

FioranoMQ™ – Messaging Server

# Concepts Guide

Version 8.01



**Fiorano Software, Inc.**

Fiorano Software, Inc.  
718, University Avenue, Suite 212,  
Los Gatos, California 95032 U.S.A.  
Tel 1.408.354.3210  
e-mail: [info@fiorano.com](mailto:info@fiorano.com)

(C) 1999-2005 Fiorano Software, Inc.  
All Rights Reserved



# Contents

---

<b>Contents</b> .....	<b>3</b>
<b>Chapter 1: Introduction to FioranoMQ</b> .....	<b>5</b>
FioranoMQ - Fastest Messaging Server .....	6
Choice of Topologies and Implementations .....	7
Security in FioranoMQ .....	8
Key features of FioranoMQ .....	9
Design Goals .....	14
JMS Communication Models .....	16
Publish/Subscribe Interactions .....	16
Request/Reply Interactions .....	18
FioranoMQ System Architecture .....	20
Hub and Spoke Topology .....	20
FioranoMQ Internal Architecture Details .....	21
Thread Management in FioranoMQ .....	22
Memory Management and Publisher Throttling in the FioranoMQ Server .....	24
Typical FioranoMQ Server Setup .....	24
FioranoMQ Server-to-Server Communication and Clustering Architecture .....	25
Server-to-Server Communication Mechanism - Repeaters .....	26
Subscription Mode and Choice of Selectors .....	28
Load Balancing, Failover Protection and High Availability .....	29
Load-balancing Architecture .....	29
Balancing Client Connections .....	30
Multiple Dispatcher Service .....	31
Automatic Failover Protection .....	31
High Availability .....	31
LPC mode for Clients in FioranoMQ Server .....	32
Scalability .....	32



# Introduction to FioranoMQ

The explosion of the Internet and the rapid proliferation of computer networks have created the need for a comprehensive software platform that specifically addresses the requirements of event-based network-centric applications. FioranoMQ is a software platform, developed to meet these requirements.

Today's enterprise networks typically deploy hundreds of applications from different vendors. There is little standardization of communication protocols between individual distributed systems and exchanging information between applications from different vendors is surprisingly difficult. The lack of a standard platform for network-centric computing increases the cost and complexity of developing and deploying distributed systems. FioranoMQ was designed to alleviate these and other problems.

FioranoMQ has been designed over JMS (Java Message Service) standard. JMS provides a common technique for Java programs to create, send, receive and read messages in an enterprise messaging system.

Enterprise messaging products (sometimes called, Message Oriented Middleware products) are becoming an essential component for integrating intra-company operations. They allow separate business components to be combined into a reliable, yet flexible system. In addition to the traditional Message Oriented Middleware (MOM) vendors, enterprise messaging products are provided by several database vendors and a number of Internet related companies. Java language clients and Java language middle tier services must be capable of using these messaging systems. JMS provides a common technique for Java language programs to access these systems.

JMS is a set of interfaces and associated semantics that define how a JMS client accesses the facilities of an enterprise messaging product. A JMS provider is an entity that implements the JMS specifications and the APIs for providing an enterprise messaging solution. FioranoMQ is such a JMS provider that has support for all the JMS specifications. In addition, it provides numerous proprietary features such as scalable connection management, file based data store and load balancing that make it one of the fastest, most scalable JMS servers currently available.

The JMS messaging model consists of two basic domains PTP (point to point) and Pubsub (publish/subscribe).

Point-to-point (PTP) products are built around the concept of message queues. Each message is addressed to a specific queue; clients extract messages from the queue(s) established to hold their messages. Publish and subscribe (Pub/Sub) clients address messages to some node in a content hierarchy. Publishers and subscribers are generally anonymous and may dynamically publish or subscribe to the content hierarchy. The system takes care of distributing the messages arriving from the multiple publishers of a node to its multiple subscribers.

FioranoMQ implements both the messaging domains and also the unified domain concept that has been introduced in JMS 1.1.

## FioranoMQ - Fastest Messaging Server

FioranoMQ is a JMS Messaging platform that supports the PTP, pubsub and unified domains. FioranoMQ is a communication platform that dramatically reduces the development time for network applications. FioranoMQ incorporates a 100% pure Java implementation of JMS (the Java Message Service API released by Sun Microsystems, Inc.), which isolates developers from the details of low-level network programming and provides a standards-based method to access distributed-system services.

JMS supports location independent application development through its Publish/Subscribe and PTP (Point to Point) communication APIs. It further includes support for critical network services such as transactions and guaranteed message delivery. This considerably eases the process of developing and deploying Internet, Intranet and Extranet applications. JMS APIs and services allow developers to add value to an application by focusing more on its core business logic, rather than its low-level infrastructure.

FioranoMQ implements all of the JMS Publish/Subscribe and Point-to-Point (PTP) APIs, together with support for security, massive scalability. This allows thousands of concurrent connections to the server, remote administration, guaranteed message delivery, scalability and language interoperability. The security implementation includes integrated SSL support (with digital certificates) for encrypted data transfers with a full implementation of Java REALMS. This allows an external administrator to control access to the system resources. An integrated C++ runtime library, based on JNI-bindings to the standard Java interfaces, allows C++ and Java programs to communicate seamlessly. Automatic store-and-forward across multiple servers ensures guaranteed message delivery across faulty networks.

# Choice of Topologies and Implementations

FioranoMQ is implemented as a “message broker” (based on a hub-and-spoke topology). The choice of a particular topology depends on the particular problem being solved.

The Hub-and-spoke topology is useful where information has to be exchanged between a large number of clients and a central hub or server is required for catering to different client requests. In a messaging scenario, this central hub or server is often referred to as the message broker.

The message-broker architecture is typically used in situations where communication over the Internet, guaranteed message delivery and remote administration are important requirements.

## Use Case

A simple real world use case of the Hub and Spoke model would be a banking scenario. There are many small branches (or ATM machines) of a bank and a customer can access and use his/her account from any of these branches. In case of a point-to-point topology, every bank branch will have to maintain a connection with every other branch so that the information about every account is available in each branch. This method is definitely not adequate as there can be any number of branches and customers.

In this case, the Hub-and-Spoke model solves the problem. All customer information is kept at a centralized repository, which can be accessed by a central hub or server. All the bank branches need to connect only with this server and obtain real time information about the customer account.

This reduces the number of connections that have to be maintained in the system and also provides a centralized control over all the banking transactions. Another important advantage of this model is the use of failover support. A central hub can have one or more backups in case it fails due to some reason.

## Security in FioranoMQ

The secure version of the FioranoMQ server includes integrated SSL support for encrypted data transfers. It supports 40-bit (export version) and 128-bit (U.S. version) encryption of all data transfers. Authentication is provided by digital certificates using server and client-side callbacks. In addition, Java REALMS-based security that is built on Java 2 standards, allows the system administrator to restrict access to the server resources. Access is restricted through an external administration tool.

The FioranoMQ server runtime environment is lightweight and robust. Each FioranoMQ application maintains a single persistent connection with the server. Messages are sent to different Destinations (Topics or Queues in JMS parlance), which are all multiplexed over a single connection. The runtime library component embedded within each application is highly compact, allowing JMS APIs to be used effectively in Java applets and applications running over the Internet.

## Key features of FioranoMQ

FioranoMQ offers a unique set of advantages to the developers. Besides reducing the development time for applications, FioranoMQ imparts significant network functionality to applications, ultimately benefiting the end-users. The following section describes the key features of FioranoMQ:

### Internet Ready

FioranoMQ software is implemented in 100% pure Java, both on the server and client (i.e. runtime library) sides. Leveraging Java on the server-side offers cross-platform application support, and, in the case of downloaded applets, removes the need to pre-provision client nodes with the application code. This greatly reduces client administration and overhead, while increasing application accessibility across enterprise Intranets and the Internet.

In addition to the 100% pure Java implementation of the default connection manager, FioranoMQ software incorporates a special Scalable Connection Management (SCM) module. This allows a single FioranoMQ server to support several thousand concurrent client connections on a variety of operating platforms (including Windows NT, Solaris and other Unix platforms). FioranoMQ allows the user to choose between the default Java connection management and Scalable Connection Management, as described in detail below.

### Event-based Communication System

To more closely model how business systems function, the JMS API incorporates complete support for the Publish/Subscribe communications model. The Publish/Subscribe communications model supports event-based computing. Event-based computing using publish/subscribe, more closely mirrors how business-events flow in the real world. Events are disseminated automatically to relevant subscribers as the event happens, without requiring each subscriber to continuously poll a centralized repository of information.

FioranoMQ's implementation of JMS is based on highly efficient "true push" technology, which does not waste any network bandwidth or computational cycles. As such, it is ideal for the deployment of Internet-based applications.

### Location Transparency

FioranoMQ applications communicate with each other over Destinations (Topics or Queues). Sending applications publish data to Destinations rather than to a fixed physical location or network address; receiving applications subscribe to Destinations of interest to receive data on those Destinations. The FioranoMQ Server then dynamically routes data from the sender to the receiver, eliminating the need for applications to locate clients or determine network addresses. Because FioranoMQ applications

are location independent, they can be easily ported between machines and across different platforms. For example, one can move a server application from one machine to another without disrupting the existing clients. Clients can be added to (or removed from) the FioranoMQ network dynamically, without interrupting other clients or servers.

## **Ease of Change and Application Modification**

Distributed systems (such as FioranoMQ) based on the publish/subscribe paradigm are very easy to change and modify. This is because event dependencies are resolved through runtime event names, rather than by compilers, linkers or operating systems. Publishers and subscribers of FioranoMQ messages operate independently of and asynchronously from each other, and may change without recompilation. All communication between applications is through messages that are routed through the FioranoMQ daemon on each machine. This allows new FioranoMQ applications to be added to the distributed system later, without recompiling, relinking or even stopping and restarting any current applications.

## **Reduced Complexity**

Publish/Subscribe and message queuing greatly reduce complexity in distributed system design and implementation. Events are published to Topics (or Queues) as they happen. Business processes that depend on these events, automatically receive them as they occur. By allowing change notifications to be easily propagated through an enterprise, FioranoMQ software reduces the complexity in distributed system implementation.

## **Scalability, Extensibility and Performance**

The FioranoMQ platform combines all the benefits of standards-based JMS APIs with high performance and massive scalability. The FioranoMQ server architecture is highly scalable and extensible by design.

FioranoMQ includes a special native Scalable Connection Management (SCM) module that allows thousands of concurrent clients to connect to a single FioranoMQ server, on Windows NT as well as on Solaris and other Unix platforms. FioranoMQ gives users the choice of using the default server-side connection management, or the native SCM module, which allows an unlimited number of concurrent clients to connect to the server (while keeping server resources constant). The default server-side connection management involves a new thread on the server for each additional client connecting to the server, a model used by default on all Java server implementations.

In addition to the scalability enhancements on a single server, multiple FioranoMQ servers can be connected together to support large numbers of concurrent client connections. In addition, the FioranoMQ environment includes message repeaters to efficiently route data between multiple FioranoMQ servers, with facilities to filter the data using message selectors.

These servers could be placed locally or across geographically distributed locations on a WAN

## **Robust Features for Distributed Application Development**

The use of publish/subscribe and queuing techniques allow FioranoMQ distributed application systems to scale smoothly as you add new component processes. FioranoMQ-based distributed Java systems are easier to maintain and have longer useful lifetimes than traditional networked application systems.

## **Standards-based Pub/Sub and PTP (Queuing) Communication APIs**

FioranoMQ software supports both the Publish/Subscribe and PTP communication models, as required by the JMS specification. These communications can operate with either synchronous or asynchronous timing. This depends on whether the developer wants a thread to block while waiting for information or set a listener, which invokes a callback whenever any information is available.

## **Reliable and Guaranteed Message Delivery**

FioranoMQ supports both reliable and guaranteed delivery of messages, as required by the JMS API. Guaranteed delivery is specified using the Durable Subscription API, as specified by JMS. FioranoMQ software ensures that all the persistent messages are delivered once and only once to all the subscribers marked durable, even in the presence of intermittent or long term network failures. Once and only once message delivery is also guaranteed across multiple FioranoMQ servers over a WAN, allowing the deployment of reliable distributed systems across geographically dispersed locations. FioranoMQ supports both guaranteed as well as reliable delivery.

With reliable (at-most-once) delivery semantics, FioranoMQ delivers messages to all the connected clients, assuming there are no network or application failures. Messages sent with reliable delivery are not logged to persistent storage and so may be lost on occasion. For several applications, reliable delivery offers the most appropriate quality of service. For example, if an application drops a single message in a web-based chat application, users might not worry, since another message will arrive in a few seconds.

## Transactions

Several applications often require grouping several individual tasks into a single logical unit of work, called a Transaction. FioranoMQ supports basic transactions (with commit/rollback of messages) as specified in the JMS API as well as distributed transactions (specified by the XA classes in the JMS specification).

## Language Interoperability

FioranoMQ messaging services can be accessed by an available C++ runtime library to support access to the legacy systems. The C++ runtime library is a thin wrapper, which uses JNI to call the corresponding Java APIs. This allows Java and C++ programs to exchange and share data seamlessly.

## Open Architecture

FioranoMQ software is designed to support Integration with existing directory servers (based on JNDI and LDAP) and with Java application servers. Several application server vendors have already integrated messaging services (based on FioranoMQ) with their core platforms.

## Comprehensive Security Features

FioranoMQ software incorporates a comprehensive security system that allows organizations to easily implement their security policies with respect to identification and authentication, authorization and access control, and privacy/integrity protection using encryption.

The secure version of FioranoMQ Server includes 40-bit (export version) and 128-bit (U.S. version) encryption, using SSL sockets. Authentication is implemented through client and server-side digital certificates. Client digital certificates may be passed to a customized callback function on the server, allowing developers to implement a layer of access control, based on digital certificates. FioranoMQ also implements the normal Username/Password authentication API, as specified by JMS.

FioranoMQ implements access control based Java 2 Security APIs. For instance, in a FioranoMQ network it is possible to do the following:

- Restrict applications from creating publishers, subscribers and/or durable subscriptions on a given Topic or Queue
- Allow certain applications to publish information only on a fixed set of Topics/Queues,
- Allow the FioranoMQ Administrator to restrict the neighbors of MQ Server by setting incoming and outgoing permissions for them.

The FioranoMQ security system is comprehensive and highly customizable, fitting the needs of any enterprise.

## Remote Management

FioranoMQ includes a comprehensive administration API and external tools for creating and configuring ConnectionFactories, Topics, Queues, Users, ACLs(Access Control Lists) and user permissions. The administration API together with available external tools allow to monitor the server at any time from any location on the web.

## True Push Technology

Traditional applications search for information by polling, which overloads server machines and wastes machine cycles in general. FioranoMQ implements standards-based JMS APIs through true-push technology. This allows applications to wait efficiently for events to occur and take actions on those events (through installed callback functions referred to as listeners).

The push technology was created to alleviate problems like information overload and low bandwidth that are associated with the usual pull technology. The push technology is a data distribution technology, in which selected data is automatically delivered to the user's computer at prescribed intervals or based on some event that occurs. The push technology can be used to make information more accessible for the user. By applying the push technology, FioranoMQ designs and implements user-friendly and effective information delivery systems.

## Design Goals

The specific design goals of FioranoMQ include portability, standards-based communication services, web-integration, comprehensive security, firewall tunneling, robustness in the face of network problems as well as performance and scalability.

### Portability

Most enterprises today have a variety of computer systems, ranging from thin-clients and Windows desktop PC's to high-end UNIX servers and mainframe systems for reliable backend processing. Portability and ease of communication between different operating environments was thus a predominant concern for FioranoMQ.

### Standards-based, Extensible Communication Service

Studies have shown that over 60% of the development time for a distributed system is spent on the communications layer. A reliable, scalable, robust and location-independent communications system dramatically reduces the development time of distributed applications while increasing reliability.

FioranoMQ incorporates a standards-based communication layer that implements JMS - the Java Message Service API, released by Sun Microsystems, Inc. The system is completely event-based and supports both Publish/Subscribe and PTP (Point to Point) communication models. Additional design goals for the platform include:

- **Features** Support for Transactions, Guaranteed Messaging, Persistent Messages, Security and other key features, all based on the standard JMS API published by Sun Microsystems Inc.
- **Massive scalability** Today's Internet applications require hundreds if not thousands of concurrent connections to centralized servers. FioranoMQ is specifically designed for massive scalability. A pluggable, Scalable Connection Management (SCM) module supports thousands of concurrent connections to a single FioranoMQ server. Additionally, multiple FioranoMQ servers can be clustered together to support large numbers of concurrent client connections - clients connected to one server can exchange information with clients connected to remote servers.

---

## Web Integration

The FioranoMQ runtime library is designed to allow JMS communication APIs to be used within Java applets. JMS API calls in an applet communicate directly with a FioranoMQ server on the machine from which the applet was downloaded. This allows a single FioranoMQ server installed on the web-server machine to serve multiple applets over an Intranet (or over the Internet).

## Security

Limiting and controlling access to resources over a network is of paramount importance to enterprises. For example, one might not want a developer to run any network application on the finance network (which links up various financial related applications within an organization). Any platform that is used to deploy network applications over an enterprise network needs to have a security system with a high degree of configuration.

FioranoMQ incorporates a comprehensive security system that allows organizations to easily implement their security policies with respect to identification and authentication, authorization and access control, and privacy/integrity protection using encryption. Version 5 onwards of FioranoMQ implements the Java REALMS APIs, with access-control based on destinations. This allows an administrator to control access to destinations using an external tool, without any intervention from the application developer or the end-user.

## Remote Monitoring, Administration, Tracing and Logging Facilities

A key design goal of FioranoMQ was the ability to administer the server and monitor it from any location on the web. To this end, FioranoMQ incorporates a comprehensive Administration API, allowing administrators to configure ConnectionFactories, Topics, Queues, Users, Groups, ACLs (Access Control Lists) and user/group permissions. In addition, it allows to monitor the server statistics such as memory and thread usage, queue sizes and server throughput. The administrator can dynamically set different tracing options for each individual FioranoMQ component with the sophisticated tracing and logging facilities, provided by FioranoMQ. Trace levels can be set at start-time using Admin Studio.

## JMS Communication Models

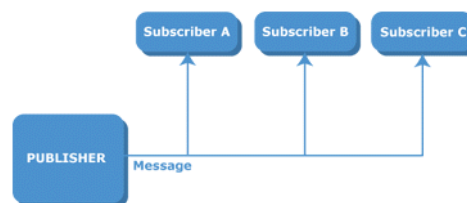
Traditional communication models involve demand-driven request/reply “polling” to access information, which is then transferred over a point-to-point topology. This section discusses how JMS combines the benefits provided by traditional models with event-based publish/subscribe “push” technology, offering real-time transmission of business events.

### Publish/Subscribe Interactions

Publish/Subscribe interactions, illustrated in Figure 1, are event driven rather than demand driven. In this paradigm, producers (also called publishers) send data to one or more consumers (also called subscribers) over the network. Publish/subscribe interactions are driven by events (usually the creation or arrival of data) in a producer. When an event occurs, messages containing event data are published to Topics. Consumers place a standing request for the data by subscribing to Topics and installing message listeners (callback functions), which allow them to filter the exact set of data to be delivered to the application. Through this process, producers are completely decoupled from consumers and they do not coordinate data transmission with each other.

Example applications that use publish/subscribe interactions include:

- Sales information from various outlets flows continuously to accounting, marketing, purchasing and other departments in the branch office of a large retail chain.
- Stock prices are simultaneously published to hundreds of traders on a trading floor.
- A just-in-time inventory management system publishes low-inventory data to purchasing, production and sales executives in real-time.
- Desktop machines in an enterprise network publish statistics (software audits and license usage) and alerts; systems administrators can deploy monitoring applications on any node at any time to receive and analyze the required information.



**FIGURE 1** PubSub Interaction

## Benefits of Publish/Subscribe

Distributed systems based on the publish/subscribe paradigm are very easy to change and modify because event dependencies are resolved through runtime event names, rather than by compilers, linkers or operating systems. Publishers and subscribers of JMS messages operate independently of and asynchronously from each other; and may change without recompilation. All communication between applications occur through messages that are routed through the FioranoMQ server, which implements the JMS API. This allows new JMS applications to be added to the distributed system later without recompiling, relinking or even stopping and restarting any existing applications.

Publish/Subscribe greatly reduces complexity in distributed system design and implementation. Events are published to channels as they happen; business processes that depend on these events automatically receive them as they occur. Key advantages of publish/subscribe include:

- **Robustness to change** Applications on heterogeneous platforms communicate and interoperate transparently, yet each application operates independently of and asynchronously from others. Communicating programs remain separately maintained and individually replaceable.
- **Location transparency** Applications can be moved between machines without disrupting other component applications. The developers use a logical, consistent naming convention; and are completely isolated from low-level details of network programming.
- **Incremental, dynamic extensibility** New application components can be added to, and existing components removed from the network dynamically and transparently.

## Publish/Subscribe Implementation in FioranoMQ

FioranoMQ coordinates all communication between publishing and subscribing applications by using a set of shared Topics as keys to distributing messages. Client applications, both publishers and subscribers, connect to the server to send and receive messages. For each topic, the client may act as a publisher, subscriber, or both.

The process of publishing involves sending messages to the JMS server. Whenever a JMS client wants to send a message to other clients, it simply publishes the information as a message to the server. There may be many publishers on a given Topic.

The process of subscription involves notifying the JMS server of a client application's interest in one or more Topics. The subscription must occur before a message on that topic gets published, in order for the client to receive that message. A subscriber can subscribe to one or more topics; a topic may have many subscribers. A subscribing client executes callback functions (called `MessageListeners` in JMS) to filter messages and process the ones of interest.

## Advantages of Topic-based Addressing

Topic-based addressing eases the development and deployment of distributed systems. Any application can send messages to any other application over the network in a location-independent manner. Senders and receivers can be anywhere on the network. Application components can be moved from one machine to another without causing any disruption in functionality. Server applications can migrate or replicate (to share heavy loads, for example) without disrupting the existing clients. Topic-based addressing allows you to build distributed systems that adapt easily to change and growth.

## Request/Reply Interactions

Traditional client/server applications fall under this category. A request/reply interaction consists of two point-to-point messages: a request and a reply. For example, a client requests data from a given server, which generates or computes an individual response and sends it back to the client. Communication is bi-directional. Examples of request/reply interactions include:

- **Remote database queries** A database server accepts incoming requests from remote clients, processes each request and returns the computed result to each client
- **Transaction** General transaction processing, such as ATM banking

- **Getting permissions** Getting permission for purchase requests in an enterprise; getting permission from a supervisor on a factory floor (to manage machines and/or control equipment, for example).



**FIGURE 2** Request/reply interaction (demand driven).

In a request/reply interaction, producers and consumers of data interact closely, as shown in Figure 2. A client sends a request to a server; the server receives the request, processes it, and sends back a reply only to the client that made the request. The requesting client waits until it receives a reply, and then continues processing. Requesting clients can listen in either blocking or non-blocking mode.

## Request/Reply Implementation in FioranoMQ

In traditional implementations, request/reply is based on polling, which requires an application or process to request event information at regular intervals. A poll requests information from a database, which either returns the requested information or a message indicating that the information is unavailable. As long as the application runs, it will continuously issue demands for information.

In FioranoMQ, request/reply is based on the publish/subscribe engine. When a client application publishes a request message on a subject, multiple subscribers registered to that subject may reply. This is unlike a poll-based request, which typically specifies a single information source. Most messages in the FioranoMQ system use a Topic for routing, but reply messages do not have a topic. The server routes these directly to the requestor. The publishing client application accepts the first reply to the request; all other responses are ignored.

## FioranoMQ System Architecture

Large-scale Internet and Intranet applications require hundreds if not thousands of concurrent connections to the server, a number that is expected to grow even higher over the next few years. As such, an enterprise-level messaging platform needs to be able to scale smoothly to handle the increasing numbers of client connections.

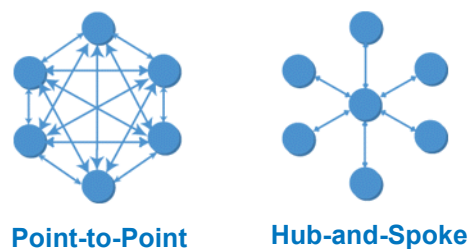
### Hub and Spoke Topology

FioranoMQ's design departs from the traditional point-to-point communications architecture to follow a "hub-and-spoke" design. The FioranoMQ Server acts as the hub and JMS-enabled applications as the spokes.

The hub-and-spoke topology has several benefits:

- Passing all the messages from the applications through the server allows the advantages of centralized administration, security, and routing of messages. For the system administrator, this implies all communications between the applications can be monitored and maintained from a single location. For the developers, this implies each application only needs to interface with the JMS-compliant FioranoMQ Server (which loosely couples applications).
- A comprehensive security model becomes much simpler to arrange, when all communications go through a single point.
- Message routing prevents users from having to screen all the messages locally to identify those that they need. Additionally, an application that publishes messages does not need to know the address of each subscriber.

Figure 3 illustrates the advantages of a hub-and-spoke topology in terms of logical connections between applications.



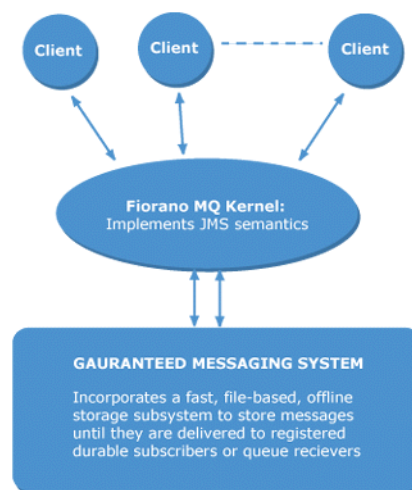
**FIGURE 3** Comparing point-to-point with hub-and-spoke topology

In Figure 3, with six applications communicating, the system requires 15 connections with the traditional point-to-point design and only six connections with the hub-and-spoke topology. The point-to-point model requires each application to include the code to interface with the five other applications; furthermore, each application requires modification if another application joins the network. In the hub-and-spoke model, each model only

needs to include a single set of code for communicating with the FioranoMQ Server at the hub. The hub decouples applications, thereby reducing system complexity and increasing system flexibility and adaptability.

## FioranoMQ Internal Architecture Details

The FioranoMQ messaging server implements both the publish/subscribe (Pub/Sub) as well as the point-to-point (PTP) subsets of the JMS API. Both of these subsets of the JMS API require some form of off-line database to store persistent messages, which have been received but not yet delivered to all the client applications. The system architecture of the FioranoMQ server is illustrated in Figure 4.



**FIGURE 4** FioranoMQ Publish/Subscribe Server - System Architecture

Following important points can be noted from the Figure 4:

1. The offline database is implemented using the default file system of the platform on which FioranoMQ is installed. The database contains multiple subdirectories, one for each topic for which messages need to be stored. As messages are consumed (i.e. when they have been delivered to all registered subscribers), the size of the database is automatically pruned using an efficient purging algorithm.
2. If a publisher marks a message as persistent, then the message is always stored in the offline database. This protects the clients against JMS provider failures. In order to ensure that a persistent message is either delivered to the JMS provider for sure or not delivered at all, the sending thread in the client program waits until the message has been completely written into the offline database of the FioranoMQ server. Hence, if either the client or the server crashes, it is ensured whether the message has been written into the offline cache.
3. Messages remain in the off-line cache until they have been delivered to all the registered durable subscribers. Information about all the durable subscriptions created on a given topic are stored in the off-line

cache. When subscriptions are removed (using the JMS “unsubscribe” call), the corresponding references are removed from the off-line database.

The off-line database storage subsystem ensures that all the persistent messages are delivered to all registered durable subscribers, with guaranteed one-time delivery. In case a provider failure occurs during message delivery or if the client application crashes, a persistent message is always redelivered. The redelivered message is marked, as required by JMS semantics.

Note that JMS allows non-persistent messages to be lost in case some subscribers are not connected to the system. Therefore, if you are developing a distributed system and want to ensure that none of your messages are ever lost, it is best for you to:

- Make all your subscribers Durable. Messages meant for a durable Subscriber are stored in the persistent cache even when the Subscriber is inactive.
- Ensure that your client applications publish only persistent messages

## Thread Management in FioranoMQ

In addition to the 100% pure Java implementation of the default connection manager, FioranoMQ software incorporates a special Scalable Connection Management (SCM) module. This allows a single FioranoMQ server to support several thousand concurrent clients connections on a variety of operating platforms, such as Windows NT, Solaris and other Unix platforms. FioranoMQ allows the user to choose between the default Java connection management and Scalable Connection Management.

Using the default Java Connection Manager, each server instance runs in its own JVM. Clients communicate with the server through sockets, with each client connection using two threads.

A single FioranoMQ server can handle as many concurrent connections as are supported by the JVM, under which the server runs. It is the responsibility of the administrator or the application designer to ensure that the clients connect to servers in the cluster in such a manner that the client load is automatically distributed between the servers.

As more clients connect to the server, the number of threads used on the server goes up linearly. This is because client connections are persistent and are not aborted/removed even if the client is not sending or receiving any messages. In case the server JVM runs out of threads, an error is reported and the FioranoMQ server refuses the requested client connection. This condition implies that the thread-limit for the JVM on the server machine has been reached. This number depends on the JVM being used, and varies depending on the target platform. On Solaris, our tests (January 2004) show that over 3500 concurrent connections can be supported using JDK 1.2, while Windows NT environments are a little less scalable. As JVM technology improves, the total number of concurrent connections that can be supported on a single server will continue to rise. The FioranoMQ

server is only limited by the JVM that it runs under, and not by any internal server limits. If your application needs more concurrent client connections than are supported by a single server, you should use the Scalable Connection Manager and/or cluster multiple FioranoMQ servers connected together, as explained in detail in the section “Load Balancing, Failover Protection and High Availability” .

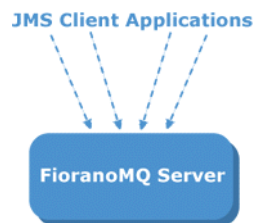
The thread limit is typically very dynamic and depends critically on the load on the server machine. By aborting compute and thread intensive applications, one can increase the number of threads available to the JVM, running the FioranoMQ server. Thus, concurrent client connections to a single FioranoMQ Server are limited in number. The exact limit critically depends on the hardware configuration and the system software (for example, the JVM) running on the FioranoMQ Server machine.

## Memory Management and Publisher Throttling in the FioranoMQ Server

Each client connection to the FioranoMQ Server uses up some amount of available memory. The FioranoMQ Server implements internal memory management algorithms in order to keep track of the approximate amount of memory being consumed by the system. When the memory limit of the JVM is reached, the FioranoMQ Server automatically starts throttling the publisher application(s) connected to the server and, where allowed by JMS semantics, overflowing the internal buffers. The publisher throttling algorithms automatically governs the speed of running applications, ensuring that the server never gets flooded at any point and that total server throughput remains constant. Publisher throttling ensures that the server scales well as more clients are added to the server.

## Typical FioranoMQ Server Setup

In a typical case, the FioranoMQ Server is used in stand-alone mode. During the setup process, the system administrator uses external tools and special administrative APIs to create a set of destinations on a FioranoMQ server. As defined by JMS, a Destination is either a Topic for Publish/Subscribe interactions or a Queue for Point to Point interactions. Each stand-alone FioranoMQ server owns a set of available destinations and clients connect directly to the server, as illustrated in Figure 5.



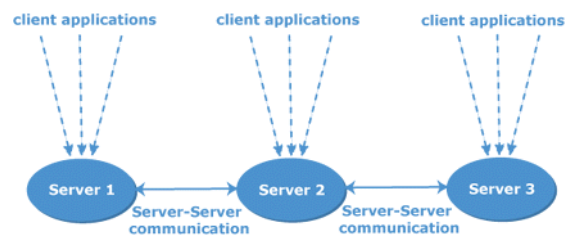
**FIGURE 5** Typical setup of a FioranoMQ Server

In the typical setup illustrated in Figure 5, all the clients connect to a single server. This setup, while adequate for a large number of applications, suffers from the inherent problem that multiple concurrent client connections can overload the server, resulting in a general slowdown in response time for all applications. To solve this problem, multiple FioranoMQ servers can be connected together, allowing clients connected to one server to exchange information with clients connected to another server. The FioranoMQ server-to-server communication architecture is explained in detail in the next section.

# FioranoMQ Server-to-Server Communication and Clustering Architecture

The FioranoMQ architecture allows multiple FioranoMQ Servers to be connected together, allowing clients connected on one server to exchange information with clients connected to any of the other servers. Servers may be connected locally or across different geographical sites, in any topology, as illustrated in Figure 6. Server to server communication is achieved using the FioranoMQ Repeater.

Server clustering allows clients connected on different FioranoMQ servers to exchange information without each client having to connect to each server. This feature is particularly useful in deploying applications that need to communicate with other applications across geographically distributed sites. For instance, an organization might have offices in New York, San Francisco and Boston, with users in each office that need to exchange information with the other offices. With server-to-server communication, each client application in Boston (that sits on an end-user's desktop) only needs to connect to the FioranoMQ server in Boston. The FioranoMQ Administrator can configure the Repeater to automatically forward relevant messages from the Boston server to the FioranoMQ servers in New York and/or San Francisco, where they are delivered to the user applications that have subscribed to the appropriate Topics. Moreover, in case there are any transient network failures across the WAN connecting Boston, New York and San Francisco, none of the client applications are affected. Publishers can continue to publish messages locally and subscribers stay connected and receive local messages. The Repeater can be configured to ensure FioranoMQ automatically buffers messages at the publisher end and delivers these messages to the appropriate neighboring servers as soon as the network comes up again.



**FIGURE 6** FioranoMQ Server Network

The Administrator can control the route for messages and can also setup the kind of subscribers (Durable/Non-Durable) that can be created on every connection. The Administrator can also control which Topics are distributed to which neighboring servers. Since topic information is available on each server, clients can publish or subscribe to any topic to send and receive information. All server-to-server communication is handled transparently by FioranoMQ and the client application does not need to be modified in any way.

Figure 6 illustrates several key points about the server-to-server communication architecture, which is discussed in the following sections.

## Server-to-Server Communication Mechanism - Repeaters

FioranoMQ Servers communicate with each other through Message Repeaters. A Message Repeater is a dedicated JMS client application that allows messages to be distributed across remote client Applications (Clients connected to different FioranoMQ Servers).

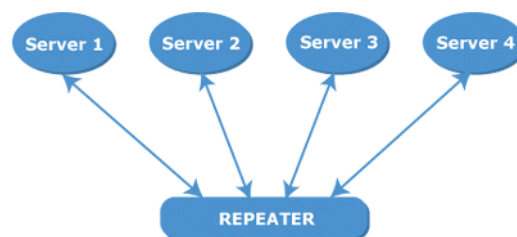
In this section, we take a closer look at how a network of multiple FioranoMQ servers is set up. For more information, read the “FioranoMQ Repeater” section of the Administration Guide.

### Connection Topology

FioranoMQ servers can be connected in any topology. The FioranoMQ Repeater, allows the Administrator to configure multiple FioranoMQ servers in any topology. Examples of popular network topologies and FioranoMQ as well as Repeater configuration in each of these topologies is discussed below.

In all the scenarios, servers can be part of the same LAN or can be spread across multiple WANs.

Figure 7 illustrates 4 FioranoMQ Servers setup up in a Hub and Spoke based topology. The Repeater is configured to send and receive messages to the all Servers (S1 to S4). The Repeater ensures that messages are delivered only once and there is no loop back of messages. So, in this case, Clients connected with Server 1 can exchange messages with clients connected to Server 2, Server 3, Server 4 and vice-versa.

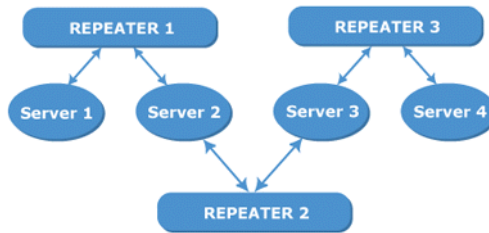


**FIGURE 7** FioranoMQ Server setup in a Hub and Spoke based topology

Alternatively, the FioranoMQ Administrator can configure the enterprise in a bus-based topology, as illustrated in Figure 8. Here Repeater1 assists clients on Server1 and Server2 to transfer messages to each other. Similarly Repeater 2 and Repeater 3 are responsible for communications between clients on Server 2 - Server 3 and Server 3 - Server 4 respectively. The Repeater allows for only 1 hop for messages i.e. messages from Server 1 will never reach Server 3.

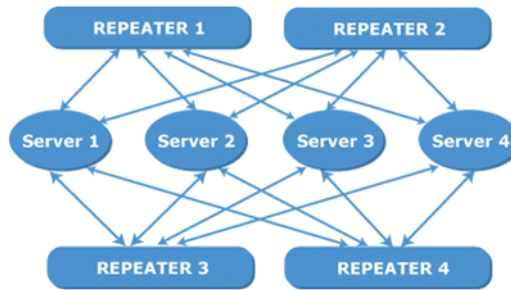
Hence, in this topology:

- Clients on Server 1 can exchange messages only with clients on Server 2
- Clients on Server 2 can exchange information only with Server 3 and Server1
- Clients on Server 3 can exchange information only with clients on Server 2 and Server 4
- Clients on Server 4 can exchange messages only with clients on Server 3.



**FIGURE 8 Repeater setup in a bus-based topology**

Figure 9 illustrates the "mesh-based" topology, where a separate instance of the Repeater can be setup for every Server. The result of this topology is similar to the one shown in Figure 7.



**FIGURE 9 Various Server to Server topologies**

## Client Connections

Client applications can connect to any FioranoMQ server in a network. Clients connected to one server can exchange messages with clients connected to another server, without each client having to explicitly connect to each server. The client application does not need to be modified in any manner. FioranoMQ manages the process of distributing Topic information across machines as required.

## **Robustness in the Presence of Network Failures**

Data transfers between multiple FioranoMQ servers, connected to each other using the “Repeater”, can be optionally made to use Persistent Messages and Durable Subscriptions. In this case, messages transferred between servers are always logged to persistent storage, making the system highly reliable and robust in the event of network failures.

## **Subscription Mode and Choice of Selectors**


The FioranoMQ Administrator can configure the Repeater to create either Durable or Non-Durable Subscription with the Source Server. Message selectors can also be set on the created Subscriptions. For more information, read the “Configuring FioranoMQ Repeater” section of the Administration Guide.

# Load Balancing, Failover Protection and High Availability

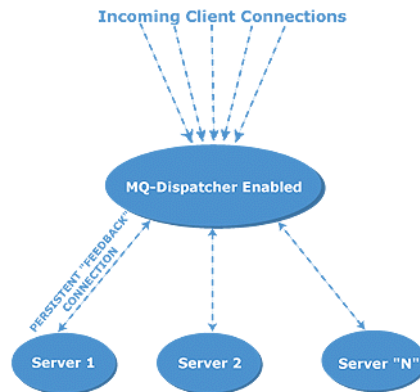
FioranoMQ implements dynamic load balancing and failover protection, making the system highly available and allowing an unlimited number of concurrent client connections to a server-cluster.

## Load-balancing Architecture

Load balancing is implemented using the Dispatcher, which routes incoming client connections to the least-loaded server in a cluster, as illustrated in Figure 10. Any FioranoMQ Server can be started to provide Dispatcher services. The FioranoMQ Administration can enable Dispatcher services for FioranoMQ using the Admin Studio.

 For more information, please refer to the chapter "FioranoMQ Clustering" of Admin Guide.

In a server cluster, any number of FioranoMQ servers can be started to provide "dispatcher" facilities.



**FIGURE 10** FioranoMQ load-balancing architecture

The Dispatcher component of FioranoMQ is typically connected to multiple FioranoMQ servers, which can run on multiple machines. All of these servers become part of the "cluster" that is serviced by the Dispatcher. Fiorano provides a special Dispatcher Administration API that allows servers to be added to and removed from the cluster. The Dispatcher maintains a persistent connection with each FioranoMQ server in its cluster, allowing servers to inform the dispatcher in real-time about varying server loads. The Dispatcher uses information (such as the total number of current connections) from the server-cluster to determine the load factor for each server.

---

## Balancing Client Connections

When a client wants to connect to a FioranoMQ server that is part of a server cluster, it always connects to the FioranoMQ server that acts as the Dispatcher. The Dispatcher automatically routes the incoming connection to the least loaded server in the cluster, which could also be the same server on which Dispatcher is executing.

Internally, the client connects to the FioranoMQ server that acts as the Dispatcher, which returns the IP address and port number of the least loaded FioranoMQ server in the cluster to the client. When the client receives the server address to connect to, it automatically disconnects from the dispatcher and binds to that server. From here, the client application is just another normal JMS client for the target FioranoMQ server. When any client disconnects from a server in a cluster, the server publishes a message to the Dispatcher informing the latter to update the internal tables appropriately. There are no changes required in the client application to connect to the FioranoMQ Server in a cluster.

The following code snippet shows how a client application connects to a dispatcher and how you can find the server with which the application is connected. It explains the concept that no changes are required in a Client application, if clients are to be load balanced across multiple servers.

```
// InitialContext Object used for looking up JMS administered
// objects on the FioranoMQ Dispatcher Server, located on the
// "serverIP".

Hashtable env = new Hashtable ();
env.put (Context.SECURITY_PRINCIPAL, "anonymous");
env.put (Context.SECURITY_CREDENTIALS, "anonymous");
env.put (Context.PROVIDER_URL, "http://serverAddress:1856");
env.put (Context.INITIAL_CONTEXT_FACTORY,
"fiorno.jms.runtime.naming.FioranoInitialContextFactory");

InitialContext ic = new InitialContext (env);
System.out.println ("Created InitialContext :: " + ic);

// Lookup Connection Factory and Topic names from the connected
// Server(leastLoaded Server in cluster)

TopicConnectionFactory tcf =
(TopicConnectionFactory) ic.lookup ("primaryTCF");

ConnectionFactoryMetaData metaData =
tcf.getConnectionFactoryMetaData();

// Print the URL of the Server to which this Client is
// connected
System.out.println (metaData.getConnectURL());
```



The client application only needs to know the IP address and port number of the server that provides Dispatcher services and not of any servers in the cluster. The IP address and port number of the server to which it connects is completely transparent to the client.

---

## Multiple Dispatcher Service

FioranoMQ Administrators can also setup multiple FioranoMQ Servers in the cluster to provide "dispatcher" facilities. This prevents "single points of failure".

Setting up multiple dispatcher servers in a cluster ensures that if one dispatcher goes down, then any one of the other dispatcher servers can take over all the client applications and load balance them accordingly. Now if the secondary dispatcher also goes down, you can have another dispatcher configured in the cluster so that the client applications never fail or stop functioning. Thus, in case of a server failure, the client applications just shift from one failover server to the other. This provides "multiple points of failure" in a cluster scenario.

## Automatic Failover Protection

Automatic Failover is implemented by Auto Revalidation and Durable Connections. Auto Revalidation ensures that the clients are connected to back-up servers. The application doesn't have to worry about reconnection logic being used. A Durable Connection coupled with Auto Revalidation takes care of storing, re-connecting and then forwarding the stored messages to the server. These extended functionalities also ensure that data is not lost in transit and is sent as soon as the connection is re-established.

## High Availability

A backup server is started along with the primary FioranoMQ server. In case the primary server becomes unavailable, all the client applications connected to it are automatically connected to the backup server. A backup server is always ready to take up all the load and resume normal messaging operations in case the primary server becomes unavailable. The process of shifting from the primary server to the backup server or vice-versa is transparent to the application.

## LPC mode for Clients in FioranoMQ Server

A client using LPC mode in FioranoMQ is a JMS client that runs within the kernel process and uses local procedure calls to send/receive messages to/from JMS destinations. In LPC mode, JMS clients hook on to FioranoMQ server locally within the kernel process without the need for socket connections.

## Scalability

Fiorano's load balancing and failover protection architecture allows unlimited scalability in terms of the number of concurrent client connections that a FioranoMQ server can service. Combined with server to server communication, the Fiorano clustering architecture provides a very robust solution for a vast set of customer problems.