FEDERATING AMQP1.0 MESSAGE-ORIENTED MIDDLEWARE (MOM) BROKERS

By

David Joe Wade

A Project Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: COMPUTER SCIENCE

Approved:

Dr. Eugene Eberbach, Project Adviser

Rensselaer Polytechnic Institute
Hartford, Connecticut

April 2014
(For Graduation May 2014)

CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENT

Special thanks to: my wife Anne, my son Jack, my daughter Erin, and my parents Brian and Catherine. Thanks for all the support and sorry for all the late nights and weekends spent working on this project. Also thanks to Professors Eugene Eberbach and Houman Younessi.

ABSTRACT

Message passing is a fundamental aspect of systems engineering. Without a methodology to pass information between separate components building a distributed system is impossible. When building a distributed system it is very common for teams/organizations to develop proprietary message passing systems. Because of their isolated development, these systems usually fill a very niche role and do not interoperate with other implementations.[1] This is a fundamental problem when trying to build larger systems from subsystems. Without a common language and transport getting different components to communicate is difficult if not impossible. There have been many attempts to develop common languages and transports with varying amounts of success. A new standard, developed by a consortium of large technology companies, the Advanced Message Queuing Protocol (AMQP) provides an open standard protocol for message orientated middleware (MOM). There are currently several active AMQP compliant MOM implementations, both open-source and proprietary. This project investigates the current state of several popular open-source AMQP message brokers, attempts to create a federated network of different implementations in order to collect benchmark information for both homogenous an heterogeneous broker networks. To achieve this goal a simple AMQP messaging benchmark was developed and data collected from different broker architectures. While it was possible to integrate some of the active broker implementations this project found that not all of the available implementations are interoperable and there are significant problems creating a reliable messaging system with the available AMQP implementations.

[1]Many middleware implementations that fail because of this - The author has specifically implemented two such systems in his professional career. Both were replaced with other systems unused because of lack of interoperability.

1. BACKGROUND

1.1 Middleware

In computer science, the term ”middleware” has many meanings. Generically, middleware can be thought as ”glue” for computer software. It facilitates communication between different software components - abstracting the complexities of intra and inter process communication from a software component. In a broad sense many things that do clearly resemble a communication context are technically middleware. User space software drivers, virtual machines (Java not VMware) and kernel Inter-process Communication (IPC) mechanisms are all forms of middleware. This paper focuses on middleware as it relates to distributed applications - typically used to share data and state between software components. Middleware at this level typically facilitates communication over a network and allows for components to work together in separate execution environments. Even with this narrow definition there are many different classes of middleware systems. Some of the common design patterns used are: enterprise application integration (EAI), data integration (DI), message-oriented middleware (MOM), object request brokers (ORB) and the enterprise service bus (ESB). This paper specifically focuses on message-oriented middleware.

1.1.1 Message Oriented Middleware (MOM)

Message-oriented middleware (MOM) is a middleware design that is focused on sending discrete blocks of information (messages) between processing components. Messages can contain state information, commands or pure input data - it is up to the processing components to identify and process the messages according to their contents. MOM systems are inherently asynchronous - processing components typically execute independently and are not synchronized to process in lockstep.

A fundamental component of some MOM systems is a message broker. A message broker is acts as an intermediary - all messages sent through a MOM system must be sent to a message broker. A message broker performs many of the same functions as a network router on an ethernet network, but at application layer [33]. Message brokers can connect to either message producers, consumers or other message brokers - and are responsible for routing messages from the producer to consumer. Frequently, message brokers are designed to do more than just route messages and are designed with the ability to store messages locally in order to provide message reliability. Message brokers are often able to handle message duplication - that is to forward a message to multiple clients and provide for quality of service (QOS) to be applied to messages.

While a message broker can provide many benefits, by adding an additional component between message producers and consumers this can reduce performance and reliability. MOM systems also frequently require a significant amount of configuration to setup properly and add an extra layer of complexity when trying to debug a system.

1.2 Advanced Message Queuing Protocol (AMQP)

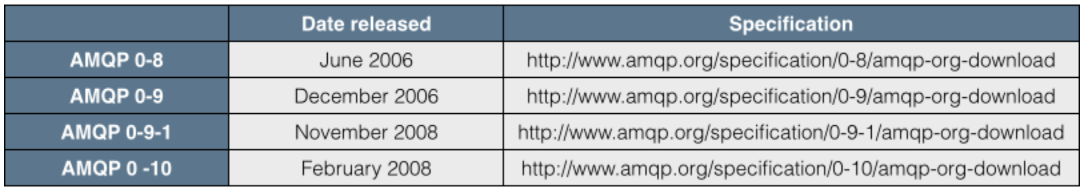
The Advanced Message Queueing Protocol (AMQP) is a standard published by the Organization for the Advancement of Structured Information Standards (OASIS) [14]. In the Open Systems Interconnection (OSI) model AMQP is a application layer protocol, it exists in the same space as other protocols as the File Transfer Protocol (FTP), Dynamic Host Control Protocol (DHCP), etc. [31]. The AMQP specifies both message format and and a communication language. This is in direct contrast to previous attempts to standardize middleware at the API level, e.g. Java Messaging Service (JMS) [16], which because did not specify a protocol limited interoperability.

1.3 AMQP History

The development of AMQP was started in 2003 by John O'Hara and others at JPMorgan Chase in London, UK [31]. JPMorgan was looking for a messaging solution with high durability that supported a very high number of small message transactions (in the range of 500,000 messages/second) [29]. In the financial sector accurate and reliable message passing is clearly very important; losing a message or delays in message processing have clear economic consequences. At the time O'Hara found that the commercial middleware products could not deliver the level of service required, and banks were known to develop their own enterprise middleware to fill in the gaps. However, developing enterprise middleware is complex and difficult, and there no standardization among the various implementations [29].

O'hara used protocols such as HTTP, TCP and FTP as models for the new protocol, and wanted to ensure that the specification for the protocol was open and royalty-free [31]. Initial development was done internally at JPMorgan Chase but was later subcontracted with iMatix Corporation for a broker and protocol documentation [28]. This work lead to the creation of the OpenAMQ message broker, which is no longer being actively developed [27].

In 2005 JPMorgan Chase partnered with companies such as: Cisco Systems, Red Hat, iMatix and the Transaction Workflow Innovation Standards Team (TWIST) to form an AMQP working group. There have been five major releases of the AMQP standard [1].



	Date released	Specification
AMQP 0-8	June 2006	http://www.amqp.org/specification/0-8/amqp-org-download
AMQP 0-9	December 2006	http://www.amqp.org/specification/0-9/amqp-org-download
AMQP 0-9-1	November 2008	http://www.amqp.org/specification/0-9-1/amqp-org-download
AMQP 0 -10	February 2008	http://www.amqp.org/specification/0-10/amqp-org-download

Figure 1.1: AMQP versions

AMQP 1.0 was released by the AMQP working group on October 30th 2011. The next day the AMQP working group announced its reorganization as a OASIS member section[23].

1.4 AMQP 1.0

AMQP is a layered protocol. The first layer defines a type system and a set of encodings that must be used to represent fundamental types. The second layer defines an binary, peer-to-peed protocol for transporting messages between two nodes on a network. The third layer defines the overall AMQP message format, its contents and header/footer structure. The fourth, defines a communication protocol and how to atomically cluster messages. The fifth later explicitly deals with security, which from the ground up is a fundamental part of AMQP [24].

AMQP treats all nodes equally - it places no constraints on roles of connections endpoints. Because of historical reasons most AMQP messaging systems are broker centric - client nodes connect to a centralized message passing mechanism to exchange messages. At the most fundamental level a broker is a trusted intermediary, in the AMQP model a broker has three primary responsibilities [31]:

1. Taking responsibility for messages sent by clients.
2. Co-ordinating client transactions.
3. Routing and distributing messages

Message brokers all for the creation of messaging networks with varying complexity. Brokers typically support both direct connections (single producer communicates with single consumer) and fanout configurations (single producer communicates with many consumers). With these building blocks it is also possible to create very complicated broker networks with many tiers of producers, consumers and brokers. A simple example of a single producer, multiple consumer network with a single message broker can be seen in Figure 1.2.

1.4.1 Specific differences

AMQP 1.0 is a major departure from earlier AMQP versions. While it imposes far fewer semantic requirements, the protocol is substantially more complex [20]. Pre AMQP 1.0 versions specify both a wire-level protocol and a broker architecture. AMQP 1.0 only specifies a protocol and does not impose any architecture model on implementations [8].

AMQP 1.0 is a symmetric protocol, all nodes are treated equally. Earlier versions were asymmetric - each connection needed a defined client and broker. AMQP 1.0 allows for broker-less point-to-point connections. AMQP versions before 1.0 also specified protocol commands to manage brokers. AMQP 0-10 specifies commands such as "Create Queue", "Delete Queue", etc. AMQP 1.0 does not

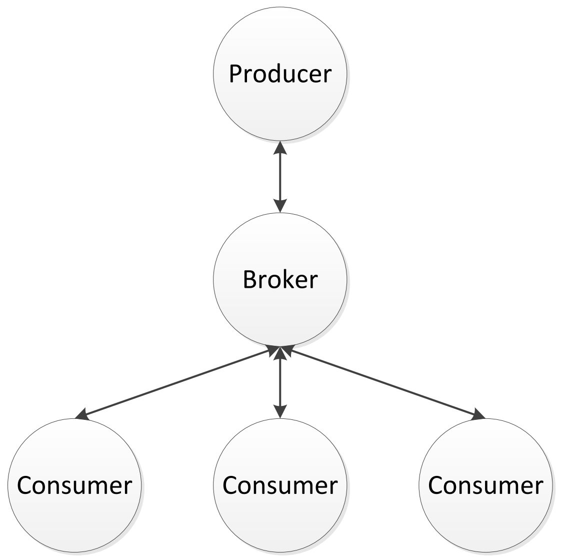


Figure 1.2: Simple broker network

define these messages as part of the protocol and assumes that this functionality will be part of each specific implementation [8]. Previous AMQP versions had strict definitions for clients and brokers. Every connection must have a defined client and server (broker). AMQP 1.0 is not backwards compatible with earlier AMQP versions[6].

1.4.2 Backlash to AMQP 1.0

The release of AMQP 1.0 was not without controversy. Version 1.0 was a major departure from the previous AMQP versions and caused significant backlash among the AMQP community. At least two of the major players in the AMQP working group have decided not to implement the 1.0 standard. Even two years after the formal adoption of the AMQP 1.0 specification RabbitMQ does not directly support the AMQP 1.0 standard. iMatix, one of the original members of the AMQP working group has moved away from the AMQP standard all together and now develops a competing messaging system ∅MQ [11, 27].

1.5 AMQP Alternatives

The AMQP specification is not unique in its goal of providing a cross-platform, ubiquitous message protocol. There are several popular specifications in current use and many of MOM implementations tested in this project support support these other protocols. Some are supported natively, while others need to be enabled as plugins.

1.5.1 STOMP

Simple Text Oriented Message Protocol (STOMP) is a text-based protocol designed to work with message oriented middleware systems. It defines a wire-level message encoding format that allows STOMP clients to communicate with any STOMP enabled message broker. It is built on top of TCP/IP and uses a protocol similar to HTTP [21].

1.5.2 OpenWire

OpenWire is a binary protocol used as the native wire format of ActiveMQ [7].

1.5.3 Extensible Messaging and Presence Protocol (XMPP)

XMPP is a communications protocol based on XML. Originally developed by the open-source community for instant messaging, presence information and contact list maintenance [22]

1.5.4 MQTT

MQ Telemetry Transport (MQTT) is a light-weight messaging protocol. It is specifically targeted for applications where a light-weight protocol is needed due to code size limitations. The protocol was invented by Andy Stanford-Clark (IBM) and Arlen Nipper (Cirrus Link Solutions). In 2013 the MQTT protocol specification was submitted to the OASIS specification body. There is a variation of the standard (MQTT-S) for embedded applications on non-TCP/IP networks such as ZigBee. Facebook Messenger uses the MQTT protocol internally for large parts of its functionality [13].

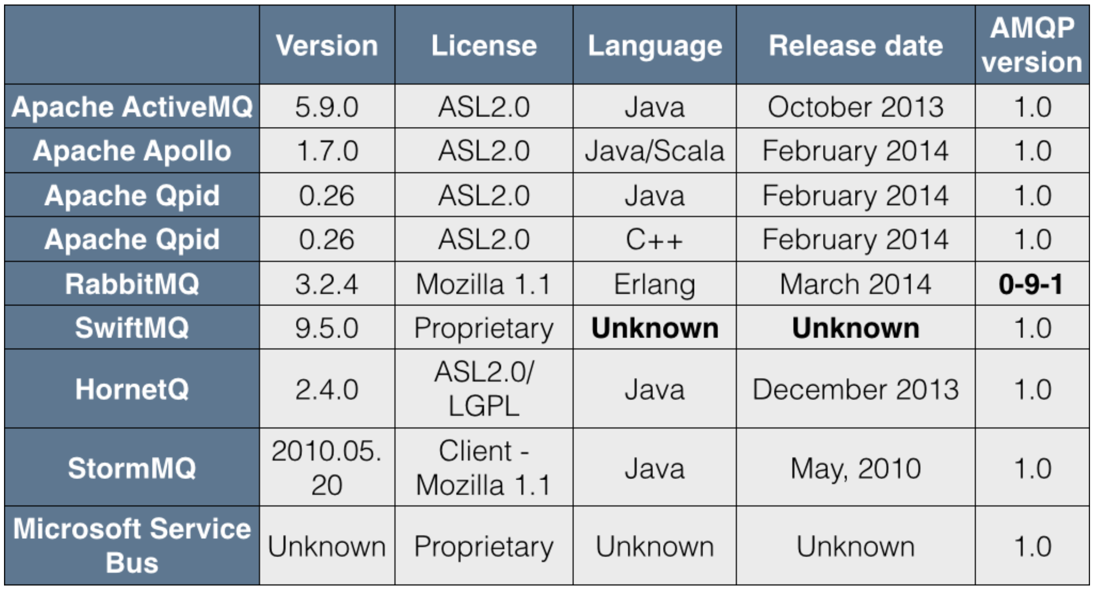
1.5.5 Java Messaging Service (JMS)

The Java EE JMS (Java Messaging Service) was the starting point for attempting to standardize MOM. Unfortunately, JMS only specifies an application programming interface (API) and does not specify a format for exchanged messages so it does not create interoperability. To achieve this desired interoperability other standards such as AMQP built upon the JMS concept.

1.6 AMQP implementations

1.6.1 Broker implementations

There are several major open source AMQP broker implementations.



	Version	License	Language	Release date	AMQP version
Apache ActiveMQ	5.9.0	ASL2.0	Java	October 2013	1.0
Apache Apollo	1.7.0	ASL2.0	Java/Scala	February 2014	1.0
Apache Qpid	0.26	ASL2.0	Java	February 2014	1.0
Apache Qpid	0.26	ASL2.0	C++	February 2014	1.0
RabbitMQ	3.2.4	Mozilla 1.1	Erlang	March 2014	**0-9-1**
SwiftMQ	9.5.0	Proprietary	**Unknown**	**Unknown**	1.0
HornetQ	2.4.0	ASL2.0/ LGPL	Java	December 2013	1.0
StormMQ	2010.05. 20	Client - Mozilla 1.1	Java	May, 2010	1.0
Microsoft Service Bus	Unknown	Proprietary	Unknown	Unknown	1.0

Figure 1.3: AMQP versions

For a long time Apache ActiveMQ was the most popular implementations, but recently RabbitMQ has recently overtaken ActiveMQ as the most popular choice. RabbitMQ is a components in the popular Java Spring framework and this is a factor in the implementation's increased popularity.

1.6.2 RabbitMQ

RabbitMQ is an open source message broker written in the Erlang programming language [19] RabbitMQ is developed by and supported by Rabbit Technolo-

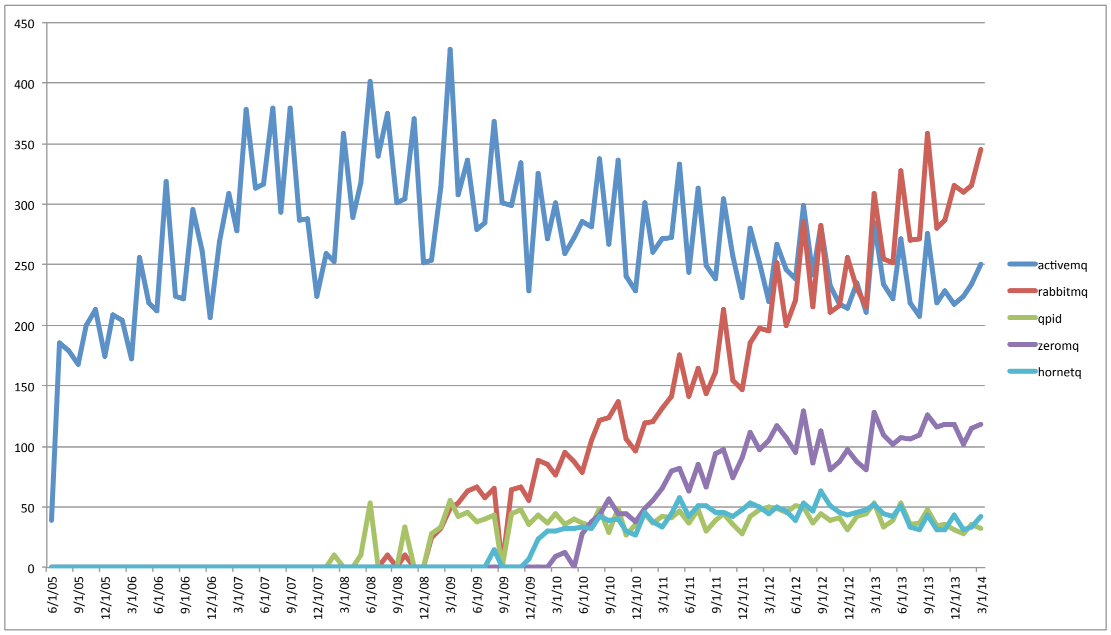


Figure 1.4: Google trend data for AMQP implementations

gies Ltd. It is actively being developed, the latest stable version (3.2.4) was released on March 4, 2014. It is licensed under the Mozilla Public License. The development history/management of RabbitMQ is complicated. Originally developed by Lshift, the project was moved to an independent company Rabbit Technologies Ltd, which was cofounded by Monadic and CohesiveFT. In 2013, Rabbit Technologies Ltd. was acquired by SpringSource, a division of VMware, Inc. [12]. As of March 2014 - commercial support for RabbitMQ was being provided by Pivotal, Ltd [17]. RabbitMQ is cross platform and has bindings/clients for a wide range of systems/programming languages [18].

1.6.3 ActiveMQ

ActiveMQ is an open source message broker written in Java. ActiveMQ is developed by the Apache Software Foundation and as of the writing of this report the most recent stable version is 5.9.0 released on October 21, 2013. ActiveMQ is part of Apache's enterprise service bus (ESB) implementations, and is leveraged in other project such as Apache Camel and Apache CXF to Service Orientated Architecture (SOA) projects [2].

1.6.4 ApolloMQ

ApolloMQ is an open source message broker written in Java. It is a fork of Apache ActiveMQ with the primary difference being the threading model and message dispatching architecture. ApolloMQ maintains multi-protocol support and supports STOMP, AMQP 1.0, MQTT, OpenWire, SSL and WebSockets [3].

1.6.5 HornetQ

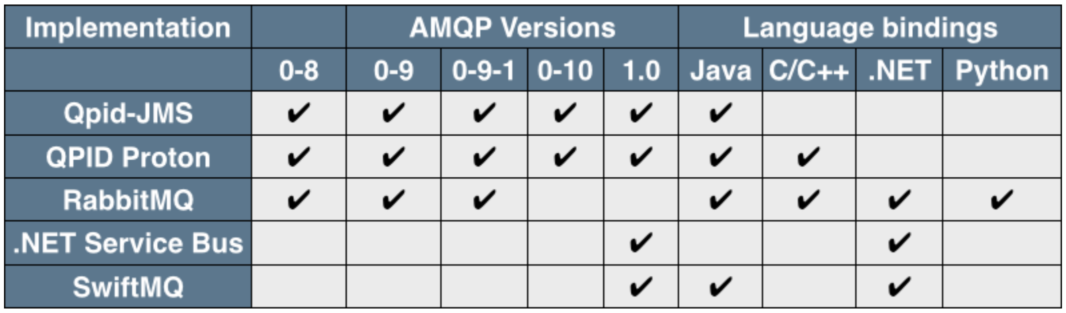
HornetQ is an open source, multi-protocol messaging system developed from JBoss, a division of Red Hat, Inc. It can be integrated with JBoss Application Server or embedded into standalone applications. As of version 2.2 it supports the AMQP1.0 specification [10]. HornetQ is released under the Apache Software License version 2.0. There are a few components that are released under LGPL but the plan is to develop a pure ASL2.0 version in the future . HornetQ is written in Java and supports and platform with a Java 5 or later runtime. It is actively being developed and maintained with the last stable version 2.4.0 being released on December 16, 2013. [9].

1.6.6 Client implementations

There are many different AMQP client libraries for connecting a software components to an AMQP broker. There are ports for most programming languages, for example RabbitMQ lists 175 different client libraries for over 21 different programming languages [18]. The maturity and functionally for each client vary greatly. Many of the client libraries are open source and have low activity, but there are several open source client libraries with active support. There are also closed source, proprietary implementations and while they may be free to use there are license restrictions that make them unusable for this project. [2] One of the most active and widely used client libraries is Apache's Qpid Proton library [5], which has both C and Java implementations.

For this project, Apache's Qpid JMS client library was used. This client provides an AMQP fluent client library using the Java JMS 1.1 API. At the time

[2]Specificially SwiftMQ has a clause in their license that prohibits publishing benchmarking results without company approval.



Implementation		AMQP Versions				Language bindings			
	0-8	0-9	0-9-1	0-10	1.0	Java	C/C++	.NET	Python
Qpid-JMS	✔	✔	✔	✔	✔	✔			
QPID Proton	✔	✔	✔	✔	✔	✔	✔		
RabbitMQ	✔	✔	✔			✔	✔	✔	✔
.NET Service Bus					✔			✔	
SwiftMQ					✔	✔		✔	

Figure 1.5: Summary of available AMQP clients

of this project's development, version 0.26 was the most recent release of this client library. Apache Qpid JMS is a pure Java implementation and supports a significant amount of AMQP concepts such as: failover, heartbeats, transactions, SSL, flow control, and authentication. There are many examples available and a pre-built library is available as a maven dependency [4].

2. Methodology

2.1 Overview

This project was decomposed into three separate phases. This approach was taken so that verification and validation of the collected data could be performed.

For the first phase, baseline performance data was collected on the performance of each AMQP broker implementation in isolation. A very simple broker network configuration was used, with a single broker, consumer and producer.

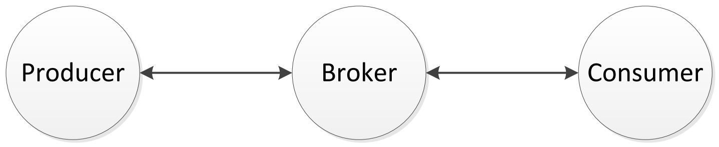


Figure 2.1: Simple AMQP broker network

There have been many experiments run with this configuration [26] [25]. While this step is not novel it allowed for comparison of the results with independent AMQP evaluations to ensure that the developed benchmark produces reliable results.

The second phase for this project created simple broker networks of homogenous AMQP broker implementations and collect benchmark information. The broker network configurations followed the designs described by Marsh, Sampat, Potluri, Panda in [30]. This again allowed for comparison independent evaluation to ensure that the collected results are valid.

The third phase was to create heterogeneous broker networks using the same broker network architectures in step two. The goal of this was to investigate the impact of integrating different AMQP broker implementations into a single system.

Each step was completed independently using the same lab setup, computing hardware and software. As much as possible outside factors were minimized to help ensure that the collected data is comparable and independent of interference.

AMQP brokers are complicated software components, with many interacting parts so care must be taken to ensure that the data collected is reliable.

2.2 Hardware

A lab environment was setup to isolate the tests and minimize outside interference. The test hardware was over-specified to ensure that the benchmark hardware was not a limiting factor in the experiment. A high-end workstation was used to host the AMQP message broker networks. Business class laptop computers were used to run the benchmarking software. All components were connected with as business class network switch. The workstation hosting the broker networks used the latest Long-Term Support version of the Ubuntu Server operating system (12.04LTS) and was configured to minimize background daemons and processes. This workstation was setup as an Network Time Protocol (NTP) reference in order to synchronize the times between all of the computers used. A full list of the computing components and network resources used for this project is included in Figure2.3.

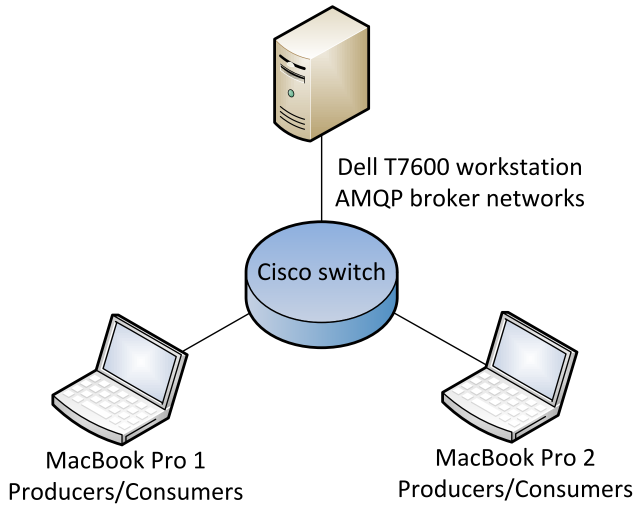
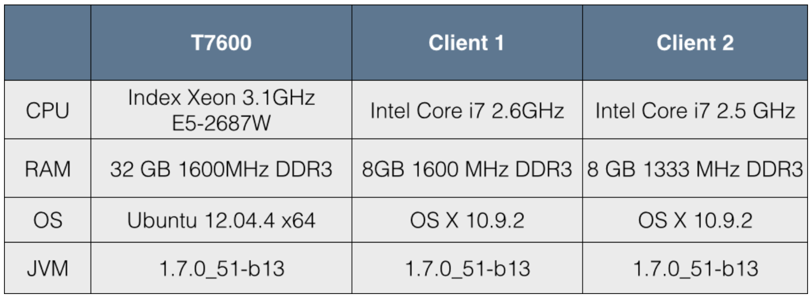


Figure 2.2: Hardware setup



	T7600	Client 1	Client 2
CPU	Index Xeon 3.1GHz E5-2687W	Intel Core i7 2.6GHz	Intel Core i7 2.5 GHz
RAM	32 GB 1600MHz DDR3	8GB 1600 MHz DDR3	8 GB 1333 MHz DDR3
OS	Ubuntu 12.04.4 x64	OS X 10.9.2	OS X 10.9.2
JVM	1.7.0_51-b13	1.7.0_51-b13	1.7.0_51-b13

Figure 2.3: Hardware components

2.3 Network design

The design of the test network was kept as simple as possible. The goal was to ensure that the network overhead was not the limiting factor in the benchmarks measurement.

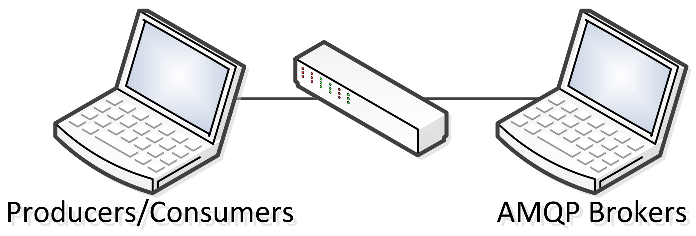


Figure 2.4: Network overview

2.4 Benchmark design

The design of the benchmarks for this experiment were modeled after the work done by Subramoni, Marsh, Narravula, Lai and Panda in [32] which in turn based their benchmarks on Ohio State University Micro-benchmarks [15]. There are two fundamental components to the benchmark application, an AMQP message producer and AMQP message consumer. A high level overview of the software components can be seen in Figure 2.5.

The message producer is a multithreaded component responsible for sending example AMQP messages and recording message acknowledgments. The function-

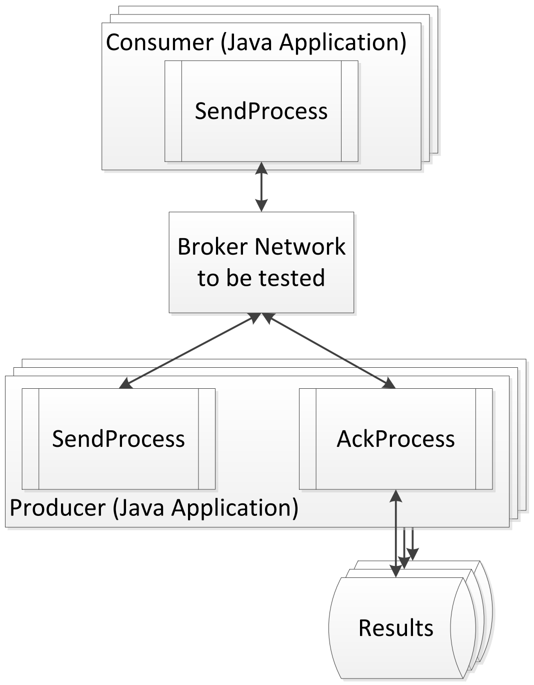

Consumer (Java Application)
SendProcess
Broker Network to be tested
SendProcess
AckProcess
Producer (Java Application)
Results

Figure 2.5: High level benchmark design

ality of sending messages was separated from the message acknowledgment management to minimize the impact of writing to disk on the benchmark's accuracy.

The message consumer is a simple component, its job is to receive AMQP messages and send a response message back to the original message producer that contains a timestamp of when the source message was received.

2.4.1 Message Format

AMQP supports a wide variety of message formats. For this experiment the "BytesMessage" format was used exclusively, this uses the binary encoding format as specified in AMQP1.0.

2.4.2 Message Consumer

The message consumer is setup to run as a simple state machine as shown in Figure2.6. The module takes a single command line option, the name of a java context configuration file. On initialization, the component reads its configuration the file specified on the command line and connects to the AMQP broker at the address specified in the configuration file. If configured, it creates the message and acknowledgment channel, and then enters a loop waiting for input messages.

As messages are received, the module creates a small response message that contains the time the input message was sent from the producer, a unique client identification number, and the time the response message was sent. If the module detects an out of order message ID, it sends a warning message back to the message producer by setting a flag to -1. If a message with ID = -1 is received, the module sends a response to the producer, setting the message ID of the response to -1, exits its main processing loop and closing the AMQP connection. Note, that for some of the broker implementations closing the AMQP connection resulted in a software exception.

The source code for the message consumer module can be found in AppendixA.

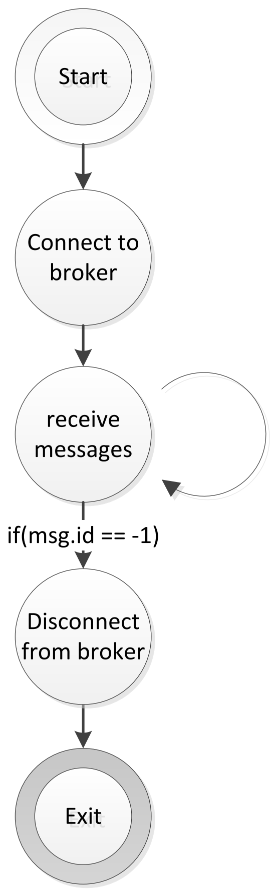


Figure 2.6: Consumer state diagram

2.4.3 Message Producer

To get accurate benchmark results the message producer needed to be split into two separate components: a bandwidth benchmark message producer and a latency benchmark message producer. Both components are very similar and follow the same basic state diagram as shown in Figure2.7. Both components take a single command line option, the name of a java context configuration file. The major difference between the bandwidth and latency benchmark message producers is how the modules send each message. In the bandwidth benchmark the module contains two threads. One thread to send messages and another to listen for responses and write benchmarking data to file. This allows the message sender thread to take advantage of buffering options internal to the AMQP client and broker and greatly increases the bandwidth that can be achieved by the broker implementation. For the latency benchmark, this buffering leads to inaccurate results, and instead the message producer has a single thread. It sends messages one at a time and waits for the response from the message consumer. This ensures thats no buffering is performed by either the AMQP client library or message broker and ensures valid latency measurements. On initialization, both components reads its configuration from the file specified on the command line, and depending on the broker implementation either creates the AMQP queues or attaches to pre-configured queues on the broker instance specified in the configuration file. The source code for the latency-benchmark message producer can be found in AppendixB.The source code for the bandwidth-benchmark message producer can be found in AppendixC.

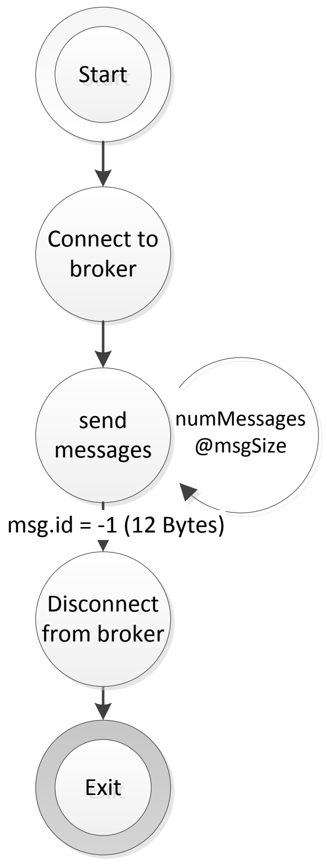


Figure 2.7: Producer state diagram

2.4.4 Format of collected data

For simple analysis, data will be written from the benchmark application into comma-separated-value (CSV) files. The effective bandwidth is calculated by the message producer component and written to the file. The time that each message is sent and the response time for each message is written to the CSV file so that the round trip time (RTT) and one-way time (OWT) can easily be calculated.

3. Results

Creating a simple AMQP producer and consumer proved to be more difficult than expected. Even though all of the AMQP broker implementations chosen to benchmark advertise support for the AMQP 1.0 standard, the client programs needed specific customizations to successfully send messages through the AMQP broker. So much configuration was needed for the benchmarking modules that it was necessary to change from command line parsing for configuration options to using separate context files. At one point, both the producer and consumer modules required 10 separate command line arguments. Using context files greatly simplified management of program options. Even with a context file, specific configuration sections were required in the source code of the consumer and producer components in order to support the different broker implementations, refer to the source code below for the level of customization required.

```
//APOLLO
if(brokerType.equalsIgnoreCase("APOLLO"))
{
   ackChannelDest = new QueueImpl("queue://acks");
   msgChannelDest = new QueueImpl("queue://msgs");

   consumer = consumerSession.createConsumer(msgChannelDest, null);
   ackProducer = producerSession.createProducer(ackChannelDest);
}
//HORNETQ
else if(brokerType.equalsIgnoreCase("HORNETQ"))
{
   // Create message consumer
   consumer = consumerSession.createConsumer(msgQueue, "color = red");
   // Create message ack
   ackProducer = producerSession.createProducer(ackQueue);
}
```

```
else
{
   // Create message consumer
   consumer = consumerSession.createConsumer(msgQueue);
   // Create message ack
   ackProducer = producerSession.createProducer(ackQueue);
}
```

This goes against the very idea of having a common API to access the messaging services. Part of this was from the different use cases that each broker implementation addresses. Apache Qpid (cpp) and Apache Qpid (Java) are both designed to be setup and statically configured. In order to get these brokers working it was necessary to use an external configuration utility (command-line based for Qpid(cpp), web-based for Qpid(Java)) to setup the broker exchanges and queues. ActiveMQ and Apollo were much more flexible for dynamic exchange creation.

3.1 Single Broker Benchmark Results

From the original list of nine AMQP broker implementations 1.3, only four (ActiveMQ, Qpid-cpp, Qpid-Java, Apollo) resulted in successful data collections. The brokers that failed did so for a variety of reasons. RabbitMQ was a nonstarter. While they claim to have an AMQP 1.0 plugin to enable the protocol on their broker implementation, no amount of configuration of either the broker or the client resulted in a working message passing instance. Investigation into the issue revealed that in spite of adding and enabling the AMQP 1.0 plugin to RabbitMQ the broker was identifying the underlying transport as 0-9-1. Frustratingly, although the Qpid JMS Client library also claimed to be configurable to support AMQP 0-9-1 no amount of client side configuration or modification of the benchmark software resulted in a viable message passing system. StormMQ is totally cloud based - they do not let you instantiate a broker instance locally and was therefore untestable. SwiftMQ was better, and while using their broker did result in a working configuration, their license specifically disallows the release of benchmarking information. The real surprise was HornetQ, which while it supports AMQP1.0 natively never worked.

Several configurations were tried, both building from source and using pre-built binaries. Their built in clients seemed to work which are run as part of their build process, but no matter the options that were added to the benchmark software, it was never able to create a valid AMQP channel. The specific error message seemed to be an issue with accessing the broker using a JMS style client API. For the implementations that remain the results of benchmarking can be see in Figure3.1 and Figure3.2. The results collected make sense, as the size of the message increases, the throughput drops, but the bandwidth increases. In general all four broker implementations were fairly evenly matched. Surprisingly, all of the brokers had issues sending larger messages. As the test message size was increased, both client and broker errors were observed.

3.2 Homogenious Broker Network Benchmark Results

3.2.1 ActiveMQ-ActiveMQ

For ActiveMQ-ActiveMQ bandwidth results see Figure3.3. For ActiveMQ-ActiveMQ throughput results see Figure3.4. For ActiveMQ-ActiveMQ latency results see Figure3.6.

3.2.2 QPID-QPID

For Qpid-Qpid bandwidth results see Figure3.7. For Qpid-Qpid throughput results see Figure3.8. For Qpid-Qpid latency results see Figure3.10.

3.3 Hetergenious Broker Network Benchmark Results

Unfortunately, because ActiveMQ did not support federation using the AMQP protocol, the ultimate goal of measuring the performance of heterogeneous broker networks was not possible.

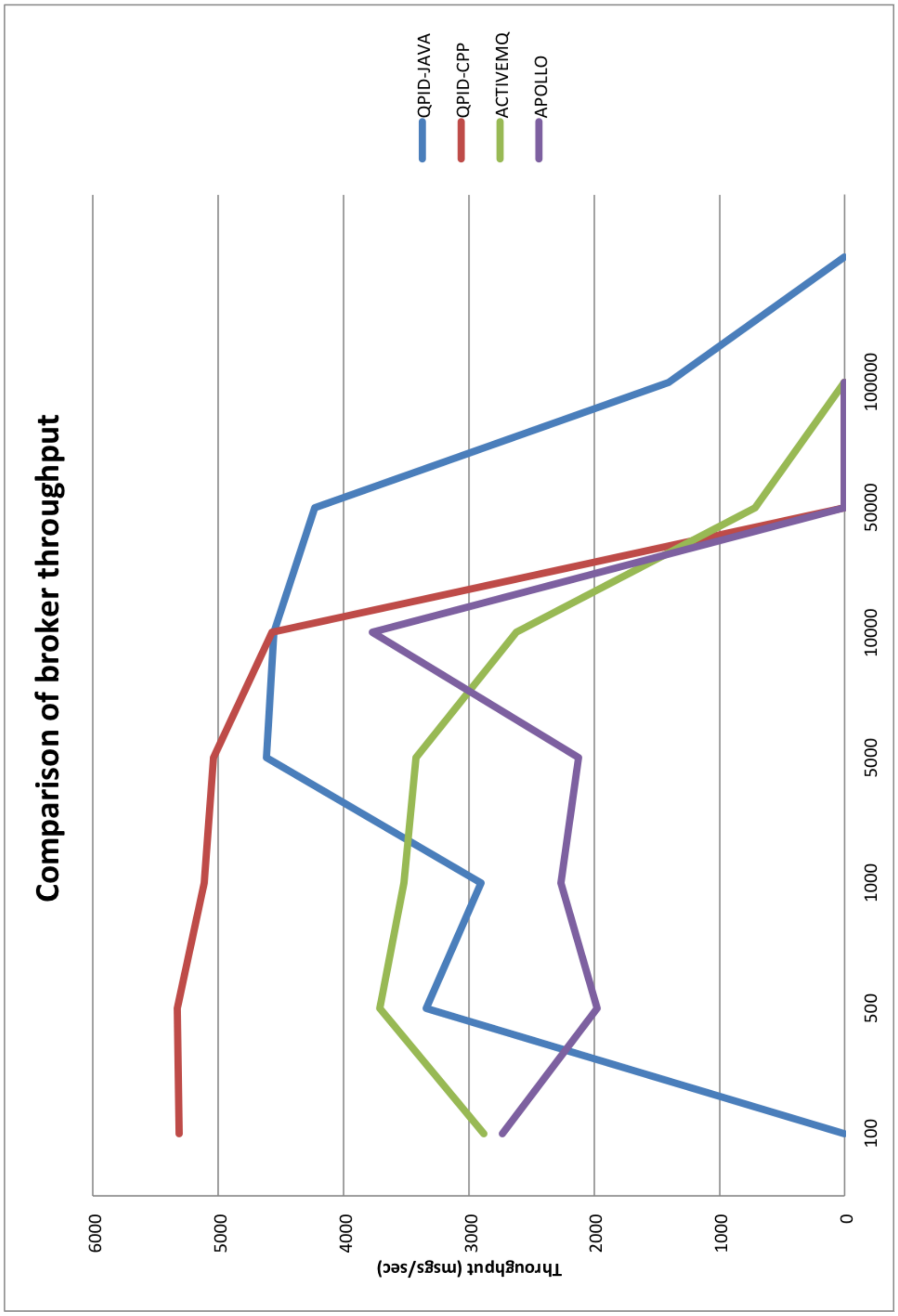


Figure 3.1: Throughput

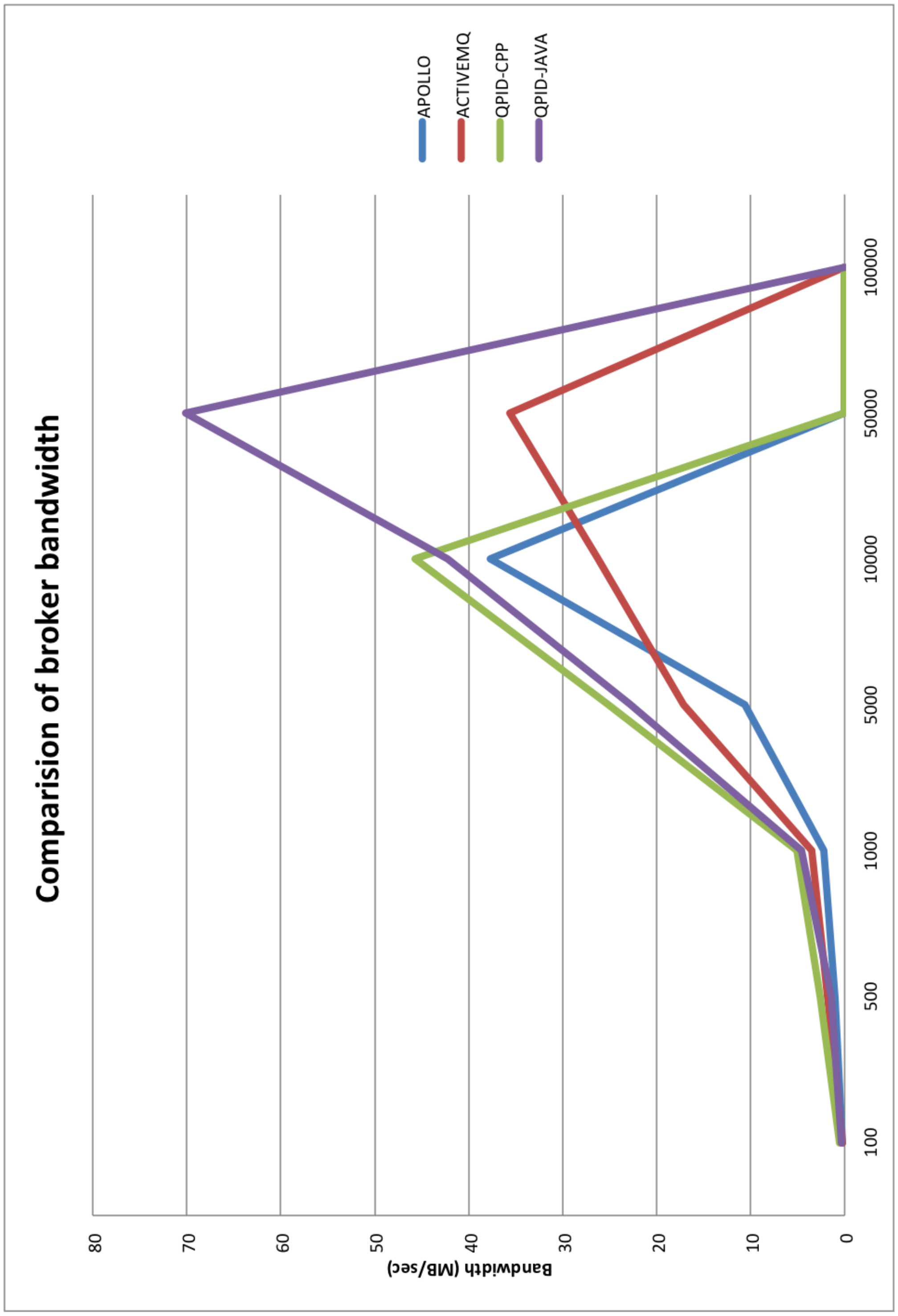


Figure 3.2: Bandwidth

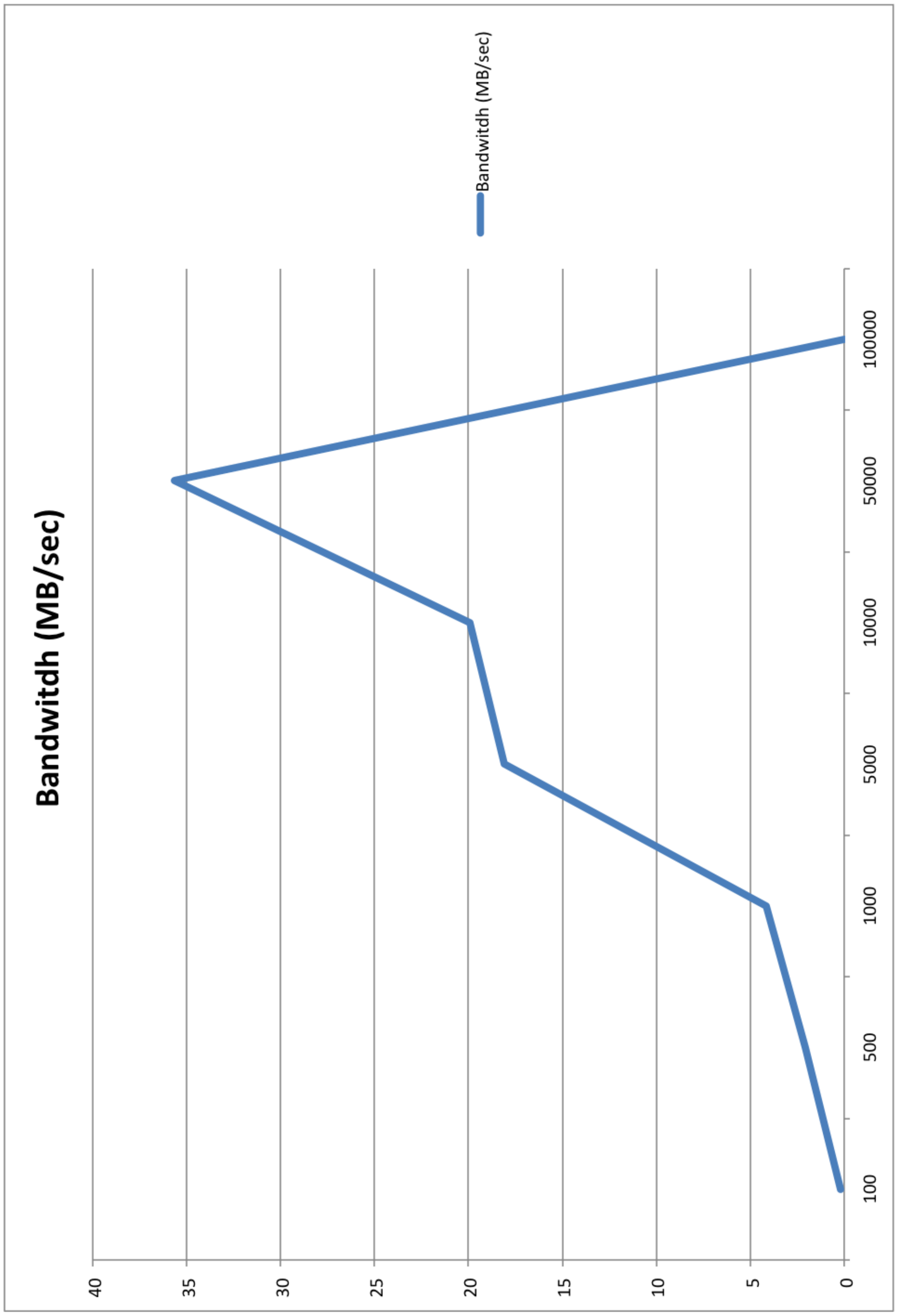


Figure 3.3: ActiveMQ-ActiveMQ bandwidth

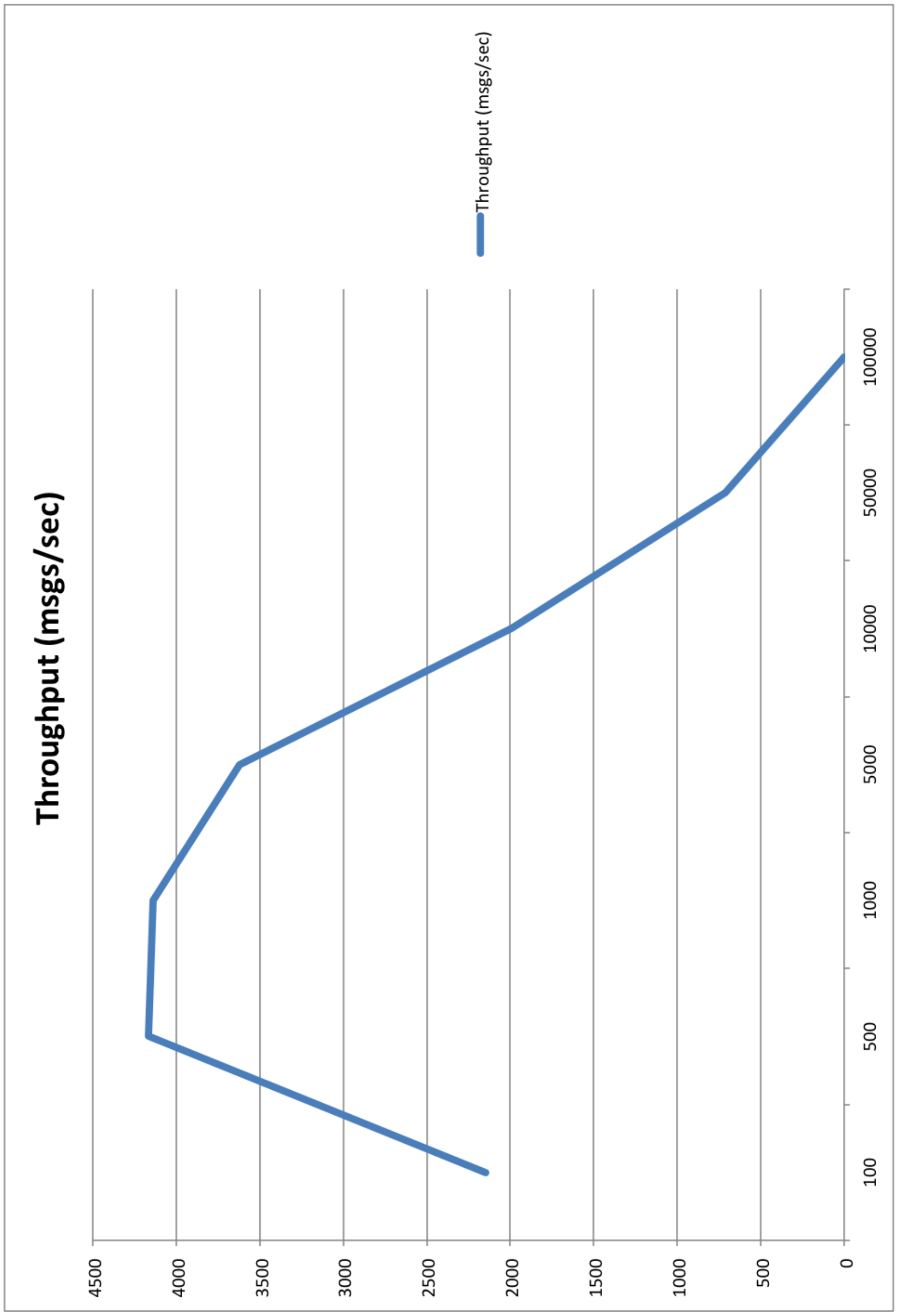


Figure 3.4: ActiveMQ-ActiveMQ throughput

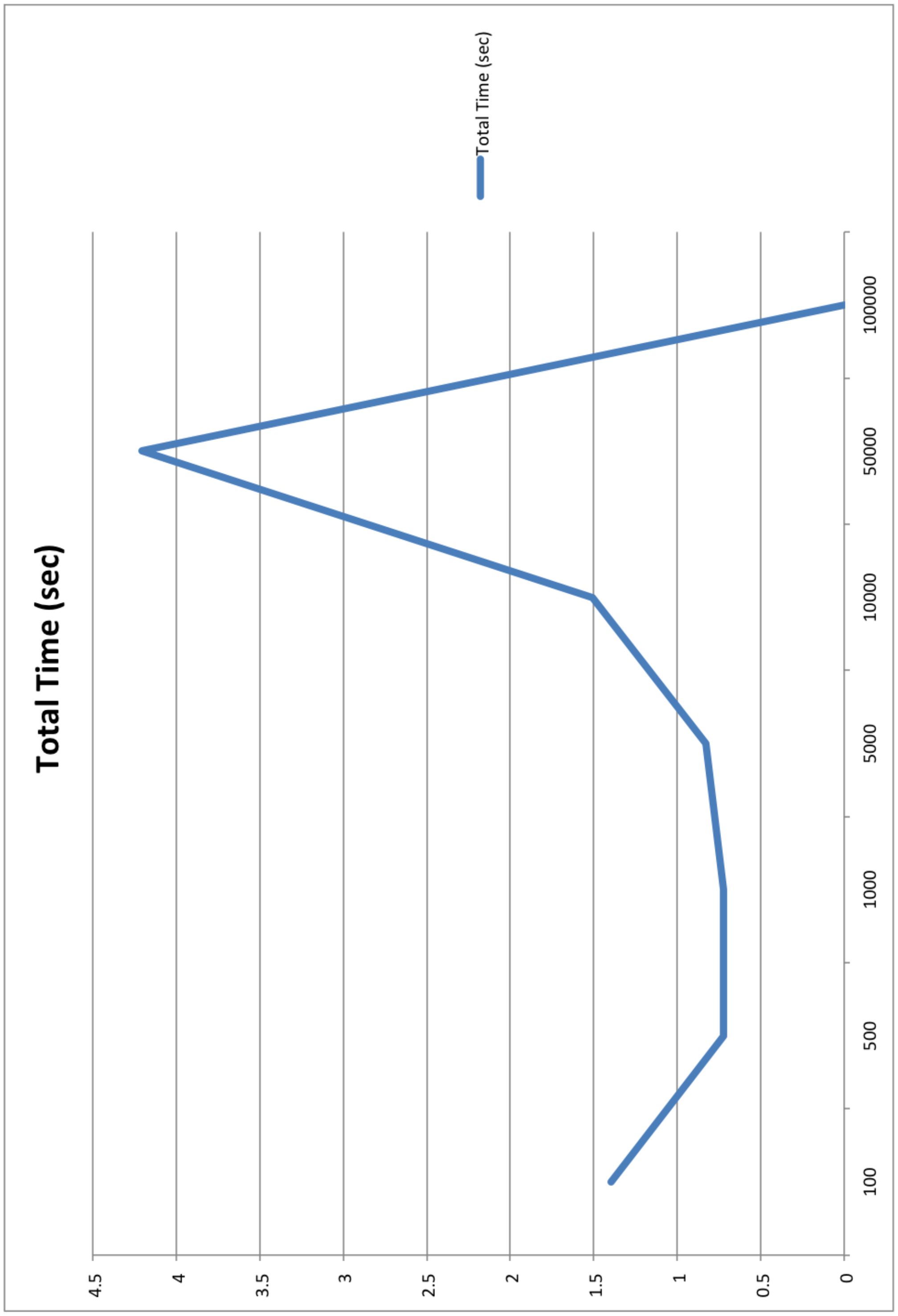


Figure 3.5: ActiveMQ-ActiveMQ total time

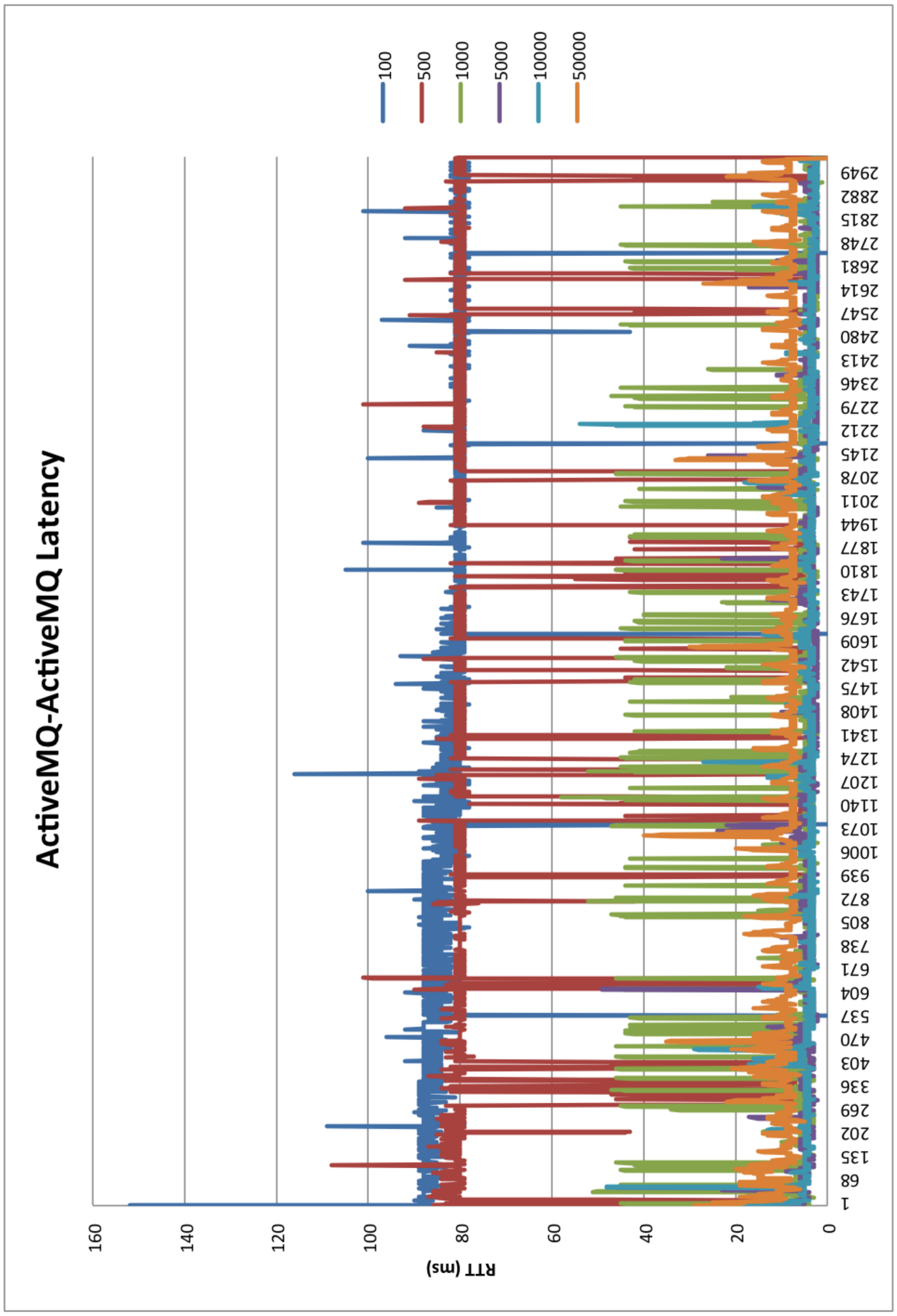


Figure 3.6: ActiveMQ-ActiveMQ latency

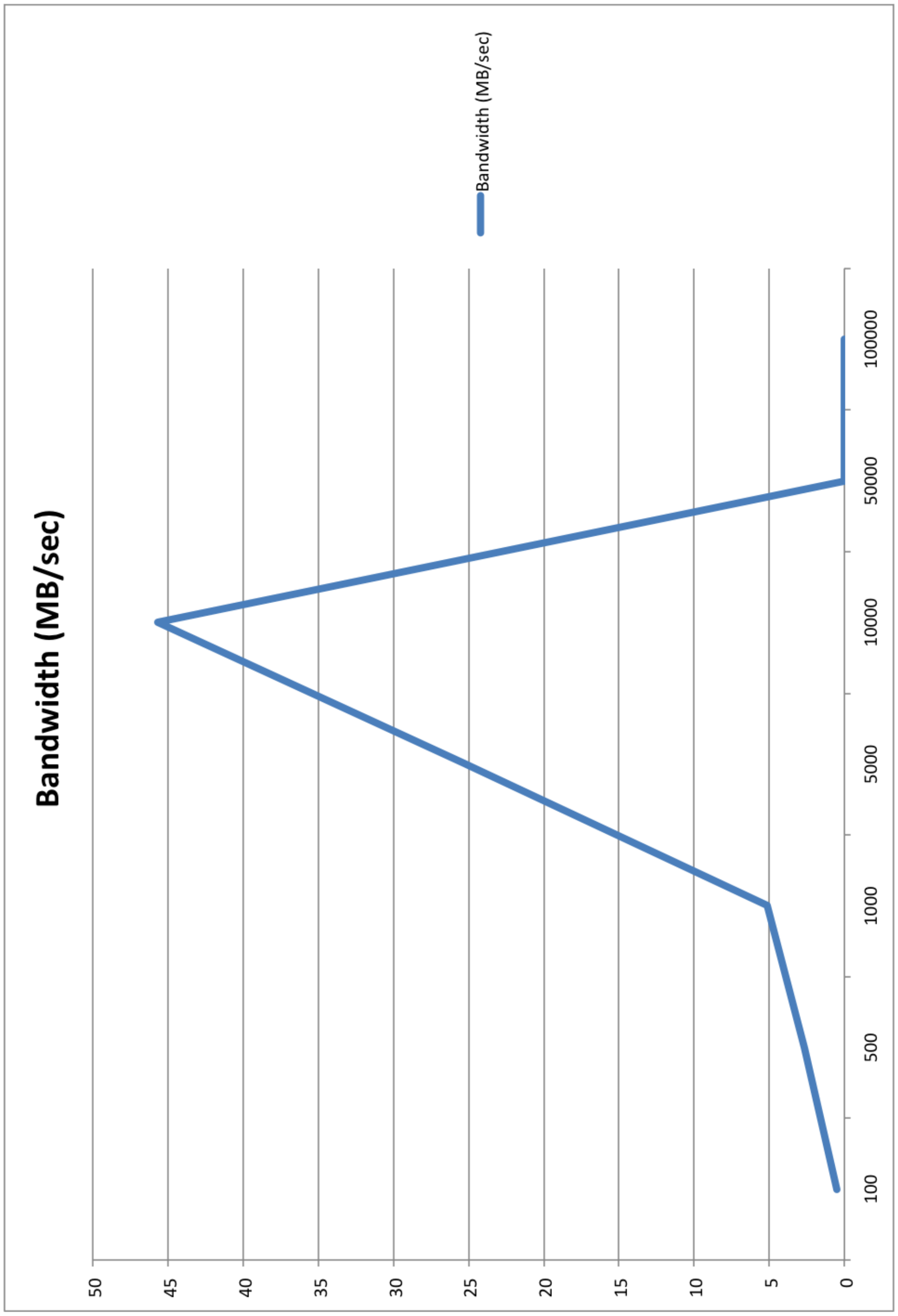


Figure 3.7: Qpid(CPP)-Qpid(CPP) bandwidth

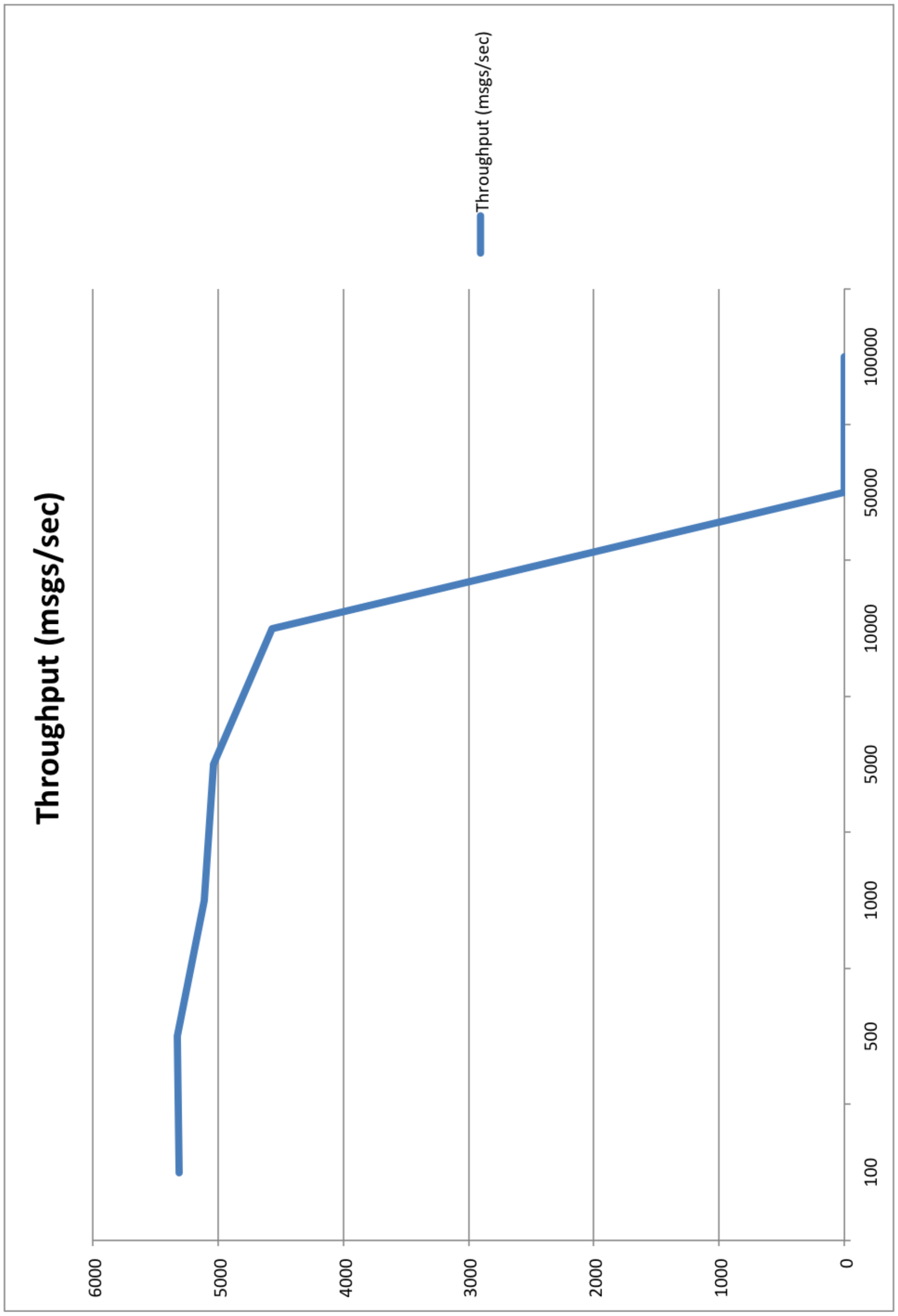


Figure 3.8: Qpid(CPP)-Qpid(CPP) throughput

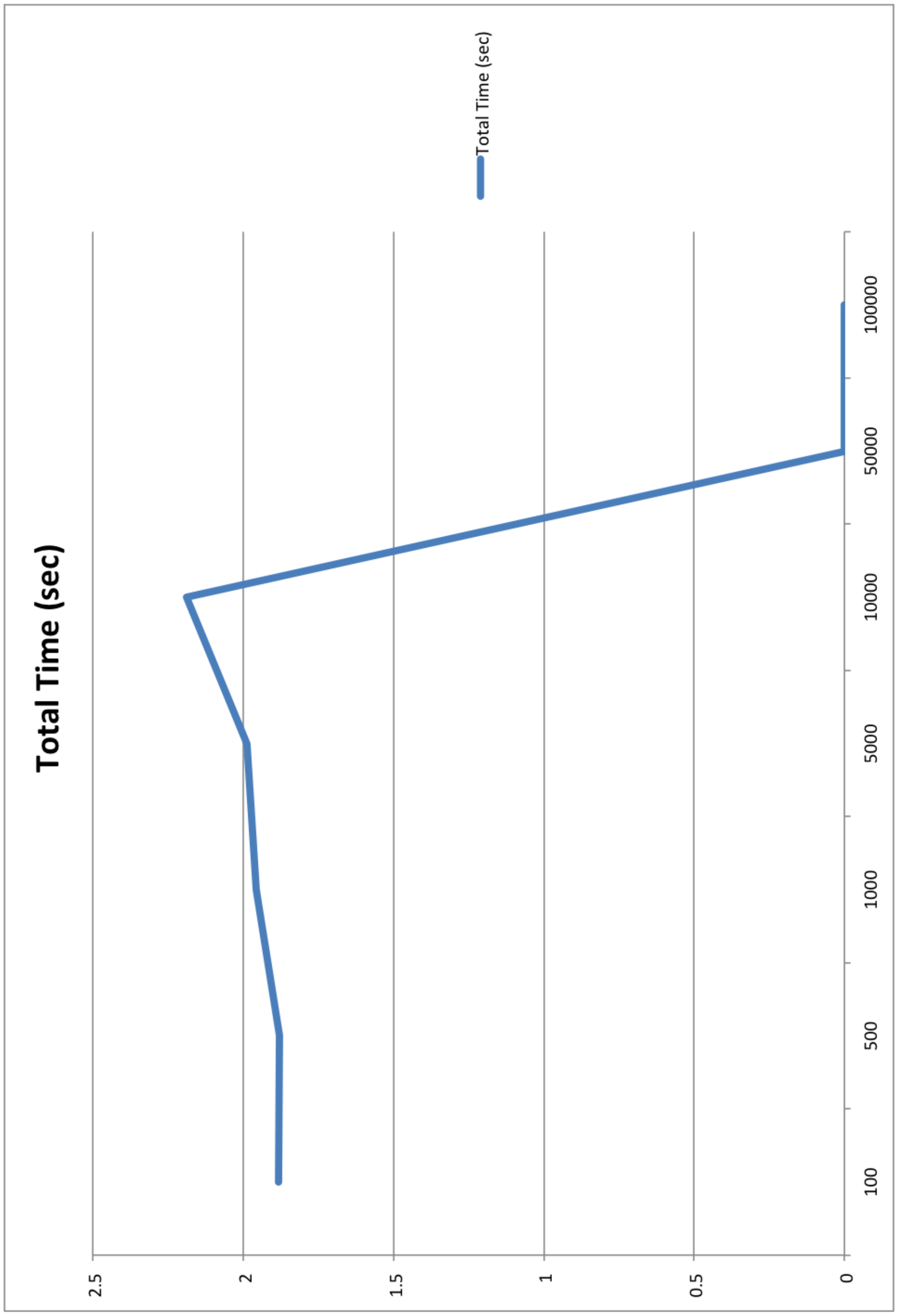


Figure 3.9: Qpid(CPP)-Qpid(CPP) total time

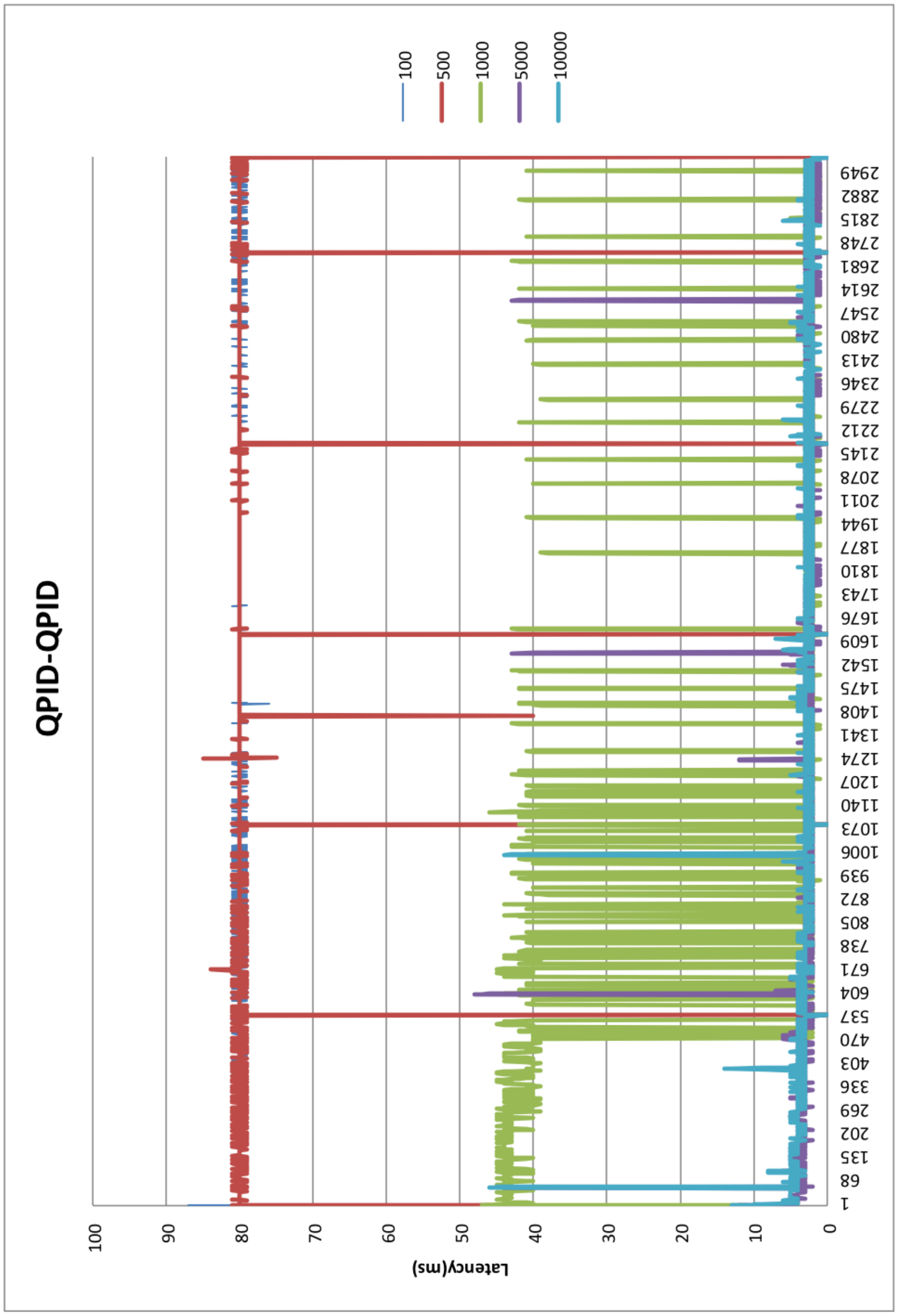


Figure 3.10: Qpid(CPP)-Qpid(CPP) latency

BIBLIOGRAPHY

[1] AMQP Specifications. http://www.amqp.org/specification. [Online; accessed 17-April-2014].

[2] Apache ActiveMQ Homepage. http://activemq.apache.org. [Online; accessed 17-April-2014].

[3] Apache Apollo Homepage. https://activemq.apache.org/apollo. [Online; accessed 17-April-2014].

[4] Apache Qpid JMS library. https://qpid.apache.org/components/qpid-jms/index.html. [Online; accessed 17-April-2014].

[5] Apache Qpid Proton Library. http://qpid.apache.org/components/messenger/index.html. [Online; accessed 17-April-2014].

[6] Backwards compatibility of AMQP versions. http://www.amqp.org/resources/download. [Online; accessed 17-April-2014].

[7] Default protocol of ApacheMQ. http://activemq.apache.org/openwire.html. [Online; accessed 17-April-2014].

[8] Differences between AMQP 0-10 and AMQP 1.0. https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_MRG/2/html/Messaging_Programming_Reference/Differences_between_AMQP_0-10_and_AMQP_1.0.html. [Online; accessed 17-April-2014].

[9] HornetQ homepage. http://www.jboss.org/hornetq. [Online; accessed 17-April-2014].

[10] HORNETQ Roadmap. https://community.jboss.org/wiki/Roadmap. [Online; accessed 17-April-2014].

[11] iMatix ZeroMQ homepage. http://zeromq.org/. [Online; accessed 17-April-2014].

[12] LShift Homepage. http://www.lshift.net/blog/2010/04/13/rabbitmq-2. [Online; accessed 17-April-2014].

[13] MQTT Telemetry Transport. http://mqtt.org/. [Online; accessed 17-April-2014].

[14] OASIS Homepage. https://www.oasis-open.org/org. [Online; accessed 17-April-2014].

[15] Ohio State University micro benchmarks. http://mvapich.cse.ohio-state.edu/benchmarks. [Online; accessed 17-April-2014].

[16] ORACLE JMS Tutorial. http://docs.oracle.com/javaee/6/tutorial/doc/bncdq.html. [Online; accessed 17-April-2014].

[17] PIVOTAL Homepage. http://www.gopivotal.com/support. [Online; accessed 17-April-2014].

[18] RabbitMQ clients. http://www.rabbitmq.com/devtools.html. [Online; accessed 17-April-2014].

[19] RabbitMQ Homepage. http://www.rabbitmq.com. [Online; accessed 17-April-2014].

[20] RabbitMQ transport protocols. http://www.rabbitmq.com/protocols.html. [Online; accessed 17-April-2014].

[21] STOMP homepage. http://stomp.github.io/. [Online; accessed 17-April-2014].

[22] XMPP Standards Foundation. http://xmpp.org/. [Online; accessed 17-April-2014].

[23] AMQP Working Group Transitions to OASIS Member Section. http://www.amqp.org/node/54, 2011. [Online; accessed 17-April-2014].

[24] OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-transport-v1.0-os.xml, OCT 2012. [Online; accessed 17-April-2014].

[25] Thomas Bayer. ActiveMQ, Qpid, HornetQ and RabbitMQ in Comparison. http://predic8.com/activemq-hornetq-rabbitmq-apollo-qpid-comparison.htm, 2013. [Online; accessed 17-April-2014].

[26] H Chirino. STOMP Messaging Benchmarks: ActiveMQ vs Apollo vs HornetQ vs RabbitMQ. http://hiramchirino.com/blog/2011/12/stomp-messaging-benchmarks-activemq-vs-apollo-vs-hornetq-vs-rabbitmq/, 2011. [Online; accessed 17-April-2014].

[27] Pieter Hintjens. iMatix will end OpenAMQ support by 2011. http://lists.openamq.org/pipermail/openamq-dev/2010-March/001598.html, 2010. [Online; accessed 17-April-2014].

[28] B. Cameron J. Apps. AMQP and RabbitMQ - Message Queueing as an Integration Mechanism. http://www.openvms.org/skonetski/Webinar_2012_09_AMQP_RabbitMQ_Message_Queuing_as_Integration_Mechanism.pdf, 2012. [Online; accessed 17-April-2014].

[29] Joshua Kramer. Advanced message queuing protocol (amqp). *Linux Journal*, 2009(187):3, 2009.

[30] Gregory Marsh, Ajay P Sampat, Sreeram Potluri, and Dhabaleswar K Panda. Scaling advanced message queuing protocol (amqp) architecture with broker federation and infiniband. Technical report, technical report, available at ftp://ftp. cse. ohio-state. edu/pub/tech-report/2009/TR17. pdf, accessed on Sept. 8th, 2010.

[31] John O'Hara. Toward a commodity enterprise middleware. *Queue*, 5(4):48–55, 2007.

[32] Hari Subramoni, Gregory Marsh, Sundeep Narravula, Ping Lai, and Dhabaleswar K Panda. Design and evaluation of benchmarks for financial applications using advanced message queuing protocol (amqp) over infiniband. In *High Performance Computational Finance, 2008. WHPCF 2008. Workshop on*, pages 1–8. IEEE, 2008.

[33] Hubert Zimmermann. Osi reference model–the iso model of architecture for open systems interconnection. *Communications, IEEE Transactions on*, 28(4):425–432, 1980.

APPENDIX A
CONSUMER SOURCE CODE

```
package benchmarks;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Properties;

import org.apache.qpid.amqp_1_0.jms.impl.*;

import javax.jms.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.Logger;

class Consumer
{
static final Logger logger = Logger.getLogger(Consumer.class.getName());

public static void main(String []args) throws JMSException,
    NamingException
{
boolean running = true;
int clientId = 0;
int nextExpectedMsgId = 1;
int messageId = 0;
int producerId = 0;
long sentTime = 0;
int msgSize = 0;
int messagesRecvd = 0;

logger.trace("Entering application.");

Destination msgChannelDest = null;
Destination ackChannelDest = null;
Connection connection;
Session consumerSession;
Session producerSession;
String contextFileName = null;
MessageConsumer consumer;
MessageProducer ackProducer;
String brokerType = null;
InitialContext context = null;
Hashtable<String, String> env = new Hashtable<String, String>();
env.put(Context.INITIAL_CONTEXT_FACTORY,
    "org.apache.qpid.amqp_1_0.jms.jndi.PropertiesFileInitialContextFactory");

// Read command line args
if(args.length == 1)
{
   contextFileName = args[0];
}
else
{
   System.out.println("Usage: Consumer contextFileName");
   System.exit(-1);
}

logger.debug("context file name is: " + contextFileName);
```

```
env.put(Context.PROVIDER_URL, contextFileName);

try
{
   context = new InitialContext(env);
} catch (NamingException e1)
{
   e1.printStackTrace();
}

Properties properties = new Properties();
try
{
   properties.load(new FileInputStream(contextFileName));
} catch (IOException e)
{
   e.printStackTrace();
}

// Lookup ConnectionFactory and Queue from the context factory
ConnectionFactory connectionFactory = (ConnectionFactory)
    context.lookup("brokerURI");
connection = connectionFactory.createConnection();
Queue msgQueue = (Queue) context.lookup("MSGS");
Queue ackQueue = (Queue) context.lookup("ACKS");

clientId = Integer.parseInt(properties.getProperty("clientId"));
brokerType = properties.getProperty("brokerType");

logger.debug("clientId is: " + clientId);

//session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

```
producerSession = connection.createSession(false,
    Session.AUTO_ACKNOWLEDGE);
consumerSession = connection.createSession(false,
    Session.AUTO_ACKNOWLEDGE);

connection.start();

//APOLLO
if(brokerType.equalsIgnoreCase("APOLLO"))
{
   ackChannelDest = new QueueImpl("queue://acks");
   msgChannelDest = new QueueImpl("queue://msgs");

   consumer = consumerSession.createConsumer(msgChannelDest, null);
   ackProducer = producerSession.createProducer(ackChannelDest);
}
//HORNETQ
else if(brokerType.equalsIgnoreCase("HORNETQ"))
{
   // Create message consumer
   consumer = consumerSession.createConsumer(msgQueue, "color = red");
   // Create message ack
   ackProducer = producerSession.createProducer(ackQueue);
}
else
{
   // Create message consumer
   consumer = consumerSession.createConsumer(msgQueue);
   // Create message ack
   ackProducer = producerSession.createProducer(ackQueue);
}

BytesMessage ack = null;
```

```
try
{
   ack = producerSession.createBytesMessage();
} catch (JMSException e)
{
   e.printStackTrace();
}

long start = System.currentTimeMillis();
nextExpectedMsgId = 1;
System.out.println("Waiting for messages...");
while(running == true)
{
   //System.out.println("calling receive...");
   Message msg = consumer.receive();
   messagesRecvd++;
   //System.out.println("returned from receive...");
   if( msg instanceof BytesMessage )
   {
      msgSize = (int) ((BytesMessage) msg).getBodyLength();
      messageId = ((BytesMessage) msg).readInt();
      producerId = ((BytesMessage) msg).readInt();
      sentTime = ((BytesMessage) msg).readLong();

      if(nextExpectedMsgId != messageId)
      {
         int numDroppedMessages = messageId - nextExpectedMsgId;
         System.out.println(String.format("Dropped %d messages (expected:
             %d | received: %d)", numDroppedMessages, nextExpectedMsgId,
             messageId));
      }

      // send response message
      ack.writeInt(messageId);
      ack.writeInt(producerId);
      ack.writeInt(clientId);
      // message type
      ack.writeInt(1);
      ack.writeLong(sentTime);
      ack.writeLong(System.currentTimeMillis());

      try
      {
         //System.out.println("sending ack");
         ackProducer.send(ack);
      } catch (JMSException e)
      {
         e.printStackTrace();
      }

      ack.clearBody();


      if((messageId % 50) ==0)
      {
         System.out.println(String.format("message id: %d - sent at: %d
             (%d bytes)", messageId, sentTime, msgSize));
      }

      nextExpectedMsgId = messageId + 1;
      }
      else
      {
         System.out.println("Unexpected message type: "+msg.getClass());
      }
```

```
      if(messageId == -1)
      {
         running = false;
         System.out.println(String.format("Received %d messages",
             messagesRecvd));
      }
      }

      consumer.close();
      System.exit(0);
}
```

APPENDIX B
LATENCY BENCHMARK SOURCE CODE

```
package benchmarks;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Hashtable;
import java.util.Properties;

import org.apache.qpid.amqp_1_0.jms.impl.*;

import javax.jms.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

class LatencyTest
{
public static String readFile(String path) throws IOException
{
   StringBuilder sb = new StringBuilder();
   try (BufferedReader br = new BufferedReader(new FileReader(path)))
```

```
    {
      String sCurrentLine;
      while ((sCurrentLine = br.readLine()) != null)
      {
        sb.append(sCurrentLine);
      }
    }

    return sb.toString();
  }

  public static class SendProcessor implements Runnable
  {
    private int producerId = 0;
    private int numMessages = 0;
    private String stringData;
    private byte[] bytesData;
    private long sendTime;
    private int msgSize;
    ConnectionFactory factory = null;
    Queue msgQueue = null;
    Queue ackQueue = null;
    Destination msgDest = null;
    Destination ackDest = null;
    boolean createQueue = false;
    BufferedWriter writer;
    boolean running = true;
    Session session = null;
    Connection connection = null;
    MessageProducer producer = null;
    MessageConsumer consumer = null;
    StringBuffer testData = new StringBuffer();
    int messageId;
```

```
int consumerId;
int msgType;
long sentTime;
long ackTimeSent;
long ackTimeRecvd;

public void setOuputFile(BufferedWriter writer)
{
   this.writer = writer;
}

public void setProducerId(int id)
{
   this.producerId = id;
}
public void setCreateQueue()
{
   createQueue = true;
}

public void setFactory(ConnectionFactory factory)
{
   this.factory = factory;
}

public void setMsgQueue(Queue queue)
{
   this.msgQueue = queue;
}

public void setAckQueue(Queue queue)
{
   this.ackQueue = queue;
```

```
}

public void setNumMessages(int numMessages)
{
   this.numMessages = numMessages;
}

public void setMsgSize(int msgSize)
{
   this.msgSize = msgSize;
}

private void receiveAck()
{
   Message msg = null;
   try
   {
      msg = consumer.receive();
     } catch (JMSException e)
     {
        e.printStackTrace();
     }

     if( msg instanceof BytesMessage )
     {
        messageId = 0;
        producerId = 0;
        consumerId = 0;
        msgType = 0;
        sentTime = 0;
        ackTimeSent = 0;
        ackTimeRecvd = 0;
        try
```

```
{
    messageId = ((BytesMessage) msg).readInt();
    producerId = ((BytesMessage)msg).readInt();
    consumerId = ((BytesMessage) msg).readInt();
    msgType = ((BytesMessage) msg).readInt();
    sentTime = ((BytesMessage) msg).readLong();
    ackTimeSent = ((BytesMessage) msg).readLong();
    ackTimeRecvd = System.currentTimeMillis();
} catch (JMSException e)
{
  e.printStackTrace();
}
try
   {
    //System.out.println("writing some stuff");
    writer.write(messageId + ",");
    writer.write(producerId + ",");
    writer.write(consumerId + ",");
    writer.write(msgType + ",");

    if(msgType == 1)
    {
       writer.write(sentTime + ",");
       writer.write(ackTimeSent + ",");
       writer.write(ackTimeRecvd + ",");
    }
    else
    {
       writer.write(0 + ",");
       writer.write(ackTimeSent + ",");
       writer.write(ackTimeRecvd + ",");
    }
    writer.newLine();
```

```
                }
                catch (IOException e)
                {
                   e.printStackTrace();
                }
                if(messageId == -1)
                {
                   running = false;
                }
             }
             else
             {
                 //System.out.println("Unexpected message type:
                     "+msg.getClass());
             }
          }

@Override
public void run()
{
   for (int i=0; i < this.msgSize; i++)
   {
      testData.append("x");
   }

   stringData = testData.toString(); // If you wanted to go char by
       char
   bytesData = stringData.getBytes();

   try
   {
      connection = factory.createConnection();
      connection.start();
```

```
 session = connection.createSession(false,
    Session.AUTO_ACKNOWLEDGE);
 if(createQueue == false)
 {
   producer = session.createProducer(msgQueue);
   consumer = session.createConsumer(ackQueue);
 }
else
{
   msgDest = new QueueImpl("queue://msgs");
   ackDest = new QueueImpl("queue://acks");
   producer = session.createProducer(msgDest);
   consumer = session.createConsumer(ackDest);
}

producer.setPriority(9);
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);

} catch (JMSException e)
{
   e.printStackTrace();
}
BytesMessage msg = null;
try
{
   msg = session.createBytesMessage();
} catch (JMSException e)
{
   e.printStackTrace();
}
for( int i=1; i <= numMessages; i ++)
{
   // Put data into the output message
```

```
    try
    {
       msg.writeInt(i);
       msg.writeInt(producerId);
       msg.writeLong(System.currentTimeMillis());
       msg.writeBytes(bytesData);

    } catch (JMSException e)
    {
       e.printStackTrace();
    }

//System.out.println(String.format("msg id: %d -> send time: %d",
    i, sendTime));
// Send the message
try
{
   producer.send(msg);
} catch (JMSException e)
{
   e.printStackTrace();
}

if((i % 100) == 0)
{
   System.out.println(String.format("Sent %d messages", i));
}

// Receive ack
receiveAck();

try
{
```

```
            msg.clearBody();
         } catch (JMSException e)
         {
            e.printStackTrace();
         }
      }

      //send shutdown message
      try
      {
         msg.writeInt(-1);
         msg.writeInt(producerId);
         msg.writeLong(System.currentTimeMillis());
         producer.send(msg);
      } catch (JMSException e1)
      {
         e1.printStackTrace();
      }

       /*
       // close the connection
       try
       {
          connection.close();
       }
       catch (JMSException e)
       {
          e.printStackTrace();
       }
     */
   }
}
```

```
public static void main(String []args) throws Exception
{
   SendProcessor sendProcessor = new SendProcessor();
   Thread sendThread = null;
   String brokerType = "unknown";
   int numMessages = 1000;
   int msgSize = 1000;
   long startTime = 0;
   long endTime = 0;
   String propertyFileName = null;

   InitialContext context = null;
   // Read command line args
   if (args.length != 1)
   {
        System.out.println("Usage: LatencyTest propertiesFileName");
        System.exit(-1);
   }
   else
   {
        propertyFileName = args[0];
   }

   Hashtable<String, String> env = new Hashtable<String, String>();
   env.put(Context.INITIAL_CONTEXT_FACTORY,
       "org.apache.qpid.amqp_1_0.jms.jndi.PropertiesFileInitialContextFactory");
   env.put(Context.PROVIDER_URL, propertyFileName);
   try
   {
       context = new InitialContext(env);
   }
   catch (NamingException e1)
   {
     e1.printStackTrace();
}

Properties properties = new Properties();
try
{
       properties.load(new FileInputStream(propertyFileName));
}
catch (IOException e)
{
     e.printStackTrace();
}

ConnectionFactory connectionFactory = (ConnectionFactory)
    context.lookup("brokerURI");
//connection = connectionFactory.createConnection();
Queue msgQueue = (Queue) context.lookup("MSGS");
Queue ackQueue = (Queue) context.lookup("ACKS");

brokerType = properties.getProperty("brokerType");
numMessages = Integer.parseInt(properties.getProperty("numMessages"));
msgSize = Integer.parseInt(properties.getProperty("msgSize"));

// setup producer
sendProcessor.setFactory(connectionFactory);
sendProcessor.setMsgQueue(msgQueue);
sendProcessor.setAckQueue(ackQueue);
sendProcessor.setNumMessages(numMessages);
sendProcessor.setMsgSize(msgSize);
sendProcessor.setProducerId(1);

if(brokerType.equals("APOLLO"))
{
     sendProcessor.setCreateQueue();
 }

 // Create output file
 DateFormat df = new SimpleDateFormat("yyyy-MM-dd_HH.mm.ss");
 File csvFile = new File(brokerType + "_latency_" + numMessages + "_" +
    msgSize + "_" + df.format(new Date()) +".csv");
 FileWriter csvOutput = new FileWriter(csvFile);
 BufferedWriter writer = new BufferedWriter( csvOutput );
 writer.write("Message ID,");
 writer.write("Producer ID,");
 writer.write("Consumer ID,");
 writer.write("Result,");
 writer.write("Sent time,");
 writer.write("Ack Sent Time,");
 writer.write("Ack Recv Time,");
 writer.newLine();

// set output file in ackProcessor
sendProcessor.setOuputFile(writer);
// Create processing threads
sendThread = new Thread(sendProcessor);

// start the message producer
startTime = System.currentTimeMillis();
sendThread.start();

// wait for threads to finish
sendThread.join();
endTime = System.currentTimeMillis();

 System.out.printf("Start time: %d\n", startTime);
 System.out.printf("End time: %d\n", endTime);
```

```
    float timeDeltaSec = ((float)(endTime - startTime)/1000);

    System.out.printf("runtime: %f sec\n", timeDeltaSec);
    writer.newLine();
    writer.write("runtime," + timeDeltaSec + ",sec");
    writer.newLine();

    // close output files
    writer.flush();
    writer.close();
    csvOutput.close();
    System.exit(0);
  }
}
```

APPENDIX C
BANDWIDTH BENCHMARK SOURCE CODE

```
package benchmarks;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Hashtable;
import java.util.Properties;

import org.apache.qpid.amqp_1_0.jms.impl.*;

import javax.jms.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

class BandwidthTest
{
public static String readFile(String path) throws IOException
{
  StringBuilder sb = new StringBuilder();
  try (BufferedReader br = new BufferedReader(new FileReader(path)))
  {
    String sCurrentLine;
    while ((sCurrentLine = br.readLine()) != null)
    {
       sb.append(sCurrentLine);
    }
  }
  return sb.toString();
}

public static class SendProcessor implements Runnable
{
    private int producerId = 0;
    private int numMessages = 0;
    private String stringData;
    private byte[] bytesData;
    private long sendTime;
    private int msgSize;
    ConnectionFactory factory = null;
    Queue msgQueue = null;
    Queue ackQueue = null;
    Destination msgDest = null;
    Destination ackDest = null;
    boolean createQueue = false;
    BufferedWriter writer;
    boolean running = true;
    Session session = null;
    Connection connection = null;
    MessageProducer producer = null;
    MessageConsumer consumer = null;
    StringBuffer testData = new StringBuffer();
    int messageId;
    int consumerId;
    int msgType;
    long sentTime;
    long ackTimeSent;
    long ackTimeRecvd;

    public void setOuputFile(BufferedWriter writer)
    {
        this.writer = writer;
    }

    public void setProducerId(int id)
    {
       this.producerId = id;
    }

    public void setCreateQueue()
    {
       createQueue = true;
    }

    public void setFactory(ConnectionFactory factory)
    {
       this.factory = factory;
    }

    public void setMsgQueue(Queue queue)
    {
       this.msgQueue = queue;
    }

    public void setAckQueue(Queue queue)
    {
        this.ackQueue = queue;
       }

       public void setNumMessages(int numMessages)
       {
          this.numMessages = numMessages;
       }

      public void setMsgSize(int msgSize)
      {
         this.msgSize = msgSize;
      }

    @Override
    public void run()
    {
         for (int i=0; i < this.msgSize; i++)
         {
            testData.append("x");
         }

      stringData = testData.toString(); // If you wanted to go char by char
      bytesData = stringData.getBytes();

      try
      {
         connection = factory.createConnection();
         connection.start();
         session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

         if(createQueue == false)
         {
            producer = session.createProducer(msgQueue);
         }
      else
      {
        msgDest = new QueueImpl("queue://msgs");
        producer = session.createProducer(msgDest);
      }
      producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);

    } catch (JMSException e)
    {
      e.printStackTrace();
    }

    BytesMessage msg = null;
    try
    {
      msg = session.createBytesMessage();
    } catch (JMSException e)
    {
      e.printStackTrace();
    }

    for( int i=1; i <= numMessages; i ++)
    {
      // Put data into the output message
      try
      {
        msg.writeInt(i);
        msg.writeInt(producerId);
        msg.writeLong(System.currentTimeMillis());
        msg.writeBytes(bytesData);
      } catch (JMSException e)
      {
        e.printStackTrace();
        }

        //System.out.println(String.format("msg id: %d -> send time: %d",
            i, sendTime));
        // Send the message
        try
        {
          producer.send(msg);
        } catch (JMSException e)
        {
         e.printStackTrace();
        }

        if((i % 100) == 0)
        {
          System.out.println(String.format("Sent %d messages", i));
        }
        try
        {
          msg.clearBody();
        }
        catch (JMSException e)
        {
          e.printStackTrace();
        }
      }

      //send shutdown message
      try
      {
        msg.writeInt(-1);
        msg.writeInt(producerId);
        msg.writeLong(System.currentTimeMillis());
```

```
      producer.send(msg);
    }
    catch (JMSException e1)
    {
      e1.printStackTrace();
    }

    /*
    // close the connection
    try
    {
      connection.close();
    } catch (JMSException e)
    {
      e.printStackTrace();
    }
    */
  }
  }

  public static class AckProcessor implements Runnable
  {
    BufferedWriter writer;
    boolean running = true;
    MessageConsumer consumer = null;
    ConnectionFactory factory = null;
    Queue queue = null;
    Destination dest = null;
    boolean createQueue = false;

    public void setCreateQueue()
    {
      createQueue = true;
```

```
}

public void setFactory(ConnectionFactory factory)
{
   this.factory = factory;
}

public void setQueue(Queue queue)
{
   this.queue = queue;
}

public void setOuputFile(BufferedWriter writer)
{
     this.writer = writer;
}

public void terminate()
{
   running = false;
   try
   {
      consumer.close();
   }
   catch (JMSException e)
   {
      e.printStackTrace();
   }
 }

@Override
public void run()
{
```

```
Session session = null;
Connection connection = null;
try
{
   connection = factory.createConnection();
   connection.start();
   session = connection.createSession(false,
       Session.AUTO_ACKNOWLEDGE);

   if(createQueue == false)
   {
     consumer = session.createConsumer(queue);
   }
  else
  {
     dest = new QueueImpl("queue://acks");
     consumer = session.createConsumer(dest);
  }
} catch (JMSException e)
{
   e.printStackTrace();
}

System.out.println("Waiting for messages...");
int messageId;
int consumerId;
int producerId;
int msgType;
long sentTime;
long ackTimeSent;
long ackTimeRecvd;
while(running == true)
{
 Message msg = null;
 try
{
   msg = consumer.receive();
}
catch (JMSException e)
{
   e.printStackTrace();
}
if( msg instanceof BytesMessage )
{
  messageId = 0;
  producerId = 0;
  consumerId = 0;
  msgType = 0;
  sentTime = 0;
  ackTimeSent = 0;
  ackTimeRecvd = 0;
  try
  {
     messageId = ((BytesMessage) msg).readInt();
     producerId = ((BytesMessage)msg).readInt();
     consumerId = ((BytesMessage) msg).readInt();
     msgType = ((BytesMessage) msg).readInt();
     sentTime = ((BytesMessage) msg).readLong();
     ackTimeSent = ((BytesMessage) msg).readLong();
     ackTimeRecvd = System.currentTimeMillis();
  } catch (JMSException e)
  {
    e.printStackTrace();
  }

  try
  {
    writer.write(messageId + ",");
    writer.write(producerId + ",");
    writer.write(consumerId + ",");
    writer.write(msgType + ",");

    if(msgType == 1)
    {
       writer.write(sentTime + ",");
       writer.write(ackTimeSent + ",");
       writer.write(ackTimeRecvd + ",");
    }
    else
    {
       writer.write(0 + ",");
       writer.write(ackTimeSent + ",");
       writer.write(ackTimeRecvd + ",");
    }

    writer.newLine();
  }
 catch (IOException e)
 {
    e.printStackTrace();
 }

if(messageId == -1)
{
   running = false;
}

}
else
```

```
      {
         //System.out.println("Unexpected message type:
            "+msg.getClass());
      }

    }
  }
}

public static void main(String []args) throws Exception
{
  AckProcessor ackProcessor = new AckProcessor();
  Thread ackThread = null;
  SendProcessor sendProcessor = new SendProcessor();
  Thread sendThread = null;
  String brokerType = "unknown";
  int numMessages = 1000;
  int msgSize = 1000;
  long startTime = 0;
  long endTime = 0;
  String propertyFileName = null;
  InitialContext context = null;

  // Read command line args
  if (args.length != 1)
  {
    System.out.println("Usage: Producer propertiesFileName");
    System.exit(-1);
  }
  else
  {
    propertyFileName = args[0];
  }
```

```
Hashtable<String, String> env = new Hashtable<String, String>();
env.put(Context.INITIAL_CONTEXT_FACTORY,
    "org.apache.qpid.amqp_1_0.jms.jndi.PropertiesFileInitialContextFactory");
env.put(Context.PROVIDER_URL, propertyFileName);
try
{
     context = new InitialContext(env);
} catch (NamingException e1)
{
      e1.printStackTrace();
}

Properties properties = new Properties();
try
{
   properties.load(new FileInputStream(propertyFileName));
} catch (IOException e)
{
     e.printStackTrace();
}

ConnectionFactory connectionFactory = (ConnectionFactory)
    context.lookup("brokerURI");
//connection = connectionFactory.createConnection();
Queue msgQueue = (Queue) context.lookup("MSGS");
Queue ackQueue = (Queue) context.lookup("ACKS");

brokerType = properties.getProperty("brokerType");
numMessages = Integer.parseInt(properties.getProperty("numMessages"));
msgSize = Integer.parseInt(properties.getProperty("msgSize"));

// setup producer
```

```
sendProcessor.setFactory(connectionFactory);
sendProcessor.setMsgQueue(msgQueue);
sendProcessor.setAckQueue(ackQueue);
sendProcessor.setNumMessages(numMessages);
sendProcessor.setMsgSize(msgSize);
sendProcessor.setProducerId(1);

if(brokerType.equals("APOLLO"))
{
   ackProcessor.setCreateQueue();
   sendProcessor.setCreateQueue();
}

ackProcessor.setFactory(connectionFactory);
ackProcessor.setQueue(ackQueue);

// Create output file
DateFormat df = new SimpleDateFormat("yyyy-MM-dd_HH.mm.ss");
File csvFile = new File(brokerType + "_bandwidth_" + numMessages + "_"
    + msgSize + "_" + df.format(new Date()) +".csv");
FileWriter csvOutput = new FileWriter(csvFile);
BufferedWriter writer = new BufferedWriter( csvOutput );
writer.write("Message ID,");
writer.write("Producer ID,");
writer.write("Consumer ID,");
writer.write("Result,");
writer.write("Sent time,");
writer.write("Ack Sent Time,");
writer.write("Ack Recv Time,");
writer.newLine();

// set output file in ackProcessor
ackProcessor.setOuputFile(writer);
```

```
 // Create processing threads
 ackThread = new Thread(ackProcessor);
 sendThread = new Thread(sendProcessor);

 ackThread.start();

Thread.sleep(2000);

// start the message producer
startTime = System.currentTimeMillis();
sendThread.start();

// wait for threads to finish
sendThread.join();
ackThread.join();
endTime = System.currentTimeMillis();

System.out.printf("Start time: %d\n", startTime);
System.out.printf("End time: %d\n", endTime);
float timeDeltaSec = ((float)(endTime - startTime)/1000);
float throughput = numMessages / timeDeltaSec;
float bandwidth = ((float)(numMessages * msgSize) /
   timeDeltaSec/1000000);

System.out.printf("runtime: %f sec\n", timeDeltaSec);
System.out.printf("throughput: %f messages/sec\n", throughput);
System.out.printf("bandwidth: %f MB/sec\n", bandwidth);

writer.newLine();
writer.write("runtime," + timeDeltaSec + ",sec");
writer.newLine();
writer.write("throughput," + throughput + ",msgs/sec");
```

```
  writer.newLine();
  writer.write("bandwidth," + bandwidth + ",MB/sec");
  writer.newLine();

  // close output files
  writer.flush();
  writer.close();
  csvOutput.close();
  System.exit(0);
}

}
```

APPENDIX D
CONSUMER CONFIGURATION

```
#
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements. See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership. The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License. You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied. See the License for the
# specific language governing permissions and limitations
# under the License.
#
java.naming.factory.initial =
   org.apache.qpid.amqp_1_0.jms.jndi.PropertiesFileInitialContextFactory

# register some connection factories
# connectionfactory.[jndiname] = [ConnectionURL]
connectionfactory.brokerURI = amqp://admin:admin@10.9.1.30:10001/


# Register an AMQP destination in JNDI
# destination.[jniName] = [Address Format]
queue.MSGS = msgs
queue.ACKS = acks


clientId = 1
brokerType = QPID
```

APPENDIX E
PRODUCER CONFIGURATION

```
#
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements. See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership. The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License. You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied. See the License for the
# specific language governing permissions and limitations
# under the License.
#
java.naming.factory.initial =
   org.apache.qpid.amqp_1_0.jms.jndi.PropertiesFileInitialContextFactory

# register some connection factories
# connectionfactory.[jndiname] = [ConnectionURL]
connectionfactory.brokerURI = amqp://admin:admin@10.9.1.30:10001/

# Register an AMQP destination in JNDI
# destination.[jniName] = [Address Format]
queue.MSGS = msgs
queue.ACKS = acks
```

```
producerId = 1
numMessages = 3000
msgSize = 50000
brokerType = QPID-QPID
```

APPENDIX F
ACTIVEMQ BROKER CONFIGURATION - SINGLE INSTANCE

```
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
  http://activemq.apache.org/schema/core
      http://activemq.apache.org/schema/core/activemq-core.xsd">

    <bean id="logQuery"
        class="org.fusesource.insight.log.log4j.Log4jLogQuery"
          lazy-init="false" scope="singleton"
          init-method="start" destroy-method="stop">
    </bean>

    <broker xmlns="http://activemq.apache.org/schema/core"
        brokerName="localhost" dataDirectory="${activemq.data}"
        persistent="false">

        <destinationPolicy>
            <policyMap>
              <policyEntries>
                <policyEntry queue=">" producerFlowControl="true"
                    memoryLimit="10 mb">
                </policyEntry>
              </policyEntries>
            </policyMap>
        </destinationPolicy>
        <managementContext>
            <managementContext createConnector="false"/>
        </managementContext>

        <systemUsage>
            <systemUsage>
                <memoryUsage>
                    <memoryUsage percentOfJvmHeap="70" />
                </memoryUsage>
                <storeUsage>
                    <storeUsage limit="1000 mb"/>
                </storeUsage>
                <tempUsage>
                    <tempUsage limit="1000 mb"/>
                </tempUsage>
            </systemUsage>
        </systemUsage>

        <transportConnectors>
            <transportConnector name="amqp"
                uri="amqp://10.9.1.30:5672?maximumConnections=1000&amp;wireFormat.maxFram
        </transportConnectors>

        <shutdownHooks>
            <bean xmlns="http://www.springframework.org/schema/beans"
                class="org.apache.activemq.hooks.SpringContextHook" />
        </shutdownHooks>

    </broker>
</beans>
```

APPENDIX G
ACTIVEMQ BROKER CONFIGURATION - FEDERATED

```
<beans
 xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
 http://activemq.apache.org/schema/core
    http://activemq.apache.org/schema/core/activemq-core.xsd">



  <bean id="logQuery"
      class="org.fusesource.insight.log.log4j.Log4jLogQuery"
        lazy-init="false" scope="singleton"
        init-method="start" destroy-method="stop">
  </bean>

  <broker xmlns="http://activemq.apache.org/schema/core"
      brokerName="localhost" dataDirectory="${activemq.data}"
      persistent="false">

      <destinationPolicy>
          <policyMap>
            <policyEntries>
              <policyEntry queue=">" producerFlowControl="true"
                  memoryLimit="10 mb">
              </policyEntry>
            </policyEntries>
          </policyMap>
```

```
</destinationPolicy>

<networkConnectors>
    <networkConnector duplex="true" staticBridge="true"
        uri="static:(tcp://10.9.1.31:61002)" >
        <staticallyIncludedDestinations>
            <queue physicalName="msgs"/>
            <queue physicalName="acks"/>
        </staticallyIncludedDestinations>
    </networkConnector>
</networkConnectors>

<managementContext>
    <managementContext createConnector="false"/>
</managementContext>

<systemUsage>
    <systemUsage>
        <memoryUsage>
            <memoryUsage percentOfJvmHeap="70" />
        </memoryUsage>
        <storeUsage>
            <storeUsage limit="1000 mb"/>
        </storeUsage>
        <tempUsage>
            <tempUsage limit="1000 mb"/>
        </tempUsage>
    </systemUsage>
</systemUsage>

<transportConnectors>
    <transportConnector name="amqp"
        uri="amqp://10.9.1.30:5672?maximumConnections=1000&amp;wireFormat.maxFram
```

```
            <transportConnector name="openwire"
                uri="tcp://10.9.1.30:61002" />
        </transportConnectors>

        <shutdownHooks>
            <bean xmlns="http://www.springframework.org/schema/beans"
                class="org.apache.activemq.hooks.SpringContextHook" />
        </shutdownHooks>
    </broker>
</beans>
```

APPENDIX H
APOLLO BROKER CONFIGURATION - FEDERATED

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<broker xmlns="http://activemq.apache.org/schema/activemq/apollo">
  <notes>
    The default configuration with tls/ssl enabled.
  </notes>
  <log_category console="console" security="security"
      connection="connection" audit="audit"/>

  <authentication domain="apollo"/>
  <access_rule allow="*" action="*"/>

  <virtual_host id="test">
    <host_name>test_broker</host_name>
    <host_name>localhost</host_name>
    <host_name>127.0.0.1</host_name>
    <host_name>10.9.1.30</host_name>
    <null_store/>
  </virtual_host>

  <web_admin bind="http://0.0.0.0:61680"/>
  <web_admin bind="https://0.0.0.0:61681"/>

  <connector id="amqp" bind="tcp://10.9.1.30:5672"/>

</broker>
```