



16.3.2001

This document assists you in optimizing iBus//MessageBus performance, and in avoiding message loss in certain high load situations. This information applies only to the iBus//MessageBus product, but not to iBus//MessageServer.

Introduction

iBus//MessageBus is the JMS Publish-Subscribe solution which fully exploits IP multicast communication. This means that a message published on a JMS topic results in a near-constant amount of IP traffic on the network, no matter how many subscribers are listening to the topic. The iBus//MessageBus architecture is fully distributed, scalable, and fault-tolerant.

iBus//MessageBus automatically assigns IP multicast addresses (Class D) to JMS topics. This is accomplished in a distributed and fault-tolerant manner. Optionally, the iBus//MessageBus developer can control the assignment of IP multicast addresses to JMS topics. This is described in the iBus//MessageBus Programmer's Manual.

iBus//MessageBus implements a fully reliable multicast protocol as well as a failure-detection mechanism. This implies that

- *all JMS messages published on a given topic are received by all TopicSubscribers subscribed to that topic, in exactly the same order as they were published. iBus//MessageBus automatically recovers from any message loss occurring at the network level, by using a negative acknowledgements multicast protocol.*
- *Optionally, iBus//MessageBus can inform the JMS application layer when an application subscribes to a topic, unsubscribes, or crashes. This allows developers to devise application specific fault-tolerance and clustering mechanism.*

more



Failure Detection

The `iBus//MessageBus` reliable multicast as well as the `iBus//MessageBus` fault-tolerance subsystem, are based on a distributed failure detection (FD) protocol. The FD protocol operates by letting each `iBus//MessageBus` application multicast a heartbeat message at regular time intervals, on a specific IP multicast channel. For example, if application A receives the heartbeat of application B for the first time, it will notify the reliable multicast subsystem (as well as the application layer, optionally) that application B is alive. From now on, A expects to receive B's heartbeat. It will permit B to delay its heartbeat from time to time, due to system overload or packet loss at the network level. However, after A has not received B's heartbeat for a certain period of time, it will inform the reliable multicast subsystem (as well as the application layer, optionally) that application B has terminated.

The key tuning parameters of the FD protocol are:

- The time interval (TI) at which an application emits its heartbeat.
- The FD missing heartbeats threshold (MH). If application A thinks it has missed MH of B's heartbeats in a row, B is tagged as "terminated".

What does it mean for A to "think" it has lost heartbeats? A notes the interval between heartbeats it receives from B and tracks the average time interval (AT). AT can differ from TI if heartbeats are lost. Using AT rather than TI helps failure detection to adapt to network conditions and overloaded CPUs, and reduces the need for special tuning.¹ When $MH * AT$ milliseconds have passed without receiving a heartbeat, A considers B to have failed.

If TI is set to a low value (e.g., 200 milliseconds), then excessive traffic is generated on the network. If TI is set to a high value (e.g., 20.000 milliseconds), then it might take application A too long to detect the termination of B. `iBus//MessageBus` uses a default TI value of 4.000 milliseconds.

¹The adaptive element was added in `MessageBus 4.1.1`. Thus MH replaces the parameter TO mention in earlier versions of this document.



If MH is set to a low value (e.g. 2) then A will tag B as failed on the slightest overload of the system. This means, A will suspect B to have terminated, even though B is still up and running but working off a peak in its application load. The FD is then said to be too "aggressive". This can lead to message loss at the application level or to other unwanted side effects. Actually this is the only situation where an iBus//MessageBus subscriber can suffer from message loss at the application level: when the producer erroneously suspects the subscriber to have failed. Tuning the FD parameters described below can solve this problem.

If MH is set to a high value (e.g. 15), then failure detection is very accurate, but it might take unacceptably long to detect a failure. iBus//MessageBus uses a default MH of 5.

Failure Detection Tuning

If your iBus//MessageBus application suffers from message loss under high load, then it's a sign that the TI or MH value is too aggressive for your application. iBus//MessageBus applications report message loss by issuing a warning such as

```
"upHandleEvent: message loss on channel /fx/quotes/MSFT".
```

In such situations we recommend leaving TI unchanged, and to increase MH to 10. This is accomplished by adding the following line to the config.ibusjms.txt file located in the iBus//MessageBus JAR file:

```
ibusjms.topic.qos=DISPATCH(threadPerChannel=1,highmark=10):FRAG:FIFO:NAK:REACH(t  
imeoutheartbeats=10):IPMCAST
```

If you have more than 50 subscriber applications (running in different Java VMs) listening to the same JMS topic, then you might want to increase both the TI and the MH value:

```
ibusjms.topic.qos=DISPATCH(threadPerChannel=1,highmark=10):FRAG:FIFO:NAK:REACH(h  
beat=8000,timeoutheartbeats=10):IPMCAST
```

more



Reliable IP Multicast

The iBus//MessageBus reliable IP multicast protocol (RMP) uses a combination of negative and positive acknowledgements. The goal of our RMP is to provide the highest possible delivery guarantee in a multicast environment, while scaling to multicast groups of up to a few 100 applications.

The iBus//MessageBus reliable multicast protocol uses a rate-based flow control scheme, reducing the message throughput when UDP datagram loss occurs at the network level. The key tuning parameters of the RMP are:

- *The negative acknowledgement (NAK) epoch size (ES)*
- *The send-delay (SD)*

The default value for ES is 200. When a TopicPublisher starts publishing on a given topic, it relies purely on NAKs sent by the subscribers, to detect that certain datagrams have to be retransmitted to the subscribers. At the end of the epoch (e.g., after ES messages have been sent), the producer waits until it has received a positive acknowledgement from each subscriber, confirming that the subscriber has received all the messages sent during that epoch. The producer will then discard its retransmission-cache containing the messages sent during that epoch, and will start a new epoch. In summary, the RMP relies purely on negative acknowledgements during an epoch, and requires exactly one round of positive confirmation, when the epoch has terminated. This allows a producer to transmit messages at full speed and with the highest possible delivery guarantee.

The default value for SD is 0. SD defines an amount of milliseconds that the RMP will use to delay the transmission of datagrams, thus artificially throttling the message transmission rate. This tuning parameter can help in decreasing message loss and thus to dramatically improve iBus//MessageBus performance in certain situations. This is explained below.

more



RMP Tuning

Experience shows that the iBus//MessageBus RMP default parameterization can turn out to be too "aggressive" in certain situations. This means that flow control might not work effectively and lead to excessive NAK retransmit requests. Note that this will not lead to JMS message loss at the application level, but the throughput and scalability of your application will suffer. If you have the impression that your iBus//MessageBus application is trashing due to internal message loss, you will first have to analyze your level of message loss, and then to adapt the ES and SD parameters if necessary.

Analyzing Message Loss

Add the following line to the config.ibusjms.txt file located in the iBus//MessageBus JAR file:

```
ibusjms.topic.qos=DISPATCH(threadPerChannel=1,highmark=10):FRAG:FIFO:NAK(statistics=1):REACH:IPMCAST
```

Now subscriber applications will print message retransmission statistics to System.err. Let your subscriber application transmit a couple of 1.000 messages, then analyze the statistics printed to the console window.

```
DEBUG[nak.ChannelInfo]: messages sent: 29008
DEBUG[nak.ChannelInfo]: retransmit req: 227 (0.7825%)
DEBUG[nak.ChannelInfo]: epoch ack ave ms: 254
DEBUG[RAMP_Flowctl]: current send delay      : 0.30257
DEBUG[RAMP_Flowctl]: ms slept this interval  : 53
DEBUG[RAMP_Flowctl]: average elapsed btwn sends: 0.47829
```

Optimizing Multicast Throughput

If, on average, the "messages lost" number is higher than 20% of the totally transmitted messages ("messages received"), then you should set the SD value to 1 and run the test again:

```
ibusjms.topic.qos=DISPATCH(threadPerChannel=1,highmark=10):FRAG:FIFO:NAK(statistics=1):REACH:IPMCAST(senddelay=1)
```

more



If message loss is still higher than 10%, then you should keep on increasing the SD value by 1 until the message loss rate stabilizes at about 10% or lower. If you have reached an SD value of 10 without stabilizing the message loss rate, then you should try to decrease the epoch size to, say, 50 messages:

```
ibusjms.topic.qos=DISPATCH(threadPerChannel=1,highmark=10):FRAG:FIFO:NAK(statistics=1,epochsz=50):REACH:IPMCAST(senddelay=5)
```

If all of this fails there might be a problem with your network card, hub, or router. Please request additional advice using <http://www.softwired-inc.com/products/support.html> .

more