.

### ***Client/Federate***

The Client program is, in a nutshell, your application program. Clients are also known as Federates. The CybernetRTI currently supports federates residing on a Win32 machine, compiled and linked against the CybernetRTI Client-side API to communicate out to the network

### ***Host Federate***

A federation (*network game*) is created upon federate request to the LobbyManager. The lobbymanager responds to this request by commanding its FedHosts to create a federation that this and other federates can join. Typical player federates can be given Host Federate capability or not, depending on your business and/or game paradigms (Do you want players to be able to start up their own multiplayer games?). If not, the ability to host federations is typically left to another type of client called a game server.

### ***Game Server Federate***

The Game Server(s) is an optional Windows based system(s). It is a federate that, in the network sense, functions the same as just another player client. The game server is also similar to the player client in that the game developer must implement it. Game servers are implemented to control aspects of the game not implemented in the clients. Typically these are things that are common to the game universe or aspects of play that you want to isolate from the players for security reasons (e.g. book keeping for score, weapons status, etc.)

Game developers can implement culling rules to route data differently for game servers than for regular client federates. It is easy to implement culling rules that disallow direct client-to-client communication, forcing clients to communicate only with game servers.

An added benefit to using the Openskies SDK to implement a Game Servers approach is that they can be added on the fly. The LobbyManager and FedHost(s) can handle several Game Servers (games) at once. This is especially advantageous if you implement your game servers such that the number of players scales with the number of game servers.

Typically the first game server can serve as the Host Federate. However, if there are no Game Servers, the first client can initialize the federation (network game) at startup/entry.

### *Database – Authentication server*

The Database (also known as Authentication server) is usually a Linux based system that runs a database application such as MySQL or Oracle. You may chose to not have any databases or even multiple databases. The communications to and from the module that handles passing information to and from the database(s) can be either encrypted via secure socket layer (SSL) or not.

Purposes of the database include:

- Login (user and password) restriction/verification
- Demographic User information
- Logged into system time (useful for time based billing)
- Number and time of login/logout (useful for billing and other stats)
- Game or application specific scoring/information
- Game Server statistics
- FedHost statistics
- Client defined information

Any statistic that you want to keep track of can be placed into the database(s) for retrieval and analysis. This retrieval and analysis can be done via .html or .shtml pages from within an internet browser.

The fields that a database is made up are determined by the client's application. All that we need to know is the variable name(s) and type(s), when to query or update the database for each, and possibly a few other variables.