DataPower XM70 Use Cases and Patterns

DataPower XM70 Overview

Messaging Patterns and Use Cases

Example Configurations

Gerald Kaplan
Cindy Claudant

Redpaper
First Edition (May 2009)

This document created or updated on May 26, 2009.
## Contents

**Notices** ................................................................. v  
Trademarks .............................................................. vi  

**Preface** ................................................................. vii  
The team that wrote this paper. ........................................ vii  
Become a published author ............................................. viii  
Comments welcome ...................................................... viii  

**Chapter 1. WebSphere MQ Low Latency Messaging overview** ........... 1  
1.1 WebSphere MQ LLM overview ........................................ 2  
1.2 Message transports .................................................. 3  
  1.2.1 Multicast ......................................................... 5  
  1.2.2 Unicast UDP ..................................................... 5  
  1.2.3 Unicast TCP ..................................................... 5  
1.3 Reliable protocol .................................................... 5  
  1.3.1 Wait1-ACK-based multicast ...................................... 7  
1.4 Availability .......................................................... 7  
  1.4.1 Reliable and consistent message streaming ...................... 8  
1.5 Message filtering .................................................... 10  
  1.5.1 TurboFlow bitmap support ...................................... 10  
  1.5.2 Message selection ............................................... 11  
1.6 Monitoring ........................................................... 11  
1.7 Congestion management .............................................. 12  

**Chapter 2. DataPower XM70 overview** .................................. 13  
2.1 General overview .................................................... 14  
  2.1.1 Reliable messaging .............................................. 16  
2.2 DataPower XM70 LLM Objects ...................................... 16  
  2.2.1 LLM Multicast Receive .......................................... 17  
  2.2.2 LLM Multicast Transmit ........................................ 17  
  2.2.3 LLM Unicast ..................................................... 18  
  2.2.4 LLM Tier Group ................................................ 18  
  2.2.5 LLM Instance ................................................... 19  
  2.2.6 LLM Policy ....................................................... 20  
  2.2.7 LLM Route ....................................................... 24  
2.3 LLM Object reuse ................................................... 27  
2.4 Message flow ......................................................... 28  
2.5 Late joining receivers .............................................. 30  
2.6 Repair ports .......................................................... 30  

© Copyright IBM Corp. 2009. All rights reserved.
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

- AS/400®
- DataPower®
- IBM®
- Redbooks®
- Redbooks (logo) ®
- WebSphere®

The following terms are trademarks of other companies:

InfiniBand, and the InfiniBand design marks are trademarks and/or service marks of the InfiniBand Trade Association.

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Expression, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.
Preface

This IBM® Redpaper publication provides a broad understanding of the WebSphere® DataPower® XM70 Low Latency Messaging appliance. After a brief introduction to low latency messaging, various messaging patterns are discussed along with configuration details for implementing each pattern. The paper wraps up with a section on troubleshooting and performance monitoring specific to the XM70.

This Redpaper publication is intended for individuals who require a better understanding of low latency messaging and how it is implemented on the DataPower XM70. It is assumed that the reader is familiar with concepts and configuration of DataPower appliances.

The team that wrote this paper

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Gerald Kaplan is an IT Specialist in the United States. He has more than 25 years of experience in software development on a wide array of IBM platforms including the System/34, Series/1, AS/400®, and desktop workstations. He has worked at IBM for three years and is a subject matter expert for the IBM WebSphere DataPower appliance family. He is the author of the DataPower Proof of Technology. His history with IBM began as an intern for Federal Systems Division at NASA on the Space Shuttle project.

Cindy Claudant is a Technical Support Engineer in the United States. She has been working at IBM for 3 years. Before joining IBM, she worked as an Instruction designer / E-Learning consultant. She holds a Bachelors degree in Computer Science and a Masters in Management Information Systems from Florida International University. Her interests include networking and MQ.

Thanks to the following people for their contributions to this project:

Chris Rayns
International Technical Support Organization, Raleigh Center

Nicholas Maniskas
IBM Software Group, Application and Integration Middleware Software
Luis Vega-Zayas  
IBM Software Group, Application and Integration Middleware Software  

Gari Singh  
Product Manager, DataPower SOA Appliances  

Become a published author  

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.  

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.  

Find out more about the residency program, browse the residency index, and apply online at:  

ibm.com/redbooks/residencies.html  

Comments welcome  

Your comments are important to us!  

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks® publications in one of the following ways:  

- Use the online Contact us review Redbooks form found at:  
  ibm.com/redbooks  
- Send your comments in an e-mail to:  
  redbooks@us.ibm.com  
- Mail your comments to:  
  IBM Corporation, International Technical Support Organization  
  Dept. HYTD Mail Station P099  
  2455 South Road  
  Poughkeepsie, NY 12601-5400
WebSphere MQ Low Latency Messaging overview

The WebSphere DataPower XM70 Low Latency Messaging (LLM) appliance is based on the IBM WebSphere MQ Low Latency Messaging program product and thus provides a nearly identical feature set. For this reason, a general understanding of WebSphere MQ LLM is beneficial when architecting and configuring the DataPower XM70.

This chapter provides an introduction and overview of the WebSphere MQ LLM program product and LLM concepts.

**Note:** Because the names of the DataPower LLM objects are similar to words used when discussing networking, DataPower, and messaging, this book will reference specific LLM objects with a capital letter, while the generic meanings will remain in lower case. For example, when discussing an LLM Policy, this book will use the word Policy (capitalized). Similarly, a Route refers to an LLM Route object, and an Instance refers to an LLM Instance object.
1.1 WebSphere MQ LLM overview

IBM WebSphere MQ LLM is a messaging transport that is highly optimized for the high-volume, low-latency requirements of financial markets firms. Applications include the high-speed delivery of market data, transactional data, reference data, and event data in or between front-, middle-, and back-office. Although initially designed to meet the high-speed and throughput requirements of financial services firms, WebSphere MQ LLM is suitable for use by other industries with similar requirements. Characteristic applications require extremely low (submillisecond) latency and high message volumes (ranging from many thousands to millions of messages per second), with positive or negative acknowledgement reliability, but not requiring the assured delivery provided by WebSphere MQ.

WebSphere MQ LLM is a high-throughput, low-latency transport fabric designed for one-to-one, one-to-many or many-to-many data exchange in a message-oriented middleware publish/subscribe fashion. LLM exploits the IP multicast infrastructure to ensure scalable resource conservation and timely information distribution. Both Ethernet and InfiniBand® networks are supported.

Reliability and traffic control are added on top of high performance UDP and TCP networking implementations. LLM takes this one step further and enables the support of highly available unicast or multicast data distribution, by implementing a number of stream failover policies that allow seamless migration of transmission from failed to backup processes.

In summary, WebSphere MQ LLM offers the following advantages:

- Multiple and flexible message transports
- Reliability
- High Availability and reliability in a tiered, replicated environment using LLM’s Reliable and Consistent Message Streaming (RCMS) functionality:
  - Component replication, automatic failure detection and failover.
  - Total order sequencing.
  - Automatic state synchronization for joining applications.
  - Single acknowledgment multicast reliability.
  - Support for multiple levels of redundancy, with inter-tier communications.
  - Support for both duplex and replay modes.
- Message filtering
- Monitoring
- Congestion management
- High Performance
- Support for multiple platforms and programming languages
1.2 Message transports

LLM provides a multicast transport for high-speed, one-to-many communications using the User Datagram Protocol (UDP) with receiver feedback. Although typical multicast implementations offer only best-effort, unreliable message delivery, the addition of delivery options for receiver feedback enables reliable delivery with minimal loss of speed. See Figure 1-1.

Multicast transmissions are typically used to implement a publish-subscribe topology. A server publishes messages to a multicast group with a given topic. Subscribers filter incoming streams and accept only multicast groups and topics of interest. The publisher (or transmitter) has no knowledge of how many subscribers are listening. Subscribers provide either positive (ACK) or negative (NAK) acknowledgement depending on configuration parameters.

![Figure 1-1  Reliable multicast transport](image)

In addition to multicast, WebSphere MQ LLM offers two transports that provide the ability to deliver a stream of data across a WAN or through a firewall reliably, at high speeds.

The first alternative is a lightweight, point-to-point UDP transport with either positive- or negative-feedback reliability and traffic control features similar to the multicast offering. With positive acknowledgment, all packets are acknowledged, whereas negative acknowledgment provides feedback only if a packet is lost. See Figure 1-2 on page 4.

Unicast transmissions are destined for a specific client. The server has explicit knowledge of each client that is receiving messages. Unlike multicast messages where the message is destined for a multicast group, unicast messages are destined for a specific IP address. Receivers provide either positive (ACK) or negative (NAK) acknowledgement depending on configuration parameters.
The second alternative offers reliable, point-to-point, unicast messaging using the TCP/IP protocol, in which reliability and traffic control are primarily handled by the TCP protocol. These alternatives provide the ability to deliver a stream of data across a wide area network (WAN) or through a firewall reliably, at high speeds. See Figure 1-3.

The two point-to-point extensions of RMM are designed for applications that need point-to-point communication and applications that must run over an infrastructure where IP multicast is not available.

WebSphere MQ LLM provides the following three modes of communication:

- Multicast
- Unicast UDP
- Unicast TCP
1.2.1 Multicast

This communication mode is referred to as Reliable Multicast Messaging (RMM). RMM exploits the IP multicast infrastructure to ensure scalable resource conservation and timely information distribution. RMM implements different levels of reliability on top of the standard multicast infrastructure. Multicast is delivered using either UDP or raw IP datagrams.

1.2.2 Unicast UDP

This mode is an extension to RMM that enables point-to-point communication over the UDP protocol. RMM provides different levels of reliability including ACK- and NAK-based reliability on top of UDP. The unicast UDP and multicast modules of RMM use the same set of APIs and are integrated into the same library. Unicast UDP streams have the advantage that they provide little overhead in terms of both configuration and communication. The use of UDP provides greater flexibility over traffic control and the level of reliability. Unicast UDP streams can be configured to work in lightweight mode, in which the overhead of the messaging middleware is reduced, allowing hundreds of transport streams to be supported.

1.2.3 Unicast TCP

This mode is referred to as Reliable Unicast Messaging (RUM). This mode provides the same messaging service on top of regular TCP sockets. The reliability and traffic control are handled by the TCP protocol. Using a TCP-based protocol has the advantage that it is more suitable for WAN communication and can be used to communicate even in the presence of firewalls. RUM exploits the reliability and flow and congestion control features of TCP, which means it can consume less memory. RUM adds high availability on top of the TCP reliability.

1.3 Reliable protocol

WebSphere MQ LLM features a reliable protocol. A reliable protocol is one that ensures reliability properties with respect to the delivery of data to the intended recipients. Reliable in the name Reliable Multicast Messaging (RMM) originally referred to the ability to tolerate transient network disruption or application failure without compromising the integrity of the data stream.
Common examples are as follows:

- Packets lost in the network.
- Packets re-ordered in the network or kernel.
- Temporary network disconnections.
- Packets lost in the kernel (usually caused by application failure to consume messages fast enough).
- The reliability of RUM is handled by TCP. Reliability for RMM is due to the inherent unreliability of UDP.

The reliability level is defined on a per-topic basis. Reliability levels you can set for a transmitter are as follows:

- Unreliable (messages are sent only once with no retransmissions).
- Reliable (retransmission is enabled).
- Reliable using the failover server as primary or backup.

Reliability levels you can set for a receiver are as follows:

- Unreliable (no requests are made for messages that were not received).
- Reliable-ordered or reliable-unordered (indicates whether messages are to be received in their original order).
- Reliable failover (re-sent messages are received in their original order and can be retrieved from the backup server).

RMM achieves higher levels of reliability by buffering outgoing data and incoming data. The total memory footprint of RMM and the individual buffer sizes are highly configurable.

Buffering outgoing data in the history buffer tolerates traffic spikes that may overwhelm system buffers and allows for retransmission of missed data.

Buffering incoming data tolerates lost messages or spikes in message traffic that may overwhelm system buffers due to the application’s ability to consume messages at pace.

RMM supports both negative and positive acknowledgment (NAK and ACK, respectively) modes of receiver feedback.

- NAK mode is supported on both multicast and unicast topics. It allows receivers to request a retransmission of missed data from the transmitter’s history buffer. Messages are removed from the transmitter’s buffer when it is full and new messages are sent. As a result, unrecoverable packet loss can occur if the receiver sends a NAK for a message that has already been removed from the transmitter’s buffer.
ACK mode is supported on both multicast and unicast topics. It requires a positive acknowledgment of each packet the receiver receives before the packet is removed from the transmitter's history buffer. This requires more transmitter resources, but can ensure no message loss in the case of application failure. ACK mode eliminates the chance of unrecoverable packet loss and ensures no loss of messages due to a failure in the receiver application.

1.3.1 Wait1-ACK-based multicast

LLM offers Wait1-ACK based multicast as a way to increase the reliability of NAK-based multicast streams, and has a backpressure mechanism that can prevent data loss.

In Wait1-ACK-based multicast, the RMM transmitter needs to get at least one ACK before it can remove packets from the stream's history queue. This ensures that when the transmitter clears a packet from the queue, that packet has been received by at least one receiver. Wait1-ACK is much lighter in weight compared to standard ACK-based multicast where the transmitter should get an ACK from all the receivers before it can remove the packet from the history queue. Thus, compared to full ACK-based multicast, Wait1-ACK provides a lower level of reliability but avoids all the complications and scalability issues associated with full ACK-based multicast.

The feature is particularly suitable for solutions such as RCMS where components are replicated for high-availability (HA) reasons. In RCMS the sender is typically sending data to a tier that is made of two or more components that are a replica of one another. As long as the data is received by one of the servers it will not be lost. However, if all the servers are not receiving data, no ACKs will be sent and the transmitter stops sending new data on the stream.

1.4 Availability

RMM supports highly available messaging features with several stream failover reliability levels that allow for active-active and active-backup transmitter configurations. Receivers can switch from one transmitter stream to another with little or no message loss when a transmitter fails. Stream failover requires transmitter coordination to synchronize message sequence numbers from the application developer.
1.4.1 Reliable and consistent message streaming

Reliable and consistent message streaming (RCMS) enhances the HA capabilities of RMM by using component replication. This means that multiple instances of the application run simultaneously. RCMS ensures that all instances of the application get exactly the same input and can therefore maintain an identical state. If one of the instances of the application fails, RCMS detects the failure and automatically takes corrective actions. RCMS also provides the service of synchronizing a new instance of the application (for example, to replace an instance that failed).

The main features of RCMS are as follows:

- **Fast failure detection and failover**
  
  RCMS supports highly available message streaming. A solution developer can replicate an application instance for fault tolerance purposes, and use RCMS tools to manage the group of replicated components automatically. RCMS detects a component failure and automatically takes corrective actions.

- **Total ordering**
  
  This feature enforces a consistent order of message delivery from a number of independent data transmitters to multiple receivers, so all receivers see exactly the same order of incoming messages.

- **New component priming**
  
  RCMS allows dynamic addition of a new application component to a tier of existing components. RCMS automatically synchronizes the state of the components' incoming and outgoing traffic and the state of the application itself. As a result, the new component can start full functioning in parallel with its existing running peers, and support them in case of failure.

- **Management of failover streams**
  
  RCMS can be configured to work in either duplex or replay mode. RCMS manages the RMM stream so that in replay mode only one of the components in the tier is actually sending data out while in duplex mode all components are sending the data.

- **Easy to use and a set of additions to the RMM API**
  
  RCMS is built as an extension to RMM so that most of the RMM APIs (monitoring, congestion management, and so on) can be used while working with RCMS.
Tiers
A tier consists of a group of components (that is, the tier members) where each component is an identical replica. Each replica executes the application's logic as though it was the only component. RCMS connects the tier members and ensures that the application remains available in the case of a failure. The application can define the level of redundancy it wants to use. If there are X tier members running, then even if X-1 members fail at the same time, the application continues to function normally.

A tier instance includes an RMM receiver instance and an RMM transmitter instance. The two instances are created automatically when the tier is created. The application can access the receiver and transmitter instances of the tier to execute any of the RMM APIs.

A tier topic is a topic that RCMS maintains identically in all tier members. For receiver tier topics, in all tier members, all the tier receiver topics deliver exactly the same set of messages to the application. If the tier enforces total order, all the messages from all topics are delivered in exactly the same order. For transmitter tier topics, all the topics send the same set of messages and RCMS automatically handles the reliability level of these topics to work in either primary (sending data to the network) or backup mode (just buffer data in RMM).

Figure 1-4  Tiers

A tier topic is a topic that RCMS maintains identically in all tier members. For receiver tier topics, in all tier members, all the tier receiver topics deliver exactly the same set of messages to the application. If the tier enforces total order, all the messages from all topics are delivered in exactly the same order. For transmitter tier topics, all the topics send the same set of messages and RCMS automatically handles the reliability level of these topics to work in either primary (sending data to the network) or backup mode (just buffer data in RMM).
Intra-tier communication
Intra-tier communication is designed to provide the application a tool by which it can coordinate actions taken by the different tier members. Intra-tier communication is performed by a synchronized exchange of messages between tier members. Intra-tier communication can be used for example to coordinate time-based events and non-deterministic actions performed by the application.

1.5 Message filtering

WebSphere MQ LLM allows fine-grained message filtering. Filtering is performed by RMM and RUM using the TurboFlow or message selection technology.

Filtering within RMM occurs at three levels:
- Multicast group filtering is performed by the network interface card (NIC).
- Topic name filtering is performed by RMM on a per-packet basis.
- TurboFlow label filtering is performed by RMM on a per-message basis.

Filtering within RUM occurs at three levels:
- An isolated unicast connection.
- Topic name filtering is performed by RUM on a per-packet basis
- TurboFlow label filtering is performed by RUM on a per-message basis

The proper filtering scheme will be application-dependent. A market data application, for example, filters as follows:
- Feeds (and sometimes feed sources) by multicast group
- Redistributable data products by topic
- Instruments by TurboFlow label

1.5.1 TurboFlow bitmap support

You can filter messages based on a bitmap that is attached to each message. RMM will filter messages by performing a bitwise AND between the bitmap provided in the message and the bitmap filter the application provided; only messages for which the AND operation yields a result different than zero will be delivered to the application.

If the length of the bitmap in the message and the length of the filter bitmap the receiver has are different, the first bytes of the shorter bitmap will be padded with zeros to match the length.

**Note:** TurboFlow bitmap support in LLM is for RMM only.
1.5.2 Message selection

Message selection technology enables an application to set properties on each outgoing message and performs message filtering based on these properties on the receiver side.

The message selection mechanism can be used by the application to implement a solution to other requirements:
- Wildcard subscription
- Message authorization and entitlement

Message selection is implemented in both RMM and RUM.

1.6 Monitoring

WebSphere MQ LLM provides internal transmitter and receiver latency monitoring.

Latency monitoring provides information about the internal latency within the transmitter and receiver. It also provides statistics on the status of key queues, which may indicate accumulation of packets or messages leading to increased latency.

The application can set (dynamically) the latency monitoring parameters (in particular the monitor level) to balance between the resources consumed by the monitoring process and the amount of information it provides.

WebSphere MQ LLM provides access to the following statistics:
- Instance statistics
  General transmitter or receiver instance statistics such as overall message rate, bytes sent, buffer and memory utilization, and current configuration.
- Topic statistics
  Receiver topic statistics such as accepted streams, queue utilizations, and topic configuration.
- Stream statistics
  Transmitter or receiver stream statistics on information such as messages and packets processed, received, filtered, duplicated or lost, NAKs generated, and stream configuration.
1.7 Congestion management

WebSphere MQ LLM offers congestion management and traffic control.

Receiver-side congestion management features allow you to limit the amount of messages that RMM retains in the topic buffer based on space or time limits. This allows the receiver application to trim stale data automatically. Congestion due to loss is managed by configuration parameters that limit the rates at which NAKs are sent and the timeout value after which data is reported as lost.

Congestion management features on the transmitter side allow you to limit the number of times RMM attempts to request ACKs. You can also limit the amount of data that is held by RMM for topic history.

A transmitting application can monitor the behavior of its receivers and take actions against certain receivers that are reducing the performance of topic or causing congestion. Typically, an application would want to suspend or expel receivers that are sending too many NAKs and are thus overloading the network and slowing other receivers.
DataPower XM70 overview

This chapter provides a detailed introduction to the WebSphere DataPower XM70 Low Latency Messaging (LLM) appliance. In addition to a general overview of the appliance, it contains detailed descriptions of the LLM objects required to build reliable LLM configurations.
2.1 General overview

The IBM WebSphere DataPower XM70 Low Latency Appliance simplifies, consolidates, and accelerates LLM environments. The XM70 is a purpose-built LLM hub for simplified deployment, content-based routing, and extreme performance.

The XM70 extends the IBM WebSphere DataPower XI50 Integration Appliance by adding protocol support for high-performance Reliable Multicast and unicast messaging. As an architectural extension to the XI50, the XM70 inherits the XI50's rich feature set such as high speed encryption/decryption, digital signature processing, XSL acceleration, message filtering, and built-in support for a wide array of security services. The XM70 maintains the XI50's portfolio of supported protocols such as HTTP(S), WebSphere MQ, WebSphere JMS, FTP(S), and Tibco EMS.

![Multicast and unicast processing](image)

Unlike the XI50, the XM70 includes built-in support for the WebSphere MQ LLM protocol and can be configured to both receive and transmit messages through the simple web-based user interface (WebGUI). Much like the XI50, protocols can be mixed and matched as necessary, resulting in highly flexible messaging solutions that can be tailored to almost any messaging requirement. For example, an LLM Policy can listen for a specific topic on a multicast group, and upon receiving a message, transform it and put it on a WebSphere MQ queue.

See Figure 2-2 on page 15.
Chapter 2. DataPower XM70 overview

15

Figure 2-2 Protocol bridging between RMM, WebSphere MQ, and HTTP

The WebSphere MQ LLM protocol provides a reliable transport that is built on top of UDP. This results in reliability for both point-to-point UDP and multicast transmissions. The XM70's support for these reliable protocols allows it to participate as a transmitter or receiver in MQ LLM environments, or act as a conduit into MQ LLM environments for non-LLM protocols such as HTTP and WebSphere MQ.

Unlike traditional queue-based messaging scenarios, the concept of a server is non-existent in high-speed messaging environments. Messages flow from peer to peer with no intermediary. As a result, it is the responsibility of both the transmitter and receiver to manage the details associated with reliability. WebSphere MQ LLM provides a runtime library, and a published API for both C and Java™ programming languages that help developers make use of this functionality.

The DataPower XM70 accelerates time to market by providing a configuration-based approach, rather than a programmatic one. There are no APIs or functions to call and no source code to edit or maintain. The entire solution is created using the XM70's drag-and-drop graphical interface. Once a change is entered in a form, it becomes active immediately without the need for recompilation. For example, if it becomes necessary for an XM70 LLM Policy to listen for additional topics, the additional topic names can simply be added to the policy. No other changes would be required.

DataPower XM70 appliances can work in concert with one another, providing stream failover for high availability. This ensures no messages are lost in the event of a device failure.
In summary, the DataPower XM70 LLM Appliance offers the following features:

- **Appliance model**
  - Predictive, low-latency, drop-in, network-based messaging
  - User interface (WebGUI) for configuration rather than code libraries
  - Advanced fast path routing
  - Message transformation capability using standard DataPower policies
  - Protocol bridging

- **Multicast and unicast**
  - Reliable UDP Multicast (RMM)
  - Reliable UDP Unicast (point-to-point RMM or RUDP)
  - Reliable TCP Unicast (RUM)

- **Low latency and high throughput**

- **Reliable**
  Zero or minimal messages lost due to transient network or application failure

- **Stream failover for high availability**
  Zero messages lost during failover

- **Fine-grained message filtering**
  Allows for many logical message flows

- **Statistics and performance monitoring**

- **Congestion and traffic rate control**

### 2.1.1 Reliable messaging

The WebSphere DataPower XM70 implements a reliable transmission and receiving protocol. A reliable protocol is one that ensures reliability properties with respect to the delivery of data to the intended recipients. For more information about reliable protocols, see 1.3, “Reliable protocol” on page 5.

### 2.2 DataPower XM70 LLM Objects

There are seven basic object types that are utilized when creating reliable LLM configurations on the XM70. It is important to understand these objects and their relationship to each other.
2.2.1 LLM Multicast Receive

This object defines the specific network details associated with receiving UDP streams. It contains a single network interface and port that will be used to receive streamed messages for both UDP unicast (RUDP) and multicast (RMM).

2.2.2 LLM Multicast Transmit

This object defines the specific network details associated with transmitting UDP streams. It transmits both UDP unicast (RUDP) and multicast (RMM) transmissions.
The Multicast Transmit object contains a collection of the network interfaces used for outbound transmission. Messages are forwarded to the Multicast Transmit object from the Instance and are published on all of the specified network interfaces. The port specified in this object identifies the port on which the receiver will be listening.

### 2.2.3 LLM Unicast

The LLM Unicast object contains both receive and transmit configurations for Reliable Unicast Messaging (RUM) and enables the sending of low latency messages to a single receiver using a TCP transport. RUM is for applications that need point-to-point communications, or an infrastructure where IP multicast is not available. Message reliability and traffic control are handled by the underlying TCP protocol, ensuring that the receiver does not lose any data. RUM exploits the reliability, flow, and congestion control features of TCP, resulting in a smaller memory footprint. RUM adds high availability on top of the TCP reliability.

### 2.2.4 LLM Tier Group

The LLM Multicast Tier Group identifies a configuration that combines a group of two or more machines called tier members into a tier failover group for stream-based Reliable and Consistent Message Streaming (RCMS). A tier consists of a group of application components in a configuration where each component is an identical replica. Each replica executes the application logic as if it was the only component. RCMS, by synchronizing inbound and outbound traffic for each tier member, ensures that the application remains available in the case of a failure. As a result, any new tier component can start fully functioning in parallel with its existing running peers, and support them in case of failure.

RCMS is a part of LLM that provides high availability and consistent, ordered message delivery. It supports the following features:

- Failure detection and failover
- Total ordering
- State synchronization
- Component group management

RCMS is based on a component replication model where all members of a tier group perform the following actions:

- Receive the same input
- Perform same processing
- Produce the same output
RCMS enhances the high availability capabilities of RMM by ensuring that all instances of the application get exactly the same input and therefore can maintain identical state. For receiver-tier topics this means that in all tier members all the tier-receiver topics deliver exactly the same set of messages to the application. If the tier enforces total order, all the messages from all topics are delivered in exactly the same order. Multiple instances of the application run simultaneously. If one of the instances of the application fails, RCMS detects the failure and automatically takes corrective actions (for example, to replace an instance that failed).

Stream failover allows receivers to switch from one transmitter stream to another with little or no message loss when a transmitter fails. Both RMM and RUM support stream failover. RCMS implements basic stream failover by coordinating transmitters and synchronizing application state and message sequence numbers.

### 2.2.5 LLM Instance

The LLM Instance is a container object that consolidates interface/port-related objects for inbound and outbound communications. In one respect, it is analogous to a typical Front-side Handler (FSH) in that it manages inbound communications, however it is different in that it also manages outbound communications. The Instance acts as a container for the following LLM objects:

- **LLM Multicast Receive**
  - Used by Reliable Multicast (RMM) and Reliable UDP protocols.
- **LLM Multicast Transmit**
  - Used by Reliable Multicast (RMM) and Reliable UDP protocols.
- **LLM Unicast**
  - Used by Reliable Unicast Message (RUM) protocol.
- **LLM Tier Group**

A Policy subscribes to an instance for a specific type of stream. For example, if the Policy’s specified protocol is RMM, it will only receive streamed messages from the associated Instance’s Multicast Receive object (even if the Instance has a Unicast receive interface defined). Similarly, the Route will forward outbound messages to an Instance for its defined protocol. Therefore, if a Route defines its protocol as Reliable Unicast, the associated Instance will transmit the message using the Unicast transmit object.
2.2.6 LLM Policy

The LLM Policy acts as the main coordinator for message processing. It defines which protocol to use for inbound messages (RMM, RUM, RUDP, and Local), which LLM Instance to receive messages from, and which messages to accept through topic and message content filters. It specifies one or more LLM Routes where messages should be forwarded for transmission. The Policy itself does not define the interfaces used for communication. Rather, it specifies an LLM Instance object that manages the networking details. Essentially, the LLM Policy is subscribing to the LLM Instance for messages.

A Policy can subscribe to messages on one of several different protocols:

- **Reliable Multicast Message (RMM)**
  When this protocol is selected, the Policy will receive messages from the LLM Multicast Receive object identified in the Instance. The Policy will only accept messages for the specified multicast group.

- **Reliable Unicast Message (RUM)**
  When this protocol is selected, the Policy will receive messages from the LLM Unicast object identified in the Instance. The Policy can override the interface to use for inbound messages. However, the inbound port is specified in the LLM Unicast object.

- **Reliable UDP Message**
  When this protocol is selected, the Policy will receive messages from the LLM Multicast Receive object identified in the Instance. The Policy can override the interface to use for inbound messages, however the port is specified in the LLM Multicast Receive object.

- **Reliable Local Message**
  Specifying this protocol directs the policy to subscribe to locally generated messages. Local messages may originate from a LLM Route, Multi-Protocol Gateway, or Web Service Proxy.

**Topic selection and filtering**

Upon receiving a message, the LLM Policy must decide whether to process or ignore the message. The selection process follows this order:

1. If the protocol is RMM, the inbound message must be for the same multicast group as specified in the Policy, otherwise the message is ignored.

2. If the Policy specifies a Topic Selection field, the SQL92 expression will be evaluated and, if FALSE, the message will be ignored. If the Policy does not specify a Topic Selection, the Topic Name is compared to the topic in the message and if they are not equal, the message is ignored.
3. The Message Selection expression is evaluated and if FALSE, the message will be ignored.

4. If the message has not been ignored in the previous three tests, the message is accepted by the Policy.

The Topic Selection field is used to determine whether to accept or ignore an inbound message based on its topic and some additional transport-related details. The topic selection is expressed as an SQL92 expression. A topic selection string matches when the expression evaluates to TRUE. Example 2-1 shows several topic selection SQL92 expressions that will accept messages for topics ibmDataPower, ibmWebSphere, and ibmTivoli.

**Example 2-1  Topic selection**

```
llm:Topic LIKE 'ibm%'
llm:Topic IN ('ibmDataPower', 'ibmWebSphere', 'ibmTivoli')
```

Topic selection isn't limited to just inspecting the topic. Selection can be based on a variety of criteria. Table 2-1 shows the permissible identifiers that can be used in the Topic Selection field.

**Table 2-1  Topic selection criteria**

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>llm:Topic</td>
<td>The topic or queue name on which the message is transmitted.</td>
</tr>
<tr>
<td>llm:LocalAddr</td>
<td>The local address on which the message arrived.</td>
</tr>
<tr>
<td>llm:LocalPort</td>
<td>The local port on which the message arrived.</td>
</tr>
<tr>
<td>llm:SourceAddr</td>
<td>The source address from which the message was transmitted.</td>
</tr>
<tr>
<td>llm:SourcePort</td>
<td>The source port from which the message was transmitted.</td>
</tr>
<tr>
<td>llm:Authorization</td>
<td>A string sent from the transmitter to identify the sender. This can be used to authenticate the stream.</td>
</tr>
</tbody>
</table>

**Message selection and filtering**

The Message Selection field gives the Policy additional filtering capabilities by allowing inspection of the message payload. SQL92 expression syntax is used to inspect the message contents and make the decision to accept or reject.

A message selector expression consists of one or more field names and operators. It can contain identifiers for input properties, local variables, and parsed content values.
A message selector string matches a message when the selector expression evaluates to TRUE. The message selector shown in Example 2-2 evaluates to TRUE when property myprop starts with the letters "ibm" (for example. ibm, ibmDataPower, ibmTivoli).

Example 2-2  Example Message Selector

myprop like 'ibm%'

If the inbound message has a property named myprop and it contains any value starting with the letters "ibm", the message selector in Example 2-2 will evaluate to TRUE and the Policy will accept it. The order of evaluation of a message selector is from left to right within precedence level. Use parentheses to change the order of evaluation. The default value for the message selector is TRUE, resulting in all inbound messages being accepted.

Table 2-2 shows the various namespaces and identifiers that can be used in the message selector.

Table 2-2  Message selector namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>If a namespace is not provided, the name is assumed to be a property from the input message. For example, myprop would retrieve the value of the myprop property from the message.</td>
</tr>
<tr>
<td>fix</td>
<td>The name is either a number or the canonical name of a FIX field. For repeating fields, a value in square brackets ([]) starting with 1 can be used to specify which instance.</td>
</tr>
<tr>
<td></td>
<td>fix:8</td>
</tr>
<tr>
<td></td>
<td>FIX 8 (same as BeginString)</td>
</tr>
<tr>
<td></td>
<td>fix:BeginString</td>
</tr>
<tr>
<td></td>
<td>The FIX BeginString field</td>
</tr>
<tr>
<td></td>
<td>fix:216[3]</td>
</tr>
<tr>
<td></td>
<td>The third occurrence of FIX field 216</td>
</tr>
<tr>
<td>Namespace</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| llm       | The following fields are available in the LLM namespace:  
  - llm:0 to 32767  
    LLM property  
  - llm:TopicLLM  
    Topic name  
  - llm:Label  
    LLM Label  
  - llm:Length  
    Length (in bytes) of the message  
  - llm:Sequence  
    The sequence number of the message  
  - llm:Time  
    Current time stamp  
  - llm:From  
    Source IP (if known)  
  - llm:Text  
    The text of the message  
  - llm:Domain  
    The name of the domain  
  - llm:Instance  
    The name of the LLM Instance object  
  - llm:Route  
    The name of the LLM Route object  
  - llm:Policy  
    The name of the LLM Policy object  
  - llm:Bitmap  
    The Turboflow bitmap  
  - llm:Byte[#]  
    Integer value of indexed byte in the message. |
| var       | The field is a variable. Field names with an integer value from 0 to 9 are local variables, otherwise they are global variables. |
| xml       | A simple value (not a list or nodeset) from the content parse of an XML document. This can represent the name of an element, the value of an attribute, or the first content value of an element.  
  - "xml:/"  
    "Content of root element. Quotes are needed so the slash is not interpreted as a division operator.  
  - xml:symbol  
    Contents of the first element named symbol.  
  - xml:symbol[2]  
    Contents of 2nd occurrence of element symbol.  
  - xml:symbol@v  
    Value of attribute v from first symbol element.  
  - xml:/root/next/  
    Name of element starting at the root and within next.
Overriding inbound network interfaces
The Policy has provisions to allow overriding the inbound network interface for both RUM and RUDP protocols. This allows a single instance to be used by multiple Policies, where only the inbound interface varies but the other settings remain the same. This reduces the number of instances to be defined.

2.2.7 LLM Route
LLM Policies forward accepted messages to one or more LLM Routes. If the message selection statement evaluates to TRUE, the Route will assign a topic, process the message, and forward it to its designated Instance for transmission. This can be the same Instance as specified in the Policy, or a different one altogether.

The route supports the following outbound protocols:

- Reliable Multicast Message (RMM)
  For this outbound protocol, a topic name, multicast group, and data port are specified. Messages will be transmitted using the associated Instance’s Multicast Transmit object to the multicast group and data port identified in the Route. The data port specified in the Route is the same port that the receivers should be listening on. If the data port is not specified in the Route, the data port from the Instance’s Multicast Transmit object will be used.

- Reliable Unicast Message (RUM)
  For this outbound protocol, a topic, destination, and port are specified. Messages will be transmitted using the associated Instance’s Unicast object to the destination address and port identified in the Route. If a port is not specified in the Route, the port from the Unicast object will be used.

- Reliable UDP Message
  For this outbound protocol, a topic, destination, and port are specified. Messages are transmitted using the associated Instance’s Multicast Transmit object to the destination address and port identified in the Route. If a port is not specified in the Route, the port from the Multicast Transmit object is used.

- Reliable Local Message
  For this protocol, a topic is specified. The message will be rebroadcast within the same device for consumption by a LLM Policy or MQ LLM Front Side Handler on the same device.

- Null Routing
  This selection is primarily for debugging and prevents any routing of messages.
Message filtering
Like the Policy, a Route can be configured with message selection criteria for outbound message filtering. At first, this may seem redundant to the filtering provided in the Policy, but a Route can be used by multiple Policies and cannot rely on the filtering performed by any one particular Policy. In addition, a single Policy may employ many Routes, each of which may broadcast a specific variation of the original message.

Figure 2-3 illustrates message filtering at the Route level. In this scenario, a Policy receives a message with a topic of ibm. The message contains a property named myProp. The Policy forwards the message to two Routes. The first Route uses message selection to accept only messages where myProp equals DataPower, and broadcasts them with a new topic of ibmDP. The second Route uses message selection to accept only messages where myProp equals WebSphere, and broadcasts them with a new topic of ibmWAS.

![Message filtering based on property](image)

Property mapping and variables
Property mapping rules control the propagation of properties from inbound to outbound messages. They are optional and specified in the Route object.

The Copy Message Properties toggle, when enabled, causes message properties to be copied from the original received message to the output properties of the outbound message. A property map allows you to change or suppress the propagation of properties. Typically, outbound properties get converted as needed based on the selected output protocol. The TurboFlow label and bitmap are always copied from input to output independently of the copy properties setting.

Mapping rules provide a set of actions for modifying certain properties. Each rule consists of a property name or field that you can specify, such as llm:Length, followed by a new value, which can be a SQL92 expression to be evaluated. If the SQL92 expression is blank or resolves to unknown, no assignment is done, and the specified property is removed from the output. Otherwise, expressions used in mapping rules resolve to integers, strings, or floating points.
Mapping rules execute in a sequential order. Output properties can reside in the unspecified, llm, or var namespaces. Properties in the fix and xml namespaces cannot be mapped. You can modify or suppress TurboFlow values using a map properties rule. After configuring a field, such as llm:Length, all future references to that field will use its modified value. Table 2-3 show the various identifiers that can be used in property mapping rules.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>llm:Label</td>
<td>Sets the TurboFlow numeric label.</td>
</tr>
<tr>
<td>llm:Length</td>
<td>Sets the length of the message in bytes.</td>
</tr>
<tr>
<td>llm:Text</td>
<td>Replaces the text of the message with the expression value.</td>
</tr>
<tr>
<td>llm:Bitmap</td>
<td>Sets the TurboFlow bitmap.</td>
</tr>
<tr>
<td>llm:Byte[#]</td>
<td>Modifies the specified byte, when # is a number from 0 to the length of the message. The expression value is interpreted as an integer.</td>
</tr>
<tr>
<td>var:integer</td>
<td>Sets a local variable, a number between 0 and 9. Used to save a result in a temporary location. The saved value can be accessed in subsequent mappings within the same map. Local variables are never part of the message.</td>
</tr>
<tr>
<td>var:name</td>
<td>Sets a string used to create a variable that is global to the domain. Used to save a result in a temporary location. The saved value can be accessed in subsequent mappings across all maps in the domain. Global variables are never part of the message.</td>
</tr>
<tr>
<td>property</td>
<td>The name of a property in the unspecified property namespace (not llm or var) of an input message.</td>
</tr>
</tbody>
</table>

For the examples in Table 2-4 on page 27, assume that a property from the inbound message named myprop was copied to the outbound properties and has a value of 3.
Table 2-4  Examples of property mapping expressions

<table>
<thead>
<tr>
<th>Identifier</th>
<th>SQL92 Expression</th>
<th>New value of property</th>
</tr>
</thead>
<tbody>
<tr>
<td>myprop</td>
<td>myprop + 3</td>
<td>6</td>
</tr>
<tr>
<td>myprop</td>
<td>myprop # 3</td>
<td>33</td>
</tr>
<tr>
<td>myprop</td>
<td>‘MyProp’ # 3</td>
<td>MyProp3</td>
</tr>
<tr>
<td>myprop</td>
<td>myprop + 4.0</td>
<td>7</td>
</tr>
<tr>
<td>myprop</td>
<td>‘MyProp’ # myprop</td>
<td>MyProp3</td>
</tr>
<tr>
<td>var:0</td>
<td>myprop + 50</td>
<td>local variable var:0 = 53</td>
</tr>
<tr>
<td>var:1</td>
<td>var:0 + myprop * 2</td>
<td>local variable var:1 = 53 + 3 * 2 = 59</td>
</tr>
<tr>
<td>newprop</td>
<td>var:0 + var:1</td>
<td>outbound property newprop = 59 + 53 = 112</td>
</tr>
<tr>
<td>var:lastLen</td>
<td>llm:Length</td>
<td>global variable lastLen set to length of current message.</td>
</tr>
</tbody>
</table>

2.3 LLM Object reuse

A DataPower XM70 LLM configuration uses any number of the objects introduced in this chapter. Each LLM object is atomic and can be mixed and matched as necessary. For example, an Instance object may be used by multiple Policies and Routes. Figure 2-4 illustrates a configuration where two Policies and three Routes are all using the same Instance object for communications.

Figure 2-4   Instance and Route reuse
A messaging environment may employ the use of more than one instance. Figure 2-5 shows a complex configuration with a second Instance sharing the Multicast Receive object (used by the Instance1) and using a Unicast2 and Multicast Transmit2 for outbound communications.

![Diagram of complex interrelationship of LLM objects]

**Figure 2-5 Complex interrelationship of LLM objects**

### 2.4 Message flow

On the XM70, there are three points of entry where a message may originate:

- LLM Multicast Receive object
- LLM Unicast Object
- Multi-protocol Gateway Service (through a variety of protocols)
Figure 2-6 illustrates the various entry points and flows for messages through the LLM appliance. The following is a generalized sequence of events that occur for each message.

1. A message arrives into the device in one of two ways:
   a. A message arrives on the interface/port specified by either the Multicast Receive object or the Unicast object. The message is handed over to the containing Instance object for publishing. The instance object publishes (or broadcasts) the message for consumption by any associated Policies. This message will only be consumed by Policies that have specified this Instance.
   b. A message arrives through a supported protocol (HTTP, MQ, and so forth). It is received and processed by a Multi-protocol Gateway service. The message is assigned a topic and published as a Reliable Local Message for consumption by any Policies listening for that topic.

2. The instance object publishes (or broadcasts) the message to any associated Policies.

3. A Policy that receives the message will determine whether to accept or reject the message based on the multicast group, topic, or message selection criteria. If the message is rejected, no further processing occurs (the message is discarded).

4. The Policy forwards the message to all of its associated Routes.

5. The Route will determine whether to accept or reject the message based on message selection criteria. If the message is rejected, no further processing occurs (the message is discarded).
6. The Route performs parameter mapping, if specified.
7. The Route forwards the message to either its associated Instance (in the case of RMM, RUM, or RUDP) or for consumption by a Multi-protocol Gateway (Reliable Local Message).
8. The Instance will hand the message over to the appropriate transmission object based on the Route's selected protocol. If the Route specified either RMM and RUDP, the LLM Multicast Receive object is selected. For RUM, the Unicast object is selected.
9. The transmission object transmits the message over the interface.

2.5 Late joining receivers

The XM70's RMM protocol supports a late join feature that allows a newly oncoming receiver to request a configurable amount of old data from the transmitter's history buffer.

2.6 Repair ports

Multi-port configuration allows the application to send and receive data on more than one port. Each transmitter topic can define a destination port for data and repair. The receiver is able to define a list of ports on which to receive data.

Multi-port enables the following features:

- The transmitter is able to open a topic (multicast or unicast) and define destination ports for data and repair.
  - ODATA and SPM packets will be sent to the data port.
  - Repair packets (RDATA, NCF) will be sent to the repair port.
- The receiver can listen to more than one port. The application is able to define a number of ports and configure each port with a set of parameters that will define behavior such as reuse and bind address.

2.7 Thread priority

Thread priority allows the application to set high priority to the key RMM threads. Assigning higher priority can reduce overall latency and in particular reduce latency spikes.
2.8 Congestion management

The XM70 offers congestion management and traffic control. Receiver-side congestion management features allow you to limit the amount of memory that RMM uses to install for a topic based on space or time. Congestion management limits the rates at which NAKs are generated and sent. Options are available to automatically trim stale data.

Congestion management features on the transmitter side enable you to limit the number of times RMM attempts to request ACKs. You can also limit the amount of data that is held by RMM for topic history.
XM70 messaging patterns

This chapter introduces various messaging patterns and how they relate to the DataPower XM70 LLM appliance.
3.1 Point-to-point pattern

The point-to-point pattern ensures that only one receiver consumes any given message. The transmitter sends messages directly to a single receiver. See Figure 3-1.

![Figure 3-1 Point-to-point pattern](image)

The XM70 supports the point-to-point pattern using both Reliable Unicast and Reliable UDP protocols, and can act as both a transmitter, receiver, or both. It can solve the need for topic translation and message filtering with no programming.

This is useful when an existing transmitter is in place providing message streams to a specific queue, and the following circumstances are in effect:

- The messages need to be forwarded to one or more additional subnets.
- The messages need to be routed to a different queue.
- The messages require topic translation.
- The messages need to be filtered based on topic or message content before being forwarded to the receiver.
- The messages require additional processing such as that offered by DataPower’s Multistep engine (for example, transformation, security, logging, and so forth)
- The messages need to be introduced into a publish-subscribe environment. This is useful for multicast relaying across routers described later.
3.2 Point-to-point fan-out pattern

The Point-to-point fan-out pattern extends the point-to-point pattern by supporting a one-to-many distribution model rather than a one-to-one model. See Figure 3-2.

While the publish-subscribe model is often the preferred method for distributing messages to multiple subscribers, there may times where it is necessary to deliver messages to multiple receivers using unicast TCP (RUM). One such time is when the receivers are located in a subnet other than the one where the publisher resides. By default, most routers will prevent multicast messages from passing from subnet to subnet. The point-to-point fan-out model solves this problem by utilizing unicast messages over TCP.

Figure 3-2  Point-to-point fan-out pattern

The XM70 supports the point-to-point fan-out pattern using both Reliable Unicast and Reliable UDP protocols while at the same time addressing performance, scalability, and manageability. It can easily solve the need for topic translation and message filtering. This pattern is useful when an existing unicast transmitter is in place providing message streams and the following circumstances are in effect:

- The messages need to be forwarded to one or more unicast receiver.
- The messages need to be routed to different receivers based on the queue name.
- The messages need to be routed to different receivers based on message contents.
- The messages require topic translation.
- The messages need to be filtered based on topic or message content.
- The messages require additional processing such as that offered by DataPower's Multistep engine (for example, transformation, security, logging, and so forth).
3.3 Publish-subscribe pattern

The publish-subscribe (or pub/sub) is an asynchronous messaging paradigm where senders (publishers) of messages are not programmed to send their messages to specific receivers (subscribers). Rather, published messages are characterized into classes (such as multicast groups and topics), without knowledge of what subscribers there may be (if any). Subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of what publishers there are (if any). This decoupling of publishers and subscribers allows for greater scalability and a more dynamic network topology. See Figure 3-3.

![Publish-subscribe pattern](image)

The XM70 supports the publish-subscribe pattern using Reliable Multicast Messaging, and can act as both a publisher and subscriber. It can easily solve the need for group and topic translation, as well as message filtering based on message properties and message content using a configuration approach rather than a programmatic one. This is useful when an existing publisher is in place providing message streams to a group of subscribers, and the following circumstances are in effect:

- The messages need to be broadcast on one or more additional subnets.
- The messages require multicast group translation.
- The messages require topic translation.
- The messages need to be filtered based on topic or message content before being forwarded to the receiver.
- The messages require additional processing such as that offered by DataPower’s Multistep engine (for example, transformation, security, logging, and so forth).
- The messages need to be re-transmitted using Reliable Unicast or Reliable UDP to a single receiver. This is most useful for multicast relaying across routers described later in this section.
3.4 Publish-subscribe fan-out pattern

The Publish-subscribe fan-out pattern extends the publish-subscribe pattern by supporting a one-to-many multicast group distribution model rather than a one-to-one model. See Figure 3-4.

The publish-subscribe fan-out model is particularly useful when a multicast group on one subnet is publishing messages for a variety of topics, and those messages need to be routed to other subnets according to some criteria such as topic name or message contents. The logical separation of messages specific for each subnet results in reduced traffic and load in the destination subnets.

![Publish-subscribe fan-out pattern](image)

The XM70 supports the publish-subscribe fan-out pattern using Reliable Multicast messaging. It can easily solve the need for topic translation and message filtering.
This pattern is useful when an existing multicast transmitter is in place providing message streams for one or more multicast groups and topics and the following circumstances are in effect:

- The messages need to be forwarded to one or more multicast groups.
- The messages need to be routed to different groups based on the topic name.
- The messages need to be routed to different groups based on message contents.
- The messages require topic translation.
- The messages need to be filtered based on topic or message content.
- The messages require additional processing such as that offered by DataPower's Multistep engine (for example, transformation, security, logging, and so forth)

### 3.5 Publish-subscribe relay pattern

Although the publish-subscribe pattern effectively decouples the publisher and subscriber allowing subscribers to be added or removed with no effect on the publisher, it has a potential side effect of substantially increasing network traffic. Routers typically filter multicast traffic to prevent excessive traffic on neighboring subnets. It's conceivable that with a few high volume transmitters and no router restrictions, the amount of traffic broadcast over an entire network could result in excessive congestion, especially if receivers start sending NAKs and requesting retransmission of lost messages.

When messages must cross subnet boundaries, routers must be specially configured to recognize the permissible multicast traffic. This is complicated when ad hoc multicast groups or topics are introduced, resulting in the need for reconfiguring all intermediate routers.

The publish-subscribe relay pattern solves this problem by effectively combining the publish-subscribe pattern with the point-to-point pattern. See Figure 3-5 on page 39. When a published message of interest arrives on the primary subnet, it is consumed by a local proxy and retransmitted point-to-point to a remote proxy publisher. The remote proxy republishes the message on the remote subnet for consumption by remote subscribers. Proxying the message as point-to-point alleviates the need for any router reconfiguration.
The XM70 supports the publish-subscribe relay pattern using a combination of Reliable Multicast and Unicast (or UDP) Messaging. It fills the role of Proxy in both the local and remote subnets. One XM70 is deployed in the publisher’s subnet. It converts the RMM messages into RUM/RUDP and forwards them to the remote XM70 where the message is converted back to RMM and republished.
Example configurations

This chapter provides some example configurations that cover the most common messaging patterns for the XM70. Each example will explain the steps employed to create the scenario.
The DataPower XM70 WebGUI provides various ways of creating objects. To reduce the number of nested pop-up windows and keep the explanations clear, this book will use the Objects menu (Figure 4-1) in the DataPower left navigation bar to create the various LLM objects.

![Control Panel](image)

*Figure 4-1 Low Latency Objects menu*

The examples in this section build upon each other. Once created, the components can be mixed and matched for a variety of other scenarios.
4.1 Prerequisite software

If you intend to execute the example configurations in this section, you will need the following software:

- IBM WebSphere MQ Low Latency Messaging.
  The example transmitter and receiver applications will require the llmJni.jar that is part of the WebSphere MQ LLM product.

- redp4515.zip
  Download redp4515.zip from the Redbooks download site. This file contains a jar file with programs for sending and receiving messages.

- Sample XML file
  A sample XML file is included in the download, or you can use your own.

- A program such as cURL that can post a file to an HTTP URL. This is used in the final example when mediating between HTTP and RMM.

Java programs are used to transmit and receive LLM messages. The actual command line that you use may be slightly different from that shown in this book. This book assumes that llmJni.jar is located in the same directory in which the Java programs are located. Furthermore, the command may be slightly different depending on whether the host operating system is Windows® or Linux®. The commands in this book are for Linux.

For example, on Linux, the command to start the RMM transmitter might be as follows:

```
java -cp redp4515.jar:/path/to/llmJni.jar RmmTx -i 10.0.0.254
   -port 34300 -d 229.1.2.3 -t rmmTopic
```

On Windows, this may be as follows:

```
java -cp redp4515.jar;c:\path\to\llmJni.jar RmmTx -i 10.0.0.254
   -port 34300 -d 229.1.2.3 -t rmmTopic
```

The individual network topology may affect whether traffic can flow unrestricted between the sender and the DataPower XM70. Network routers or firewalls may prevent the XM70 from receiving multicast transmissions. This guide assumes that the XM70, sender, and receiver are all on the same subnet.
Interface references
The network address of the host where the transmitter will run is referred to as \(<ip_{Tx}>\).

The network address of the host where the receiver will run is referred to as \(<ip_{Rx}>\).

The DataPower XM70’s network interface used for receiving and transmitting is referred to as \(<ip_{DP}>\).

4.2 Example: Point-to-point using Reliable Unicast

For this configuration, a transmitter sends messages directly to DataPower for a specific queue using RUM (underlying transport is TCP). DataPower will transmit that message directly to another receiver for a different queue. See Figure 4-2.

Figure 4-2 Reliable unicast to Reliable Unicast

The following objects are required for this configuration (see Figure 4-3):

- LLM Unicast object
- LLM Instance object
- LLM Policy object
- LLM Route object

Figure 4-3 Object relationships
Perform the following steps to create this configuration.

1. Create the LLM Unicast object. Use the responses in Table 4-1.

   **Table 4-1 LLM Unicast configuration settings**

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>myUnicast</td>
<td>The name that identifies this Unicast object.</td>
</tr>
<tr>
<td>Data Port</td>
<td>34200</td>
<td>This is the port which DataPower will listen for Reliable Unicast transmissions from the sender.</td>
</tr>
<tr>
<td>Transmit interface</td>
<td>&lt;ip_DP&gt;</td>
<td>The network interface the XM70 will use to transmit Reliable Unicast messages to the receiver.</td>
</tr>
<tr>
<td>Receive interface</td>
<td>&lt;ip_DP&gt;</td>
<td>The network interface the XM70 will receive messages from the sender. The rumsender must send to this address.</td>
</tr>
</tbody>
</table>

2. Create the LLM Instance object. The Instance will reference the Unicast object you just created. Use the responses in Table 4-2.

   **Table 4-2 LLM Instance configuration settings**

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>myInstance</td>
<td>The name that identifies this Instance object.</td>
</tr>
<tr>
<td>LLM Unicast</td>
<td>myUnicast</td>
<td>The Instance will listen for Reliable Unicast messages from this object.</td>
</tr>
</tbody>
</table>

3. Create the LLM Route object. The Route will identify all outbound parameters such as the destination, port, and topic. Use the values in Table 4-3.

   **Table 4-3 LLM Route configuration settings**

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>rumRoute</td>
<td>The name that identifies this Route object.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Reliable Unicast Message</td>
<td>Selecting RUM here will cause the associated Instance to select the Unicast object for outbound transmissions.</td>
</tr>
<tr>
<td>LLM Instance</td>
<td>myInstance</td>
<td>Identifies which Instance should be used for outbound transmissions. This Route will share the same instance used in the other configurations.</td>
</tr>
<tr>
<td>Parameter name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Message Selection</td>
<td>TRUE</td>
<td>This is an SQL92 expression used to determine whether the message should be processed or ignored based on the contents of the message. Specifying TRUE indicates that all messages are accepted (from the Policy).</td>
</tr>
<tr>
<td>Topic Name</td>
<td>dpRumTopic</td>
<td>The topic for outbound unicast messages.</td>
</tr>
<tr>
<td>Destination</td>
<td>&lt;ip_Rx&gt;</td>
<td>The IP address of the unicast receiver.</td>
</tr>
<tr>
<td>Data Port</td>
<td>34200</td>
<td>The port that DataPower will send to. The receiver must be listening on this port.</td>
</tr>
<tr>
<td>Reliability</td>
<td>Unordered</td>
<td>Selecting <em>unordered</em> causes the Route to tell the receiver that the order of the received messages is unimportant and that out-of-sequence messages can be processed as they arrive. Missing messages will still be accounted for, but will not hold up processing at the receiver.</td>
</tr>
</tbody>
</table>
4. Create the LLM Policy object. The Policy will identify the multicast group and topic for which to listen. Use the values in Table 4-4.

Table 4-4  LLM Policy configuration settings for Reliable Unicast

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>rumPolicy</td>
<td>The name that identifies this Policy object.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Reliable Unicast</td>
<td>Selecting RUM here will cause the associated Instance to forward received messages from the Unicast object to this Policy.</td>
</tr>
<tr>
<td></td>
<td>Message</td>
<td></td>
</tr>
<tr>
<td>LLM Instance</td>
<td>myInstance</td>
<td>Identifies which Instance should manage the inbound interface details. This object will reuse the previously created Instance.</td>
</tr>
<tr>
<td>Topic Name</td>
<td>rumTopic</td>
<td>The topic for that this Policy is interested in.</td>
</tr>
<tr>
<td>Message Selection</td>
<td>TRUE</td>
<td>This is an SQL92 expression used to determine whether the message should be processed or ignored based on the contents of the message. Specifying TRUE indicates that all messages are accepted.</td>
</tr>
<tr>
<td>Route List</td>
<td>rmmRoute</td>
<td>Select the rumRoute, then click <strong>add</strong>.</td>
</tr>
<tr>
<td>Reliability</td>
<td>Unordered</td>
<td>Selecting <em>unordered</em> causes the Policy to process all incoming messages as they arrive without regard to the sequence number. If a message is lost, the Policy will still request the missing message, but will not prevent subsequent messages from being processed.</td>
</tr>
</tbody>
</table>

Table 4-5 shows the discrepancies between the transmitter and the receiver. The DataPower XM70 will act as the bridge between the transmitter and the receiver and mediate the differences.

Table 4-5  Transmitter and Receiver Parameter Comparison

<table>
<thead>
<tr>
<th></th>
<th>Transmitter</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue</td>
<td>rumTopic</td>
<td>dpRumTopic</td>
</tr>
<tr>
<td>Port</td>
<td>34200</td>
<td>34200</td>
</tr>
</tbody>
</table>

Start the RUM receiver, then the RUM sender, then view the statistics shown on the LLM main page to see the details about the traffic.
5. Start the RUM receiver with the following command:
   
   ```java
   java -cp redp4515.jar:path/to/llmJni.jar RumRx -i \<Ip\_Rx\> -port 34200
   -d \<Ip\_DP\> -q dpRumTopic
   ```

6. In another window, start the RUM sender with the following command:

   ```java
   java -cp redp4515.jar:path/to/llmJni.jar RumTx -i \<Ip\_Tx\> -port 34200
   -d \<Ip\_DP\> -q rumTopic
   ```

7. Check the LLM status panel to see that traffic is flowing through the instance. The columns labeled **Bytes Received** and **Bytes Sent** should be increasing. See Figure 4-4.

   ![Figure 4-4  LLM Instance status](image)

   Clicking on the Selections link will show details about message selection for all the defined Routes and Policies. Figure 4-5 shows that the rmmPolicy object has accepted 69266841 (through selection rules, and so forth).

   ![Figure 4-5  LLM Selection status](image)
4.3 Example: Point-to-point using Reliable UDP

For this configuration, a transmitter will send messages directly to DataPower for a specific topic over UDP. DataPower will transmit that message directly to another UDP receiver for a different topic. See Figure 4-6.

![Figure 4-6 Reliable UDP to Reliable UDP](image)

The following objects are required for this configuration (see Figure 4-7):

- LLM Multicast Transmit
- LLM Multicast Receive
- LLM Instance object (previously created)
- LLM Policy object
- LLM Route object

![Figure 4-7 Object relationships](image)
Perform the following steps to create this configuration.

1. Create the LLM Multicast Receive object. The LLM Multicast Receive object is used for UDP-based protocols (Reliable Multicast and Reliable UDP). Use the values in Table 4-6.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>rmmReceive</td>
<td>The name identifying this object.</td>
</tr>
<tr>
<td>Multicast Interface</td>
<td>&lt;ip_DP&gt;</td>
<td>This is the XM70 network interface on which multicast messages will arrive.</td>
</tr>
<tr>
<td>Data Port</td>
<td>34300</td>
<td>The port number to listen on for multicast messages. This must match the port identified by the rmmsender.</td>
</tr>
</tbody>
</table>

2. Create the LLM Multicast Transmit object. Use the responses in Table 4-7.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>rmmTransmit</td>
<td>The name that identifies this Multicast Transmit object.</td>
</tr>
<tr>
<td>Multicast Interface</td>
<td>&lt;ip_DP&gt;</td>
<td>This is the XM70 network interface on which multicast messages will be transmitted. This network address can be the same as the receive address.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The form allows for multiple network interfaces to be specified. Make sure to click add after you enter the IP address.</td>
</tr>
<tr>
<td>Data Port</td>
<td>34301</td>
<td>The port number that receivers must be listening on.</td>
</tr>
</tbody>
</table>
3. Modify the previously created Instance (myInstance). Use the values in Table 4-8.

**Table 4-8  LLM Instance settings**

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLM Multicast Receive</td>
<td>rmmReceive</td>
<td>Specifies the Multicast Receive object where inbound UDP messages will arrive.</td>
</tr>
<tr>
<td>LLM Multicast Transmit</td>
<td>rmmTransmit</td>
<td>Specifies the Multicast Transmit object where the instance will forward UDP transmissions.</td>
</tr>
</tbody>
</table>

4. Create the LLM Route object that will direct traffic to the receiver. The Route will identify all outbound parameters such as the destination and topic. The Port from the Unicast object is used unless overridden here in the Route. Use the values in Table 4-9.

**Table 4-9  LLM Route configuration settings**

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>rudpRoute</td>
<td>The name that identifies this Route object.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Reliable UDP Message</td>
<td>Selecting RUDP here will cause the associated Instance to select the Multicast Transmit object for outbound transmissions. The Multicast Transmit object is used because Reliable Multicast messaging uses UDP whereas Unicast uses TCP.</td>
</tr>
<tr>
<td>LLM Instance</td>
<td>myInstance</td>
<td>Identifies which Instance should be used for outbound transmissions. This Route will share the same instance used in the other configurations.</td>
</tr>
<tr>
<td>Topic Name</td>
<td>dpRudpTopic</td>
<td>The topic for outbound UDP unicast messages.</td>
</tr>
<tr>
<td>Destination</td>
<td>&lt;ip_Rx&gt;</td>
<td>The IP address of the RUDP receiver.</td>
</tr>
<tr>
<td>Reliability</td>
<td>Unordered</td>
<td>Selecting unordered causes the Route to tell the receiver that the order of the received messages is unimportant and that out-of-sequence messages can be processed as they arrive. Missing messages will still be accounted for, but will not hold up processing at the receiver.</td>
</tr>
</tbody>
</table>
5. Create the LLM Policy object that will receive Reliable UDP messages from the Instance. The Policy will identify the topic of interest. Use the values in Table 4-10.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>rudpPolicy</td>
<td>The name that identifies this Policy object.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Reliable UDP Message</td>
<td>Selecting RUDP here will cause the associated Instance to forward received messages from the Multicast Receive object to this Policy.</td>
</tr>
<tr>
<td>LLM Instance</td>
<td>myInstance</td>
<td>Identifies which Instance should manage the inbound interface details.</td>
</tr>
<tr>
<td>Topic Name</td>
<td>rudpTopic</td>
<td>The topic for that this Policy is interested in.</td>
</tr>
<tr>
<td>Route List</td>
<td>rudpRoute</td>
<td>Select the rudpRoute, then click add.</td>
</tr>
<tr>
<td>Reliability</td>
<td>Unordered</td>
<td>Selecting unordered causes the Policy to process all incoming messages as they arrive without regard to the sequence number. If a message is lost, the Policy will still request the missing message, but will not prevent subsequent messages from being processed.</td>
</tr>
</tbody>
</table>

Start the receiver, then the transmitter, then view the statistics shown on the LLM main page to see the details about the traffic.

6. Start the receiver with the following command:
   ```
   java -cp redp4515.jar:path/to/llmJni.jar RmmRx -i <Ip_Rx> -port 34301
   -d <Ip_DP> -t dpRudpTopic
   ```

7. In another window, start the transmitter with the following command:
   ```
   java -cp redp4515.jar:path/to/llmJni.jar RmmTx -i <Ip_Tx> -port 34300
   -d <Ip_DP> -t rudpTopic
   ```

8. Check the LLM status panel to see that traffic is flowing through the instance. It should look similar to Figure 4-4 on page 48 and Figure 4-5 on page 48.
4.4 Example: Publish-subscribe with group and topic translation

For this configuration, the XM70 will subscribe to a multicast group for a specific topic. Received messages will be forwarded to a new multicast group and new topic. See Figure 4-8.

The following objects are required for this configuration (see Table 4-9 on page 51):

- LLM Multicast Receive object (previously created)
- LLM Multicast Transmit object (previously created)
- LLM Instance object (previously created)
- LLM Policy object
- LLM Route object
Perform the following steps to create this configuration:

1. Create the LLM Route object. The Route will identify all outbound parameters such as the multicast group, port, and topic. Use the values in Table 4-11.

### Table 4-11 LLM Route configuration settings

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>rmmRoute</td>
<td>The name that identifies this Route object.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Reliable Multicast Message</td>
<td>Selecting RMM here will cause the associated Instance to select the Multicast Transmit object for outbound transmissions.</td>
</tr>
<tr>
<td>LLM Instance</td>
<td>myInstance</td>
<td>Identifies which Instance should be used for outbound transmissions.</td>
</tr>
<tr>
<td>Topic Name</td>
<td>dpRmmTopic</td>
<td>The topic for outbound multicast messages.</td>
</tr>
<tr>
<td>Multicast Group</td>
<td>239.1.2.3</td>
<td>The multicast group for outbound messages.</td>
</tr>
<tr>
<td>Data Port</td>
<td>34301</td>
<td>The messages will be broadcast for this port. Receivers must be listening on this port. If this parameter is not specified, it will default to the port specified in the LLM Multicast Transmit object.</td>
</tr>
<tr>
<td>Reliability</td>
<td>Unordered</td>
<td>Selecting unordered causes the Route to tell the receiver that the order of the received messages is unimportant and that out-of-sequence messages can be processed as they arrive. Missing messages will still be accounted for, but will not hold up processing at the receiver.</td>
</tr>
</tbody>
</table>

*Note:* This currently an optional parameter but may be required in future firmware releases.
2. Create the LLM Policy object. The Policy will identify the multicast group and topic to listen for. Use the values in Table 4-12.

**Table 4-12  LLM Policy configuration settings**

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>rmmpolicy</td>
<td>The name that identifies this Policy object.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Reliable Multicast Message</td>
<td>Selecting RMM here will cause the associated Instance to forward messages from the Multicast Receive object to this Policy.</td>
</tr>
<tr>
<td>LLM Instance</td>
<td>myInstance</td>
<td>Identifies which Instance should manage the inbound interface details.</td>
</tr>
<tr>
<td>Topic Name</td>
<td>rmmpolicy</td>
<td>The topic for that this Policy is interested in.</td>
</tr>
<tr>
<td>Multicast Group</td>
<td>229.1.2.3</td>
<td>The multicast group that this Policy is interested in.</td>
</tr>
<tr>
<td>Route List</td>
<td>rmmproute</td>
<td>Select the rmmproute, then click <strong>add</strong>.</td>
</tr>
<tr>
<td>Reliability</td>
<td>Unordered</td>
<td>Selecting <strong>unordered</strong> causes the Policy to process all incoming messages as they arrive without regard to the sequence number. If a message is lost, the Policy will still request the missing message, but will not prevent subsequent messages from being processed.</td>
</tr>
</tbody>
</table>

In this example, the same Java programs that were used in the earlier RUDP example are being used (RmmRx and RmmTx). This is because RMM and RUDP both rely on UDP as the underlying transport mechanism. The main difference is that instead of providing an IP address of the receiver, a multicast group is specified.

Table 4-13 shows the discrepancies between the transmitter and the receiver. The DataPower XM70 will act as the bridge between the transmitter and the receiver and mediate the differences.

**Table 4-13  Transmitter and Receiver Parameter Comparison**

<table>
<thead>
<tr>
<th>Transmitter</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multicast Group</td>
<td>229.1.2.3</td>
</tr>
<tr>
<td>Topic</td>
<td>rmmpolicy</td>
</tr>
<tr>
<td>Port</td>
<td>34300</td>
</tr>
</tbody>
</table>
3. Start the RMM receiver on the host with the following command:

   ```bash
   java -cp redp4515.jar:path/to/llmJni.jar RmmRx -i <Ip_Rx> -port 34301
   -d 239.1.2.3 -t dpRmmTopic
   ```

4. In another window, start the RMM transmitter with the following command:

   ```bash
   java -cp redp4515.jar:path/to/llmJni.jar RmmTx -i <Ip_Tx> -port 34300
   -d 229.1.2.3 -t rmmTopic
   ```

5. Check the LLM status panel to see that traffic is flowing through the instance. It should look similar to Figure 4-4 on page 48 and Figure 4-5 on page 48.

### 4.5 Example: RMM to RUM with topic filtering

In this example, a Policy will subscribe to a specific multicast group for any topic that starts with the prefix "ibm" (for example, ibmDataPower, ibmWebSphere, and so forth). Messages that match the topic selection criteria will be accepted and forwarded to a Route that will transmit the message as a Reliable Unicast Message with a new topic of "ibm". This effectively demonstrates the first half of the Publish-Subscribe Relay pattern. Implementing the inbound portion would follow the same steps in this example with the protocols reversed (RUM to RMM instead of RMM to RUM). See Table 4-10 on page 52.

![Figure 4-10 RMM to RUM](image-url)
The following objects are required for this configuration (see Table 4-11 on page 54):

- LLM Multicast Receive (previously created)
- LLM Unicast (previously created)
- LLM Instance object (previously created)
- LLM Policy object
- LLM Route object (previously created)

Perform the following steps to create this configuration.

1. Create the LLM Policy object. The Policy will have a topic selector to enable it to receive any topic that begins with the string “ibm”. Use the values in Table 4-14.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>ibmPolicy</td>
<td>The name that identifies this Policy object.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Reliable Multicast Message</td>
<td>Listen for RMM messages</td>
</tr>
<tr>
<td>LLM Instance</td>
<td>myInstance</td>
<td>Identifies which Instance should manage the inbound interface details.</td>
</tr>
<tr>
<td>Topic Selection</td>
<td>Llm:Topic LIKE ‘ibm%’</td>
<td>SQL92 expression to select only topics that start with the string “ibm”.</td>
</tr>
<tr>
<td>Multicast Group</td>
<td>229.1.2.3</td>
<td>The multicast group that this Policy will subscribe to.</td>
</tr>
<tr>
<td>Route List</td>
<td>rumRoute</td>
<td>Select the rumRoute, then click <strong>add</strong>.</td>
</tr>
</tbody>
</table>
To test this configuration, start the RUM receiver, then start the RMM sender specifying a topic of ibmDatapower. The receiver should start showing messages being received. Stop the sender and start it again with a topic that does not begin with ibm (such as “abc”). The receiver should not receive any messages.

2. Start the receiver with the following command:
   
   ```
   java -cp redp4515.jar:path/to/llmJni.jar RumRx -i <Ip_Rx> -port 34200 -d <Ip_DP> -q dpRumTopic
   ```

3. In another window, start the transmitter with the following command:
   
   ```
   java -cp redp4515.jar:path/to/llmJni.jar RmmTx -i <Ip_Tx> -port 34300 -d 229.1.2.3 -t ibmDatapower
   ```
   
   The receiver should show messages arriving.

4. Stop the transmitter by pressing the enter key, and restart it using a different topic (such as “aaa”).
   
   ```
   java -cp redp4515.jar:path/to/llmJni.jar RmmTx -i <Ip_Tx> -port 34300 -d 229.1.2.3 -t aaaTopic
   ```
   
   This time, the receiver remains idle. The Policy is ignoring the messages due to the topic selection rule.

5. Stop the transmitter by pressing the enter key, and restart it with the topic ibmWebsphere.
   
   ```
   java -cp redp4515.jar:path/to/llmJni.jar RmmTx -i <Ip_Tx> -port 34300 -d 229.1.2.3 -t ibmWebsphere
   ```
   
   Once again, the receiver should show messages arriving.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>Unordered</td>
<td>Selecting <strong>unordered</strong> causes the Policy to process all incoming messages as they arrive without regard to the sequence number. If a message is lost, the Policy will still request the missing message, but will not prevent subsequent messages from being processed.</td>
</tr>
</tbody>
</table>
4.6 Example: RMM to RMM with multistep

This example demonstrates how received messages can be forwarded to a local multi-protocol gateway (MPG) for multistep processing, then rebroadcast over a new multicast group and topic.

Introducing multistep into the configuration requires that the inbound RMM message be republished within the device as a Reliable Local Message (RLM) for consumption by a MQ LLM Front Side Handler (FSH). The FSH's associated MPG can process the message and republish it (with a new topic) as a reliable local message for consumption by a LLM Policy on the same device. That Policy will forward the message to a Route which will publish the message as RMM over the specified interface. See Figure 4-12.

![Multicast to Multicast with Intermediate Processing](image)

**Figure 4-12** Multicast to Multicast with intermediate processing.

There are three parts to this configuration (see Figure 4-13 on page 60):

- The LLM configuration that receives the inbound multicast message republishes it as a local message.
- The MPG configuration that receives and processes the local message publishes the results as a new local message with a different topic.
- The LLM configuration that receives the new local message from #2 publishes it as a Reliable Multicast message on the specified network interface.
4.6.1 Configuration part 1: RMM to reliable local message (RLM)

Perform the following steps to create this configuration:

1. Create the Route. Use the values from Table 4-15.

Table 4-15  LLM Route configuration settings

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>rlmRoute</td>
<td>The name that identifies this Route object.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Reliable Local Message</td>
<td>RLM causes the Route to publish the message locally for consumption by a MQ LLM Front Side Handler.</td>
</tr>
<tr>
<td>LLM Instance</td>
<td>myInstance</td>
<td>Identifies which Instance should be used for outbound transmissions. This field is required but not used when the RLM protocol is selected.</td>
</tr>
<tr>
<td>Topic Name</td>
<td>localTopic1</td>
<td>The topic for the new outbound message.</td>
</tr>
</tbody>
</table>
2. Create the Policy that will receive the inbound RMM messages. Use the values in Table 4-16.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>rmmToRlm</td>
<td>The name that identifies this Policy object.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Reliable Multicast Message</td>
<td>Listen for RMM messages</td>
</tr>
<tr>
<td>LLM Instance</td>
<td>myInstance</td>
<td>Identifies which Instance should manage the inbound interface details.</td>
</tr>
<tr>
<td>Topic Name</td>
<td>mpgTopic</td>
<td>SQL92 expression to select only topics that start with the string “ibm”.</td>
</tr>
<tr>
<td>Multicast Group</td>
<td>229.1.2.3</td>
<td>The multicast group that this Policy will subscribe to.</td>
</tr>
<tr>
<td>Route List</td>
<td>rlmRoute</td>
<td>Select the rlmRoute, then click add.</td>
</tr>
<tr>
<td>Reliability</td>
<td>Unordered</td>
<td>Selecting unordered causes the Policy to process all incoming messages as they arrive without regard to the sequence number. If a message is lost, the Policy will still request the missing message, but will not prevent subsequent messages from being processed.</td>
</tr>
</tbody>
</table>

### 4.6.2 Configuration part 2: RLM to RLM with multistep

Perform the following steps to create this configuration:

1. Create the MQ Low Latency Handler (from the Objects menu). Use the values in Table 4-17. Note that the MQ LLH will be in a down state until it is associated with an MPG.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>llmFSH</td>
<td>The name of this front side handler.</td>
</tr>
<tr>
<td>LLM Instance</td>
<td>myInstance</td>
<td>The instance that will provide messages to this FSH.</td>
</tr>
<tr>
<td>Topic Name</td>
<td>localTopic1</td>
<td>Accepts only messages with this topic.</td>
</tr>
</tbody>
</table>
2. Create a Match Rule. From the Objects navigation menu, in the XML Processing section, click **Matching Rule**.

3. On the Main tab, specify the name as MatchAnyURL.

4. On the Matching Rule tab, click **add** to create a new rule.
   a. For the Matching Type, leave the selection as URL.
   b. For the URL Match field, type an asterisk: *
   c. Click **Apply** in the Edit Matching Rule page.

5. Click **Apply** in the Configure Matching Rule page.

6. Create a Multi-Protocol Gateway service. From the main control panel, click the Multi-Protocol Gateway icon, then click **add** to create a new MPG. Perform the following steps to configure the Multi-Protocol Gateway.

7. Specify the Multi-Protocol Gateway Name as rlmServiceMPGW.

8. For the Multi-Protocol Gateway Policy, click **plus sign** to create a new processing policy.

9. For the Policy Name, type rlmServicePolicy

10. Change the Rule Direction drop-down menu to Client to Server

11. Click **New Rule**.

12. Double-click the Matching Rule icon and select the previously created MatchAnyURL matching rule from the drop-down menu. Click **Done**.

13. Drag a transform action onto the rule after the match action. Specify the Processing Control File to be store:///identity.xsl. See Figure 4-14.

![Figure 4-14: Transform options](options.png)

14. Click **Done** to close the Transform window.

15. Click **Apply Policy**. The policy should appear similar to Figure 4-15.
16. Click **Close Window**.
17. Make sure that the Multi-Protocol Gateway Policy shows `rlmServicePolicy`.
18. For the Backend URL, type `dpllm:///localTopic2`.
19. For the Response Type, select **Pass-Thru**.
20. For the Request Type, select **XML**.
21. For Propagate URI, select **off**.
22. In the Front Side Protocol field, select `llmFSH` and click **add**.
23. Click **Apply**.
4.6.3 Configuration part 3: RLM to RMM

To create this configuration, create the Policy that will listen for topic localTopic2 (from the MPG). Use the values in Table 4-18.

Table 4-18   LLM Policy configuration settings

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>rlmPolicy</td>
<td>The name that identifies this Policy object.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Reliable Local Message</td>
<td>This Policy will listen for local messages.</td>
</tr>
<tr>
<td>LLM Instance</td>
<td>myInstance</td>
<td>Identifies which Instance that will provide the messages.</td>
</tr>
<tr>
<td>Topic Name</td>
<td>localTopic2</td>
<td>Listen for localTopic2.</td>
</tr>
<tr>
<td>Route List</td>
<td>rmmRoute</td>
<td>Reuse the previously created rmmRoute.</td>
</tr>
</tbody>
</table>

4.6.4 Testing the configuration

Follow these steps to test the end-to-end configuration:

1. Start the RMM receiver on the host with the following command:
   
   ```
   java -cp redp4515.jar:path/to/llmJni.jar RmmRx -i <Ip_Rx> -port 34301
   -d 239.1.2.3 -t dpRmmTopic
   ```

2. In another window, start the RMM transmitter. Make sure that the `-file` parameter specifies the path to an XML file (the MPGW is expecting an XML document). A sample XML file named soapMsg.xml was included in the zip file for your convenience.
   
   ```
   java -cp redp4515.jar:path/to/llmJni.jar RmmTx -i <Ip_Tx> -port 34300
   -d 229.1.2.3 -t mpgTopic -file soapMsg.xml
   ```

3. Check the LLM status panel to see that traffic is flowing through the instance. It should look similar to Figure 4-4 on page 48 and Figure 4-5 on page 48.
4.7 Example: HTTP to RMM

This configuration demonstrates mediation between non-LLM and LLM protocols. An XML message will be POSTed to an HTTP front side handler. The contents of the message are extracted from the inbound message, transformed, and broadcast as a Reliable Multicast Message to a specific group and topic. See Table 4-16 on page 61.

![Figure 4-16](image)

The basic flow of this configuration is as follows (see Table 4-17 on page 61):

1. A message is posted from the client over HTTP.
2. The message arrives in the HTTP Front Side Handler (FSH).
3. The FSH forwards the contents of the message to an MPG. For this example, the MPGW is configured to act as a pass-through. No message processing will occur.
4. The MPG forwards the message to the back end, which is defined with the special dpilm protocol. The topic is specified in the URL. This will result in broadcasting a Reliable Local Message within the device.
5. A Policy object receives and accepts the message based on the topic.
6. The Policy forwards the message to a Route.
7. The Route forwards the message to an Instance and specify a multicast group and topic.
8. The Instance, seeing that the requested protocol is RMM, forwards the message to the Multicast Transmit object, which publishes the message to any RMM listeners.
Note that the Reliable Local Message generated as a result of the MPG's backside dplm://topicName URL is consumable by all MQ LLM FSHs and LLM Policies that are subscribed to that topic. For example, if an LLM Route publishes a Reliable Local Message with a topic of IBM, all MQ LLM FSHs and LLM Policies that are subscribed to the topic of IBM receive the message.

The following objects are required for this configuration:

- HTTP Front Side Handler
- MPG
- LLM Multicast Transmit object (previously created)
- LLM Instance object (previously created)
- LLM Policy object that subscribes to messages generated by MPG
- LLM Route object (previously created)

Perform the following steps to create this configuration:

1. Create an HTTP FSH. From the Objects navigation menu, in the Protocol Handlers section, click **HTTP Front Side Handler**. Click **Add** to create a new HTTP FSH. Use the values in Table 4-19.

Table 4-19  **HTTP Front Side Handler configuration settings**

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>MyHttpFSH</td>
<td>The name that identifies this FSH object.</td>
</tr>
<tr>
<td>Local IP Address</td>
<td>&lt;lp_DP&gt;</td>
<td>Leaving 0.0.0.0 allows the use of any IP address, however you can specify any defined IP address that is defined on the device.</td>
</tr>
<tr>
<td>Port Number</td>
<td>2328</td>
<td>Specify any available port.</td>
</tr>
</tbody>
</table>
2. Create an MPG service. From the main control panel, click the Multi-Protocol Gateway icon, and click **add** to create a new MPG. Perform the following steps to configure the Multi-Protocol Gateway.

   a. Specify the Multi-Protocol Gateway Name as HttpToRlm
   b. For the Multi-Protocol Gateway Policy, click the plus sign to create a new multistep policy.
   c. For the Policy Name, type HttpToRlmPolicy.
   d. Change the Rule Direction drop-down menu to Client to Server.
   e. Click **New Rule**.
   f. Double-click the Matching Rule icon and select the previously created MatchAnyURL (created in 4.6, “Example: RMM to RMM with multistep” on page 59) matching rule from the drop-down menu.
   g. Drag a Results action onto the processing rule. The policy should appear similar to the one in Figure 4-18.
   h. Click **Apply Policy**.

   ![Figure 4-18 Completed MPG Policy](image)

   i. Click the **Close Window** link.
   j. For the Backend URL, type dpllm:///httpTopic.
   k. For the Response Type, select **Pass-Thru**.
   l. For the Request Type, select **Pass-Thru**.
   m. For the Propagate URi, select **off**.
   n. In the Front Side Protocol field, select **MyHttpFSH** and click **add**.
   o. Click **Apply**.

   The Multi-Protocol Gateway configuration is complete. An LLM Policy must be created to listen for reliable local messages with topic httpTopic.
3. Create the LLM Policy object that will subscribe to reliable local messages for topic rlmTopic. Use the values in Table 4-20.

Table 4-20  LLM Policy configuration settings

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>rlmPolicy</td>
<td>The name that identifies this Policy object.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Reliable Local Message</td>
<td>Selecting RLM indicates that this policy will listen for messages that are published from an MPG on the same device.</td>
</tr>
<tr>
<td>LLM Instance</td>
<td>myInstance</td>
<td>Identifies which Instance should manage the inbound interface details.</td>
</tr>
<tr>
<td>Topic Name</td>
<td>httpTopic</td>
<td>The topic that this Policy is interested in.</td>
</tr>
</tbody>
</table>

At this point, the configuration is now complete. The next steps are to start the RMM receiver, then POST a message over HTTP to the MPG.

4. Start the RMM receiver on the host with the following command:

   java -cp redp4515.jar:path/to/llmJni.jar RmmRx -i <Ip_RX> -port 34301
   -d 239.1.2.3 -t dpRmmTopic -dump

5. In another window, use the cURL command (or any program that can POST some data to a HTTP URL):

   curl --data-binary @soapMsg.xml <Ip_DP>:2328

   You should see the contents of the file you posted displayed in the receiver's console window.
4.8 Troubleshooting the examples

This section details some of the most common problems encountered when building and executing the examples in this book.

4.8.1 First steps

Check the logs
The logs should be the first point of troubleshooting.

- Error messages that state “host unreachable” are a clear indication that there is a network interface problem. Use the Ping tool (on the Troubleshooting page) to verify connectivity between all participants.
- Verify backside URL’s and outbound topics.
- Change the log level to Debug if necessary, but keep in mind that if the transmitter is transmitting at high speed, the logs will fill up quickly. It may be necessary to start and immediately stop the transmitter to gain a good sampling of log messages. Using syslog, or a similar logging server, may be helpful in capturing the log messages. Make sure to disable debug logging when finished.

Use the LLM Status page
The LLM Status page can help to identify (or eliminate) various network and configuration problems. The LLM Status page has two parts:

- Instances
  This part shows communication statistics on a per LLM Instance basis. This is useful when determining whether bytes are being received or transmitted by a specific Instance.
- Selections
  This part shows LLM Policy and LLM Route object activity. The information shown on this page is helpful in determining how many messages are accepted or rejected, as well as errors that are occurring.
4.8.2 Network interface and port configuration problems

Network interface and port configuration errors are easy to make. The first indication of a network interface problem can be detected in the XM70’s LLM status page.

If the transmitter is sending (or publishing) to the XM70 and the XM70’s LLM status page shows that there are no packets being received, it is likely that there is a network interface mismatch. This is also the case in the reverse, when the status page shows that the XM70 is transmitting packets but the recipient is not receiving them.

**Verify the interface settings**

Use the tables below to verify that the network interface settings are correct. The tables are grouped by protocol (RUM, RUDP, and RMM) and whether the XM70 is acting as a subscriber/receiver or a publisher.

**RUM interface settings**

*Table 4-21  DataPower XM70 as RUM Receiver*

<table>
<thead>
<tr>
<th>Sender (Java)</th>
<th>Receiver (XM70)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination (-d parameter)</td>
<td>Unicast object's Receive Interface</td>
<td>Both must be the same. The sender identifies the network address of the recipient. The Unicast object identifies what interface to listen on.</td>
</tr>
<tr>
<td>Port (-port parameter)</td>
<td>Unicast object's Data Port</td>
<td>Both must be the same. The sender identifies what port the receiver is listening on.</td>
</tr>
</tbody>
</table>

*Table 4-22  DataPower XM70 as RUM Transmitter*

<table>
<thead>
<tr>
<th>Sender (XM70)</th>
<th>Receiver (Java)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLM Route’s Destination field.</td>
<td>Interface (-i parameter)</td>
<td>Both must be the same. The XM70 identifies the destination (network address) of the recipient.</td>
</tr>
<tr>
<td>LLM Route’s Data Port</td>
<td>Port (-p parameter)</td>
<td>Both must be the same. The XM70 identifies what port the receiver is listening on.</td>
</tr>
</tbody>
</table>
### RUDP interface settings

**Table 4-23 DataPower XM70 as RUDP Receiver**

<table>
<thead>
<tr>
<th>Sender (Java)</th>
<th>Receiver (XM70)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination (-d parameter)</td>
<td>LLM Multicast Receive object’s Receive Interface</td>
<td>Both must be the same.</td>
</tr>
<tr>
<td>Port (-port parameter)</td>
<td>LLM Multicast Receive object’s Data Port</td>
<td>Both must be the same.</td>
</tr>
</tbody>
</table>

**Table 4-24 DataPower XM70 as RUDP Transmitter**

<table>
<thead>
<tr>
<th>Sender (XM70)</th>
<th>Receiver (Java)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLM Route’s Destination field.</td>
<td>Interface (-i parameter)</td>
<td>Both must be the same.</td>
</tr>
<tr>
<td>LLM Multicast Transmit’s Data Port.</td>
<td>Port (-p parameter)</td>
<td>Both must be the same.</td>
</tr>
</tbody>
</table>

### RMM interface settings

**Table 4-25 DataPower XM70 as RMM Subscriber**

<table>
<thead>
<tr>
<th>Sender (Java)</th>
<th>Receiver (XM70)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination (-d parameter)</td>
<td>LLM Policy’s Multicast Group</td>
<td>Both must be the same.</td>
</tr>
<tr>
<td>Port (-port parameter)</td>
<td>LLM Multicast Receive object’s Data Port</td>
<td>Both must be the same.</td>
</tr>
</tbody>
</table>

**Table 4-26 DataPower XM70 as RMM Publisher**

<table>
<thead>
<tr>
<th>Sender (XM70)</th>
<th>Receiver (Java)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLM Route’s Multicast Group field.</td>
<td>Destination (-d parameter)</td>
<td>Both must be the same.</td>
</tr>
<tr>
<td>LLM Route’s Data Port.</td>
<td>Port (-p parameter)</td>
<td>Both must be the same.</td>
</tr>
</tbody>
</table>
4.8.3 Topic selection failure

Assure that the topic name being transmitted by the sender matches that of the receiver. If a topic selection SQL92 string is being used, verify its correctness.

The LLM Selections page has two columns labeled True and False. These columns show how many messages have been accepted or rejected based on topic and message selection criteria (SQL92). If the selections page shows values in the False column, then it is likely that the selection criteria is causing messages to be rejected.

4.8.4 MPG backside URI propagation

When an MPG specifies a backside URL that uses the dpllm protocol, ensure that the Propagate URI option is turned off. If it is not turned off, the URI from the inbound message will be added to the dpllm URI, resulting in an incorrect (or unexpected) URL.

For example, assume that the MPGW's front side is HTTP, and the inbound URL was http://datapower:2049/my/uri/string. If the propagate URI was not disabled and the back end is specified as dpllm://myTopic, the MPGW would create a new backside URL of dpllm://my/uri/string which would result in unexpected behavior.

4.8.5 Mismatched property errors

Although the examples in this section do not demonstrate using message properties, they can easily be the cause of communication errors.

Ensure the sender and receiver are coordinated in terms of message properties. If the sender indicates that it is using properties and sends messages with properties, but the receiver is not expecting properties, a communication error will occur and messages will not be received.

The LLM Policy is more forgiving and requires no special configuration for receiving properties, but the LLM Route must explicitly indicate whether to transmit properties. Make sure that Use Properties is set appropriately for your environment.

4.8.6 Mismatched ACK/NAK errors

Ensure that the use of ACK and NAK is coordinated between the sender and receiver. Mismatched acknowledgement settings result in communication failures.
4.8.7 Source-llm error when deleting Instance

If a configuration needs to be deleted, DataPower appliances will prevent the deletion of objects that are referenced by other objects. When attempting to delete an object that is in use, the WebGUI will provide enough detail to help in determining which objects are dependent on each other.

One exception to this rule is the MQ LLM FSH. If an attempt is made to delete an Instance, a source-llm error may be displayed. Verify that any MQ LLM FSHs that are using the instance have been deleted first. Otherwise they will prevent deletion of the associated LLM Instance object.
Additional material

This paper refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this paper is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

ftp://www.redbooks.ibm.com/redbooks/Redp4515.zip

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the Additional materials and open the directory that corresponds with the IBMRedpaper form number, REDP4515.zip
Using the Web material

The additional Web material that accompanies this paper includes the following files:

Redp4515.zip  This zip file contains a jar file with programs for sending and receiving messages

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:  84KB

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 78. Note that some of the documents referenced here may be available in softcopy only.

- IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started, REDP-4327
- IBM WebSphere DataPower SOA Appliances Part II: Authentication and Authorization, REDP-4364
- IBM WebSphere DataPower SOA Appliances Part IV: Management and Governance, REDP-4366
- DataPower Architectural Design Patterns: Integrating and Securing Services Across Domains, SG24-7620
- DataPower Problem Determination Techniques, REDP-4445

Online resources

These Web sites are also relevant as further information sources:

- IBM WebSphere DataPower SOA Appliances: Product support  
  http://www.ibm.com/software/integration/datapower/support
- IBM WebSphere DataPower SOA Appliances: Product documentation  
- IBM WebSphere MQ Low Latency Messaing: Product support  
- IBM WebSphere MQ Low Latency Messaing: Product documentation  
How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services
Index

A
ACK/NAK Errors 72

D
DataPower’s Multistep engine 35

F
FTP(S) 14

H
HTTP to Reliable Multicast Message 65
HTTP(S) 14

I
IBM WebSphere DataPower XI50 14
IBM WebSphere MQ Low Latency Messaging 43

L
LLM Instance 17, 45
LLM Instance object 44
LLM Multicast Receive 17, 28, 49
LLM Multicast Transmit 17, 49
LLM Object Reuse 27
LLM objects 42
LLM Policy 17
LLM Policy object 44, 49
LLM Route 17, 24
LLM Route object 44, 49
LLM Status Page 69
LLM Tier Group 17
LLM Unicast 17
LLM Unicast object 28, 44

M
message filtering 10, 25
message selection 46
multicast 5
multi-protocol Gateway 28

N
null routing 24

R
Redbooks Web site 78
Contact us viii
Reliable and Consistent Message Streaming (RC-MS) 2
Reliable Local Message 24
Reliable Multicast and Unicast (or UDP) 39
Reliable Multicast Messaging (RMM) 5
Reliable Protocol 5
Reliable UDP Message 24
Reliable Unicast Messaging (RUM) 5
RMM API 8
RMM Interface Settings 71
RMM transmitter 43
RUDP Interface Settings 71
RUM Interface Settings 70

T
TCP 2
TCP/IP protocol 4
topic filtering 56

U
UDP transport 3
Unicast TCP 4–5
Unicast UDP 4–5
User Datagram Protocol (UDP) 3

W
WAN 3
WebGUI 14
WebSphere JMS 14
WebSphere MQ 14
WebSphere® DataPower XM70 1

X
XM70 14, 36–37, 39, 41
XM70 WebGUI 42
XSL acceleration 14
DataPower XM70 Use Cases and Patterns

DataPower XM70 Overview

Messaging Patterns and Use Cases

Example Configurations

This IBM Redpaper publication gives a broad understanding of the WebSphere DataPower XM70 Low Latency Messaging appliance. After a brief introduction to low latency messaging, various messaging patterns are discussed along with configuration details for implementing each pattern. The book wraps up with a section on troubleshooting and performance monitoring specific to the XM70.

This Redpaper publication is intended for individuals who require a better understanding of low latency messaging and how it is implemented on the DataPower XM70. It is assumed that the reader is familiar with concepts and configuration of DataPower appliances.