MigratoryData Server

Architecture Guide

Version 5.0
February 17, 2019

Copyright Information

Contents

Figures

Tables

1. Introduction

This guide provides an overview of the MigratoryData server and explains its concepts and features.

1.1 Release

This guide is part of the documentation set for MigratoryData Server version 5.0.

1.2 Related Documents

- *MigratoryData Installation Guide*
- *MigratoryData Configuration Guide*
- *MigratoryData API Developer's Guide* and *Reference Manual* for each language and technology listed in Table 3.1.

2. The Problem MigratoryData Solves

Traditionally, web servers deliver data using a *request-response interaction* model as follows:

1. A user opens a web page in a web browser
2. The web browser makes an HTTP request to the web server and displays the content of the web page as it is available in the web server at the moment of the request
3. If after the web page is displayed, the content of the page changes in the web server, then the user will not see the changes until the user manually refreshes the web page in the browser

A technique named *polling* is often used to provide users with up-to-date content without the need for the manual refresh of Step 3 above. The polling technique consists in including some JavaScript code in the web page: the polling script. The polling script is executed by the web browser in background as long as the web page is displayed by the browser. The polling script periodically sends HTTP requests to the web server to check if there are or not any changes in the content of the web page. If there are no changes, the polling script will do nothing, otherwise it will display the new content.

While the polling technique can be used to deliver live content for certain live applications, this technique could easily become very inefficient or not applicable for many other live applications.

Even if the polling script is configured to poll the web server as frequently as a web server can support, the latency of data (i.e. the time required to propagate data from the server side to the user) can still be too high for web applications such as a financial portal delivering real-time market data or a sports betting website where each millisecond counts.

Another example is a live website with many concurrent users. A traditional web server is able to handle a relative small number of concurrent users. If a website has many concurrent users, then multiple web servers must be installed on multiple machines. Moreover, each HTTP request sent by the polling script, just to check if there is or not any fresh data on the server, includes hundreds of bytes, which are automatically added by the browser to the request: the HTTP headers. This redundant overhead sent by many users at a high polling frequency produces an important bandwidth consumption. The need for multiple machines and the high bandwidth

usage significantly increase the total cost of ownership for the websites with many concurrent users when using the polling technique to deliver live content.

2.1 The MigratoryData's Solution

MigratoryData Server delivers data according to a *subscribe-publish interaction* model as follows:

1. A user sends a WebSocket / HTTP request to the server
2. A persistent TCP connection is created between the user's browser and the MigratoryData server
3. The MigratoryData server uses this persistent TCP connection to push data available at the moment of the request as well as any subsequent data, without any additional requests from the user.

MigratoryData implements the WebSockets protocol (see the RFC 6455 standard at `http://tools.ietf.org/html/rfc6455`). The recent browsers have support for WebSockets and MigratoryData creates the streaming TCP connection described at Step 2 above using WebSockets. For the older browsers which do not have support for WebSockets, MigratoryData implements techniques which still use a single TCP connection for streaming – like WebSockets do behind the scene – and it's 100% JavaScript-based (no browser plug-in is required).

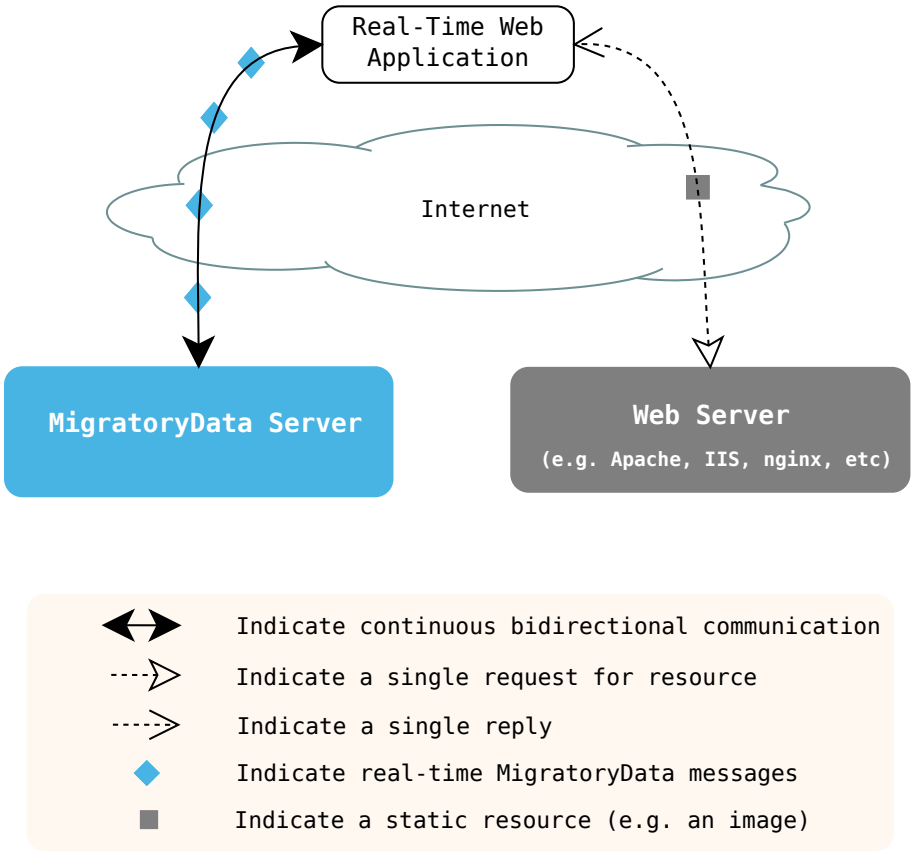
Unlike classical web servers or other WebSocket implementations, MigratoryData Server was designed to scale to a huge number of concurrent users. It has been benchmarked to stream real-time data to 10 million concurrent clients from a single 1U server in a particular test scenario and it is now used in production to successfully stream real-time data to millions of end-users every day.

> **Note** — The goal of MigratoryData Server is not to replace a classical web server such as Apache, Microsoft IIS, or nginx, but to integrate with such a web server. On one hand, the web server will be used to provide any static resources necessary for a web application such as images, text, and server-side scripting. On the other hand, MigratoryData Server will be used to provide the real-time content to the web application. Figure 2.1 shows how MigratoryData Server integrates with an existing web server.

The solution MigratoryData proposes not only is a very scalable solution for adding real-time capability to web applications, but it also offers many other functionalities.

The same MigratoryData JavaScript API used to build highly scalable real-time web applications is also available for other technologies. You can use the same API to build native mobile real-time

Figure 2.1: Integration of MigratoryData Server with a Web Server.



applications for iOS and Android, as well as other real-time Internet applications written in Java, .NET, C++, PHP, Python, and Ruby.

MigratoryData can be deployed as a fault-tolerant cluster to enable high availability and guaranteed delivery of data even in presence of sudden failures such as hardware failures and network disconnections. It scales horizontally with built-in load balancing to meet any growth in number of users. Advanced monitoring and security and many other features are also offered by MigratoryData.

See Chapter 3.1 to learn more about the main features of MigratoryData Server.

3. Product Overview

This chapter introduces MigratoryData Server.

Topics

3.1 Features

This section describes the main functionalities of MigratoryData Server.

Zero Installation for Web Clients

No installation is necessary on the desktops or mobile devices of users, real-time data is delivered directly to any web browsers via pure JavaScript (no plug-in required).

Real-Time Bidirectional Data Delivery

Messages are published from server to clients and from clients to server with low message latency and continuously, as fresh data is available.

Client APIs for Web, Mobile, Desktop, and Server Applications

A common API with libraries for the the most popular technologies – JavaScript, iOS, Android, Java, .NET, C++, PHP, Python, and Ruby – can be used to simply add real-time features to your applications.

Extreme Vertical Scalability

MigratoryData Server has been benchmarked to support 10 million concurrent connections in a particular test scenario on a single 1U server (see our blog) and it is now used in production to stream data to millions of end-users daily.

Horizontal Scalability

Besides its high vertical scalability, MigratoryData Server scales horizontally with built-in load balancing to meet any growth in number of users of your real-time application.

Weighted Load Balancing

The load balancing can be controlled to take into account the hardware differences of the machines which host the instances of MigratoryData Server.

Fault Tolerance

Multiple instances of MigratoryData Server can be deployed as a fault tolerant cluster with no single point of failure offering 24x7 high availability. Please refer to Section 5.7 for a detailed presentation.

Guaranteed Message Delivery

The cluster of MigratoryData servers can be configured to guarantee the end-to-end delivery of messages even in the event of unexpected events such as hardware failures or network disconnections. Guaranteed Message Delivery is presented in detail in Section 5.8.

Low Bandwidth

MigratoryData implements an efficient proprietary communication protocol that adds to each message a small constant overhead.

Milliseconds Latency

Fresh data available on the server is delivered to users in milliseconds.

High Throughput

MigratoryData Server scales up to 10 Gbps on 10 Gigabit Ethernet while streaming 2 million messages per second on a single 1U server.

Security

MigratoryData Server uses HTTPS, TLS/SSL encryption, dual firewalls, and entitlement. Section 5.3 is focused on security.

Advanced Monitoring

Secure (TLS/SSL) and password-protected JMX, HTTP, and PUSH monitoring services are exposed by MigratoryData Server. See Section 5.2 for details.

Internationalization

MigratoryData Server accepts and supports content through the Unicode character set.

Multi-Core CPU Optimized Architecture

MigratoryData Server can be configured to distribute its activity on a configurable number of CPU cores.

Running on a dedicated server, MigratoryData Server will automatically take advantage of all CPU cores available (no configuration tuning is needed). Otherwise, if MigratoryData Server runs on a shared server, it can be configured to use only a part of the CPU cores available.

Capacity Planning

You can estimate the hardware and the number of MigratoryData Server instances required for your use case by using *MigratoryData Benchmark Kit*, a software tool able to:

- connect a configurable number of concurrent clients
- subscribe to a configurable number of subjects per client
- publish messages of a configurable size at a configurable frequency

3.2 Language Interfaces

MigratoryData offers a common client API for various languages and technologies. You can use the different editions of the MigratoryData Client API to quickly add real-time features to your existing Internet applications. The available API editions are listed in Table 3.1.

Application Type	Client API	Functions
Web Applications	JavaScript	publish and subscribe
Mobile Applications	iOS	publish and subscribe
	Android	publish and subscribe
Desktop / Server Applications	Java	publish and subscribe
	.NET	publish and subscribe
	C++	publish and subscribe
	PHP	publish
	Python	publish
	Ruby	publish

Table 3.1: MigratoryData API Editions

3.3 Platform Support

Written in Java, MigratoryData Server runs on all major operating systems: Linux/Unix, Mac OS, Windows, and potentially any other platform having support for OpenJDK Java Runtime Environment (JRE) version 8 or later.

MigratoryData Server comes with 64-bit installers for RPM-based and DEB-based Linux distributions. Also, a platform-independent tarball is available.

The recommended operating systems for production deployments are Red Hat Enterprise Linux or Centos 6 or later and Debian 7 or later.

4. Concepts

This chapter describes the MigratoryData concepts: messages and publish-subscribe model.

Topics

4.1 Messages

A MigratoryData *message* has several pieces of information. Table 4.1 shows the message components exposed to the APIs.

Component Name	Required Component	Description
Subject	Mandatory	The subject of the message
Content	Mandatory	The content of the message
Fields	Optional	The fields of the message, as a dictionary. The keys are the field names, and the corresponding values are the field values.

Table 4.1: The Message Components

> **Note** — Depending on the configuration of MigratoryData Server and depending on the API usage, messages may include other pieces of information. For example, supposing the Guaranteed Message Delivery feature is enabled, then MigratoryData messages will be enriched with information related to sequence numbers used to achieve Guaranteed Message Delivery. However, such information is used internally only by the MigratoryData server and its APIs, it is not exposed to the developers in the API.

MigratoryData APIs provide methods to create messages from application-specific data, publish messages, and retrieve the application-specific data from messages when received by the clients.

While the content of the message, the field names, and field values can be any sequences of bytes, the subject of the message must respect a particular syntax as described in the next section.

4.1.1 Subjects

The subjects of the messages are used by both subscribers (to listen for specific messages) and by publishers (to publish messages on particular subjects).

A *subject* is a string of characters that respects a syntax similar to the Unix absolute paths. It consists of an initial slash (/) character followed by two or more character strings – called *segments* – separated by a single slash (/) character. Within a segment, the slash (/) character is reserved. Each subject must have two or more segments.

For example, the following character string, composed by the segments `Stocks`, `NYSE`, and `IBM`, is a valid subject for MigratoryData Server:

/Stocks/NYSE/IBM

Table 4.3 shows several examples of invalid subjects.

Invalid Subject	Reason
/Stocks//IBM/BID	The slash (/) character is not allowed in a segment or because the second segment is empty (any segment must contain at least one character)
Stocks/IBM/BID	The subject does not start with a slash (/) character
/Stocks/IBM/BID/	The last segment is empty
/	Subject formed from a single empty segment
/Stocks	Subject has only one segment (two or more are required)

Table 4.2: Examples of Invalid Subjects

4.1.2 Snapshot Messages

For each subscribed subject X, MigratoryData Server maintains an *snapshot message* defined as follows:

- The subject of the snapshot message is X
- The content of the snapshot message is the content of the most recent published message with subject X
- The field name set of the snapshot message is the union of the field name sets of all messages published with subject X
- The value of each field of the snapshot message is the most recent value of that field published by messages with subject X

According to this definition, the snapshot message of a subject, which has only messages without fields, coincides with the most recent published message for that subject.

Table 4.3 shows the snapshot message of the subject /Stocks/NYSE/IBM as new messages are received by MigratoryData Server.

Time	Message	Snapshot Message
10:12	subject=/Stocks/NYSE/IBM	subject=/Stocks/NYSE/IBM
(first message)	content=193	content=193
	field-volume=6.62M	field-volume=6.62M
10:13	subject=/Stocks/NYSE/IBM	subject=/Stocks/NYSE/IBM
	content=192	content=192
		field-volume=6.62M
10:15	subject=/Stocks/NYSE/IBM	subject=/Stocks/NYSE/IBM
	content=195	content=195
	field-ask=190	field-ask=190
		field-volume=6.62M
10:20	subject=/Stocks/NYSE/IBM	subject=/Stocks/NYSE/IBM
	content=193	content=193
	field-bid=194	field-bid=194
	field-ask=191	field-ask=191
		field-volume=6.62M

Table 4.3: Examples of Snapshot Messages

When a client subscribes to a subject, MigratoryData Server will firstly send to that client the snapshot message of that subject (if available), then it will send the subsequent real-time messages for that subject as they are received in MigratoryData Server from the publishers. See Section 4.2 to learn more about the publish-subscribe model.

You can disable Snapshot Messages by configuring the MigratoryData server as follows:

```
PublishSnapshotMessage = false
```

4.1.3 Retrieve Snapshots via HTTP Requests

You can get the snapshot message of a subject from the MigratoryData server via a simple HTTP request.

Supposing the MigratoryData server is accessible via the URL `https://push.example.com`, then you can retrieve the snapshot message of a particular subject `X` via the HTTP request:

```
https://push.example.com/snapshot?subject=X
```

If the entitlement feature is enabled for the MigratoryData server, then you should also include the authorization token in the HTTP request. Supposing the entitlement token of a user is `U` and the user identified by the token `U` is allowed to subscribe to the subject `X`, then, to retrieve the snapshot of the subject `X`, use:

```
https://push.example.com/snapshot?subject=X&token=U
```

Note — Currently the snapshot message retrieved via a HTTP request does not include the fields.

4.1.4 Character Encoding

The UTF-8 character encoding is used for all components of MigratoryData messages including for the message content, message subject, field names and field values. Thus, MigratoryData Server is able to handle messages with any international character set including ASCII.

4.2 Publish-Subscribe Model

A MigratoryData environment typically consists of:

- One or more MigratoryData servers
- *Subscribers* which are consumer clients
- *Publishers* which are publisher clients

Messages are asynchronously sent from publishers to the MigratoryData server and from the MigratoryData server to subscribers.

The Publish-Subscribe Model is defined as follows. A client connects to a MigratoryData server and subscribes to a subject `X`. Depending whether the subject `X` is already subscribed by other clients or not, one of the following two situations will happen:

- If `X` is already subscribed, then MigratoryData Server will send to that client the initial snapshot message for the subject `X` and any subsequent message received from publishers.
- If `X` is not subscribed by any other client, then MigratoryData Server will retain the subscription request and will publish messages with the subject `X` once received from publishers.

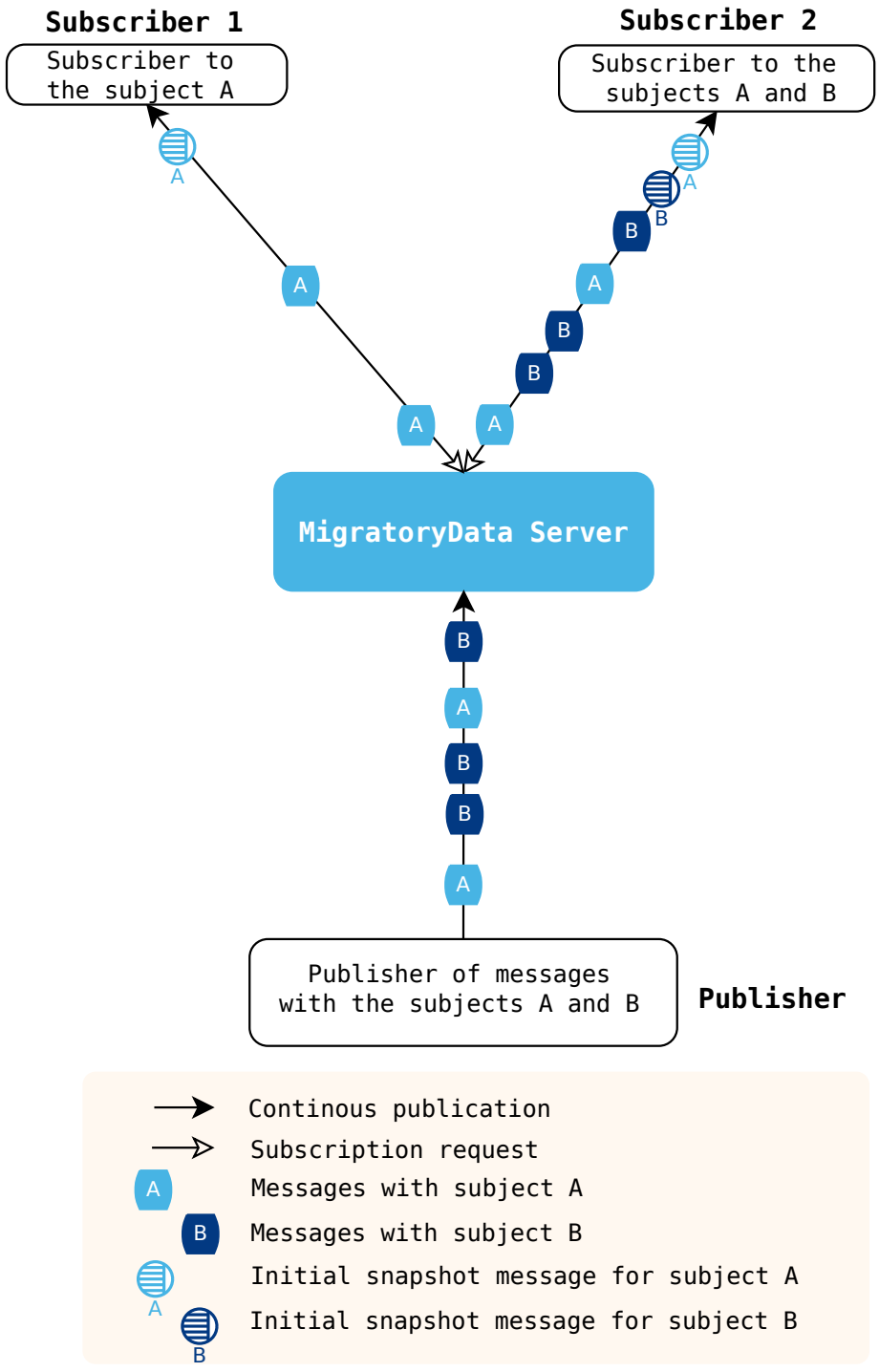
When the client is not interested anymore in messages with subject `X`, it can unsubscribe from the subject `X`. Depending whether the subject `X` is also subscribed by other clients, one of the following two situations will happen:

- If `X` is also subscribed by other clients, then MigratoryData Server will simply unsubscribe that client from the subject `X` and it will not send anymore messages with subject `X` to that client.
- If `X` is not subscribed by any other client, then MigratoryData Server will unsubscribe that client from the subject `X`. Also, it will remove the subscription interest for the subject `X`.

  Note — The interaction of the Publish-Subscribe Model described above slightly changes if any of the Entitlement, Conflation, Batching, or Guaranteed Message Delivery features is enabled. See Section 5.4 and Section 5.5 and Section 5.6 and Section 5.8 to learn how the publish-subscribe interaction works when the Entitlement or Conflation or Batching or Guaranteed Message Delivery feature is enabled.

Diagram 4.1 shows an example of the publish-subscribe interaction. Note that `Subscriber 1` which subscribes to the subject `A` receives only messages with the subject `A`. It does not receive messages with subject `B` as it does not subscribe to the subject `B`.

Figure 4.1: MigratoryData Publish-Subscribe Model



5. MigratoryData Server

This chapter describes the main features of MigratoryData Server.

Topics

5.1 Communication Ports

The web clients (built with MigratoryData Client API for JavaScript), which run in the recent web browsers having WebSockets support, will communicate with the MigratoryData server using the WebSocket protocol (see RFC 6455). The other web clients, which run in the older browsers not having WebSockets support, any other web, mobile, and desktop client will communicate with the MigratoryData server using the HTTP protocol.

Behind the scene, every client (communicating either via the WebSocket or via the HTTP protocol) uses a single persistent streaming TCP connection to communicate with the MigratoryData server. Thus, at a higher level of abstraction, MigratoryData Server can be viewed as a TCP server.

MigratoryData Server is able to listen for TCP connections on one or more ports. If the machine running the MigratoryData server is multi-homed (i.e. it has multiple IP addresses associated either with multiple network interfaces or with a single network interface but using multiple IP aliases), then MigratoryData Server can be configured to listen on one or more ports of one or more IP addresses of the machine.

Moreover, the ports can be configured to accept encrypted connections via HTTP Secure (https) or WebSocket Secure (wss). Note that MigratoryData Server can be configured to accept normal connections, encrypted connections, or both encrypted and normal connections.

Both protocols HTTP and WebSocket use the same standard port numbers: 80 for normal connections and 443 for encrypted connections.

> **Note** — The overhead introduced by the encrypted connections compared to the normal connections is a relatively small one. Thus, the recommendation is to configure MigratoryData Server to use encrypted connections. In this way, your data will be securely delivered. Encrypted connections also help to avoid the interference with certain security solutions.
>
> For example, when using normal connections, certain antiviruses may decide to block the data streaming between MigratoryData Server and the client. Such an antivirus software wrongly interprets the data streaming as a potential security attack. Otherwise, when using encrypted connections, the antivirus software is unable to inspect the data received from MigratoryData Server so it will not block the data streaming.

5.1.1 Some Recommended Port Configurations

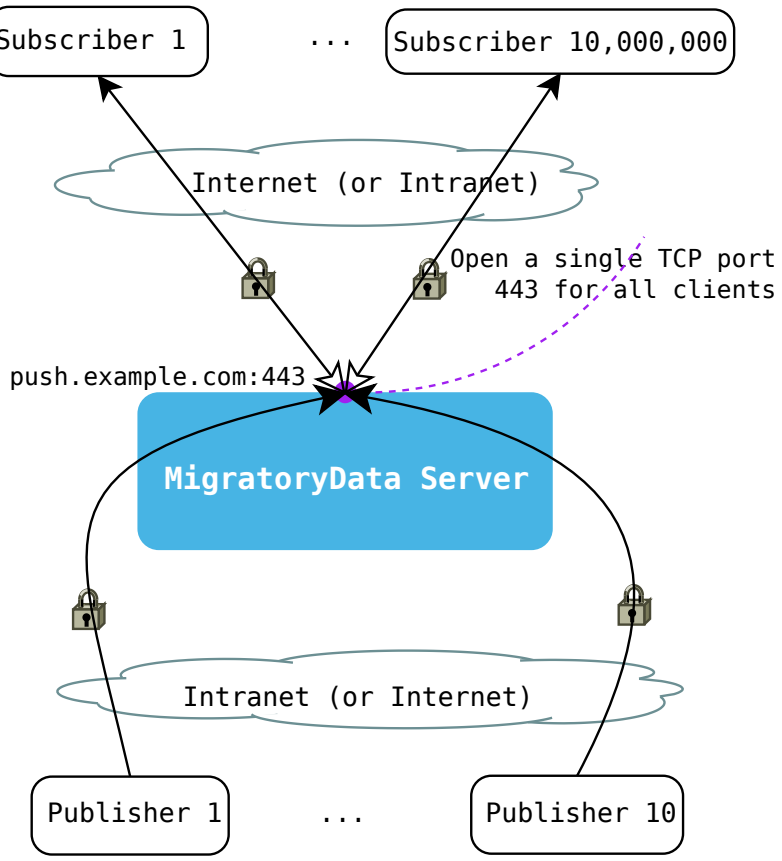
MigratoryData Server should be typically configured to accept client connections on a public address – say `push.example.com`. Ideally, it should be configured to accept only encrypted client connections via the standard `https` / `wss` port 443. Thus, it's configuration should be as follows:

```
ListenEncrypted = push.example.com:443
```

Using this configuration, any web, mobile, or desktop client will be able to connect in the same way to MigratoryData Server.

Figure 5.1 shows how both subscribers and publishers securely connect to a single open port of MigratoryData Server.

Figure 5.1: MigratoryData Using a Single Communication Port

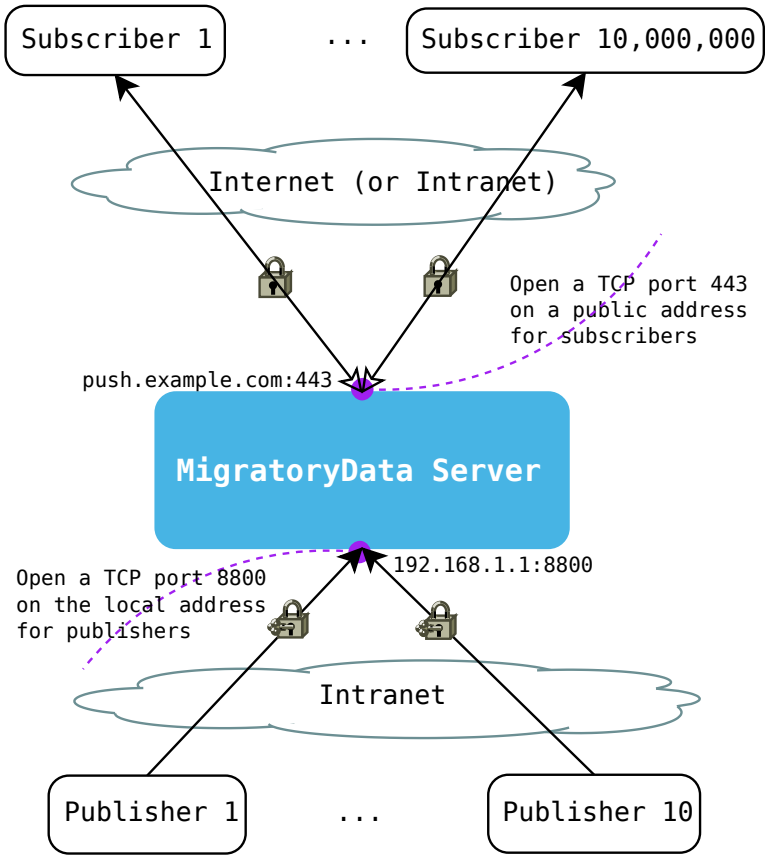


While it's perfectly valid and beneficial to use a single network address and port to accept all clients, there are setups when MigratoryData Server is deployed in the DMZ and the publisher clients are deployed behind the second firewall of the DMZ to integrate with the backend servers (see Section 5.3). In this setup, the publishers typically are not allowed to access Internet addresses, thus they will not be allowed to connect to `push.example.com:443`. For such a setup, a secondary LAN address – say `192.168.1.1` – should be configured on the machine running MigratoryData Server. For this local address, you can configure any port available to accept connections from publishers, provided however that the port is allowed by the firewall. As in the case of the client port, you can configure the publisher port to accept either normal or encrypted connections.

Figure 5.2 shows the TCP ports used by MigratoryData Server to communicate with its clients and publishers.

> **Note** — A new port should be opened if you enable the JMX monitoring feature, and another new port should be also opened if you enable the HTTP monitoring feature. These monitoring ports (not shown in Figures 5.1 and 5.2) can be any ports available, provided that these ports are allowed by your firewall. You can configure these ports on the LAN address or on the public address. As in the case of the subscriber port and the publisher port, you can configure these monitoring ports for either normal or encrypted connections. Also note that the access to the monitoring information can be configured with or without password authentication.
>
> Also, up to five ports are used for internal communication among the cluster members.

Figure 5.2: MigratoryData Communication Ports



5.2 Monitoring

MigratoryData Server supports the Java Management Extensions (JMX) technology to provide monitoring and statistics information. Monitoring and statistics information can be also retrieved with HTTP requests. Finally, monitoring information can be obtained in real-time using any MigratoryData Client API by subscribing to certain special subjects named *meta-subjects* (PUSH).

You can configure MigratoryData Server to permit access to the monitoring and statistics information with or without password authentication, via normal or encrypted connections for both HTTP monitoring and JMX monitoring.

> Access to PUSH monitoring using meta-subjects follows the security and entitlement rules used for subjects in general. So, there are no authentication and encryption parameters specifically for PUSH monitoring, unlike for JMX and HTTP monitoring.

The indicators which can be monitored are:

- The number of connected users to MigratoryData Server
- The number of connections per second established with MigratoryData Server
- The number of disconnections per second from MigratoryData Server
- The number of incoming messages per second received from publishers
- The number of outgoing messages per second sent to subscribers
- The number of incoming bytes per second received from publishers
- The number of outgoing bytes per second sent to subscribers

The following statistics are computed for the parameters above:

- Maximum
- Average
- Standard Deviation

At each moment, the values of the statistics above are available for the following periods of time:

- Last minute, last 5 minutes, and last 15 minutes
- Last hour, last 5 hours, and last 15 hours

- Last day, last 5 days, and last 15 days
- Last month, last 5 months, and last 15 months

The `jconsole` utility that is freely available with OpenJDK can be used to connect to the JMX monitoring service of MigratoryData Server. Also there are many JMX commercial tools that provide enhanced functionality like dashboards and database persistence that can be used to connect to the JMX monitoring service of MigratoryData Server.

The HTTP monitoring service can be used for charting using any available RRDTool-based graphical solutions or other HTTP monitoring tools.

5.2.1 Snapshots

A service which allows you to connect to MigratoryData Server like any other client and retrieve the snapshot message of a subject (without getting subsequent messages) is also available. This service is accessible using simple HTTP requests. For example, you can use this service to check whether:

- An instance of MigratoryData Server accepts clients quick enough
- The latest data of a subject available in MigratoryData Server correspond to the latest published data

5.3 Security

The security of MigratoryData Server is assured by having:

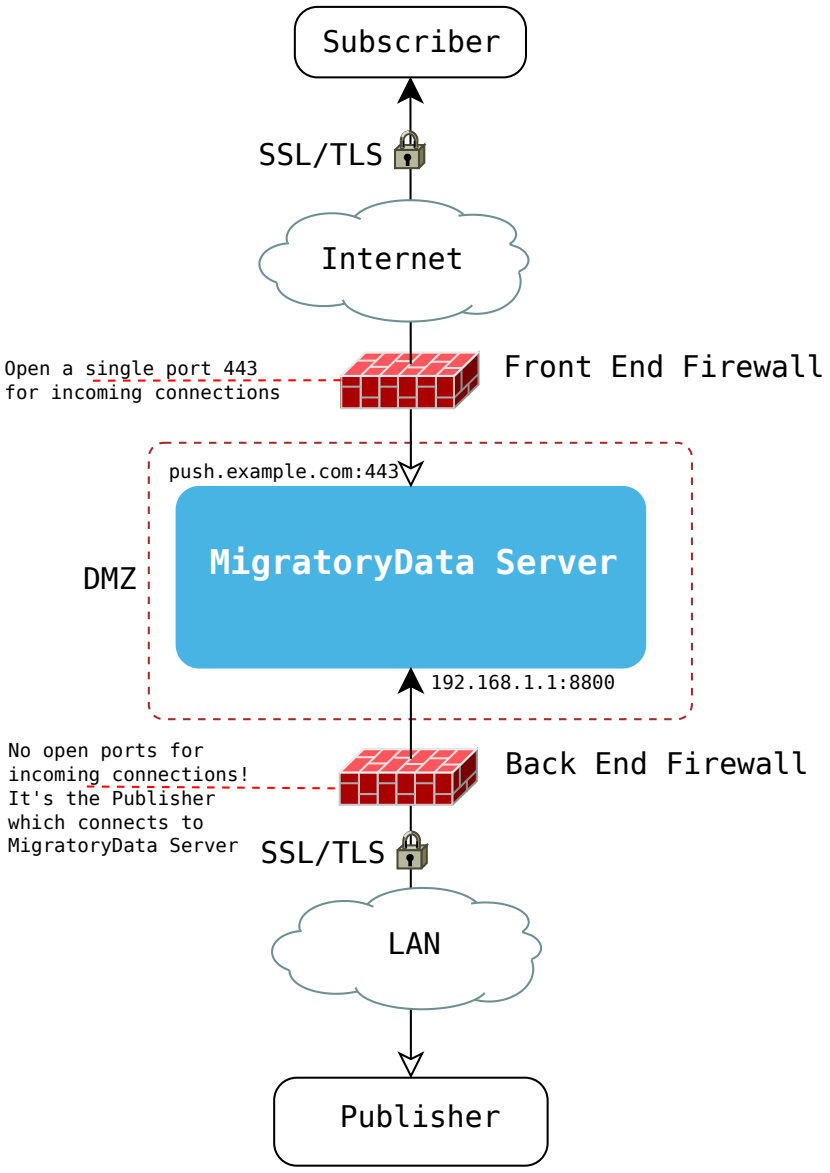
- SSL/TLS encryption of the TCP connections with the clients
- SSL/TLS encryption and authentication for both the JMX and HTTP monitoring services
- Cluster members connect each other using authentication with password key
- Entitlement (see Section 5.4)
- MigratoryData Server can be configured to run as a normal (non-root) user
- Configurable SSL/TLS ciphers to be used for the encrypted SSL/TLS connections
- Dual firewall and DMZ policy

Transport Layer Security (TLS) and Secure Sockets Layer (SSL) are cryptographic protocols that provide privacy and data integrity on TCP/IP communications (see RFC 5246). An attacker can capture the network traffic on Internet but cannot decrypt the data.

The Demilitarized Zone (DMZ), named after the military usage of the term, is a subnetwork that contains and exposes an organization's external services to a larger, untrusted network (e.g. the Internet). The purpose of a DMZ is to add an additional layer of security to an organization's Local Area Network (LAN); an external attacker only has access to equipment in the DMZ, rather than the whole of the network.

Figure 5.3 shows a secure dual firewall DMZ deployment of MigratoryData Server.

Figure 5.3: MigratoryData Secure Dual Firewall DMZ Deployment



5.4 Entitlement

The goal of the Entitlement feature is to offer a data control mechanism such that every client will access only messages with the subjects for which it was authorized to subscribe and will publish messages only for the the subjects it was authorized.

The entitlement workflow is as follows:

1. Enable the Entitlement feature in MigratoryData Server by setting its parameter `Entitlement` on `true`.
2. Use the API call `MigratoryDataClient.setEntitlementToken()` to assign an authentication token to a client. The token is any UTF-8 string (typically obtained from your backend servers following an authentication phase for that client).
3. Define your entitlement rules using the MigratoryData Extension API.

See the documentation of MigratoryData Extension API to learn how to build your own entitlement rules.

5.5 Conflation

Conflation is the process of aggregating messages with the same subject together for a period of time and sending the conflated message which results to a client. The *conflated message* for a subject `X` is defined as follows:

- The subject of the conflated message is `X`
- The content of the conflated message is the content of the most recent message with the subject `X` published during the last conflation time period
- The field name set of the conflated message is the union of the field name sets of all messages with the subject `X` published during the last conflation time period
- The value of each field of the snapshot message is the most recent value of that field published by messages with the subject `X` during the last conflation time period

When a client subscribes to a subject, it can specify, besides the subject name, a `conflationTime`. MigratoryData Server will firstly provide to the client, as in the case of a simple subscription, the snapshot message of that subject. Then it will aggregate the subsequent messages with that subject, and will publish a conflated message every `conflationTime` milliseconds if there are messages to aggregate. Note that `conflationTime` should be a multiple of `100` milliseconds. Otherwise, it will be rounded to the nearest multiple of 100 milliseconds. For example, `78` milliseconds conflation time will be rounded to `0` milliseconds (i.e. no conflation - messages are published on a one-by-one basis), `547` milliseconds conflation time will be rounded to `500` milliseconds, `887` milliseconds conflation time will be rounded to `800` milliseconds etc

The Conflation feature is especially useful for application with high volume of data such as a trading application. The financial instruments are typically represented as subjects. For some financial instruments only the last value is of interest at a given time, the previous values are not useful any more. Thus, to reduce the volume of data, certain high-volume data streaming applications may use subscriptions with conflation for some or all subjects.

Table 5.1 shows the incoming messages from publishers and the outgoing conflated messages published by MigratoryData Server to a client that subscribed with a 5-second conflation time to the subject `/Stocks/NYSE/IBM`.

Time	Incoming Message	Outgoing Conflated Message
10:12:00 (1st message for /Stocks/NYSE/IBM)	subject=/Stocks/NYSE/IBM content=193 field-volume=6.62M	NONE (client not subscribed yet)
10:12:32	subject=/Stocks/NYSE/IBM content=192 field-bid=194	NONE (client not subscribed yet)
10:12:40 (client subscribes with 5-second conflation time)	NONE	subject=/Stocks/NYSE/IBM content=192 field-volume=6.62M field-bid=194 (this is the initial snapshot message)
10:12:41	subject=/Stocks/NYSE/IBM content=195 field-ask=192	NONE (the message is aggregated in memory)
10:12:43	subject=/Stocks/NYSE/IBM content=196 field-bid=197	NONE (the message is aggregated in memory)
10:12:45	NONE	subject=/Stocks/NYSE/IBM content=196 field-ask=192 field-bid=197 (the 5-second conflation time expired, send the conflated message)
10:12:50	NONE	NONE (no new messages)
10:12:55	NONE	NONE (no new messages)
...	...	...

Table 5.1: Examples of Conflated Messages

5.6 Batching

Batching is the process of collecting messages together for a period of time or until a total size is reached before sending them in a single I/O operation to a client.

ACM

> **Note** — Unlike the Conflation feature (see Section 5.5), the Batching feature does not perform any message aggregation. When the period of time for batching expires, then *all* messages collected during the batching period are sent to the client in a single batch. On the other hand, if the Conflation feature is enabled, when the period of time for conflation expires, only a *single* message is sent to the client for each subject, which is an aggregation of all messages collected during the conflation time for that subject.

The Batching feature can be enabled or disabled in the configuration of MigratoryData Server.

To enable the Batching feature, a period of pre-configured time and/or a pre-configured size should be configured in MigratoryData Server. Once enabled, MigratoryData Server will not send individually every message to the client, instead it will send messages in batches, thus MigratoryData Server will perform a single I/O network operation for a single batch (that contains a number of messages).

Depending on your use case, batching can help you to optimize network I/O, bandwidth usage, and paradoxically even latency as explained in the following.

Batching messages implies that the individual messages will have to wait until the batching time expires or until the batching size is reached (whichever comes first, if both parameters are enabled) and then the whole bunch of messages grouped together in a batch will be sent on the network having the final destination a client.

When the rate of messages per second is high, without batching, the time spent on network I/O is significantly increased and the message latency without batching is actually worse than the latency with batching.

Figure 5.4 and Figure 5.5 show the circulation of messages without and with batching enabled.

Figure 5.4: Circulation of Messages without Batching

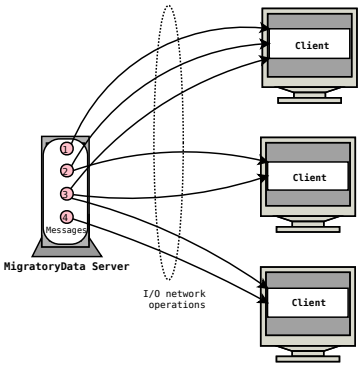
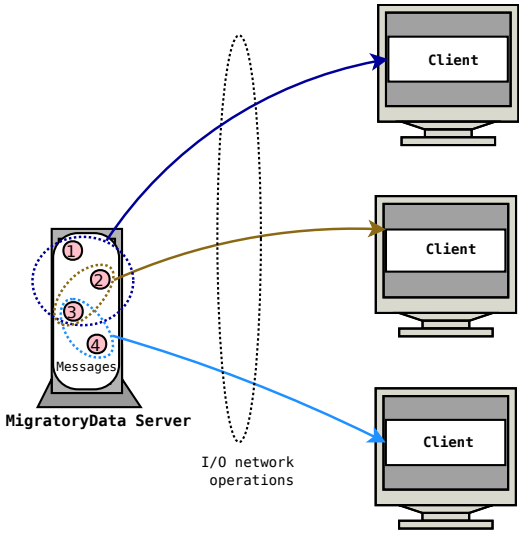


Figure 5.5: Circulation of Messages with Batching



5.7 High-Availability Clustering: Load Balancing and Fault Tolerance

You can deploy multiple instances of MigratoryData Server as a cluster to achieve:

- High Availability through *Fault Tolerance*
- Horizontal Scalability through *Load Balancing*

Both Load Balancing and Fault Tolerance are built-in features of MigratoryData Server. Thus, you don't need any load balancers or other networking hardware to achieve High Availability and Horizontal Scalability of your real-time application.

There are two clustering modes offering different qualities of service for message delivery:

- Standard Message Delivery
- Guaranteed Message Delivery

To enable Standard Message Delivery, you will need to deploy a cluster of at least two MigratoryData servers. To enable Guaranteed Message Delivery, you will need at least three MigratoryData servers for your cluster.

Both clustering modes offer reliable message delivery and high availability, including automatic client reconnection if the connection between a client and a cluster member goes down or if a cluster member goes down. Guaranteed Message Delivery offers further quality of service as detailed in Section 5.8.

5.8 Guaranteed Message Delivery

All entities which communicate with the MigratoryData server use the TCP protocol at the transport layer which is a reliable protocol. Also, in the previous section related to High Availability Clustering, we saw that MigratoryData Server can be deployed such that it will continue to work even if a sudden failure occurs.

Thus, High Availability Clustering and the use of the reliable TCP protocol already offer reliable message delivery.

Guaranteed Message Delivery offers even more reliability as follows:

- With Standard Message Delivery, when a client reconnects to another cluster member after a failover, it will get the latest messages of its subscribed subjects.
- With Guaranteed Message Delivery, when a client reconnects to another cluster member after a failover, it will get not only the latest messages of its subscribed subjects but it will also get all the messages for its subscribed subjects published during the failover period.

Figure 5.6 shows an example of data recovery with Standard Message Delivery enabled. Note that the two messages published at `10:12:00` and at `10:12:05` during the failover recovery are aggregated in the cluster member `B` and when the client reconnects to `B` it will get the most recent values for its subscribed subject `/Stocks/IBM`.

Figure 5.7 shows an example of data recovery with Guaranteed Message Delivery enabled. Note that the two messages published at `10:12:00` and at `10:12:05` during the failover recovery are received by the client when it reconnects to the cluster member `B`.

Figure 5.6: Example of Data Recovery With Standard Message Delivery Enabled

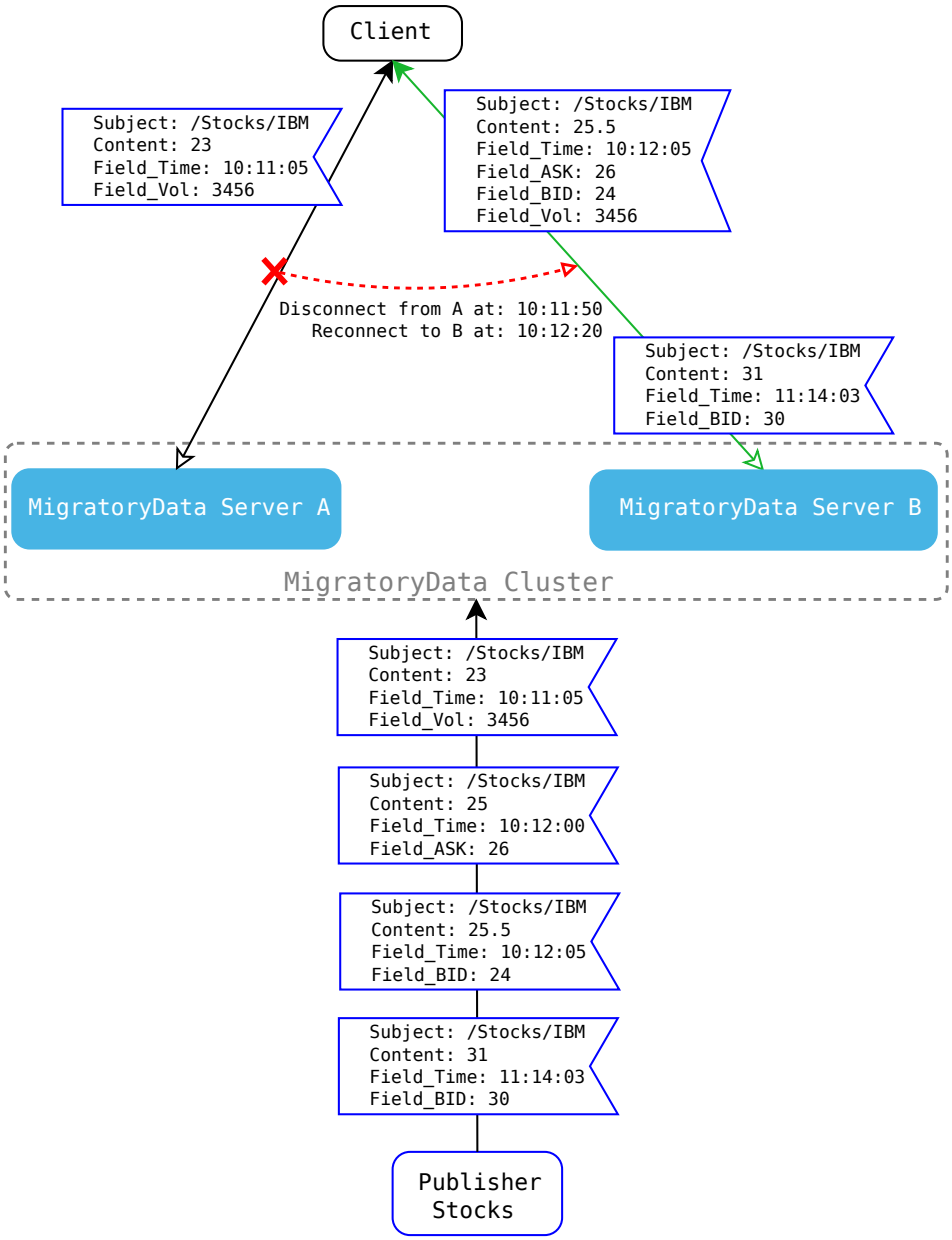
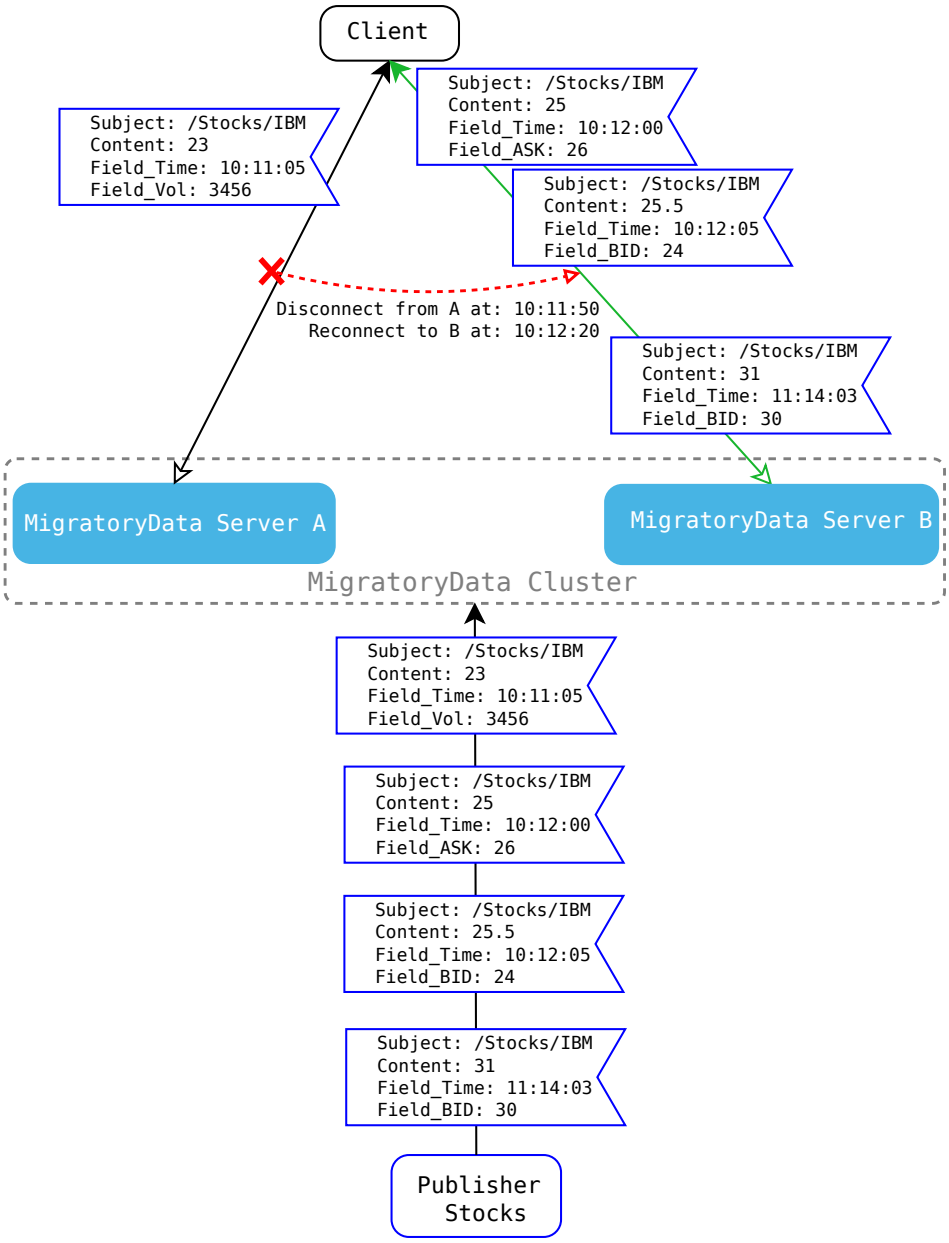


Figure 5.7: Example of Data Recovery With Guaranteed Message Delivery Enabled



5.8.1 How to Enable Guaranteed Message Delivery

Guaranteed Message Delivery is implemented using sequence numbers and a cache for messages. The sequence numbers are used to retrieve from the cache of the MigratoryData server the missing messages only when it is necessary. Thus, Guaranteed Message Delivery implementation does not use acknowledgement notifications for published messages which would add significant overhead. Thus, Guaranteed Message Delivery is very lightweight so that you can enable real-time guaranteed delivery of data with a negligible overhead.

In the MigratoryData server, configure the parameter `ClusterDeliveryMode` to `Guaranteed` as follows:

```
ClusterDeliveryMode = Guaranteed
```

While Guaranteed Message Delivery is easy to configure and use, its implementation involves sophisticated distributed algorithms to achieve in-memory cache synchronization between the MigratoryData servers of the cluster. MigratoryData Server automatically synchronizes the cache of the subjects for each cluster member, even in presence of failures. The following paper "*Reliable Messaging to Millions of Users with MigratoryData*" available at:

```
https://arxiv.org/pdf/1712.09876.pdf
```

provides more details on how Guaranteed Message Delivery is achieved internally. This is a preprint of the paper originally presented at the ACM/IFIP/USENIX International Middleware Conference 2017, University of Nevada, Las Vegas and published in the proceedings of Middleware 2017, copyright ACM, 2017.