



CYBERNET SYSTEMS CORPORATION
727 Airport Boulevard
Ann Arbor, MI 48108

27 November, 2002, Updated 8 February, 2003

Using OpenSkies to Implement Multi-Level Security in a Mixed HLA/DIS Environment

Douglas Haanpaa, dhaanpaa@cybernet.com
Charles Jacobus, chuck@cybernet.com

Background

OpenSkies™ implements the functionality of the IEEE 1516 HLA API.¹ It is a full RTI.² Clients or federates are able to subscribe to objects and interactions as well as publish them. Like any fully implemented RTI, the OpenSkies™ RTI is not restricted to any specific FOM,³ it supports the use of any FOM required by a distributed federation of simulations. In fact, the OpenSkies object definition architecture has two layers, the “.fed” file represents the HLA specified abstract layer, and another file, named “CybernetRTI.ini”, maps attribute sets to objects in client applications. The CybernetRTI.ini level can allow the simulation designer to translate information from different FOMs.

The unique aspect of OpenSkies is that it implements simulation client-to-client communications through intelligent routing processes within an application we call the Fedhost. As described in depth in *Cybernet Real Time Intelligent Routing Technology*,⁴ *Openskies Massive Multiplayer Online Game SDK*⁵ and, *Openskies Network Architecture*⁶ each client simulation enters the federation by notifying a process called the Lobby Manager. The Lobby Manager keeps account of each Fedhost and how loaded with connect clients it is. Based on a load-balancing algorithm, the Lobby Manager returns to the client the access information needed for that client to connect and authenticate to the least busy Fedhost.⁷

The simulation federation or simulation world (and potentially more than one federation running in parallel) is implemented by the network of Fedhost processes. Each Fedhost

¹ HLA stands for High Level Architecture; API is Applications Programmer’s Interface, basically a library; IEEE is the Institute for Electronic and Electrical Engineers.

² RTI stands for Real Time Infrastructure, or the portion of HLA that simulation application call to effect object sharing and message passing.

³ Federation Object Model – the model that defines all interactions and objects and their attributes for a simulation federation (i.e. the collection of simulation clients that are to mutually interact).

⁴ <http://www.openskies.net/files/Introduction.pdf>

⁵ http://www.openskies.net/files/Openskies_MMPOG.pdf; SDK is Software Development Kit – often a synonym with API.

⁶ http://www.openskies.net/files/Openskies_Network_Architecture.pdf

⁷ An OpenSkies Lobby Manager can be redundant so that a back-up manager can take over if a primary manager fails.

is attached to its set of clients (brokered by the Lobby Manager process), and is also in communications with all of its peer Fedhosts. All communications from client to the federation is through a single connection from the client to its assigned Fedhost. When client messages or object updates must flow to other clients within the cloud managed by a single Fedhost, the Fedhost distributes these messages. If clients attached to a peer Fedhost must be updated, the update traffic is routed from the originating Fedhost to its peers through Internet multicast protocol.⁸

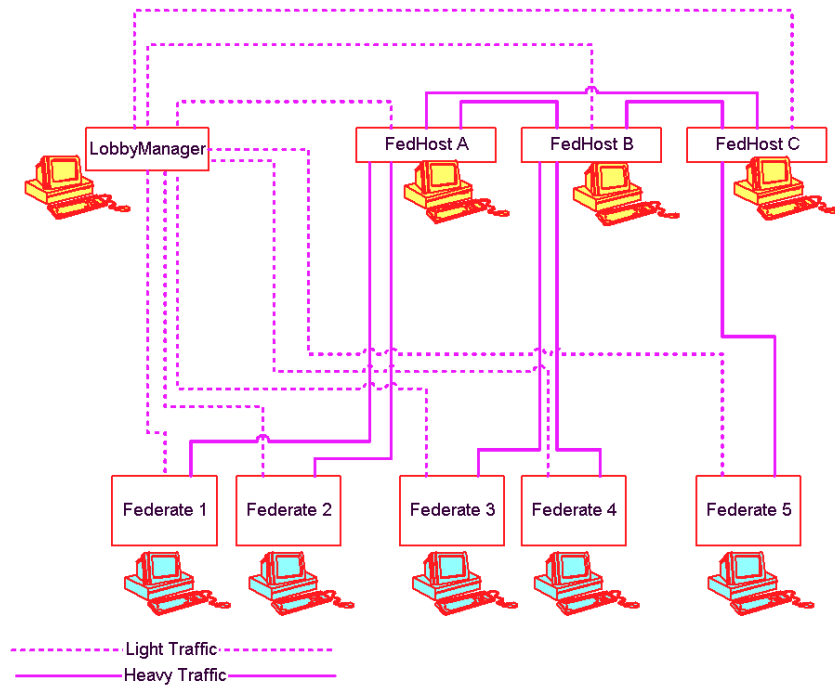


Figure 1. Clients connect to Lobby Manager and get redirected to the appropriate Fedhost

A Fedhost can optionally forward messages from client to Fedhost, to peer Fedhost, and then down to destination client or alternatively can move messages from client to Fedhost and directly to destination client. The simulation operator can use either method and chooses based on the physical architecture of his Fedhost hardware and network interconnectivity. If the Fedhost is housed on a single computer both methods are equivalent. If the Fedhosts are on multiple computers that are located close to each other (perhaps on the same LAN), allowing any Fedhost to

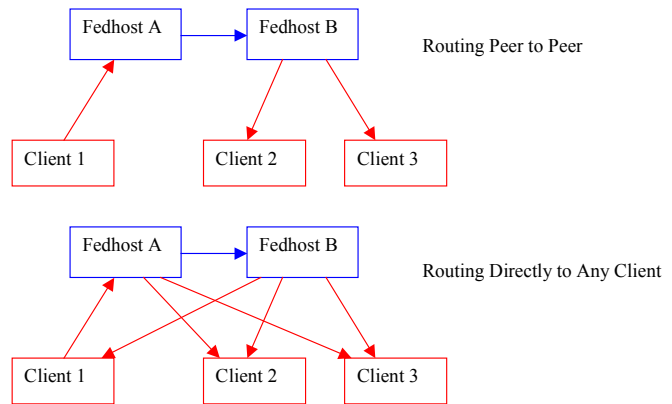


Figure 2. Alternative routing strategies supported by OpenSkies

⁸ OpenSkies libraries also implement a more traditional RTI that does not use Fedhost processes. This traditional version has an identical HLA RTI and allows developers to test code and object interactions with lower network complexity. As discussed later, in this version, network traffic is NxN where N is the number of simulated objects/clients whereas the traffic is order N or less with the version that routes through the Fedhost processes.

access any client is the most efficient communication method. However, when the Fedhosts are distributed widely over the Internet backbone or over a radio link it may be more efficient to route messaging from peer Fedhost to peer Fedhost before dropping down into the client cloud.

Culling

The original purpose for implementing the Fedhost routing system into OpenSkies was so that the normal NxN message traffic in a DIS or DMSO HLA distributed simulation⁹ is reduced to order N as in server centric simulation systems like those usually used to implement commercial gaming. This reduction in traffic is required to operate simulations with very many players or objects (OpenSkies was designed to handle 100,000+ objects – the architecture actually has no strict upper limit assuming adequate communications capacity and Fedhost memory).

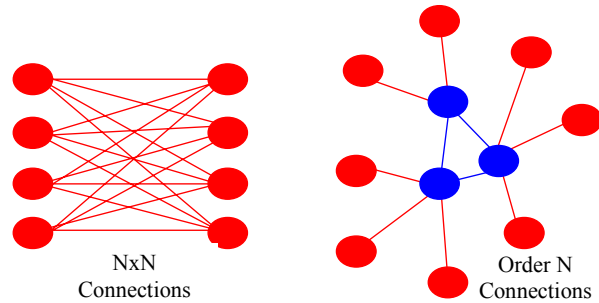


Figure 3. OpenSkies architecture reduces connections

To further reduce traffic, we assumed that in most simulations it would be possible to control messaging quality of service through the knowledge encapsulated in the object update or interactions (i.e. in the update messages). Since every status and position change associated with a simulated object in an OpenSkies HLA federation passes through the Fedhost to which the simulation client is connected, it is possible for the Fedhost to know the last set of attribute values for the objects in clients directly attached to it. Furthermore, since the Fedhost communicates attribute changes over multicast to each other, each Fedhost has a complete picture of the objects published by all clients connected to any Fedhost that implements the federation.¹⁰

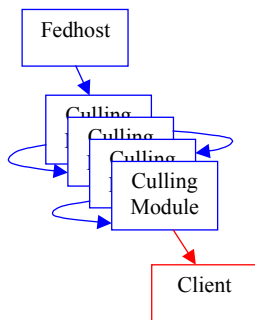


Figure 4.

This means that if we put a “filter” between each client and its Fedhost that can access the Fedhost’s object attribute database, we can control which messages pass to and from the client to the rest of simulation dynamically. We call these filter procedures, that run as Fedhost processes, culling rules, because generally they reduce message traffic to clients that do not need as high a quality of service.

The notion is when a client sends information to the Fedhost, this information is not distributed to every other client on the network. This avoids creating exponential growth in network traffic and rendering linear scaling impossible. When the

Fedhost relays a packet corresponding to an attribute set, it first subjects the packet to a list of culling rules that decide who sees what. **These culling rules can be arbitrarily**

⁹ DIS is the old DoD distributed simulation standard and stands for Distributed Interactive Simulation; DMSO is the Defense Modeling and Simulation Office – see www.dmsomil.

¹⁰ Because every Fedhost has a copy of the status of all objects in the simulation, the system is fault tolerant to Fedhost failure. If a Fedhost fails, its clients re-broker through the Lobby Manager to functional Fedhosts. Since each Fedhost remembers the last object state, the re-connect client proceeds without any effect. Adding new Fedhosts is also transparent. The new Fedhost notifies its Lobby Manager of its availability and the Lobby Manager begins brokering new players into the under utilized Fedhost.

simple or complex and are customized for a particular application. For many simulation applications, we might adopted a geographic culling paradigm. Culling, in this case, is based on a client's location in the game/simulation space. The culling rule is able to extract this information from the packets themselves. In this case, the culling module uses the LatitudeF and LongitudeF values from the Fast data packet transmitted by the Multiclient and described above. The culling module maintains an internal structure that is divided into boxes (culling bins) arranged in a two-dimensional array. The culling module also maintains a local state structure for each Federate entity, which it then places in one of the grid-bins based on the packet

information (LatitudeF and LongitudeF). With culling active, each client is sent updates for those objects in the same bin and in adjacent bins out to a distance of N from the client, thereby limiting the amount of necessary traffic.

As a practical example, if two simulations are engaged in an aerial engagement, they need interaction rates that are as close to real time and lag-free as possible. On the other hand, if an AWACS display shows a target several hundred miles away, a slower rate of position update may be acceptable. If we are simulating views from the vision blocks of an M1 tank, we only need messages that come from nearby vehicles that fall inside the viewing angle of the blocks – i.e. not messages from objects behind the tank.

In very large world models, like that for a theater operation, many of the participants (i.e. objects in the theater) may never “see” each other because they are separated by very large distances.

The culling rule system allows us to define these mutual messaging relationships by software that is operated by each Fedhost in line with the message traffic to and from each client simulation. This type of applications-specific routing is the subject of a patent pending by Cybernet. The effect of culling is that message traffic grows in ever-larger simulations no faster than linearly – and often even slower. This means that OpenSkies federations can become very large (essentially without any practical limit).

The OpenSkies routing system reduces latency or lag by reserving bandwidth for high priority, near-in interactions at the expense of slower (and possibly grainier) long-range interactions. This is effectively quality of service control based on simulation content requirements.

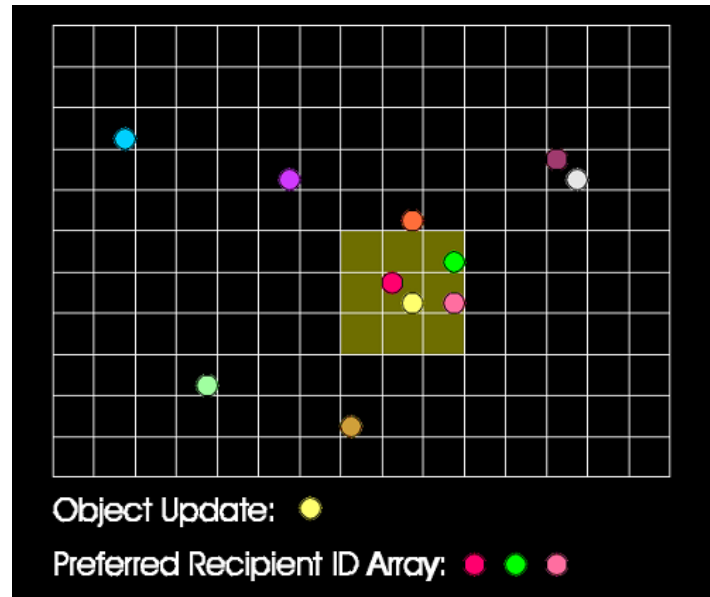


Figure 5: Culling with a Distance of One

Implementing DIS Interoperability in OpenSkies

Normally culling processes operate within the Fedhost and optimize Fedhost message routing. However, this mechanism also allows total intercept and rewrite of messages to and from the client. In HLA interactions (and their constituent attributes) and objects (and their constituent attributes) are defined in the Federation Object Model. This defines and controls a complex shared traffic between clients as they update their owned objects and share these updates with their distributed simulation partners.

In DIS the update model is simpler and all objects used in DoD simulations have been permanently defined. The foundation of DIS is a standard set of messages and rules, called Protocol Data Units (PDUs), used for sending and receiving information between clients and the network (through UDP broadcast). The most common message is the Entity State PDU, which represents all of the state information about a simulated entity that another simulator needs to know. The Entity State PDU contains data about an entity's position and velocity. DIS is typically able to limit the amount of data an average simulator transmits to approximately 250 bytes per second.

DIS does not use any central server. It is strictly a peer-to-peer architecture, in which all data is transmitted to all simulators where it can be rejected or accepted depending on the receivers' needs. DIS implementations are lighter weight than most HLA implementations. One of the largest DIS simulations executed to date was part of DARPA's Warbreaker program and had 5,400 simulated entities interacting in a single DIS virtual world. OpenSkies HLA is designed to handle over two orders of magnitude more interacting entities.

To interact between an HLA simulation federation and DIS simulators, a filter is constructed that maps the values in each DIS PDU type into the necessary object attribute updates defined in the HLA FOM. The simplest mapping is one where each PDU is associated with an object type in the FOM. Then the message mapping is one to one. Each PDU update translates into the corresponding HLA object update. Because OpenSkies allows code modules to implement the mapping, more complex mappings where the HLA FOM object definitions do not exactly map to PDUs (and values in the PDU) are possible also.

Implementing Multi-Level Security in OpenSkies

The concept of trusted systems and multilevel security are related. In a trusted system, a specification defines roles and access rights for each process or user to each data object within a system. A software module that can be formally validated and is independent of the normal operating code that executes system functions enforces this access rights model. This separation of system action from access rights enforcement is important to the concept.

Multilevel security suggests that each user of a system will be assigned an access rights level, and this level will enable or deny access to all or parts of data object that are secured to higher access levels. A trusted multilevel security system defines the access rights of users and security levels of objects independently from system operation.

How do we implement trusted multilevel security in OpenSkies? First, consider that pure data security can be inserted into any system by proper security gateway functions. For

instance, if we want a client to send encrypted message streams to and from an OpenSkies Fedhost, the Fedhost can be placed over¹¹ or behind a VPN server¹² and the client can operate over a VPN client or from behind another VPN server. Industry standard encryption is IPsec or triple 168-bit DES. Better enciphers can be used to make decrypting more difficult.

VPN data security does not control access by security level. For instance, the Air Force might want to fly high fidelity B2 simulators with or against F16 simulators in use in less secure Guard bases. In this case some of the data from the B2 simulator might be too sensitive to broadcast to the Guard. Ideally we would set Guard users to a lower security level that would stop some traffic from the B2 simulator and mask the values of other traffic (to disguise the precise performance of the B2).

Masking (i.e. re-writing) and blocking message traffic is exactly what the culling rule system described in the first section of this paper describes. Normally, a dynamic property like distance from one object to another controls this culling. However, we have implemented radio chat APIs that use the radio channel attribute to cull any traffic to channels other than the one selected (i.e. if channels from sender(s) to receiver(s) do not match, the culling rules allow no traffic to be routed by the Fedhosts).

Similarly we have implemented culling rules that pass traffic from one object to another only if the sending object is at the same security level or lower as the receiving object. When the sending object is higher, the message can be completely denied or can be rewritten according to a programmed masking rule. For instance, a B2 can be shown as one radar cross section to a high security level user but as an alternative one to a lower security level user.

The security based culling module is driven by an initialization file that closely mimics the FOM definition file so that security access can be controlled down to the attribute level.

OpenSkies System Performance

We have measured OpenSkies performance in terms of throughput. This is primarily due to the technology's intended use to support Massive Multiplayer game-play (i.e. large scale distributed simulation). All of our tests indicated that the number of participants possible in a common Federation scale linearly with the number of servers (Fedhosts) used. The capability of a Fedhosts cluster is summarized in the following table:

¹¹ Cybernet offers a Linux-based integrated server suite called NetMAX (www.netmax.com) that includes VPN, Virtual Private Networking. Processes like the OpenSkies Fedhost operating as an application on the VPN server can be set to only accept VPN connections to other Fedhosts and from clients. Clients can be secured by loading any IPsec VPN client into the network stack.

¹² Other VPN gateways like those from Cisco can be inserted in between clients or Fedhosts and the open Internet (or between secure areas). The NetMAX VPN server can be installed on a computer with two network ports to be used as an off-board VPN gateway also.

Server	IP Address	Max Clients		CPU Utilization		
		Unsaturated	Saturated	Fedhost	Idle	Traffic
Golf	123.123.123.1	432	496	63%	5%	13.6Mb/s
Quebec	123.123.123.2	496	576	62%	2%	15.5Mb/s
Delta	123.123.123.3	496	576	62%	2%	15.5Mb/s
Tango	123.123.123.4	496	576	64%	0%	12.9Mb/s
Uniform	123.123.123.5	496	576	44%	13%	15.5Mb/s

In the tests summarized, individual clients were simulated in an application called a Multiclient that ran approximately 144 simulated federates or clients per test workstation (the exact number was based on workstation performance). Each of these simulated federates was generating about 94 bytes-per-second and receiving 8 times that number (from its spatially nearest neighbors emulating a close combat application). The unsaturated numbers above show how many federates were able to connect to each Fedhost without any detrimental effect on the input/output rates experienced at the clients/Federates (the hardware running the Fedhost were 500 Mhz Pentiums). As shown, the number of federates reached a total of about 2400 when using 5 Fedhosts were interconnected. Some additional tests not reported here show that the number of federates per Fedhost is proportional to processor speed and not network adapter speed (although network router or hub transfer rate can restrict message flow overall). Therefore, with modern 3 Ghz Pentiums we could safely project that we can handle about 3000 client connects per Fedhost.

We have investigated latency. The latency between federates in a five Fedhost network was measured at less than 5 milliseconds (and has been as low as 1 msec) on a local area network. This latency does not grow with the number of Fedhosts if each Fedhost is allowed to forward directly to any client. The latency incurred by our servers is negligible compared to that incurred by the Internet (which is typically at least 150 msec).

DoD Certification and Accreditation Protection Levels

OpenSkies™ has implemented authentication of users through encryption key passing. The OpenSkies™ SDK was designed to allow developers (in-house or licensees) to create their own custom authentication schemes. Our system takes care of the key passing and allows the developer to concentrate on generating and verifying keys.

The authentication scheme authenticates the client at two points, one when connecting to the Lobby Manager server, and another when connecting to the Fedhost. Authentication information is also transported from Lobby Manager to Fedhost so that the key can be verified. We have also implemented mechanism by which the clients first get a key over an SSL (Secure Socket Layer) connection from a secure database. This key can be time-stamped so that it is only valid for a short period of time.

This architecture can be modified to provide the Fedhost public key to federates for Federate→Fedhost traffic and then to make public keys available to the Fedhost's culling/routing module facilitate Fedhost→Federate traffic.

Alternatively, OpenSkies can be used to propagate public keys to the individual clients/federates, allowing them to encrypt the sensitive parts of the data before

transmission. This has the benefit of not burdening the Fedhost, which is primarily busy routing traffic. This approach would require that a portion of each packet be encrypted differently for interpretation by the Fedhost to enable Culling.

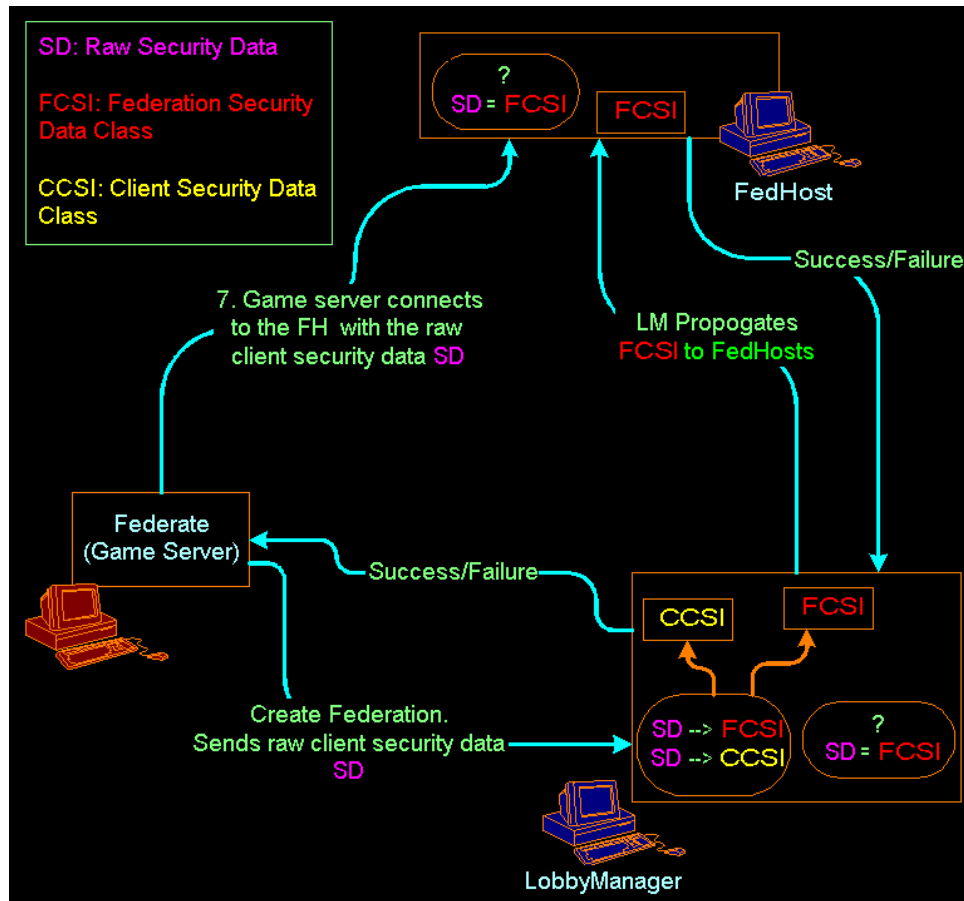


Figure 6: Authentication Key Transport System

While the security access aspect of the architecture described has not been certified in anyway, Cybernet is currently in the process of qualifying OpenSkies HLA as fully IEEE 1516 compliant. We expect this to be done no later than the end of June 2003.