MP1K: IBM MQ for z/OS version 9.0 Performance Report

June 2016

IBM MQ Performance
IBM UK Laboratories
Hursley Park
Winchester
Hampshire
SO21 2JN

Take Note!

Before using this report, please be sure to read the paragraphs on "disclaimers", "warranty and liability exclusion", "errors and omissions" and other general information paragraphs in the "Notices" section below.

First edition, June 2016. This edition applies to IBM MQ for z/OS version 9.0 (and to all subsequent releases and modifications until otherwise indicated in new editions).

Notices

DISCLAIMERS The performance data contained in this report were measured in a controlled environment. Results obtained in other environments may vary significantly.

You should not assume that the information contained in this report has been submitted to any formal testing by IBM.

Any use of this information and implementation of any of the techniques are the responsibility of the licensed user. Much depends on the ability of the licensed user to evaluate the data and to project the results into their own operational environment.

WARRANTY AND LIABILITY EXCLUSION

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.

In Germany and Austria, notwithstanding the above exclusions, IBM's warranty and liability are governed only by the respective terms applicable for Germany and Austria in the corresponding IBM program license agreement(s).

ERRORS AND OMISSIONS

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

INTENDED AUDIENCE

This report is intended for Architects, Systems Programmers, Analysts and Programmers wanting to understand the performance characteristics of **IBM MQ for z/OS version 9.0**. The information is not intended as the specification of any programming interfaces that are provided by IBM MQ. Full descriptions of the IBM MQ facilities are available in the product publications. It is assumed that the reader is familiar with the concepts and operation of IBM MQ.

Prior to IBM MQ for z/OS version 8.0, the product was known as WebSphere MQ and there are instances where these names may be interchanged.

LOCAL AVAILABILITY

References in this report to IBM products or programs do not imply that IBM intends to make these available in all countries in which IBM operates. Consult your local IBM representative for information on the products and services currently available in your area.

ALTERNATIVE PRODUCTS AND SERVICES

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.

USE OF INFORMATION PROVIDED BY YOU

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

TRADEMARKS and SERVICE MARKS

The following terms, used in this publication, are trademarks or registered trademarks of the IBM Corporation in the United States or other countries or both:

- IBM®
- z/OS®
- zSeries®
- zEnterprise®
- MQSeries®
- CICS®
- DB2 for z/OS®
- IMS™
- MVS™
- zEC12™
- z13™
- FICON®
- WebSphere®
- IBM MQ®

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

EXPORT REGULATIONS

You agree to comply with all applicable export and import laws and regulations.

Summary of Amendments

Date	Changes
2016	Version 1.0 - Initial Version.

Table of contents

Chapter 1

Performance Highlights

This report focuses on performance changes since previous versions, V7.1.0 and V8.0.0, and on the performance of new function in this release.

SupportPac MP16 "Capacity Planning and Tuning Guide" will continue to be the repository for ongoing advise and guidance learned as systems increase in power and experience is gained.

Chapter 2

Existing Function

General statement of regression

CPU costs and throughput are not significantly difference in version 9.0 for typical messaging workloads when compared with version 8.0.

A user can see how that statement has been determined by reviewing details of the regression test cases in the Regression appendix.

Storage usage

Virtual storage constraint relief has not been a primary focus of this release, however the introduction of 64-bit buffer pools in version 8.0 can offer some storage relief.

CSA usage

Common Service Area (CSA) storage usage is important as the amount available is restricted by the amount of 31-bit storage available and this is limited to an absolute limit of 2GB.

The CSA is allocated in all address spaces in an LPAR, so its use reduces the available private storage for all address spaces.

In real terms, the queue manager does not have 2GB of storage to use - as there is some amount used by MVS for system tasks and it is possible for individual customer sites to set the limit even lower.

From the storage remaining of the 2GB of 31-bit storage, a large (but configurable) amount of storage may be used by the queue manager for buffer pools. This storage usage may be reduced from version 8.0 onwards with the use of 64-bit buffer pools.

The storage remaining is available for actually connecting to the queue manager in a variety of ways and using IBM MQ to put and get messages.

Initial CSA usage

CSA usage for V900 is similar to both V710 and V800 when similarly configured queue managers are started. On our systems this is approximately 6MB per queue manager.

CSA usage per connection

CSA usage has seen little change in the following releases: V710, V800 and V900.

- For local connections, MCA channels and SVRCONN channels with SHARECNV(0), CSA usage is 2.45KB per connection.
- For SVRCONN channels with SHARECNV(1), CSA usage is approximately 4.9KB per connection.
- For SVRCONN channels with SHARECNV(5), CSA usage is approximately 2.9KB per connection, based on 5 clients sharing the channel instance.
- For SVRCONN channels with SHARECNV(10), CSA usage is approximately 2.7KB per connection, based on 10 clients sharing the channel instance.

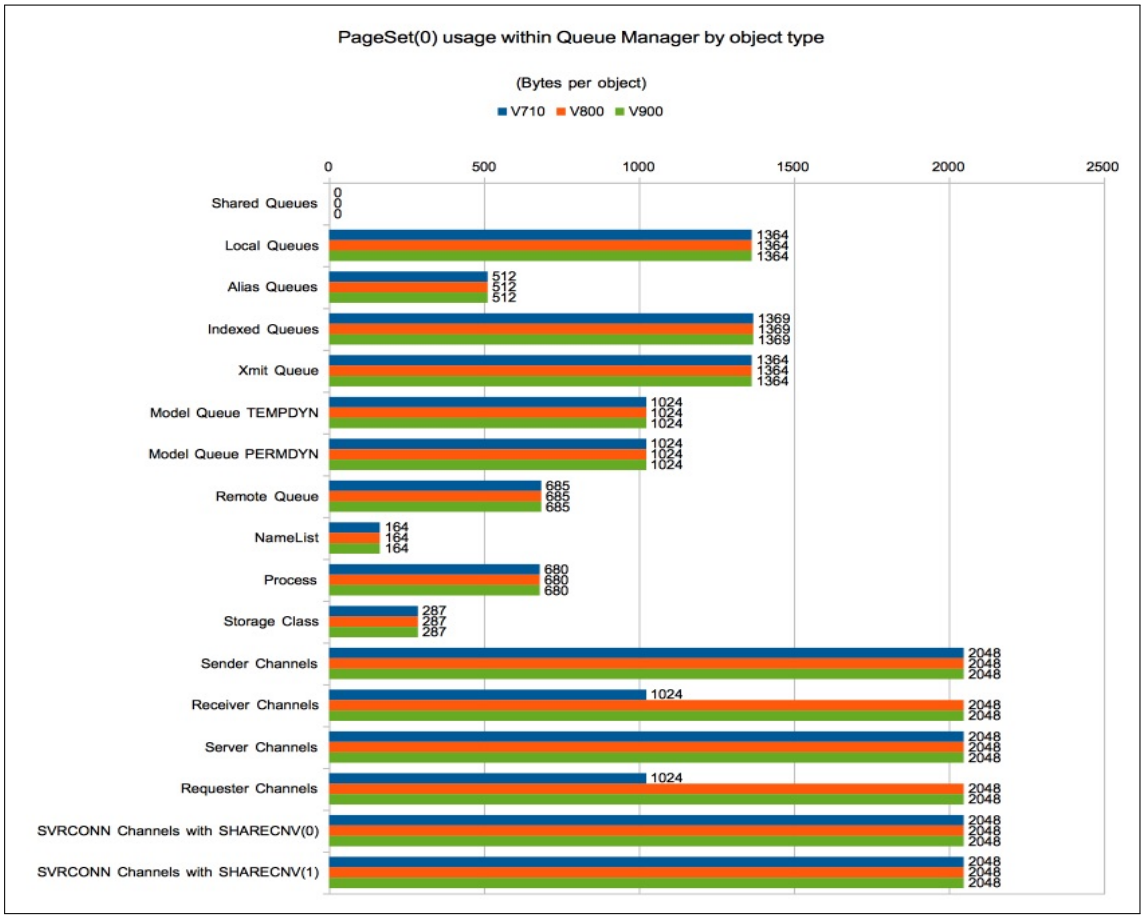
Object Sizes

When defining objects, the queue manager may store information about that object in pageset 0 and may also require storage taken from the queue manager's extended private storage allocation.

The data shown on the following 2 charts only includes the storage used when defining the objects.

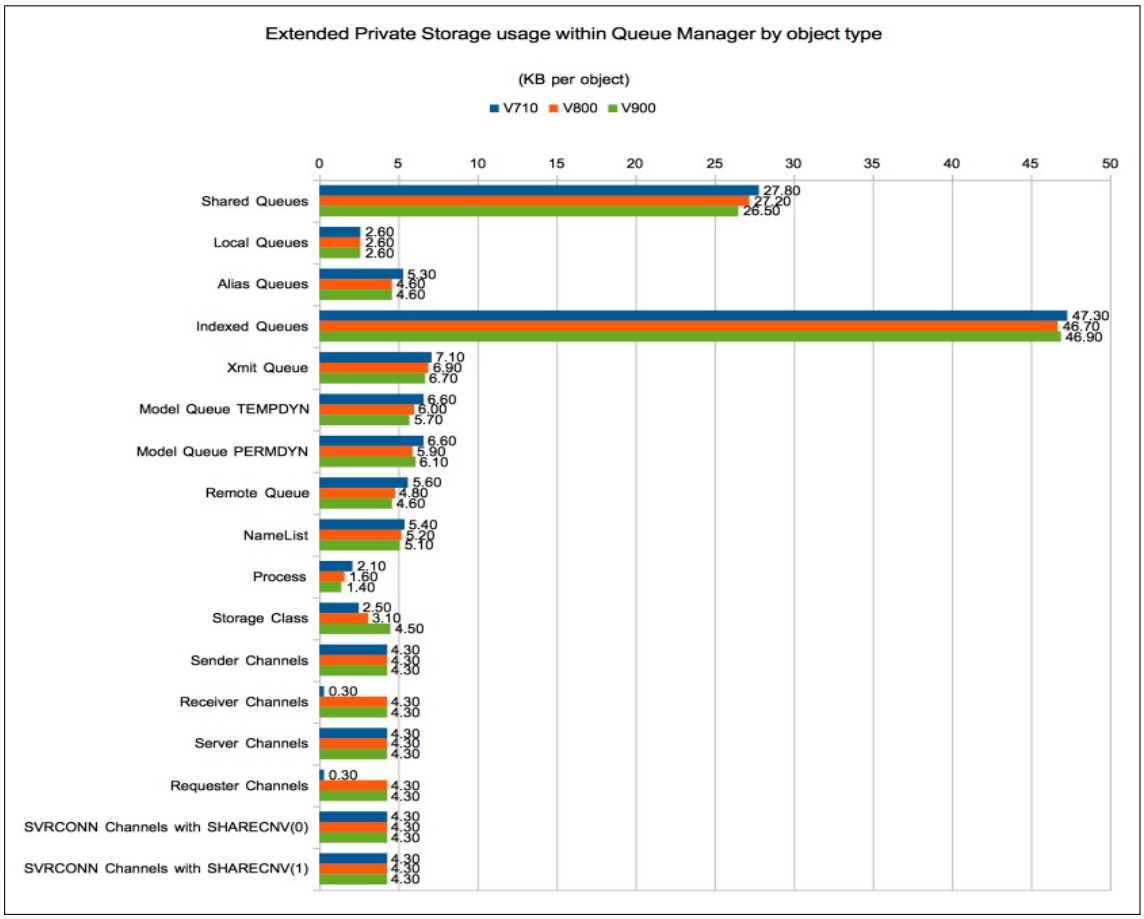
PAGESET(0) Usage

Chart: Pageset usage by object type



Virtual Storage Usage

Chart: Virtual Storage usage by object type



Capacity of the queue manager and channel initiator

How much storage does a connection use?

When an application connects to a queue manager, an amount of storage is allocated from the queue manager's available storage.

Some of this storage is held above 2GB, such as security data, and other data is storage on storage taken from that available below the 2GB bar. In the following examples, only allocations from below the 2GB bar are reported.

In V7.1.0, typical storage usage is around 22KB per connection. From V8.0.0-onwards, due to some control areas being moved above the 2GB "bar", the typical storage usage decreases to 14KB per connection, however there is additional usage in the following (non-exhaustive) cases:

- Where connection is over a SHARECNV(1) channel, the usage increases to 21KB.
- Where connection is over a SHARECNV(10) channel and has a CURSHCNV of 10, the usage is 15.5KB per connection (155KB per channel).
- Where connection is over a SHARECNV(1) channel to shared queues - either CFLEVEL(4) or CFLEVEL(5) backed by SMDS, the storage is 27.5KB, giving an additional shared queue overhead of 6.7KB.

These numbers are based upon the connecting applications accessing a small number of queues. If your application has more than 32 objects open, the amount of storage will be increased.

If the number of messages held in a unit of work is large and the size of the messages is large then additional lock storage may be required.

How many clients can I connect to my queue manager?

The maximum number of clients you can connect to a queue manager depends on a number of factors, for example:

- Storage available in the queue manager's address space.
- Storage available in the channel initiator's address space.
- How large the messages being put or got are.
- Whether the channel initiator is already running its maximum number of connected clients (either 9,999 or the value specified by channel attributes like MAXCHL).

The table below shows the typical footprint when connecting a client application to a z/OS queue manager via the channel initiator.

The value used in the SHARECNV channel attribute can affect the footprint and consideration as to the setting should be taken. Guidance on the SHARECNV attribute can be found in SupportPacs MP16 "Capacity Planning and Tuning Guide" and MP1F "WebSphere MQ for z/OS V7.0 Performance Report".

		Channel initiator footprint (KB / SVRCONN channel) Message size (KB)			
IBM MQ release	**SHARECNV**	**1**	**10**	**32**	**64**
V710	**0**	113	129	191	223
V800	**0**	114	131	191	221
V900	**0**	96	109	166	196
V710	**1**	176	199	286	352
V800	**1**	179	201	288	355
V900	**1**	179	202	291	357
V710	**10**	251	276	686	1041
V800	**10**	248	268	680	1041
V900	**10**	242	274	685	1031

Note: For the SHARECNV(10) channel measurements, the channels are running with 10 conversations per channel instance, so the cost per conversation would be the value in the table divided by 10.

Example: How many clients can I run?

When the channel initiator was started it logged the following message prior to starting channels:

"`CSQX004I Channel initiator is using 117 MB of local storage, 1355 MB are free`".

This means that the channel initiator had 1472MB of storage available and has 1355MB available for channels to be started. To avoid letting the channel initiator run short on storage, it is advisable to aim to keep the usage below 80% of the total available. In the example this means keeping the channel initiator storage usage below 1177MB, which in turn means that there is 1060 MB available for channels.

If the workload is expected to be clients connecting via SHARECNV(0) SVRCONN channels and using 32KB messages, we could predict that the channel initiator could support a maximum of 6,538 running SVRCONN channels.

How many channels can I run to or from my queue manager?

This depends on the size of the messages flowing through the channel.

A channel will hold onto a certain amount of storage for its lifetime. This footprint depends on the size of the messages.

Message Size	1KB	32KB	64KB	4MB
Footprint per channel (channel initiator)	90	99	109	1125
Overhead of message size increase on 1KB messages		+9	+18	+1034

Chapter 3

New Function for version 9.0

This release has introduced a number of items that can affect performance and these include:

- Advanced Message Security (AMS)

To extend the use of JMS on the z/OS platform, the following environments are now supported:

- Using IBM MQ classes for JMS in a CICS OSGi JVM Server
- IMS

In addition the following items have been added to aid the monitoring of the queue manager:

- Page set statistics

IBM Advanced Message Security (AMS)

There have been a number of changes to AMS in version 9.0 that affect both existing protection options and offer a new quality of protection and these are detailed in the following sections.

The existing two qualities of protection for IBM MQ Advanced Message Security, namely *Integrity* and *Privacy* have been supplemented in version 9.0 to include a third quality of *Confidentiality*.

Integrity protection is provided by digital signing, which provides assurance on who created the message, and that the message has not been altered or tampered with.

Privacy protection is provided by a combination of digital signing and encryption. Encryption ensures that message data is viewable by only the intended recipient, or recipients.

Confidentiality protection is provided by encryption only.

IBM MQ AMS uses a combination of symmetric and asymmetric cryptographic routines to provide digital signing and encryption. As symmetric key operations are very fast in comparison to asymmetric key operations, which are CPU intensive and some of the cost may be offloaded to cryptographic hardware such as Crypto Express5, this in turn can have a significant impact on the cost of protecting large numbers of message with IBM MQ AMS.

- **Asymmetric cryptographic routines**

  For example, when putting a signed message the message hash is signed using an asymmetric key operation. When getting a signed message, a further asymmetric key operation is used to verify the signed hash. Therefore, a minimum of two asymmetric key operations are required per message to sign and verify the message data. Some of this asymmetric cryptographic work can be offloaded to cryptographic hardware.

- **Asymmetric and symmetric cryptographic routines**

  When putting an encrypted message, a symmetric key is generated and then encrypted using an asymmetric key operation for each intended recipient of the message. The message data is then encrypted with the symmetric key. When getting the encrypted message the intended recipient needs to use an asymmetric key operation to discover the symmetric key in use for the message. The symmetric key work cannot be offloaded to cryptographic hardware but will be performed in part by CPACF processors.

All three qualities of protection, therefore, contain varying elements of CPU intensive asymmetric key operations, which will significantly impact the maximum achievable messaging rate for applications putting and getting messages.

Confidentiality policies do, however, allow for symmetric key reuse over a sequence of messages.

Key reuse can significantly reduce the costs involved in encrypting a number of messages intended for the same recipient or recipients.

For example, when putting 10 encrypted messages to the same set of recipients, a symmetric key is generated. This key is encrypted for the first message using an asymmetric key operation for each of the intended recipients of the message.

Based upon policy controlled limits, the encrypted symmetric key can then be reused by subsequent messages that are intended for the same recipient(s). An application that is getting encrypted messages can apply the same optimization, in that the application can detect when a symmetric key has not changed and avoid the expense of retrieving the symmetric key.

In this example 90% of the asymmetric key operations can be avoided by both the putting and getting applications reusing the same key.

AMS general performance

MP1J "IBM MQ for z/OS version 8.0.0 Performance Report" offers information on the performance of IBM MQ and AMS version 8.0 and this document will highlight differences in performance beween versions 8.0 and 9.0.

- Reduced impact of using unprotected queues when AMS enabled.
- Relative performance of protected queues
- Is the impact of cryptographic offload the same in V9.0?

Reduced impact of using unprotected queues when AMS enabled

MP1J "IBM MQ for z/OS version 8.0.0 Performance Report" reported that prior to IBM MQ for z/OS version 8.0 the cost of interacting with an unprotected queue when AMS was enabled was of the order of 25 microseconds per MQ API. These costs reduced in version 8.0 to between 2.5 to 10 microseconds per MQ API.

In IBM MQ for z/OS version 9.0, the costs have reduced further to between 0.5 to 1 microseconds per MQ API.

Relative performance of protected queues

MP1J "IBM MQ for z/OS version 8.0.0 Performance Report" reported that the overhead of running low-business logic workloads with AMS protected-queues using *privacy*-level protection against V8.0 queue managers was quite significant. In the worst-case scenario where the applications perform only messaging workload and contain little business logic, file I/O etc the impact of AMS has decreased e.g.

Batch workload

Message size	MQ V8.0	MQ V9.0
2KB	+1.1 CPU ms	+0.7 CPU ms
64KB	+2.7 CPU ms	+2.3 CPU ms
4MB	+106 CPU ms	+102 CPU ms

Notes on table:

- The value in the cell e.g. "+1.1 CPU ms" equates to the actual increase in the cost per transaction in CPU milliseconds as a result of adding AMS privacy-level protection to the queues for the indicated release.

CICS workload

For a simple short-lived CICS COBOL transaction, such as described in the regression tests for CICS workloads, the impact of AMS is much less, primarily due to the increased cost of the transaction, rather than any decrease in AMS processing costs, i.e the proportion of the application out of the total transaction cost is larger in the CICS environment.

For example IBM MQ for z/OS version 8.0 as reported in MP1J "IBM MQ for z/OS version 8.0.0 Performance Report" showed a 7 times increase in transaction cost when adding AMS privacy-level protection to the queues. Similar performance is seen in IBM MQ for z/OS version 9.0 although some reduction may be seen in the CICS transaction cost.

When the CICS COBOL transaction is replaced with a CICS JMS transaction, as described in "Using IBM MQ classes for JMS in a CICS OSGi JVM Server", the overhead of AMS privacy-level protection is 2.5 times that of the same workload running without AMS protection.

Performance of AMS Integrity-level protection

Integrity-level protection is the process of digitally signing the message, which provides assurance on who created the message, and that the message has not been altered or tampered with. It does not provide protection against viewing the data by unintended parties.

Some of the cost of digital signing can be offloaded using cryptographic hardware, such as Crypto-Express5S. Section "What if cryptographic offload is not available?" provides guidance to the impact when the cryptographic hardware offload facility is not available.

The performance data provided in this section relies on cryptographic hardware to assist in digitally signing the message as detailed in Appendix B.

Does the type of digital signature algorithm chosen affect cost?

When a policy dictates that a message is signed, there is a significant increase in the cost of the message. In the case of a long running request/reply workload running batch applications with minimal non-MQ related processing, the transaction cost increased approximately 15 times.

Varying the signing type between MD5, SHA1, SHA256 and SHA512 saw the largest increase with MD5, particularly with larger messages, as demonstrated in the following charts.

The messages are not encrypted in the workloads illustrated by the 3 charts below.

Chart: Transaction cost of 2KB message whilst varying digital signature algorithm

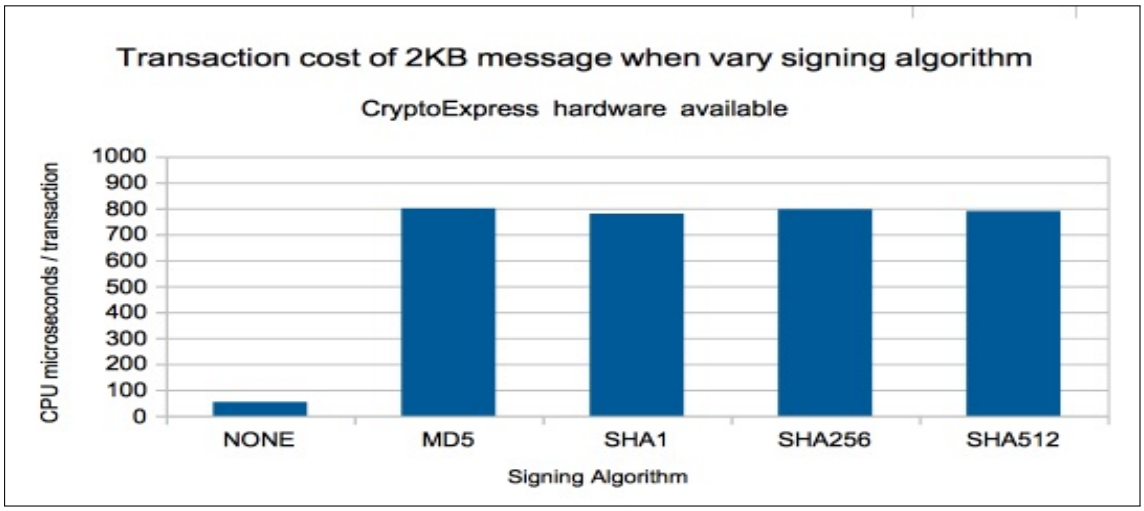


Chart: Transaction cost of 64KB message whilst varying digital signature algorithm

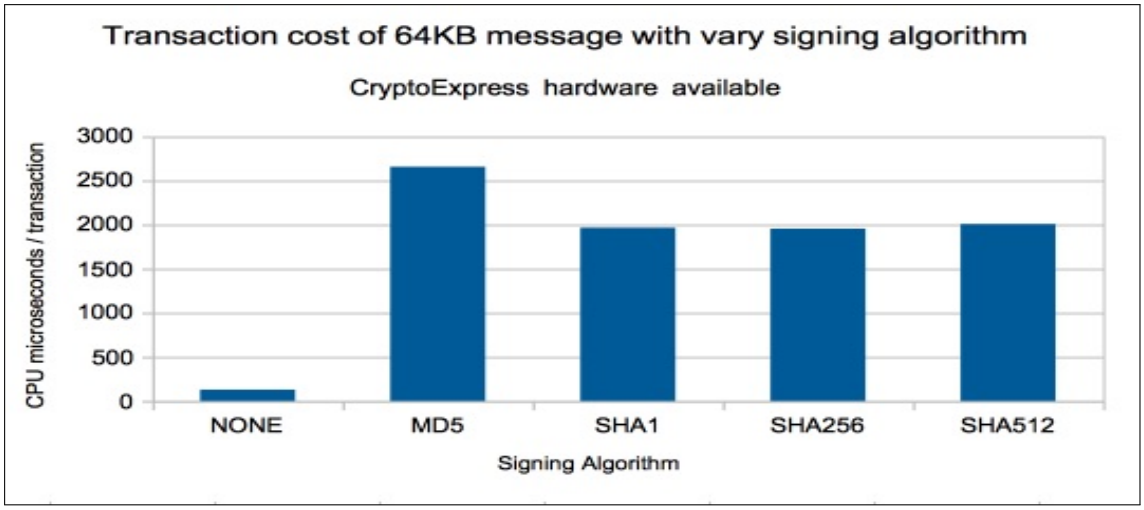
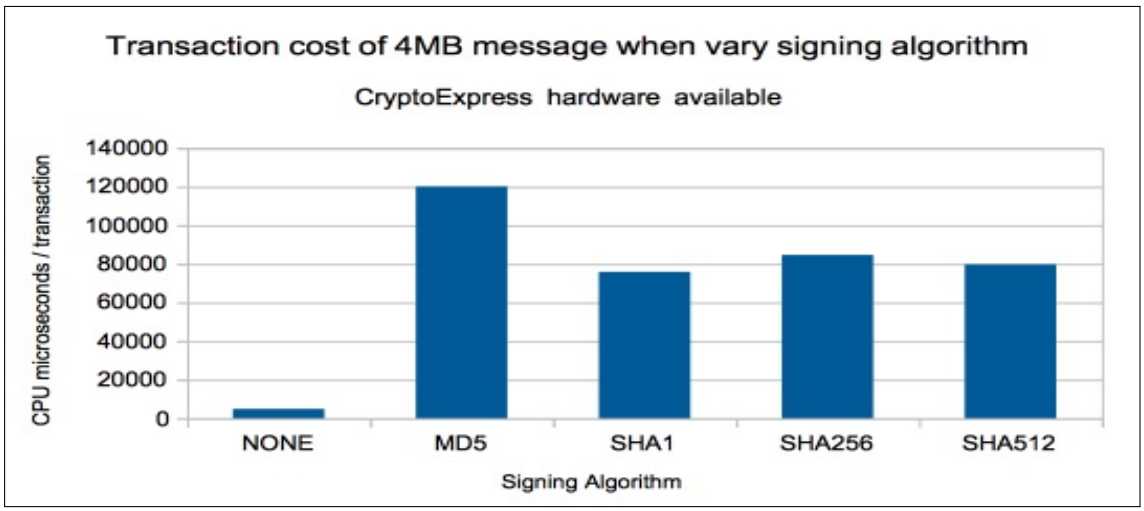


Chart: Transaction cost of 4MB message whilst varying digital signature algorithm



Notes on charts:

- Signing algorithm MD5 is sensitive to message size. The impact of MD5 signing on 2KB messages was 15 times, but as the message size increased, the impact grew to 25 times that of an unsigned message.
- Signing messages can make high usage of the CryptoExpress coprocessor and accelerators, particularly with high messaging rates such as those achieved with smaller messages. In our measurements using 2KB messages, the 2 available coprocessors chips usage was of the order of 70% utilised each, with the single available accelerator being 14% busy.

Performance of AMS Privacy-level protection

Privacy-level protection is provided by a combination of digital signing and encryption. Encryption means that the message data is viewable only by the intended recipient, or recipients.

Prior to IBM MQ for z/OS version 9.0, if a message needed to be encrypted, it also had to be signed. There is an additional cost incurred when encrypting the message in addition to those incurred from signing the message and the impact of this is similar regardless of encryption type, except for RC2.

Impact of encryption type when using SHA1 signing

Encryption Type	Message Size 2KB	Message Size 64KB	Message size 4MB
DES 3DES AES128 AES256	1.15 times	1.4 times	1.4 times
RC2	1.5 times	4.4 times	6 times

Notes on table:

- A 64KB message that is signed and encrypted using AES128 uses typically 1.4 times more CPU than a 64KB message that is just signed.
- A 64KB message that is signed and encrypted using RC2 uses typically 4.4 times more CPU than a 64KB message that is just signed.

Impact of encryption type when using MD5 signing

In the preceding table, the signing algorithm used was SHA1, which we have previously seen to have a growing influence as message size increases. For completeness sake, the following table reports the impact of the encryption type when using signing algorithm MD5.

Encryption Type	Message Size 2KB	Message Size 64KB	Message size 4MB
DES 3DES AES128 AES256	1.2 times	1.3 times	1.4 times
RC2	1.5 times	3.6 times	4.6 times

What if cryptographic offload is not available?

For messages protected using *Integrity*-level protection the impact of cryptographic hardware, such as CryptoExpress, being unavailable saw the cost of an MQPUT and MQGET increase by 3 CPU milliseconds regardless of message size or signing algorithm.

For messages protected using *Privacy*-level protection, the impact of cryptographic hardware being unavailable saw the cost of the encrypt and decrypt is increased by a further 3 CPU milliseconds for per message encrypted and decrypted on z/OS for messages up to 4MB.

From analysis of the RMF cryptographic reports from the messaging workloads when both cryptographic coprocessor and accelerators were available, the following observations have been made:

- The cryptographic coprocessors were used for signing the message.
- The cryptographic accelerator was used for verifying the digital signature of the message.
- The cost of signing the message using coprocessors was approximately 5 times that of verification of digital signature of the message using accelerator.
- The CPACF processors handle the encryption and decryption of the data.

AMS Confidentiality

AMS *Confidential* is a new level of protection introduced in IBM MQ for z/OS version 9.0, which allows the user to specify how many messages will use the same symmetric key. This is specified by setting a value for the `-c` parameter when defining a policy using the `setmqspl` program. Note that setting "`-c *`" means unlimited reuse of the key.

How many times should I reuse the key?

When using queues protected by policies with AMS *Confidential* attributes, it is possible to set values from 0 to reusing forever and the value chosen should reflect a sensible value for the security of your data. For example, it may not be advisable when transferring many customer details to use the same key for all customers, as if the encryption is broken for one message, it will be broken for all.

From a performance perspective the impact of key reuse can depend largely on the message size as can be seen in the charts on the following pages but in summary, small messages benefit more from higher key reuse values where appropriate.

The workloads used in the charts is a local request/reply workload running batch applications where there is a single requester and single server task using a request and reply queue.

In each case, the signing algorithm specified is SHA1 which we have previously demonstrated has similar performance characteristics to SHA256 and SHA512, and better performance than MD5.

With regards to encryption, we are categorising as "AES" which includes AES128 and AES256 and "DES" which includes DES and Triple-DES. Specifically AES measurements are AES256 and DES are Triple-DES.

Chart: AES-encryption - 2KB message workload with varying key reuse values

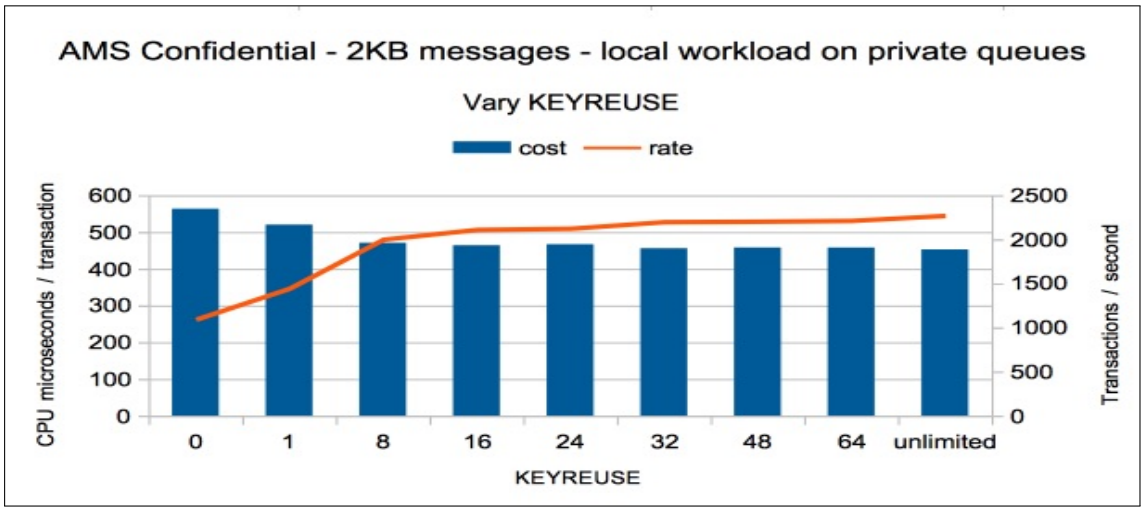


Chart: AES-encryption - 64KB message workload with varying key reuse values

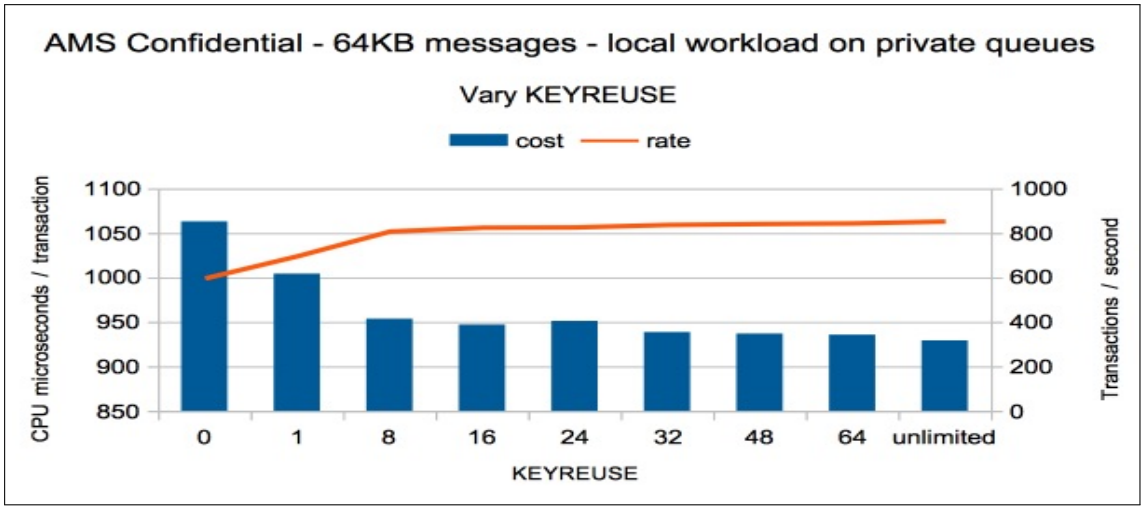
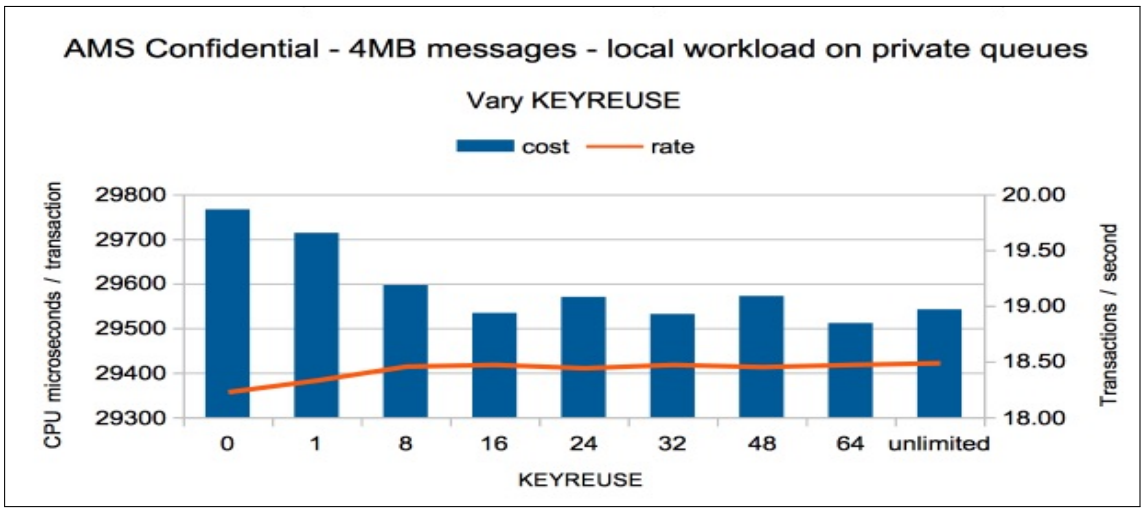


Chart: AES-encryption - 4MB message workload with varying key reuse values



Notes on AES-encryption charts

- For small messages, reusing the key 32 times or more gives the best performance
- As the message size increases, the benefits of higher key reuse values decrease.
- Large messages, such as the 4MB example, show little difference in performance regardless of the key reuse value, i.e. the difference between key reuse of 1 and unlimited is less than a 1% decrease in transaction cost.

Chart: DES-encryption - 2KB message workload with varying key reuse values

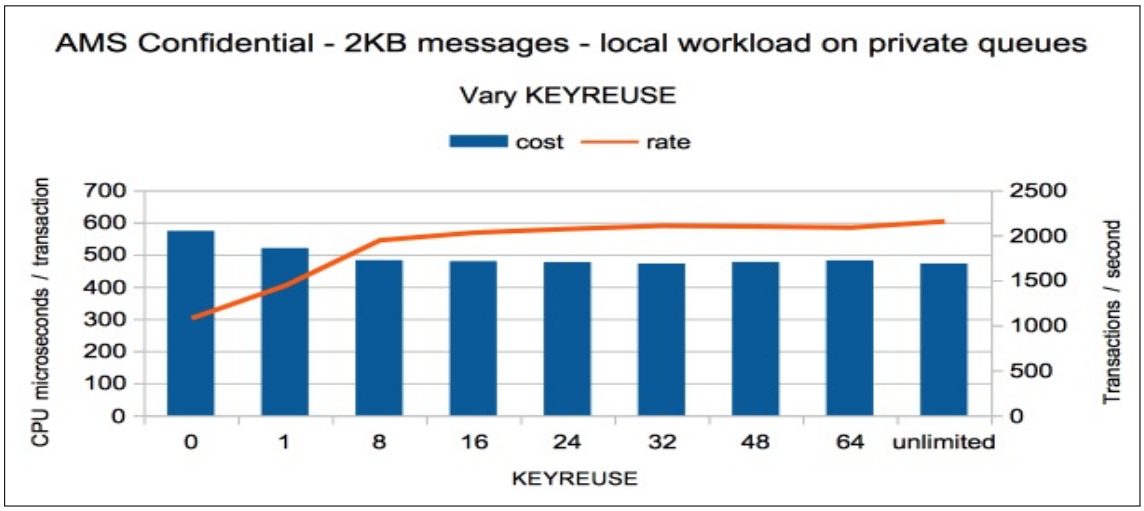


Chart: DES-encryption - 64KB message workload with varying key reuse values

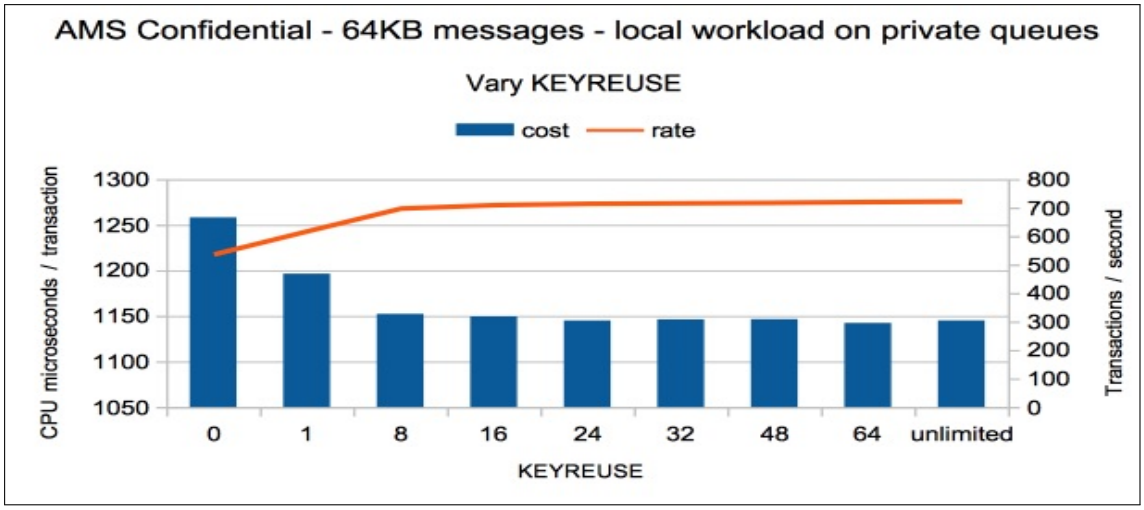
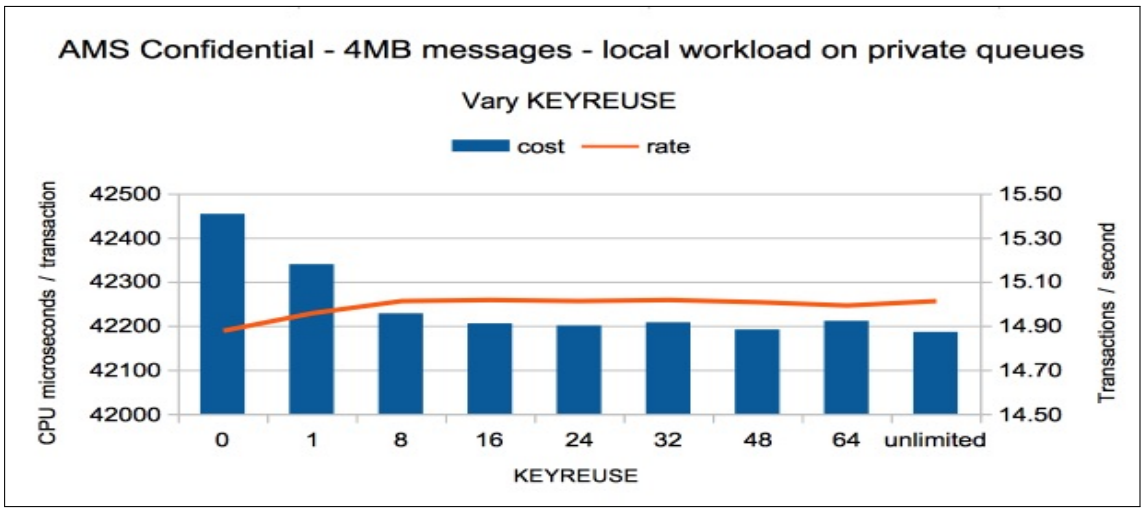


Chart: DES-encryption - 4MB message workload with varying key reuse values



Notes on DES-encryption charts

- For small messages, reusing the key 32 times or more gives the best performance
- As the message size increases, the benefits of higher key reuse values decrease.
- Large messages, such as the 4MB example, show little difference in performance regardless of the key reuse value, i.e. the difference between key reuse of 1 and unlimited is less than a 1% decrease in transaction cost.

How does AMS Confidential compare with Integrity and Privacy level protection?

This section compares the performance of the 3 available AMS qualities of service with regards to transaction cost and throughput.

For comparison purposes, a request / reply workload is being used between 2 queue managers on separate LPARs that are connected via multiple-sender receiver channels. The workloads begins with one pair of busy channels, i.e. on LPAR 1 there is a sender channel for outbound messages and a receiver channel for the inbound (reply) messages. The workload is increased until there are 5 busy outbound and 5 busy inbound channels.

For the purpose of these comparisons:

- The signing algorithm is SHA1.
- The encryption type is AES256.
- The key reuse value for the AMS *Confidential*-level measurement is 32.
- Cryptographic hardware is available.
- Each LPAR has 10 dedicated CPUs.

In our measurements with regards to throughput and cost, AMS confidential performed best, with AMS integrity next and followed by AMS privacy. However this does not take into account any audit, security or company policies that may need to be adhered to in your messaging environment.

Chart: Compare AMS Quality of Protection - Transaction Cost

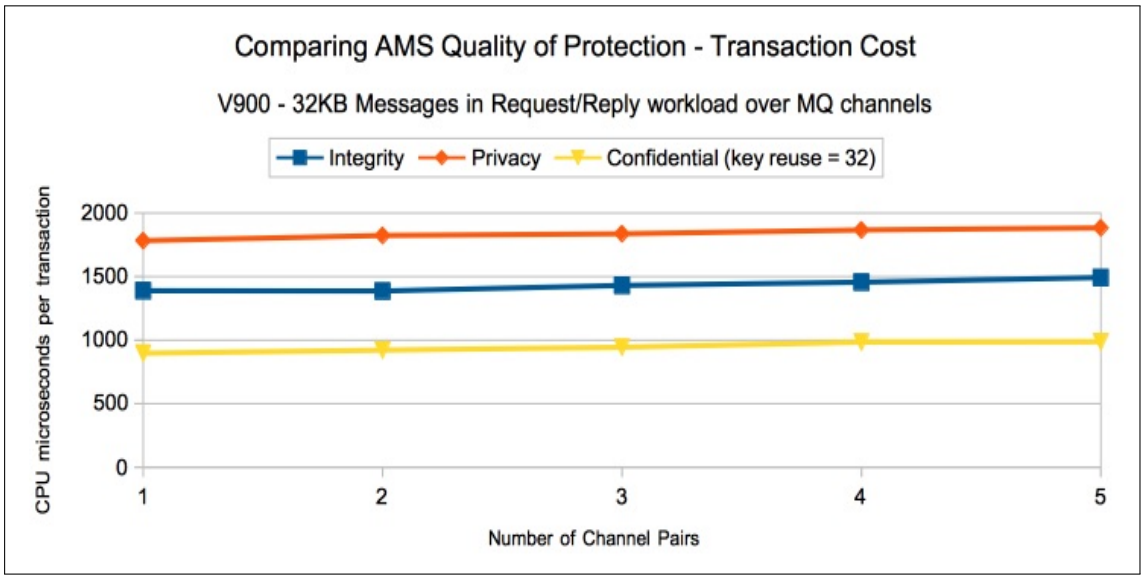
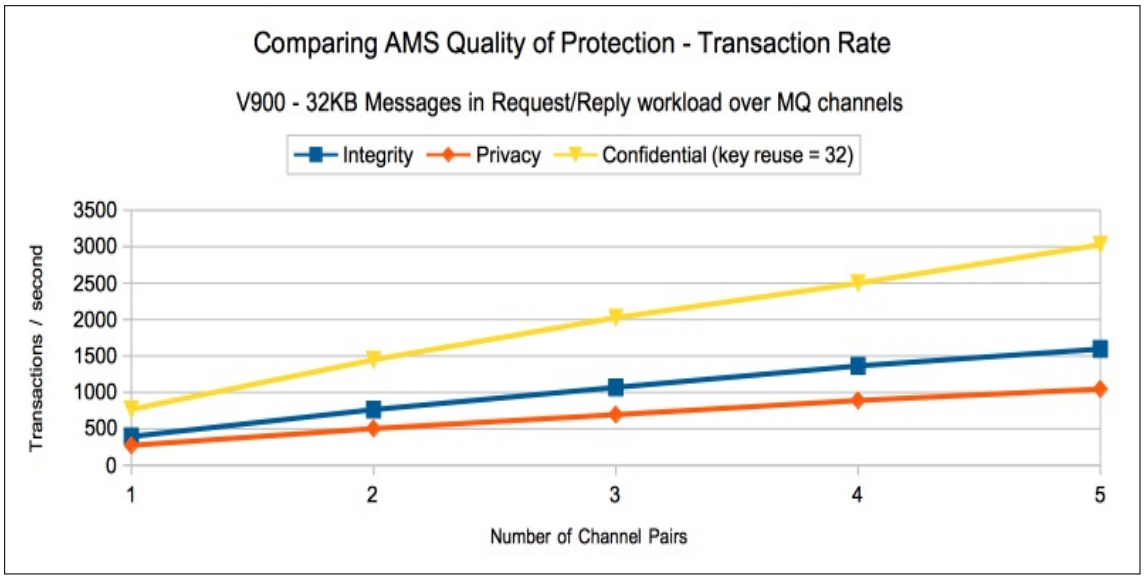


Chart: Compare AMS Quality of Protection - Transaction Rate



Notes on Quality of Protection charts

- The AMS Privacy workload costs 90% more than the AMS Confidential workload.
- The AMS Integrity workload costs 50% more than the AMS Confidential workload.
- The AMS Confidential workload achieves 3 times higher throughput than the AMS Privacy workload.
- The AMS Confidential workload achieves 2 times higher throughput than the AMS Integrity workload.

What if cryptographic offload is not available for AMS Confidential?

Encryption is largely performed by the CPACF processors and as a result the impact of having no cryptographic coprocessor or accelerators is reduced over the AMS Integrity or Privacy configurations.

There will be a higher impact from a lack of cryptographic hardware when the key is reused less.

Previously we have reported a 3 millisecond impact (per message put and got) from a lack of cryptographic hardware for AMS *Integrity* policies, which increases to 6 milliseconds when using AMS *Privacy*.

With AMS *Confidential* policies protecting a queue, we saw an increase of 1.5 CPU milliseconds per message (put and got).

How does AMS Confidential compare with other message protection options?

In the previous section we demonstrated the difference in performance between the three AMS qualities of protection, but how does this compare with a workload running today that may not have any protection, or protection offered from using SSL-enabled channels.

This section offers comparisons of the same workload being run in different configurations including:

- A basic sender-receiver channel pair
- SSL enabled channels where the secret key is negotiated only at channel start, i.e. SSLRKEYC(0).
- SSL enabled channels where the secret key is negotiated every 1MB or every 10MB processed i.e. SSLRKEYC(1048576) and SSLRKEYC(10485760).
- SSL cipher specification is "ECDHE_RSA_AES_256_CBC_SHA384".
- Where the queues are protected by AMS Integrity level protection where signing is SHA1.
- Where the queues are protected by AMS Privacy level protection where signing is SHA1 and encryption is AES256.
- Where the queues are protected by AMS Confidential level protection where the key reuse is 32, signing is SHA1 and encryption is AES256.

In these measurements, data is shown for 1 outbound and 1 inbound channel pair.

Cryptographic hardware is available in all measurements.

Chart: Impact of message protection type on transaction cost

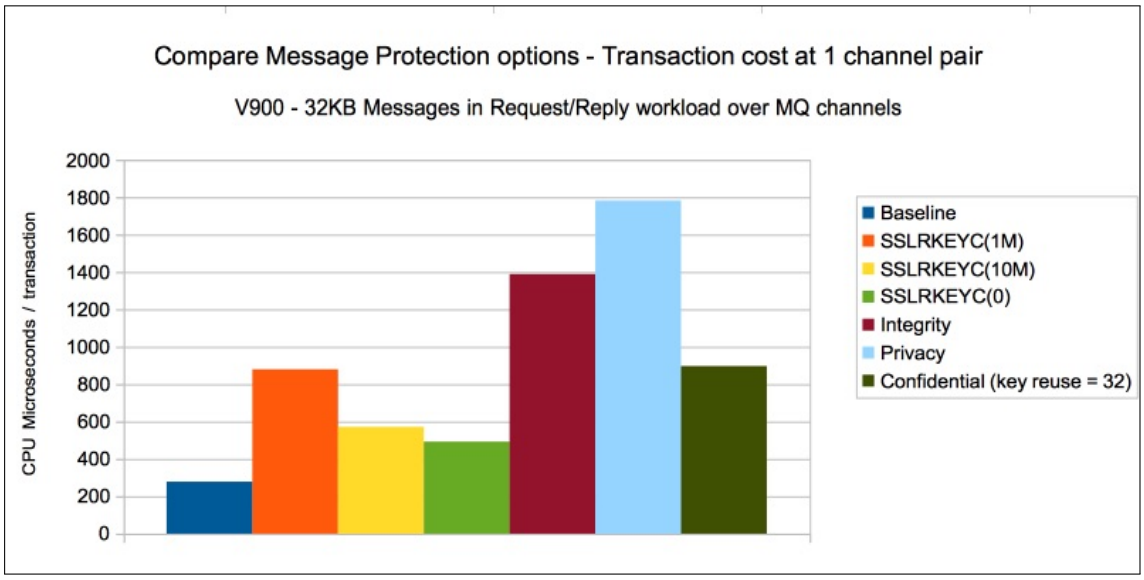
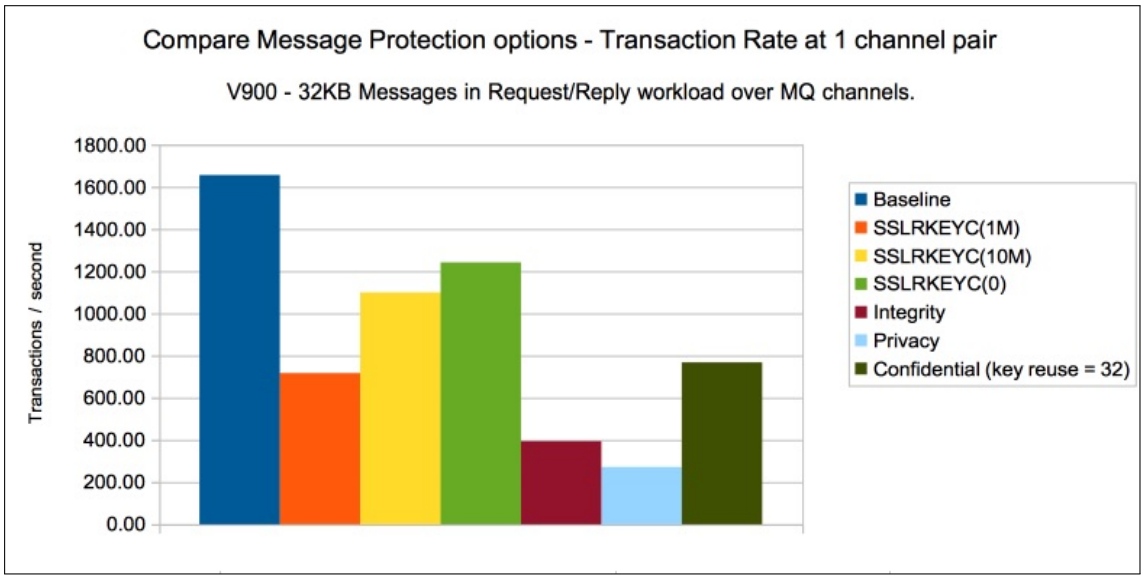


Chart: Impact of message protection type on transaction rate



Notes on impact of message protection type charts

- In these simple workloads, the AMS Confidential measurements transaction cost, which is using the same key for 1 MB of data, is similar to that achieved when using SSL channels where the secret key is negotiated every 1MB.
- The throughput of this AMS Confidential workload is higher than this SSL measurement as the workload is not delayed for renegotiating the secret key.

Whilst the AMS options Integrity, Privacy and Confidential transaction costs appear high, consider a scenario where the data flowing over the channels needs to cross (hop) over multiple queue managers, such as a gateway between the main office and a client system.

In this example, the SSL transaction cost would double, as the gateway queue manager would need to decrypt the data from the main office queue manager and re-encrypt the data for the clients system. This would result in the message being encrypted and decrypted twice, as well as potentially being

on the gateway queue manager unencrypted. Furthermore the secret key negotiation would need to be done twice as often. In this case we might expect the transaction to approximately double in cost. In the case of a frequent renegotiation such as at 1MB, this would mean the increase in transaction cost would be 880 microseconds.

With the AMS protection, the message is only encrypted and decrypted once, so if a gateway queue manager is added, the additional cost would be similar to that of the baseline measurement, i.e. of the order of 280 microseconds per transaction.

The following table demonstrates the increase in cost for the scenario of adding a gateway queue manager into the configuration. Costs shown are in CPU microseconds per transaction.

Table: Impact of adding a gateway queue manager to the protected workload measurements

Configuration	Transaction Cost	Cost of adding a gateway	Total Cost
Baseline	280	280	560
SSLRKEYC(1M)	880	880	1760
SSLRKEYC(10M)	570	570	1140
SSLRKEYC(0M)	490	490	980
Integrity	1390	280	1670
Privacy	1780	280	2060
Confidential (key reuse = 32)	770	280	1050

Using IBM MQ classes for JMS in a CICS OSGi JVM Server

IBM MQ classes for Java™ Message Service (IBM MQ classes for JMS) is the JMS provider that is supplied with IBM MQ. As well as implementing the interfaces described in the javax.jms package, the IBM MQ classes for JMS provide two sets of extensions to the JMS API. The main focus of these extensions concerns creating and configuring connection factories and destinations dynamically at run time, but the extensions also provide function that is not directly related to messaging, such as function for problem determination.

The IBM MQ classes for JMS have been built with Java 7 since IBM MQ for z/OS version 8.0.

WebSphere MQ for z/OS version 7.1 provides support for using IBM MQ classes for JMS via APAR PI29770 when using CICS version 5.2 or later.

IBM MQ for z/OS version 8.0 added support for using the IBM MQ classes for JMS in certain versions of the CICS Open Services Gateway initiative (OSGi) Java™ Virtual Machine (JVM) server.

Using the IBM MQ classes for JMS in a CICS OSGi JVM Server allows a continuation of the principle objectives of JMS:

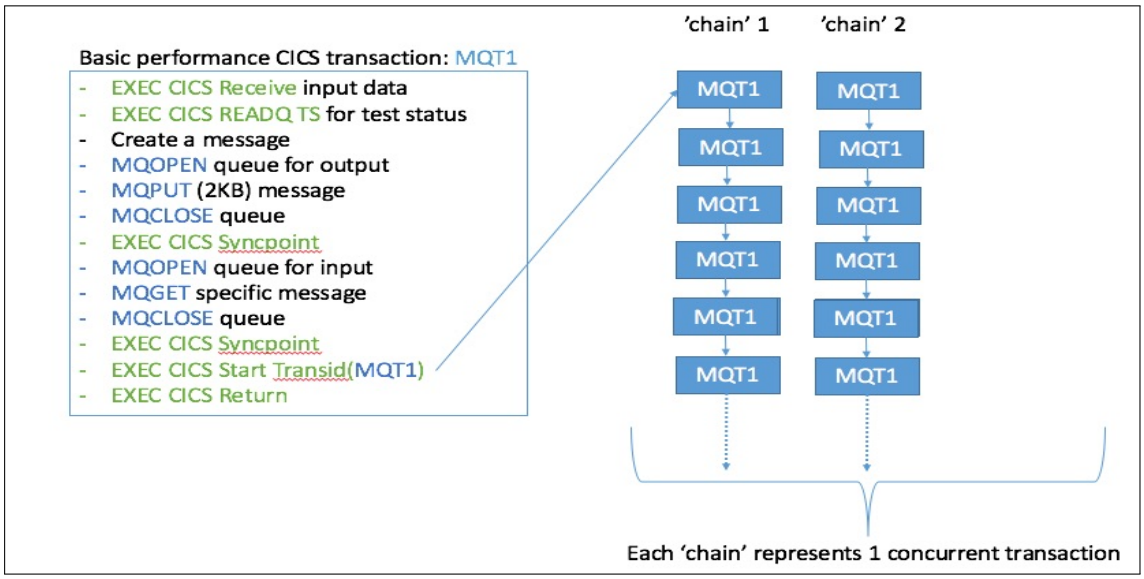
- To provide a greater level of consistency across IBM JMS providers
- To make it easier to write a bridge application between two IBM messaging systems
- To make it easier to port an application from one IBM JMS provider to another.

The ease of development however can come at a cost to performance, and the following section attempts to indicate some of that impact.

Comparing JMS performance with more traditional languages

As part of the regression tests running CICS workloads, a simple CICS COBOL program is used to drive an MQ workload. The transaction actions and model are shown in the following diagram.

Chart: Simple CICS COBOL transaction running MQ message workload



The transaction is initiated and once the message workload is complete, initiates the next transaction in the chain, before ending. Multiple chains are run to emulate concurrent transactions.

To compare the performance of JMS against this COBOL application, both a JMS and a Java™ application were created using the same model.

MQ API calls based on application programming language

Whilst the core MQ APIs invoked for logically similar transactions written in different languages remain the same, additional interaction with the queue manager may be required when implementing applications in Java or JMS.

For example the MQT1 program referred to earlier performs a simple function - put a message and get a reply message. In order to do this a number of MQ APIs were called, some directly and some as a result of the program actions. The following table shows that for similar function implemented in different languages will result in different MQ API calling patterns, i.e the table shows the number of MQ API calls per transaction.

API function	COBOL	Java	MQ JMS
Open queue	2	2	2
Put message	1	1	1
Get message	1	1	1
Commit	2	1	2
Close queue	2	2	2
Other	1	14	7
Backout	1	1	1

Notes on table:

- The Java application is only doing 1 commit, unlike both the COBOL and JMS applications which do a MQPUT followed by an EXEC CICS SYNCPOINT and then a MQGET followed by another EXEC CICS SYNCPOINT.
- Both the Java and MQ JMS applications see a significant increase in the number of "other" type calls, which include the equivalent of MQINQ of the queue manager and queues in use.
- IBM MQ Classes for Java have been functionally stabilized at the level shipped in IBM MQ for z/OS version 8.0. Existing applications that use the IBM MQ classes for Java will continue to be fully supported, but this API is stabilized, so new features will not be added.

What impact does the application language have on the MQ API cost?

The previous section shows that for similar function applications written in different languages, the amount of interaction with the MQ queue manager can vary, but how might this affect the cost of the MQ workload?

The following table reports the total MQ API cost (CPU time) for each of the functions invoked based on the average costs as reported by TRACE(A) CLASS(3) accounting data from a measurement using 2 chains of transactions. Costs shown are in CPU microseconds.

API function	COBOL	Java	MQ JMS
Open queue	4.08	4.14	4.42
Put message	5.52	6.39	15.42

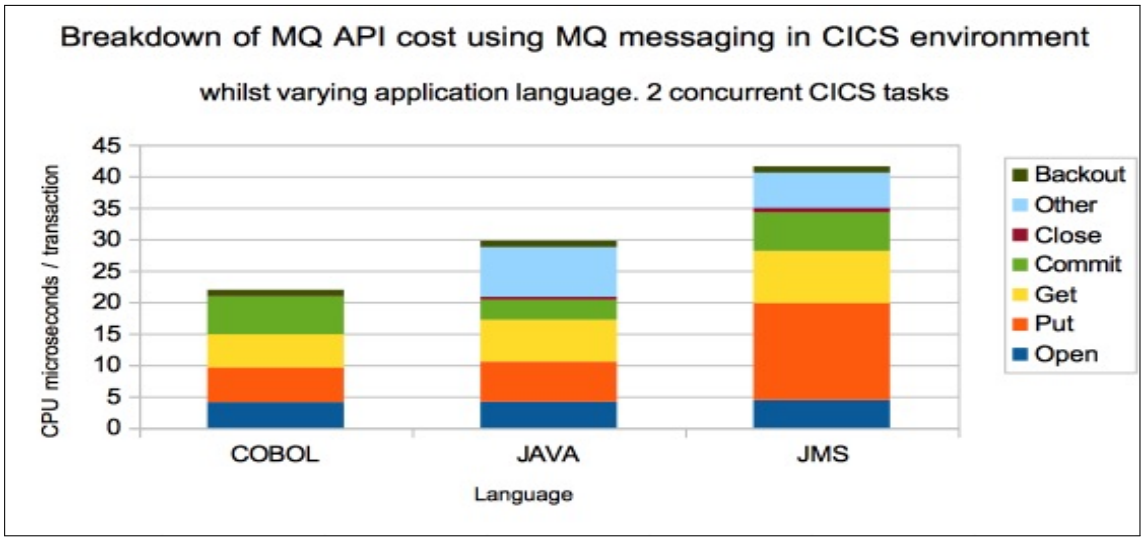
Get message	5.28	6.66	8.3
Commit	6.01	3.18	6.09
Close queue	0.05	0.48	0.73
Other	0	7.87	5.59
Backout	1.01	1.02	1.02
Total	21.95	29.74	41.57

Notes on table:

- The put message function of the MQ JMS application is significantly more expensive than either COBOL or Java, primarily due to the format of the message being created.
- The impact of the additional "other" calls does cause a marked increase in the MQ API cost which is particularly noticeable in the Java measurement.

The following chart is a representation of the data in the table.

Chart: Total MQ API cost (CPU time) for each of the functions invoked based on the average costs as reported by TRACE(A) CLASS(3) accounting data from a measurement using 2 chains of transactions.



This of course does not tell the full story - this is just the direct interaction with the queue manager.

In addition to this, there is the overhead of the running the application environment under CICS plus the work that the queue manager must do as part of commit processing. The following table shows a breakdown of the transaction cost in CPU microseconds for each of the application languages.

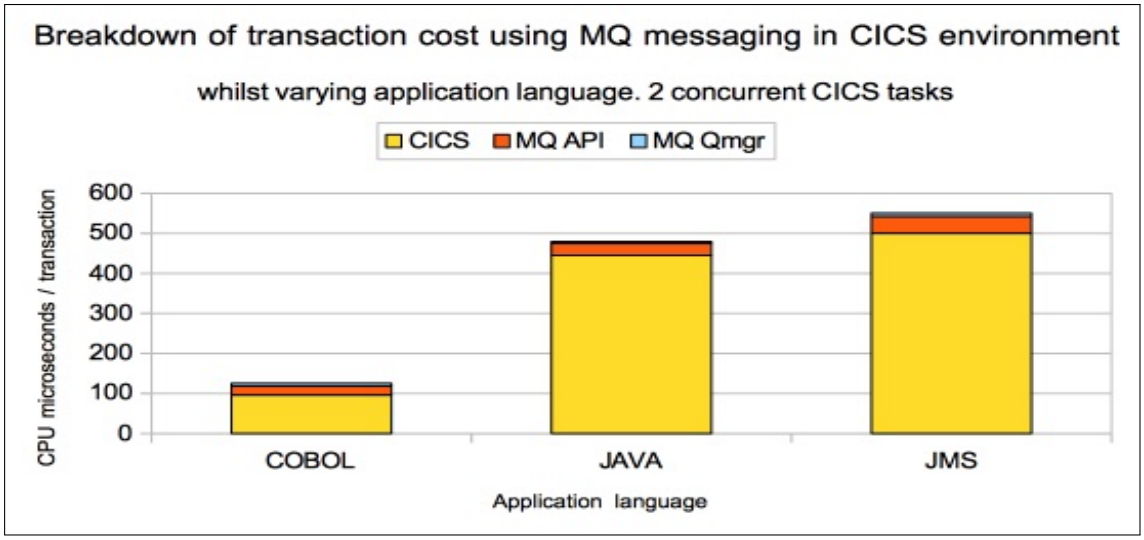
Component	COBOL	Java	MQ JMS
CICS	96.26	445.12	500.43
MQ APIs	21.95	29.74	41.57
Queue Manager (processing of commit, SRB time)	7.14	4.57	8.32
Total	125.34	479.43	550.32

What this suggests is that the cost of a Java application is of the order **3.8 times** that of the equivalent COBOL application, and the MQ JMS application might be **4.4 times** that of the

COBOL application.

The following chart is a graphical representation of the data in the chart.

Chart: Breakdown of transaction cost from measurement using 2 chains of transactions.



Comparing the performance in similarly configured systems

The charts in this section demonstrate the performance differences, when using the 3 application languages described earlier, as the workload is increased. The system under test is limited to 16 dedicated processors and the performance is limited by CPU when running with the maximum reported number of concurrent transactions.

Chart: Achieved transaction rate using MQ messaging as workload increases.

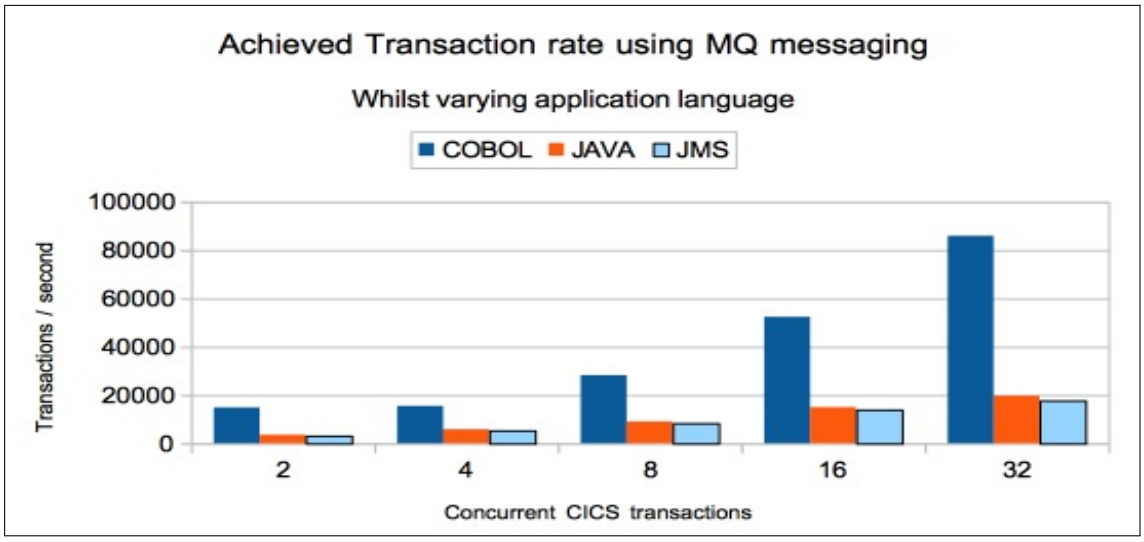
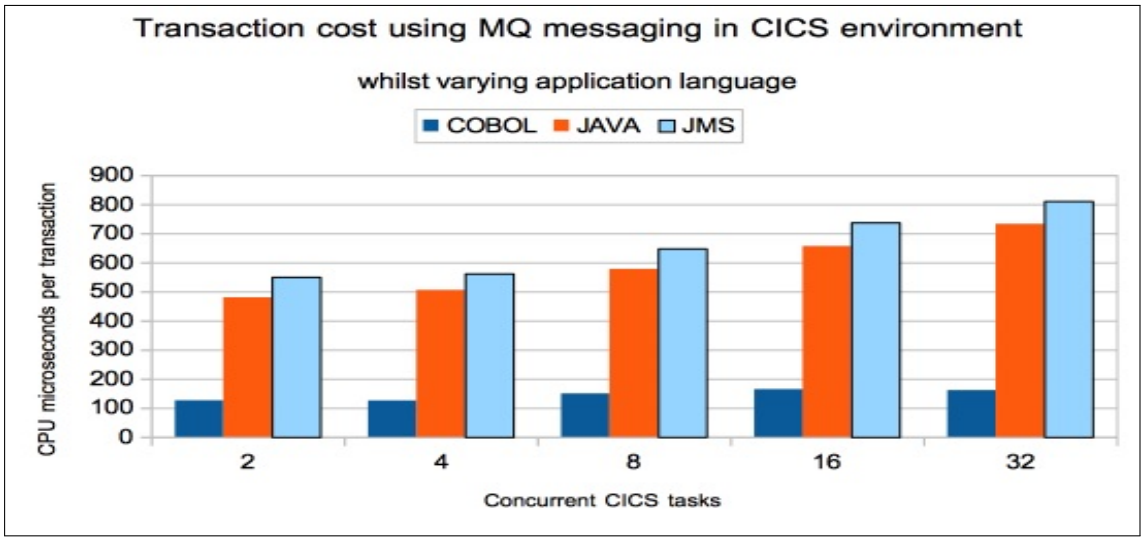


Chart: Actual transaction cost when using MQ messaging as workload increases.



Reducing transaction cost by using zIIP specialty engines

The IBM z Systems Integrated Information Processor (zIIP) has been available since the System z9 server. It is designed to free-up general computing capacity and lower overall total cost of computing for select data and transaction processing workload. From IBM z13, zIIP specialty engines also run workloads that are eligible to run on zAAP specialty engines.

In addition Simultaneous Multithreading (SMT) support for zIIP processors on z13 servers is designed to offer throughput improvements to address the growing volume of zIIP-eligible work such as Java-based transactions and XML parsing.

zIIP eligible workloads include, but are not limited to, portions of the following:

- zAAP eligible workloads, such as Java (or JMS).

For a more complete list, please refer to IBM z Systems Integrated Information Processor (zIIP)

What tuning options are available for zIIP specialty engines?

There are a number of IEAOPTxx parameters that can help or enforce usage of specialty processors including:

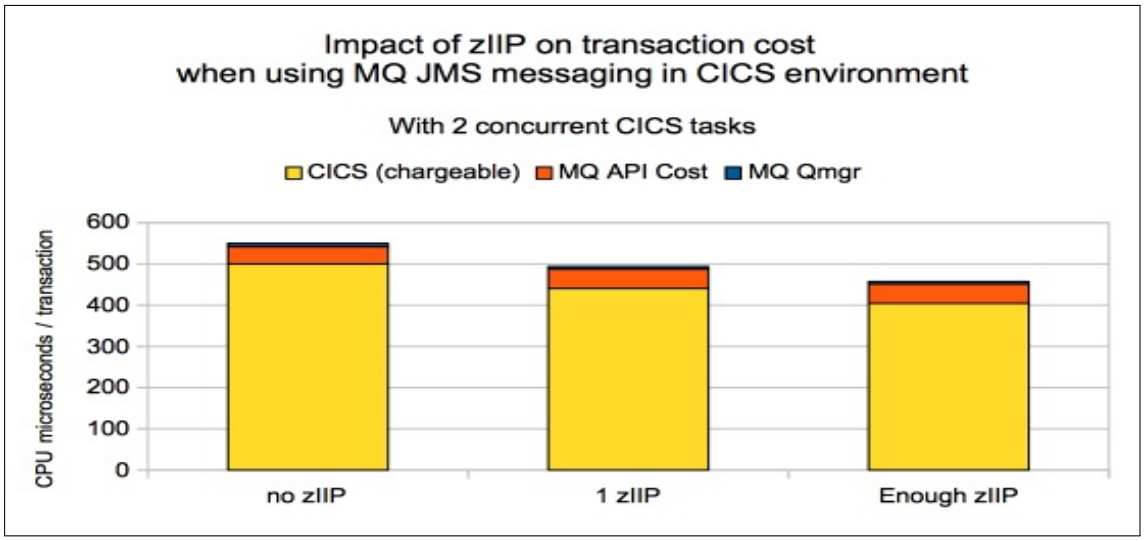
- **IFAHONORPRIORITY=YES| NO**
  - **YES:** Specifies that standard processors run both zAAP processor eligible and non-zAAP processor eligible work in priority order when the zAAP processors indicate the need for help from standard processors.
  - **NO:** Specifies that standard processors will never examine zAAP processor eligible work. This can lead to waits for zAAP processors even when there are available general purpose CPs available. Note that standard processors also run zAAP processor eligible work if it is necessary to resolve contention for resources with non-zAAP processor eligible work.
- **IIPHONORPRIORITY=YES | NO**
  - **YES:** Specifies that standard processors run both zIIP processor eligible and non-zIIP processor eligible work in priority order when the zIIP processors indicate the need for help from standard processors.
  - **NO:** Specifies that standard processors will never examine zIIP processor eligible work. Note that standard processors also run zIIP processor eligible work if it is necessary to resolve contention for resources with non-zIIP processor eligible work.
- **PROJECTCPU=YES | NO**
  - Specifies whether to activate or deactivate the projection of how work could be offloaded from regular CPs to special assist processors like zIIP. This option can be used when no zIIP is available as a guide to projected savings.
- **MT_ZIIP_MODE=1 | 2**
  - Specifies the multi-threading mode (MT) for each online zIIP core. The value is limited to what the underlying hardware and operating system support.

Example measurements using zIIP processors

The following JMS measurements have been run on a system where there is 1 zIIP processor available replacing 1 general purpose processor to demonstrate the effect of specialty processors on the transaction cost. In these measurements the following options are in specified.

- PROJECTCPU=YES
- MT_ZIIP_MODE=1

Chart: Impact of zIIP on MQ for JMS transaction in CICS OSGi CICS JVM Server.

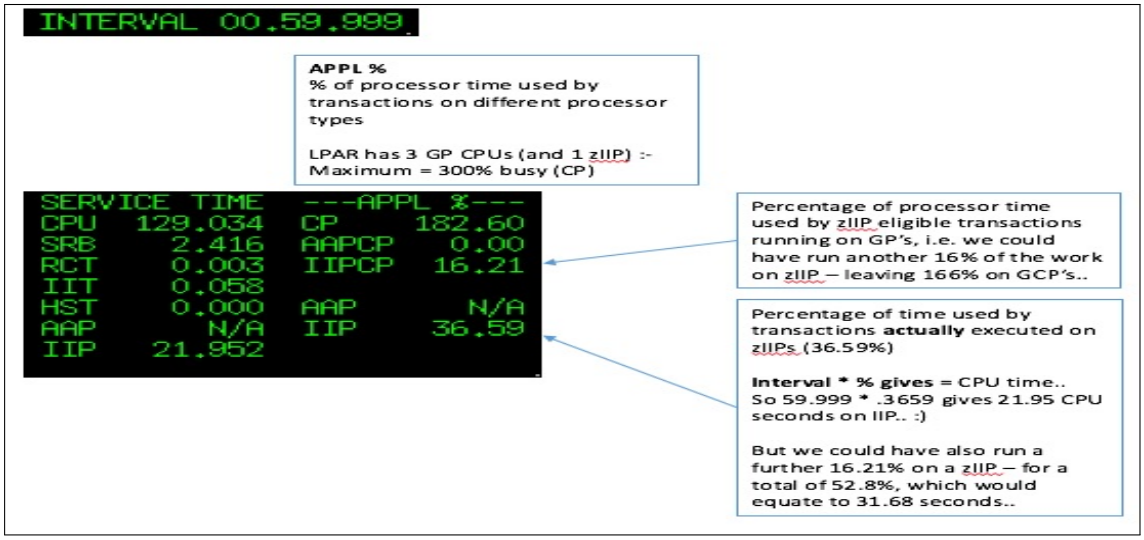


Notes on chart:

- In the configuration used, only the CICS workload is eligible for offloading onto zIIP.
- The data for the "no zIIP" measurement was run on a system with 4 dedicated general purpose CPU's.
- The data for the "1 zIIP" measurement was run on a system with 3 dedicated general purpose CPU's and 1 zIIP.
- The CICS cost per transaction for the "Enough zIIP" data is based upon the projected CPU usage, so includes the CPU that was run on the available zIIP plus the time that was used on the general purpose processors that was eligible for zIIP offload. This is the maximum saving that could be achieved given a surplus of zIIP processors.
- Providing "enough zIIP" reduces the total transaction cost on chargeable processors by 17%, which in this simple measurement equates to a 20% reduction in CICS transaction cost.
- The benefits of SMT-2 are discussed in the IBM CICS Performance Report: CICS TS for z/OS V5 Performance Report, specifically section 7.10 "Simultaneous multithreading with Java workloads".

Process used for calculating "Enough zIIP" cost savings

An extract of the RMF Workload report is provided to demonstrate how the "Enough zIIP" data was calculated.



The RMF workload report has a column headed "APPL %" which reports the percentage of processor time used by transactions on different processor types e.g. CP or IIP.

- The "IIPCP" field reports the percentage of processor time used by zIIP-eligible transactions that ran on general purpose processors.
- The "IIP" field reports the percentage of processor time **actually executed** on the zIIP processor.
- The sum of the IIP and IIPCP fields give the projected percentage for zIIP-eligible transactions.
- Multiplying the sum of the IIP and IIPCP by the interval length gives an indication of the CPU that could be offloaded to zIIP given sufficient zIIP resource, which we will call 'total projected zIIP time'.
- The CPU service time includes the seconds consumed by both general purpose CPUs and the specialty processors, so we can subtract the total projected zIIP time from the CPU service time to give an indication of the chargeable CPU time for this interval.

Using IBM MQ classes for JMS in IMS

Support for IBM MQ classes for Java™ Message Service (IBM MQ classes for JMS) was added in IBM MQ for z/OS version 8.0, via APAR PI41909, which is a cumulative fixpack, and can be used with IMS versions 13 and later.

The classes are supported in the following IMS dependent region types:

- MPR
- BMP
- IFP
- JMP
- JBP

The IBM MQ classes for JMS requires Java Platform, Standard Edition 7 (Java SE 7) or later.

This chapter will compare the performance of JBP and BMP region types and demonstrate some of the CPU savings that may be found from exploiting zIIP specialty engines.

The chapter will also compare the performance of JMP with MPR region types and consider some additional impact from using the IBM MQ classes for JMS in IMS regions.

If the IMS region e.g. BMP, MPR, JBP, or JMP runs an application that issues any MQ verbs, it is advised that that certain modules are preloaded by IMS as detailed in the "Preloading the IMS adapter" section in the IBM MQ for z/OS version 9.0 Knowledge Centre. Of particular interest is the CSQBAPPL module which is a new addition to the list.

Comparison of JBP with BMP region types

Java Batch Processing (JBP) regions run flexible programs that perform batch-like processing online and can access the IMS message queues for output, similar to non-message driven BMP applications. JBP applications are like BMP applications except that they cannot read input messages from the IMS message queue. Like BMP applications, JBP applications are started by submitting a job with JCL.

The IMS PROCLIB contains the proc DFSJBP which can be used to run JBP regions. This runs the same program as used for more traditional BMP regions, i.e. DFSRRC00. There are a number of parameters that are of particular interest when running a JBP region:

- IMSID

  The IMS subsystem ID under which to execute.

- MBR

  The application program name. To indicate which Java application to invoke when running the JBP, specify a 1-8 character name in the MBR parameter. When the MBR name is specified, the JBP tries to load the DFSJVMAP IMS proclib member. This DFSJVMAP member maps the MBR names to fully qualified Java class names.

- ENVIRON

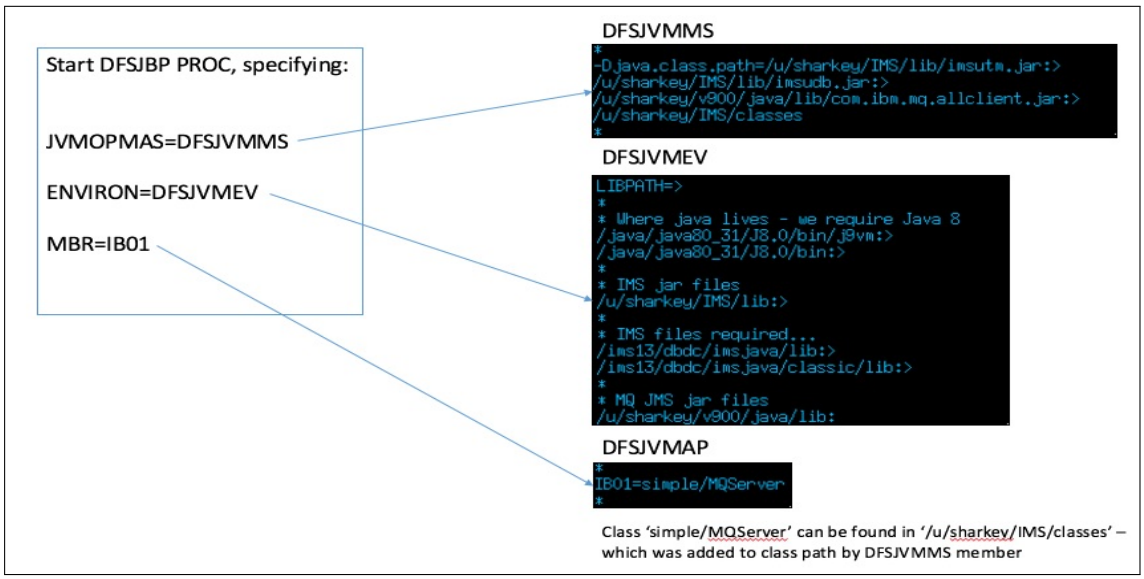
  Name of the IMS proclib member that contains the library path (LIBPATH) definition for the Java environment.

- JVMOPMAS

Name of the IMS proclib member that contains the CLASSPATH and JVM options for the master JVM.

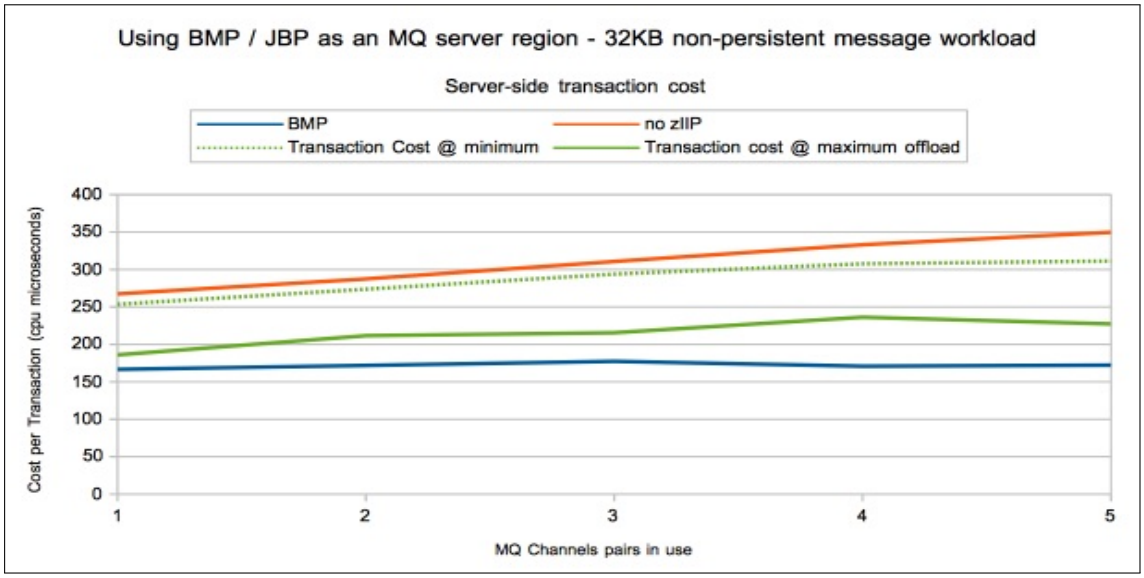
The following diagram is intended to assist clarification of how the different files are related for IMS JBP regions.

Diagram: Demonstrate JPB parameters



As with the measurements shown in the section "Using IBM MQ classes for JMS in a CICS OSGi JVM Server", the cost of the JMS applications can be reduced with the use of specialty processors such as zIIP. The extent of this is demonstrated in the following chart.

Chart: Compare cost of BMP against JBP, with zIIP offload savings



Notes on chart:

- The workload run is a set of request/reply workloads each using 1 channel pair, moving messages between LPAR 1 and LPAR 3
- The transaction costs shown are a reflection of the data from the server-side of the workload

and includes the server MQ queue manager, channel initiator, IMS control region and IMS batch processing region, whether a BMP or a JBP.

- The server application connects once to the MQ queue manager, opens the request queue once and then processes each message until requested to end. The processing of each message is simple and results in the reply message being put to a reply queue.
- The transaction costs with zIIP offload available are based on the projected savings from the RMF Workload report, where the minimum cost is calculated from the value reported in the interval with the least zIIP usage predicted, and the maximum cost is calculated from the values reported in the interval with the most zIIP usage predicted.
- The transaction rate achieved in the measurements is comparable for BMP and JBP regions as there was sufficient CPU capacity in the LPAR.
- In the optimum JBP measurement, with 5 inbound and 5 outbound channels being busy, the non-offloaded JBP workload equates to the 32% more than the BMP workload or 70 CPU microseconds per transaction.
- The BMP uses an application written in C.

Comparison of JMP with MPR region types

A JMP application program is similar to an MPP application program, except that JMP applications are written in Java or object-oriented COBOL. Like an MPP application, a JMP application is started when there is a message in the message queue for the JMP application. IMS schedules the message for processing.

JMP applications can do the following:

- Retrieve input messages from the IMS message queue.
- Access IMS and DB2 for z/OS databases, connect to an MQ subsystem and put or get messages.
- Commit or roll back transactions.
- Send output messages.

The IMS PROCLIB contains the proc DFSJMP which can be used to run a JMP region. As with the JBP region, the JVMOPMAS parameter references a file that sets the CLASSPATH and JVM options for the master JVM, the ENVIRON parameter specifies the LIBPATH, and the DFSJVMAP provides a cross reference between the transaction and the Java class to be run.

Note that the IBM MQ classes for JMS build upon existing the IBM MQ adapter support which makes use of the External Subsystem Attach Facility (ESAF). This means that the documented behaviour applies, including all open handles being closed by the IMS adapter when a syncpoint occurs. For this reason it is advisable to ensure that the MQ preload list is passed to the JMP (or MPR) when the IMS transaction makes MQ API calls, to avoid having to load the files for each unit of work.

JMP performance compared with MPR

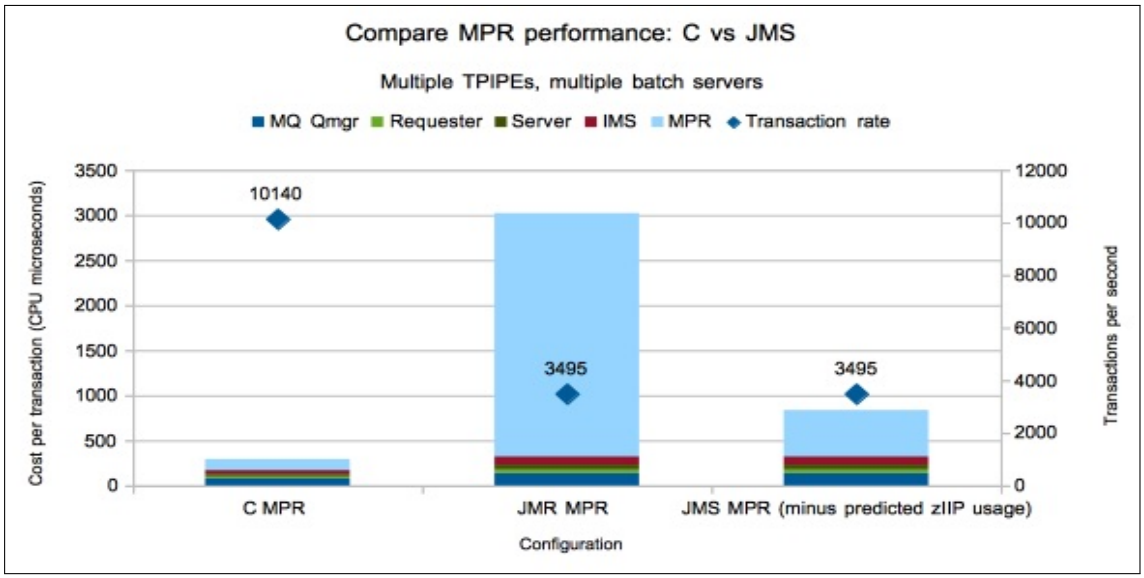
The tests detailed in this section run an IMS Bridge workload:

- The workload is driven by multiple batch applications that put a 2KB message to an IMS Bridge queue.
- These IMS Bridge queues are served by a local IMS region with multiple message processing regions, either MPR or JMP.

- Once the message is gotten from IMS using a GETUNIQ call, the region connects to an MQ queue manager, opens pair of request and reply queues, puts a message to the request queue and waits for the reply message.
- These request messages are processed by long running batch tasks that MQGET-next the request message, process the message contents and MQPUT a reply message
- Once the reply message is received by the IMS message processing region, the application closes the queue and returns a modified reply message to IMS using the IMS ISRT verb.
- The inserted message is returned to the original batch application via the IMS Bridge.

The measurements interchange between an MPR running a C program and a JMP running a JMS application.

Chart: Compare performance of JMP against MPR



Notes on chart comparing performance of JMP with MPR

- The JMP transaction cost was nearly 22 times higher than the equivalent MPR transaction cost, although a large proportion of this cost can be reduced through the use of zIIP offload. The PROJECTCPU predicted the JMP cost could be reduced such that the JMP cost 4 times that of the MPR transaction.
- In order to achieve the reported transaction rates, the MPR test used 4 processors whereas the JMP test used 12 processors.

Page set statistics

Page set statistics have been added to IBM MQ for z/OS version 9.0 and are enabled as part of `CLASS(1)` statistics trace.

The gathered data can be formatted using program MQSMF, which is available as part of SupportPac MP1B "Interpreting accounting and statistics data".

The data collected as part of page set statistics includes the following:

- How full each page set was during each interval
- Whether the page set needed to expand, and if so, how many times
- How many pages were written / read in the interval
- How long the queue manager took to write data to the page set. MQ can write multiple pages in one I/O.
- How long the queue manager took to read data from the page set. MQ will read 1 page per I/O.

Guidance on how to use this data will be provided in MP1B "Interpreting accounting and statistics data".

What affects the cost of page set statistics?

The processing performed by page set statistics is similar to that by the command `DISPLAY USAGE TYPE(PAGESET)`.

The cost of this processing is affected by the total number of pages in the page sets allocated to the queue manager. For example on a 2964-NE1 (z13), an MQ queue manager was able to process 600GB of page set data at the cost of 1 CPU second.

This means that for a queue manager with 100 page sets each of 64GB we would predict a CPU cost 10.66 CPU seconds at the end of each SMF interval. This work is performed by a single TCB running at the queue managers' dispatching priority.

A more typically sized queue manager might have 10-20 page sets with a total of 600 GB of page set disk space allocated. The following table uses LSPR capacity data to provide a guide to how much page set statistics (or indeed `DISPLAY USAGE TYPE(PAGESET)`) might cost on different z/OS machines:

Machine	Model	Average Relative Nest Intensity	CPU seconds to process 600GB
z13	2974-703	8.3	1
zEC12	2827-703	7.42	1.12
zEnterprise 196	2817-703	5.92	1.4
z10	2097-703	4.41	1.88
z9	2094-703	2.97	2.79

Note: The cost has been calculated by dividing the average relative nest intensity (RNI) of the z13 (2964-703) by the appropriate model e.g. for the z9 cost, 8.3 / 2.97 = 2.79, which as the cost on z13 is 1, means the cost of z9 is expected to be approximately 2.79 CPU seconds and this cost would be incurred at the end of each SMF interval when TRACE(S) CLASS(1) is enabled on an IBM MQ for z/OS version 9.0 queue manager.

Appendix A

Regression

When performance monitoring IBM MQ for z/OS version 9.0, a number of different categories of tests are run, which include:

- Private queue
- Shared queue
- Moving messages using MCA channels
  - SSL
  - Channel compression (ZLIBFAST / ZLIBHIGH)
- Moving messages using Cluster channels
- Client
- Bridges and adaptors
- Trace

These tests are run against version V7.1 (V710), V8.0 (V800) and V9.0 (V900) and the comparison of the results is shown in subsequent pages.

The statement of regression is based upon these results.

All measurements were run on a performance sysplex of a z13 (2964-NE1) which was configured as described in "System Configuration".

Given the complexity of the z/OS environment even in our controlled performance environment, a tolerance of +/-6% is regarded as acceptable variation.

Private Queue

Non-persistent out-of-syncpoint workload

Maximum throughput on a single pair of request/reply queues

The test uses 5 batch requester tasks that each put a message to a common request queue and wait for a specific response on the reply queue. Once they have gotten the message, they put another message to the request queue. The messages are put and got out of syncpoint.

There are 4 batch server tasks that action MQGET-with-wait calls on the request queue, get the message and put a reply to the known (and pre-opened) reply queue and then the application goes back into the MQGET-with-wait call. The messages are put and got out of syncpoint.

Chart: Transaction rate for non-persistent out-of-syncpoint workload

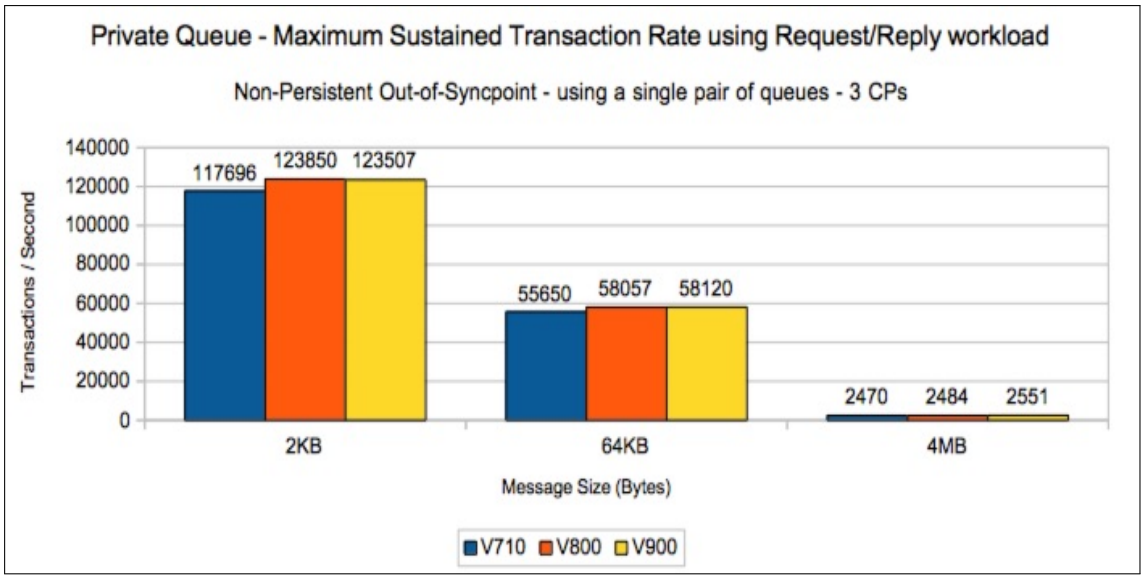
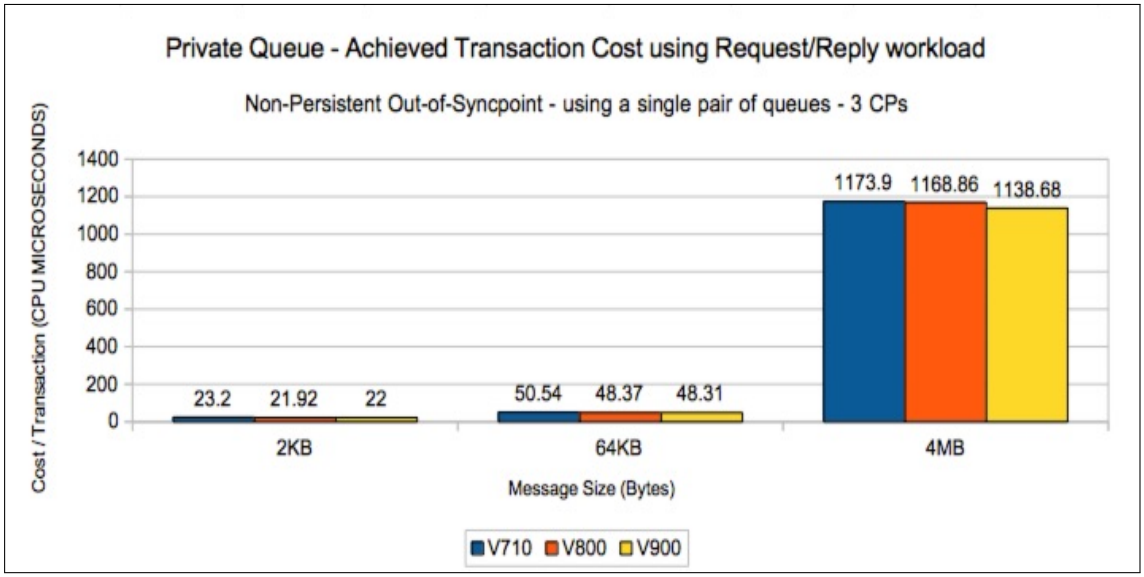


Chart: Transaction cost for non-persistent out-of-syncpoint workload



Scalability of request/reply model across multiple queues

The queue manager is configured with pagesets 0 through 15 and with 1 buffer pool per pageset.

On each of pagesets 1 to 15, a pair of request and reply queues are defined. The test starts up 1 requester and 1 server task accessing the queues on pageset 1 and runs a request/reply workload. At the end, the test starts a second pair of requester and server tasks which access the queues on pageset 2 and so on until there are 15 requester and 15 server tasks using queues on all 15 pagesets with the application queues defined.

The requester and server tasks specify NO_SYNCPOINT for all messages.

The measurements are run on a single LPAR with 16 dedicated processors online on the z13 (2964) used for testing.

Chart: Transaction rate for non-persistent out-of-syncpoint scalability workload

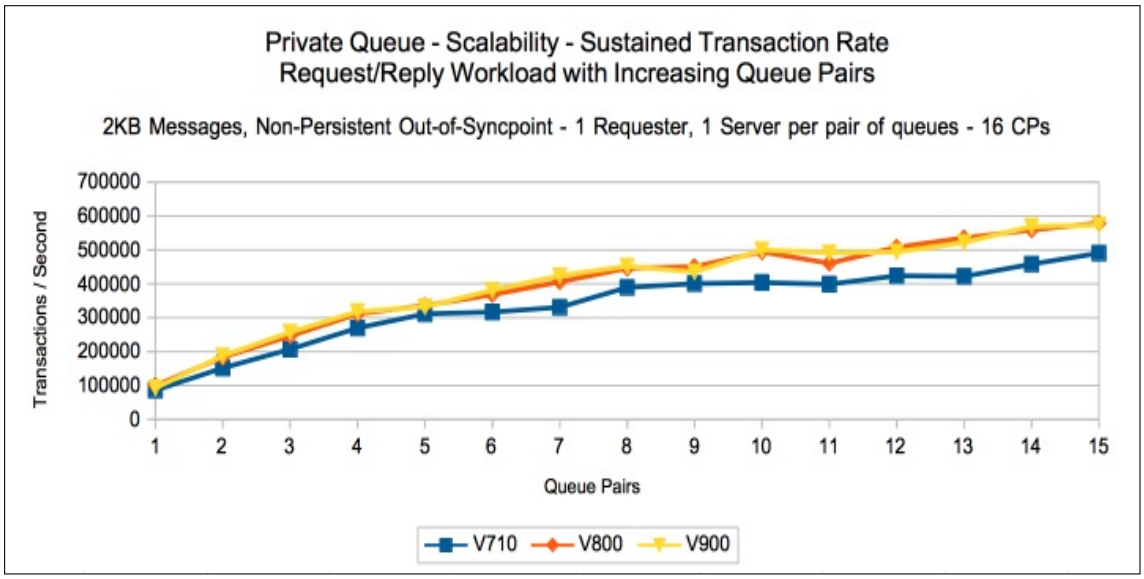
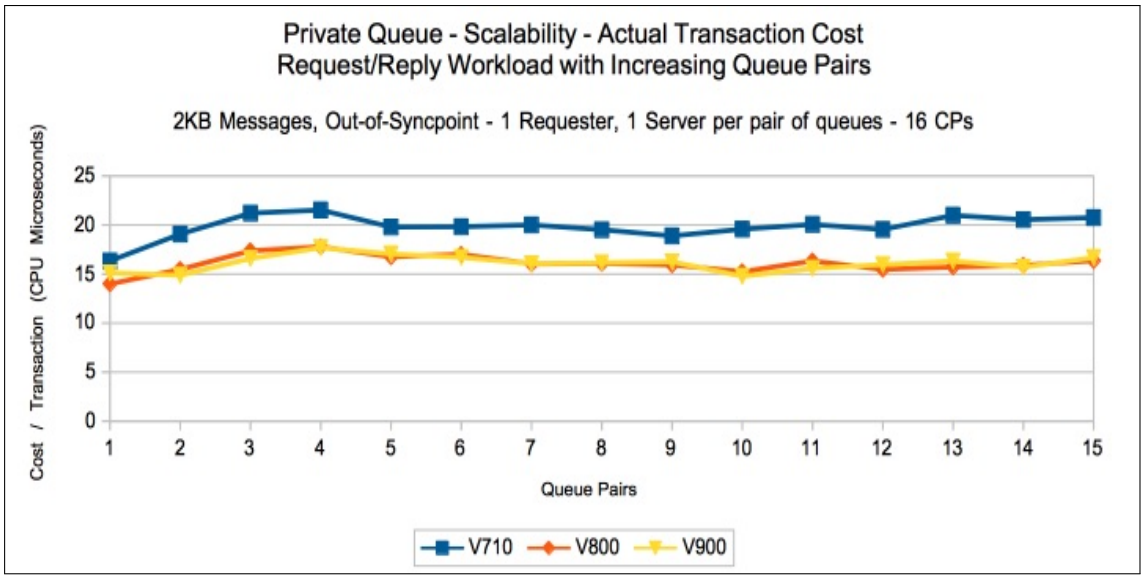


Chart: Transaction cost for non-persistent out-of-syncpoint scalability workload



The preceding 2 charts show that a single queue manager is able to drive in excess of 575,000 transactions per second - or 1,150,000 non-persistent messages per second on a 16-way LPAR.

Non-persistent server in-syncpoint workload

Maximum throughput on a single pair of request/reply queues

The test uses 5 batch requester tasks that each put a message to a common request queue and wait for a specific response on the reply queue. Once they have gotten the message, they put another message to the request queue. The messages are put and got out of syncpoint.

There are 4 batch server tasks that action MQGET-with-wait calls on the request queue, get the message and put a reply to the known (and pre-opened) reply queue and then the application goes back into the MQGET-with-wait call. The messages are got and put in syncpoint with 1 MQGET and 1 MQPUT per commit.

Chart: Transaction rate for non-persistent in syncpoint workload

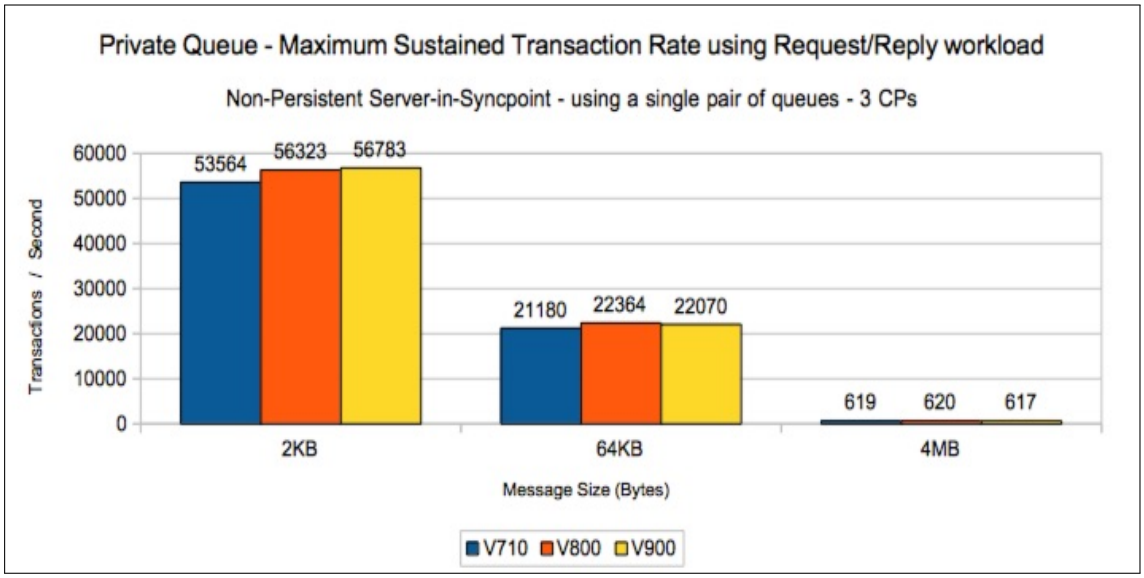
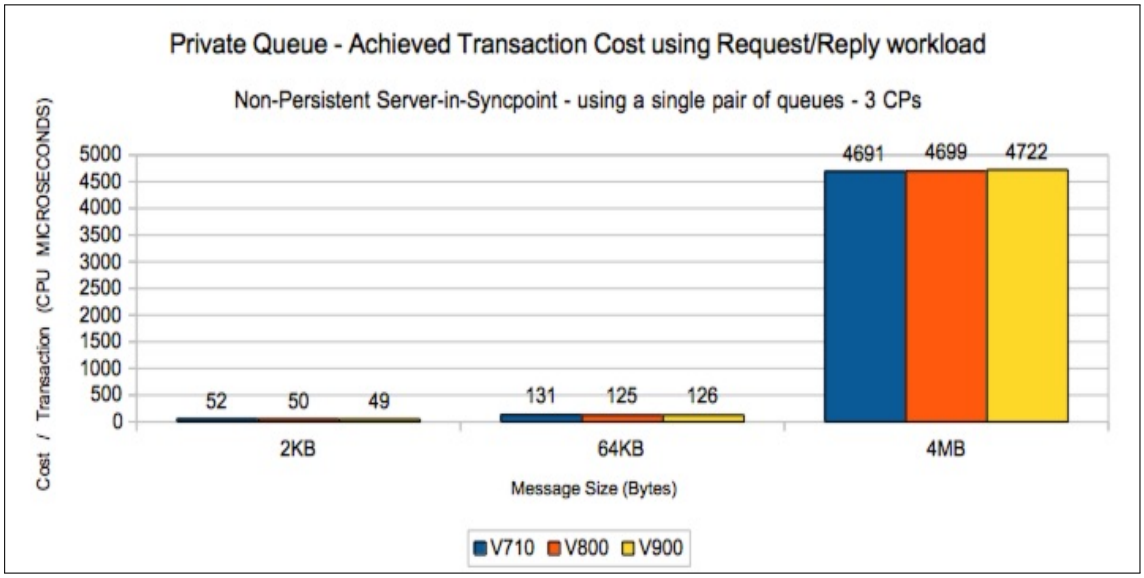


Chart: Transaction cost for non-persistent in syncpoint workload



Scalability of request/reply model across multiple queues

The queue manager is configured with pagesets 0 through 15 and with 1 buffer pool per pageset.

On each of pagesets 1 to 15, a pair of request and reply queues are defined. The test starts up 1 requester and 1 server task accessing the queues on pageset 1 and runs a request/reply workload. At the end, the test starts a second pair of requester and server tasks which access the queues on pageset 2 and so on until there are 15 requester and 15 server tasks using queues on all 15 pagesets with the application queues defined.

The requester tasks specify NO_SYNCPOINT for all messages and the server tasks get and put messages within syncpoint.

The measurements are run on a single LPAR with 16 dedicated processors online on the z13 (2964) used for testing.

The measurements are run using 2KB, 64KB and 4MB messages. The 4MB message measurement uses a maximum of 6 sets of queues and tasks.

Chart: Transaction rate for non-persistent in syncpoint scalability workload with 2KB messages

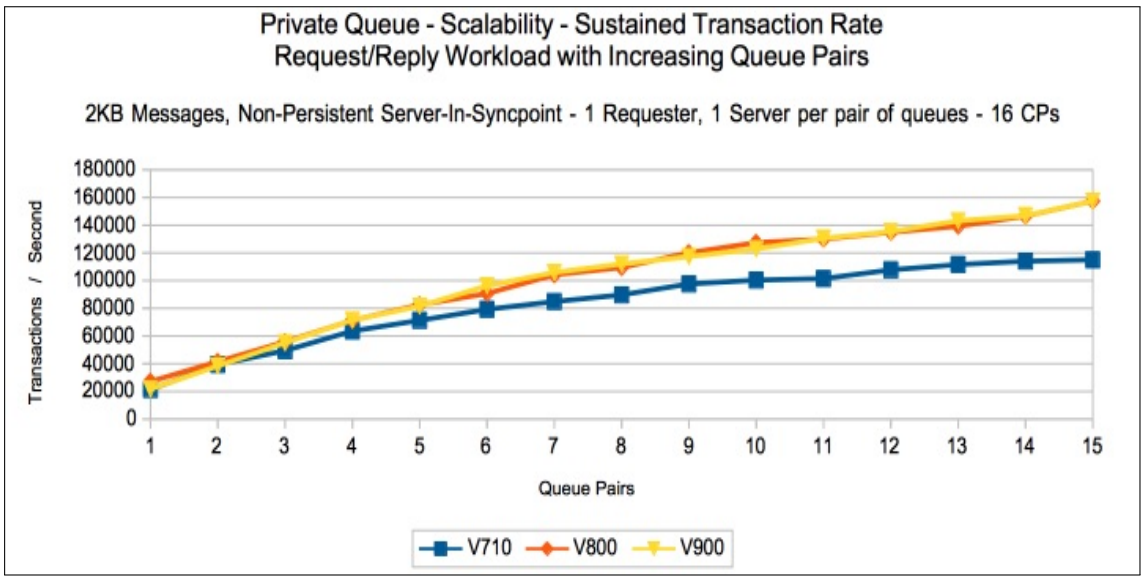


Chart: Transaction cost for non-persistent in syncpoint scalability workload with 2KB messages

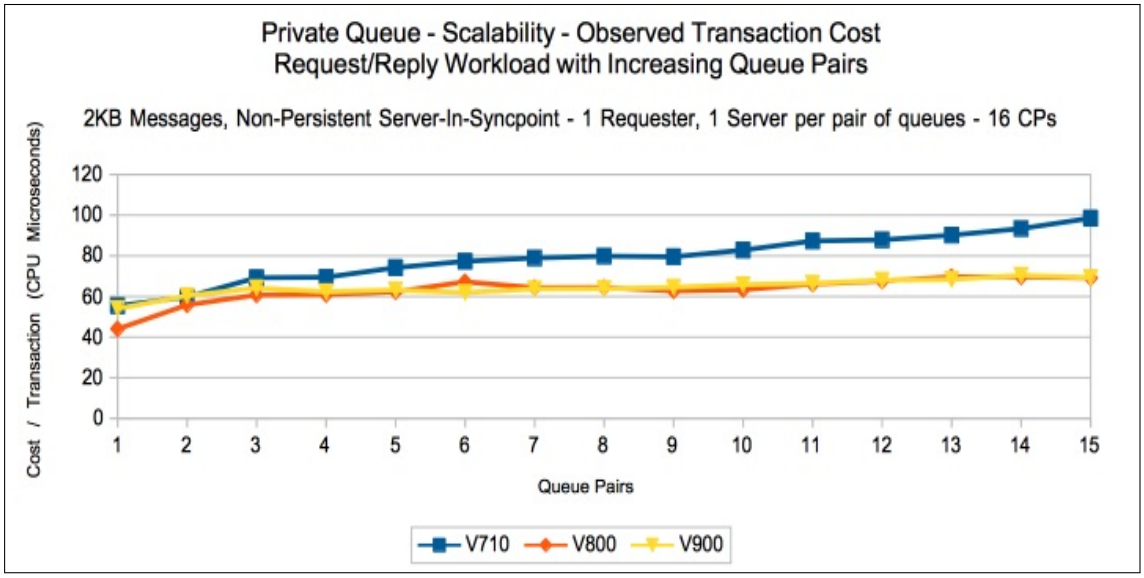


Chart: Transaction rate for non-persistent in syncpoint scalability workload with 64KB messages

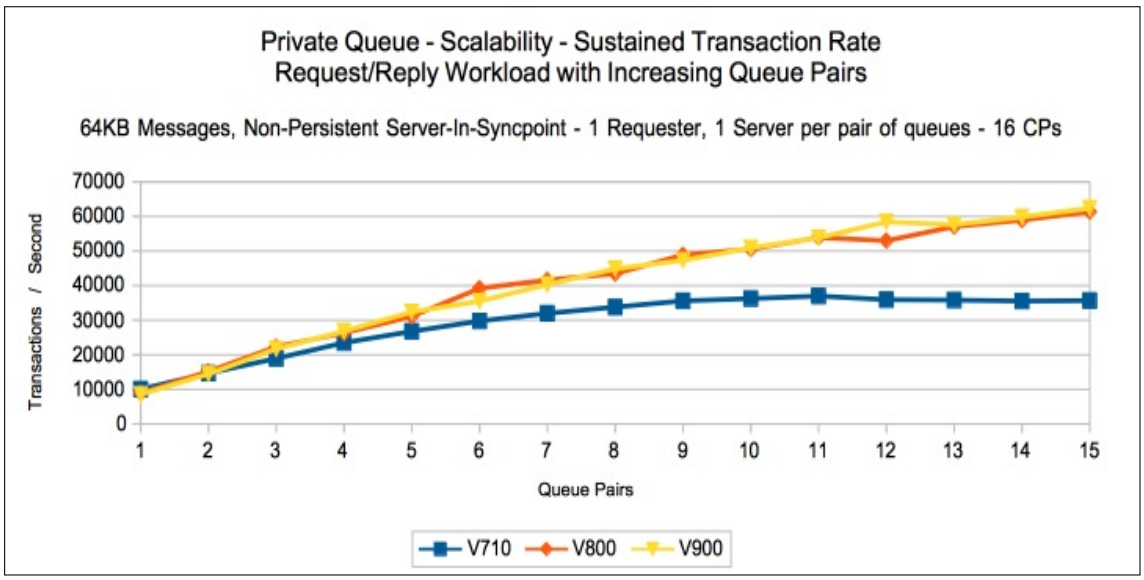


Chart: Transaction cost for non-persistent in syncpoint scalability workload with 64KB messages

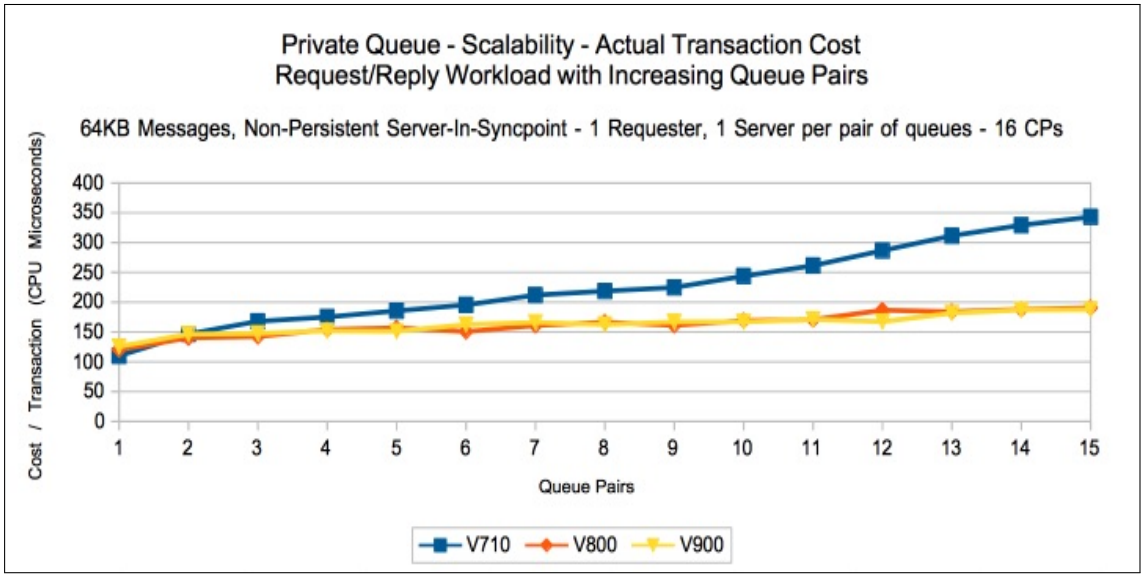


Chart: Transaction rate for non-persistent in syncpoint scalability workload with 4MB messages

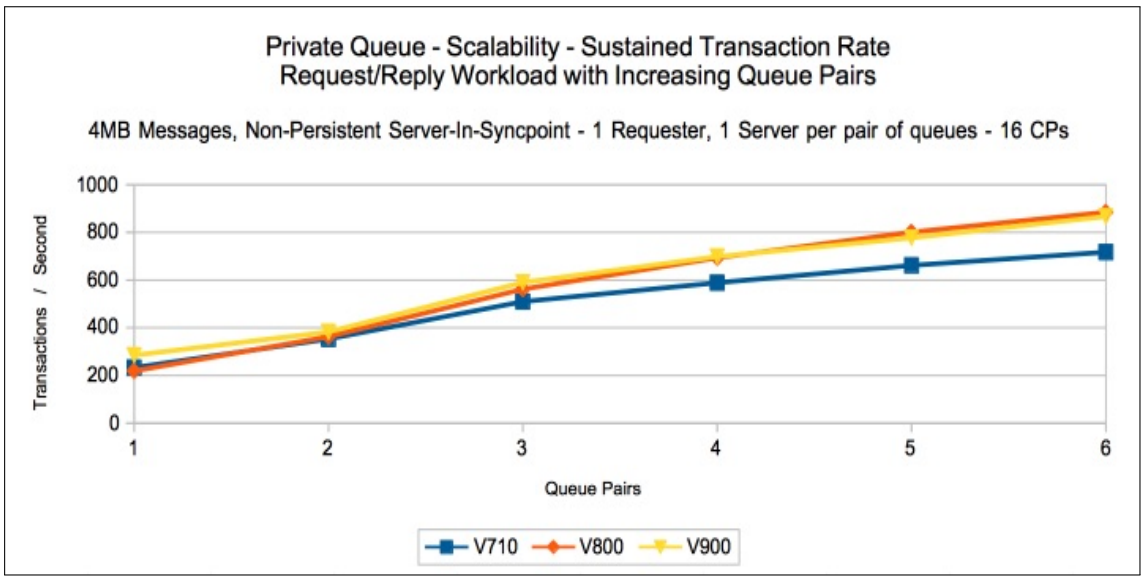
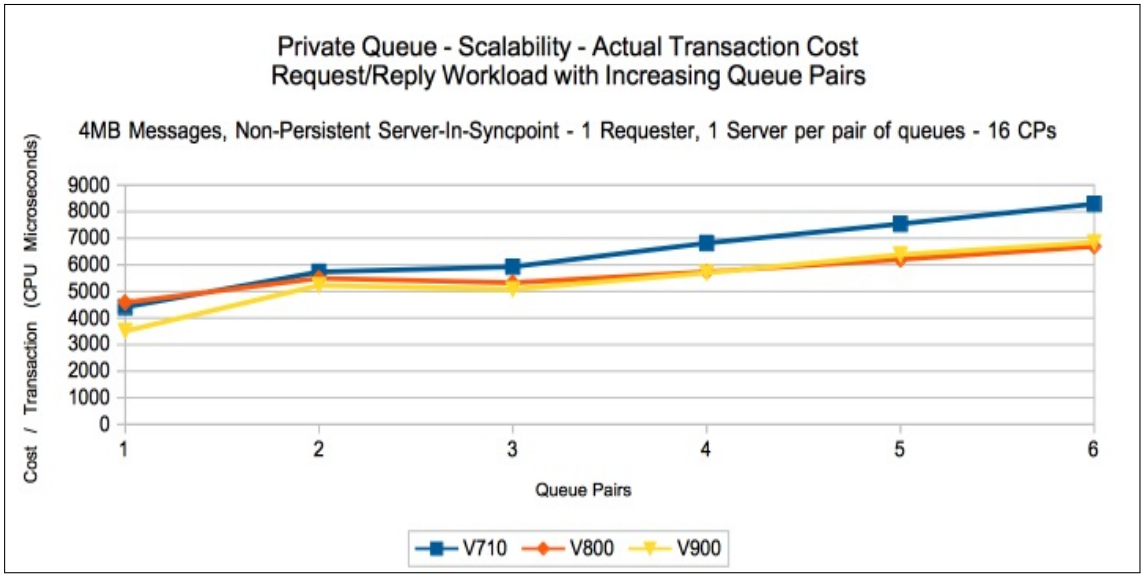


Chart: Transaction cost for non-persistent in syncpoint scalability workload with 4MB messages



Persistent server in-syncpoint workload

Maximum throughput on a single pair of request/reply queues

The test uses 60 batch requester tasks that each put a message to a common request queue and wait for a specific response on the reply queue. Once they have gotten the message, they put another message to the request queue. The messages are put in-syncpoint and got in-syncpoint.

There are 10 batch server tasks that action MQGET-with-wait calls on the request queue, get the message and put a reply to the known (and pre-opened) reply queue and then the application goes back into the MQGET-with-wait call. The messages are got and put in syncpoint with 1 MQGET and 1 MQPUT per commit.

Chart: Transaction rate for persistent in syncpoint workload

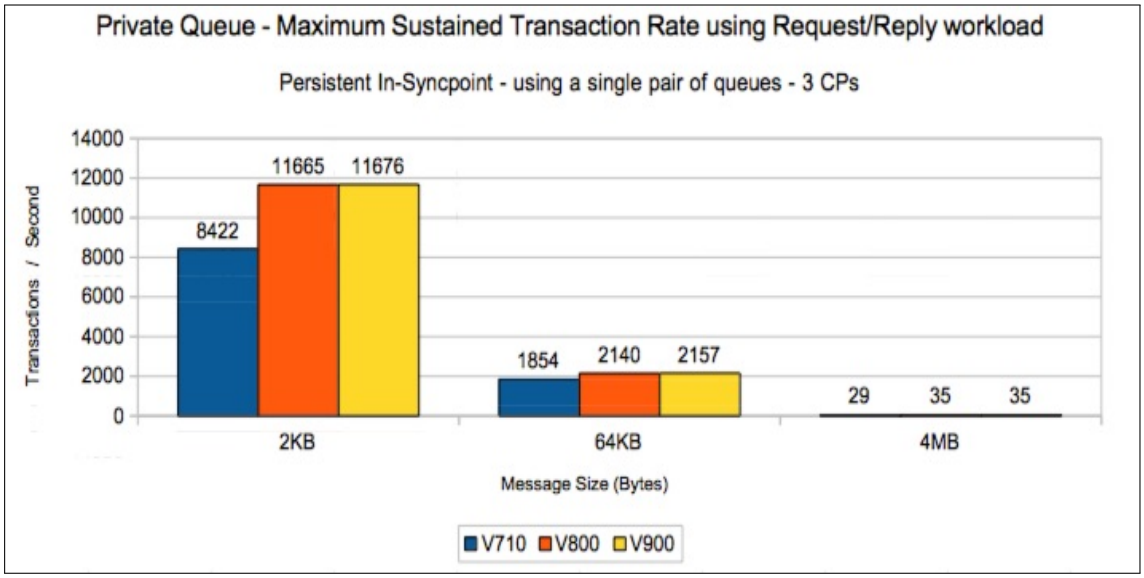
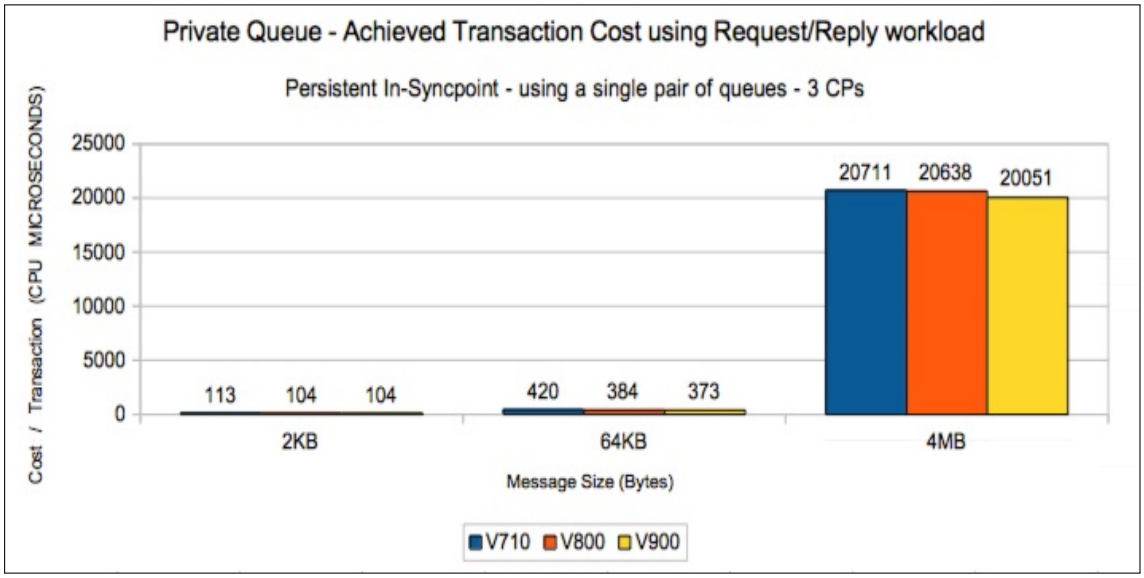


Chart: Transaction cost for persistent in syncpoint workload

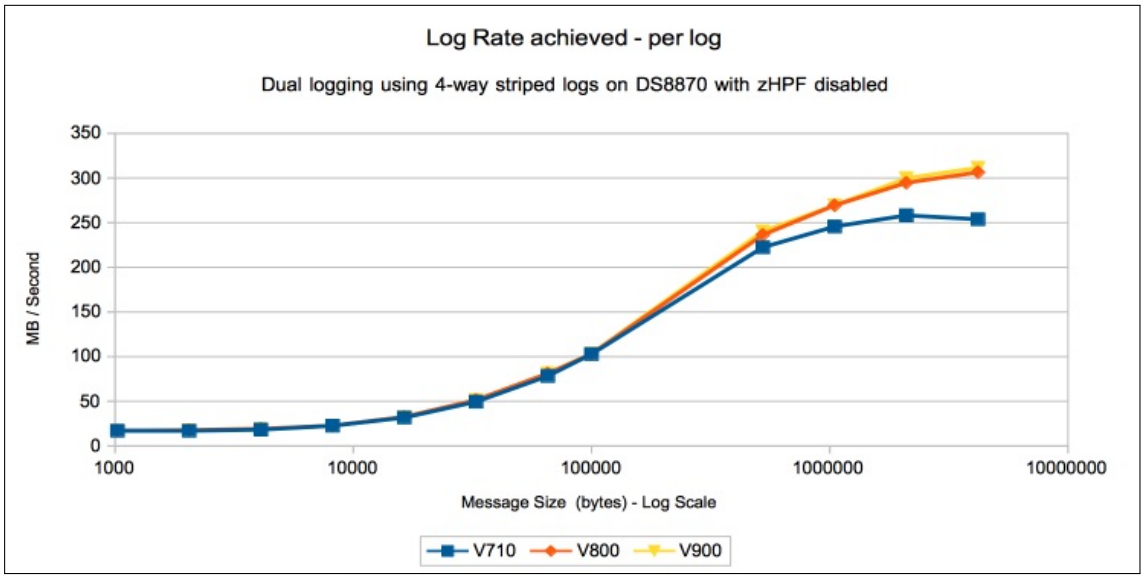


Upper bounds of persistent logging rate

This test uses 3 batch tasks that each put and get messages from a common queue. One of the tasks only uses a 1KB persistent message. The remaining 2 tasks vary the size of their message from 1KB to 4MB.

The following chart shows the achieved log rate on a 3-way LPAR of the z13.

Chart: Peak log rate achieved during persistent message workload



Notes:

The peak log rate in excess of 300MB per second is for each copy of the dual logs, i.e. there is more than 600 MB of data being written to IBM MQ log data sets per second.

These logs are configured with 4 stripes.

CICS Workload

When a CICS transaction makes a call using an MQ API, the resulting processing performed at end of task can have a significant effect on the transaction cost. This cost is most noticeable when adding the first MQ call e.g. MQPUT1 to the application.

In version 8.0.0, the processing performed by the queue manager at end of CICS task was optimised for scalability purposes with intention of reducing the impact of adding the first MQ call.

In previous releases on high N-way machines with high volume CICS transactions, the end of task processing would elongate as the workload increased. This was due to more storage being allocated for CICS tasks, and subsequently taking more time to scan that storage. In the worst case scenarios, the queue manager storage could rapidly be used, resulting in a queue manager failure. One solution was to limit the number of CICS transactions running at any point by configuring the MXT parameter.

From IBM MQ for z/OS version 8.0.0, the storage is more efficiently re-used which reduces the CPU cost per transaction and increases the throughput capacity.

The following charts use a set of simple CICS transactions that open queues, put a 2KB non-persistent message, get a 2KB non-persistent message, close queues and initiate a new CICS transaction. Running multiple sets of these transactions simulates a number of concurrent transactions across multiple CICS regions.

In each case the system is running with 16 dedicated processors and becomes CPU constrained with 512 concurrent CICS tasks.

Notes on charts:

- The scale used on each chart is consistent.
- For version 7.1.0, the queue manager cost associated with the CICS workload is approximately 23 CPU microseconds and this rises as the workload increases.
- For versions 8.0.0 and 9.0.0, the queue manager cost associated with the CICS workload is 7 CPU microseconds and rises at a similar rate as the workload increases.
- There is also an associated saving in the transaction cost, which means that in a CPU constrained system, the upper bound on transaction rate is raised by nearly 50% in these measurements.
- As the V800 and V900 data is similar, only V900 data is shown.

Chart: V710 CICS workload

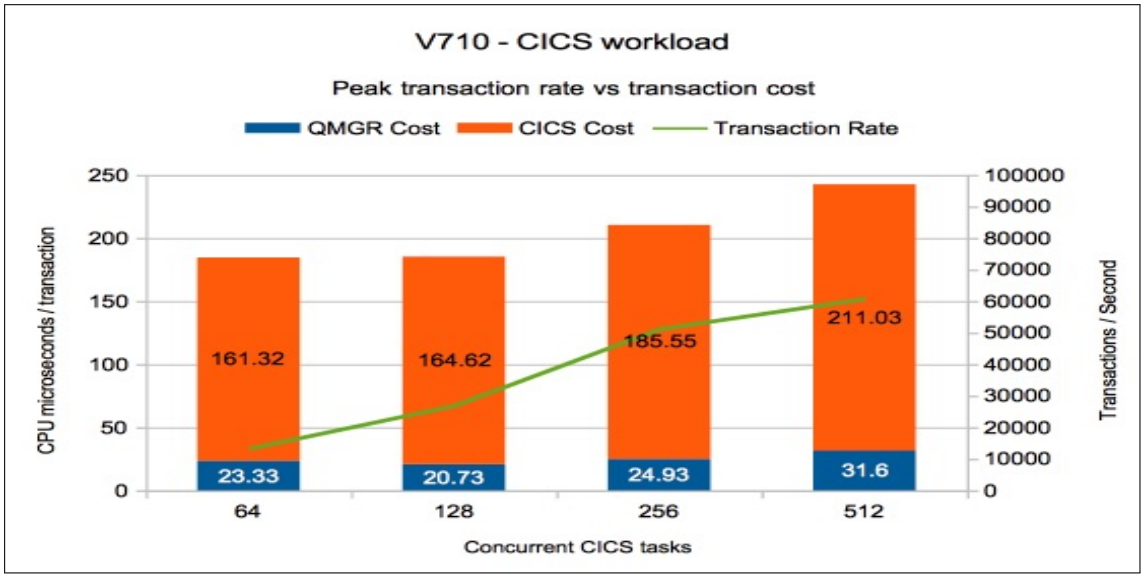
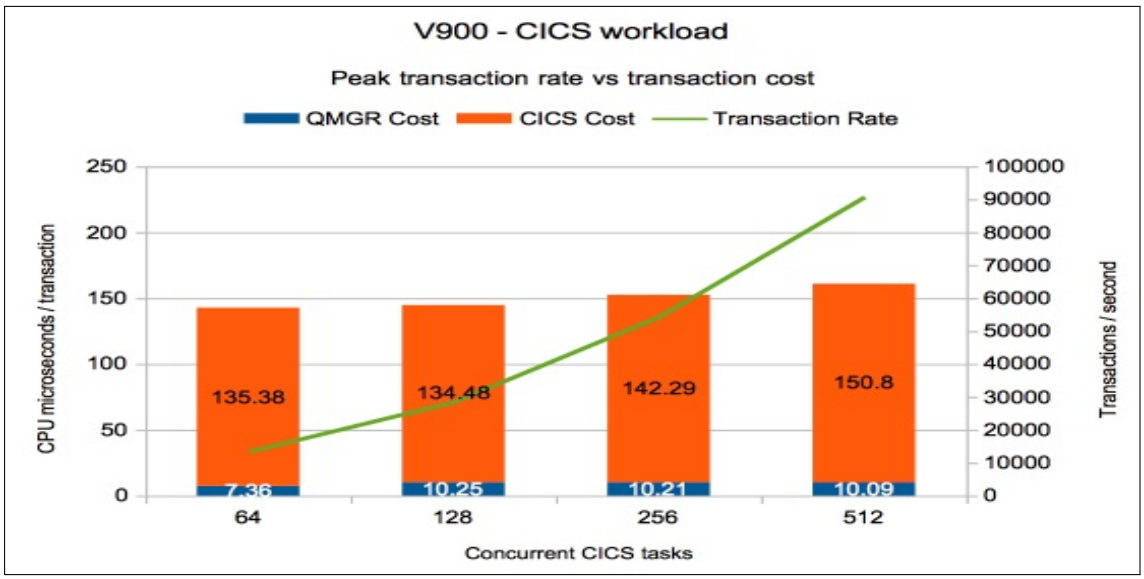


Chart: V900 CICS workload



Shared Queue

Version 7.1.0 introduced CFLEVEL(5) with Shared Message Data Sets (SMDS) as an alternative to DB2 as a way to store large shared messages. This resulted in significant performance benefits with both larger messages (greater than 63KB) as well as smaller messages when the Coupling Facility approached its capacity by using tiered thresholds for offloading data. For more details on the performance of CFLEVEL(5), please refer to performance report MP1H "WebSphere MQ for z/OS V7.1 Performance Report".

CFLEVEL(5) with SMDS with the default offload thresholds has been used for the shared queue measurements.

Non-persistent out-of-syncpoint workload

Maximum throughput on a single pair of request/reply queues

The test uses 5 batch requester tasks that each put a message to a common request queue and wait for a specific response on the reply queue. Once they have gotten the message, they put another message to the request queue. The messages are put and got out of syncpoint.

There are 4 batch server tasks that action MQGET-with-wait calls on the request queue, get the message and put a reply to the known (and pre-opened) reply queue and then the application goes back into the MQGET-with-wait call. The messages are put and got out of syncpoint.

An increasing number of queue managers are allocated to process the workload. Each queue manager has 5 requester tasks and 4 server tasks.

Chart: Transaction rate for non-persistent out-of-syncpoint workload - 2KB messages.

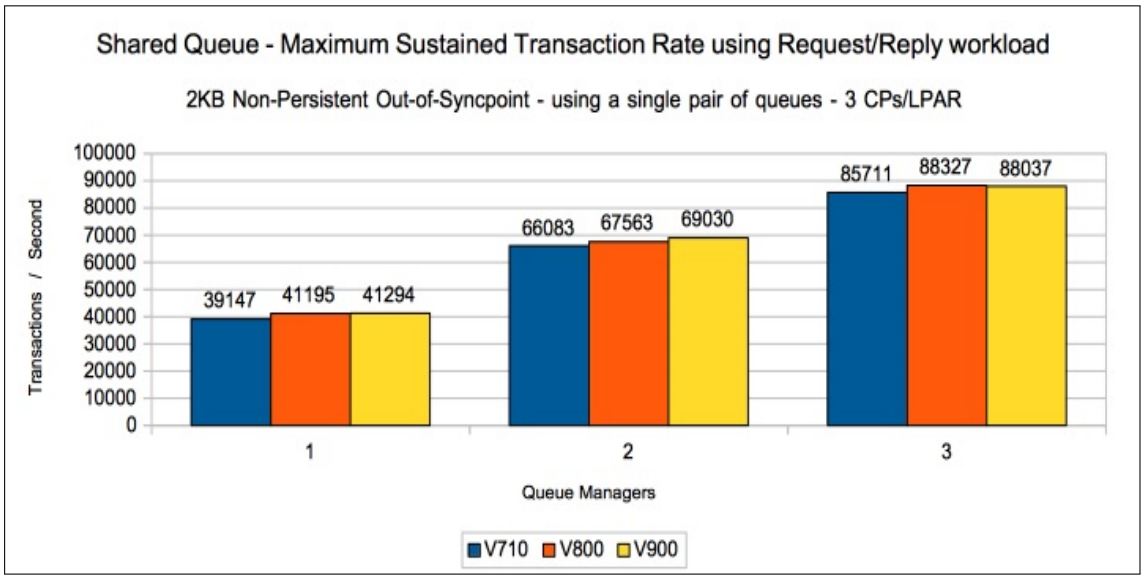


Chart: Transaction cost for non-persistent out-of-syncpoint workload - 2KB messages.

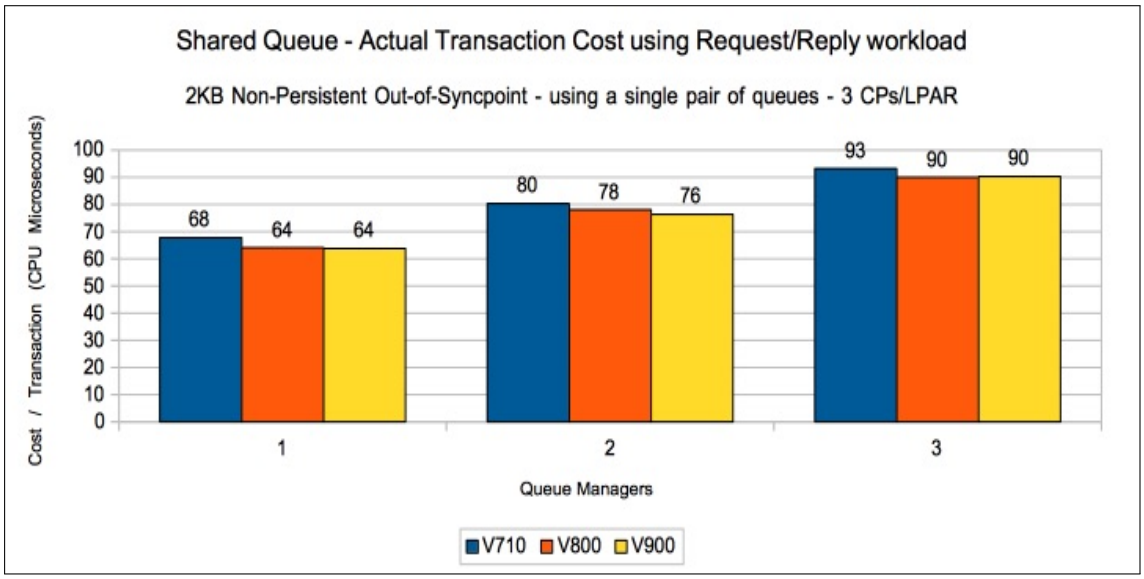


Chart: Transaction rate for non-persistent out-of-syncpoint workload - 64KB messages.

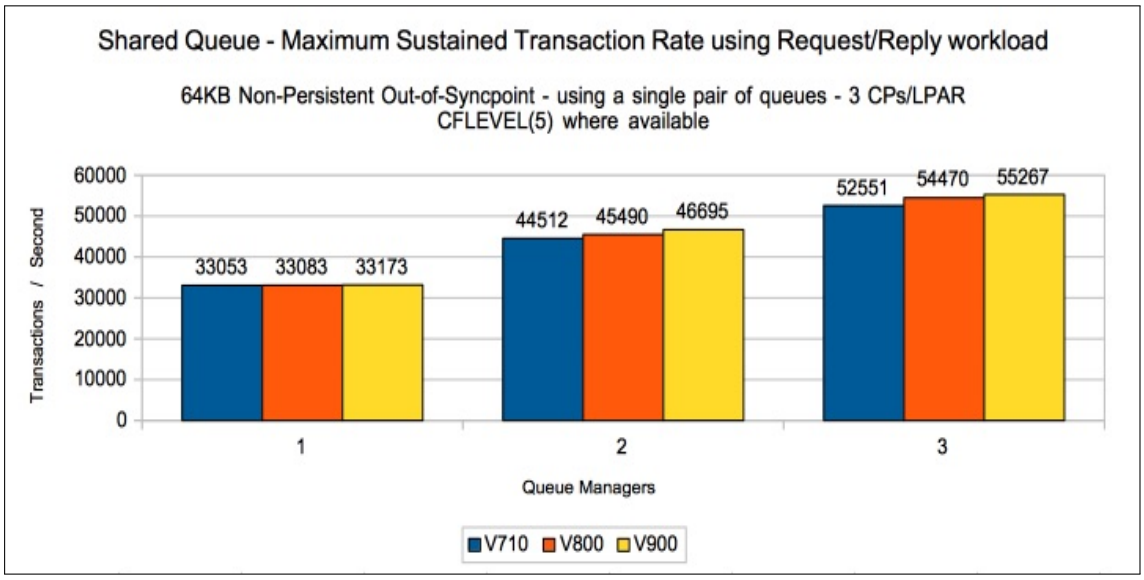


Chart: Transaction cost for non-persistent out-of-syncpoint workload - 64KB messages.

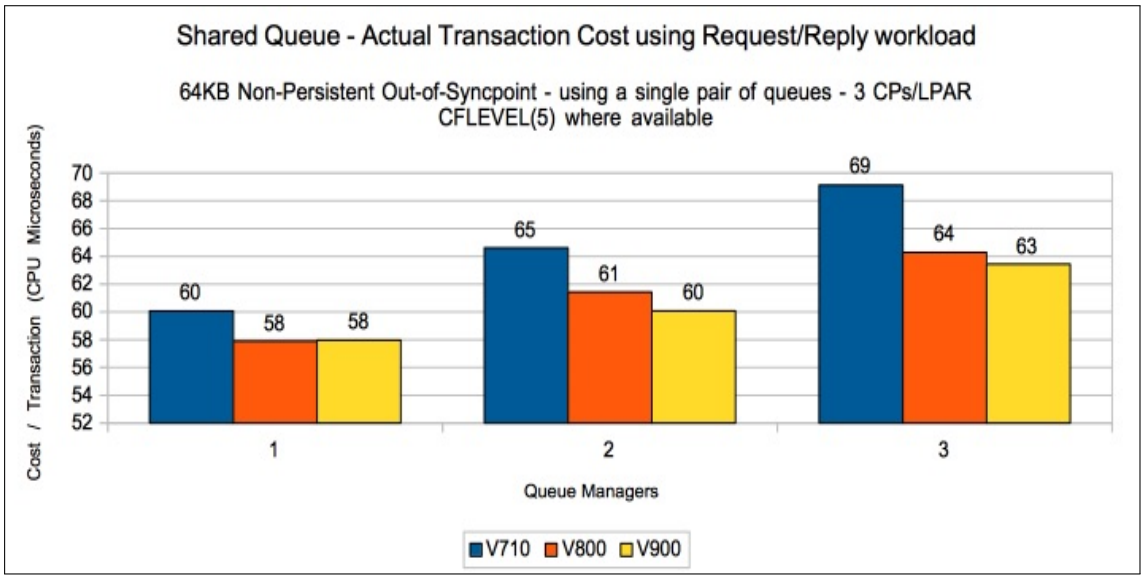


Chart: Transaction rate for non-persistent out-of-syncpoint workload - 4MB messages.

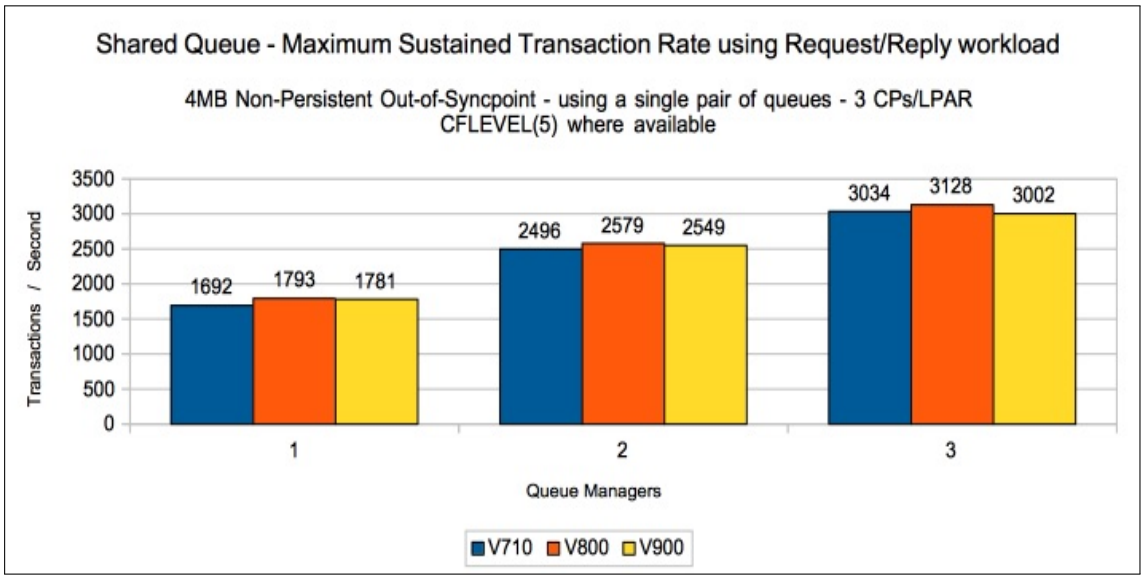
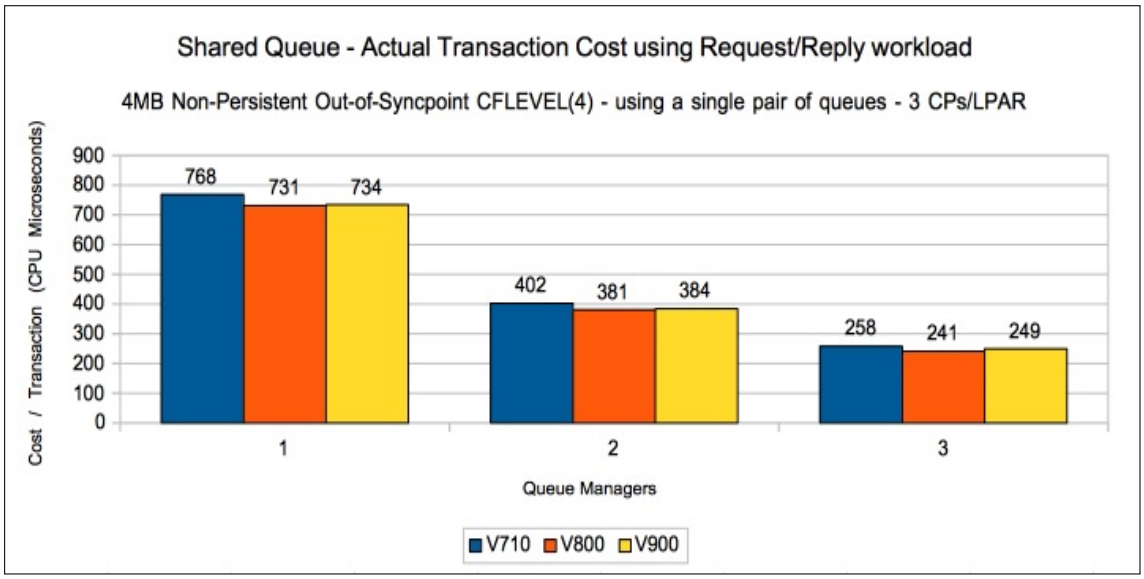


Chart: Transaction cost for non-persistent out-of-syncpoint workload - 4MB messages.



Non-persistent server in-syncpoint workload

Maximum throughput on a single pair of request/reply queues

The test uses 5 batch requester tasks that each put a message to a common request queue and wait for a specific response on the reply queue. Once they have gotten the message, they put another message to the request queue. The messages are put and got out of syncpoint.

There are 4 batch server tasks that action MQGET-with-wait calls on the request queue, get the message and put a reply to the known (and pre-opened) reply queue and then the application goes back into the MQGET-with-wait call. The messages are got and put in syncpoint with 1 MQGET and 1 MQPUT per commit.

An increasing number of queue managers are allocated to process the workload. Each queue manager has 5 requester and 4 server tasks.

Chart: Transaction rate for non-persistent in syncpoint workload - 2KB

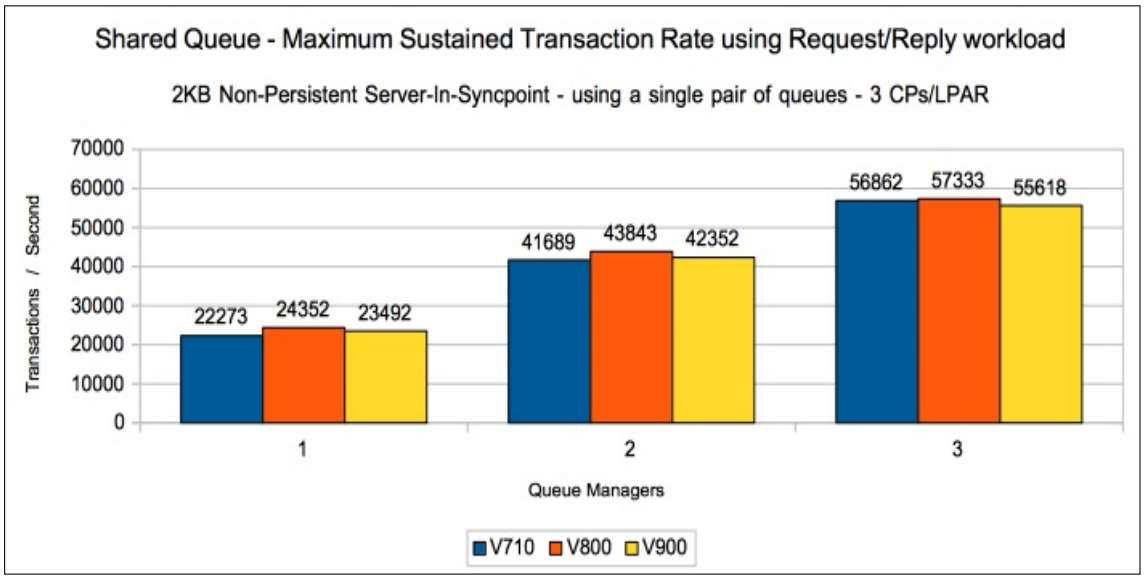


Chart: Transaction cost for non-persistent in syncpoint workload - 2KB

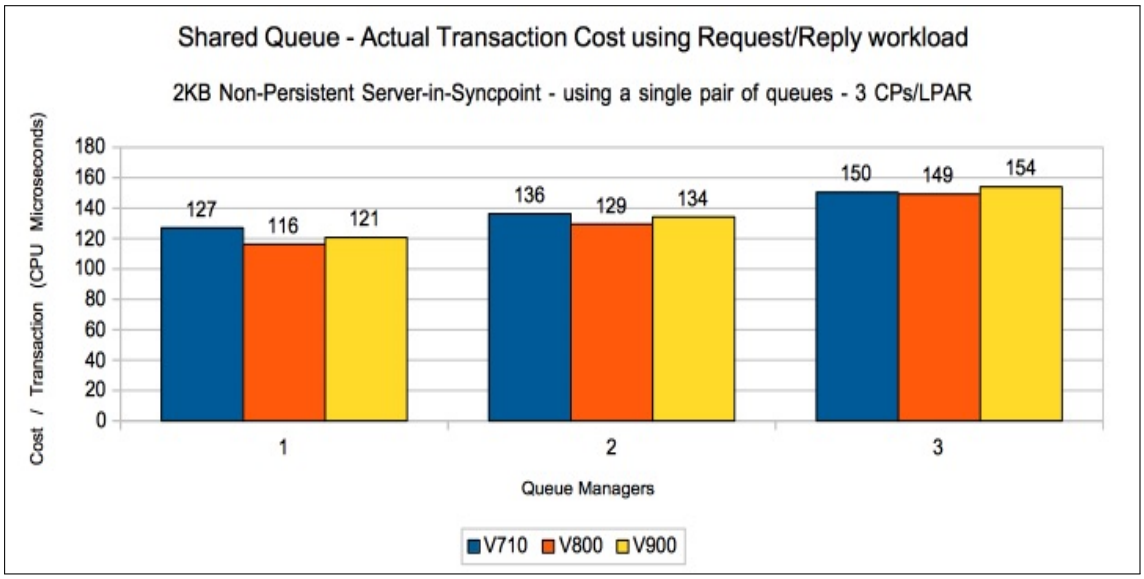


Chart: Transaction rate for non-persistent in syncpoint workload - 64KB

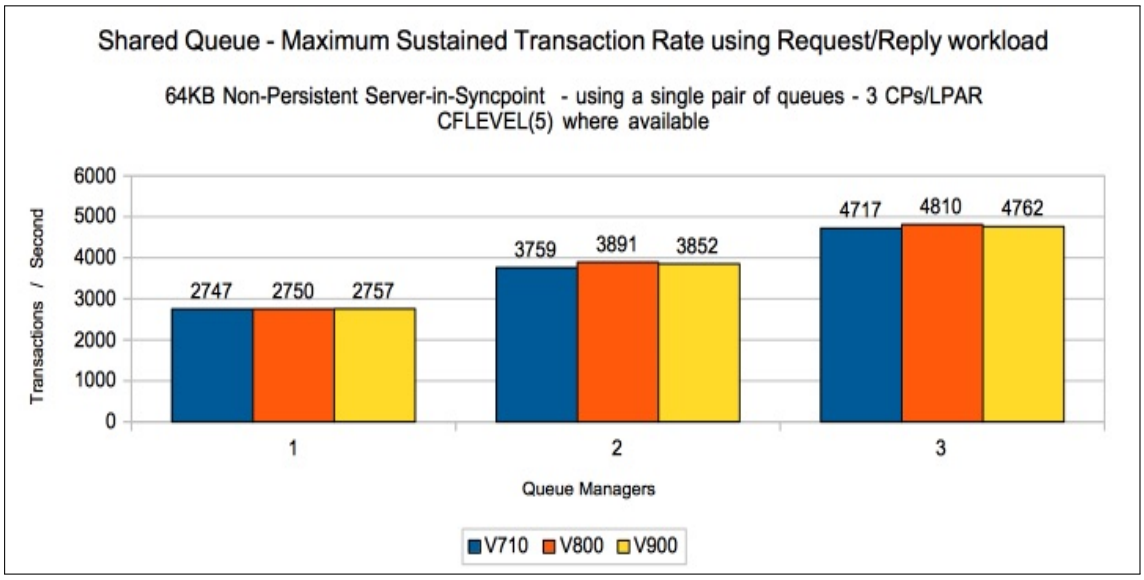


Chart: Transaction cost for non-persistent in syncpoint workload - 64KB

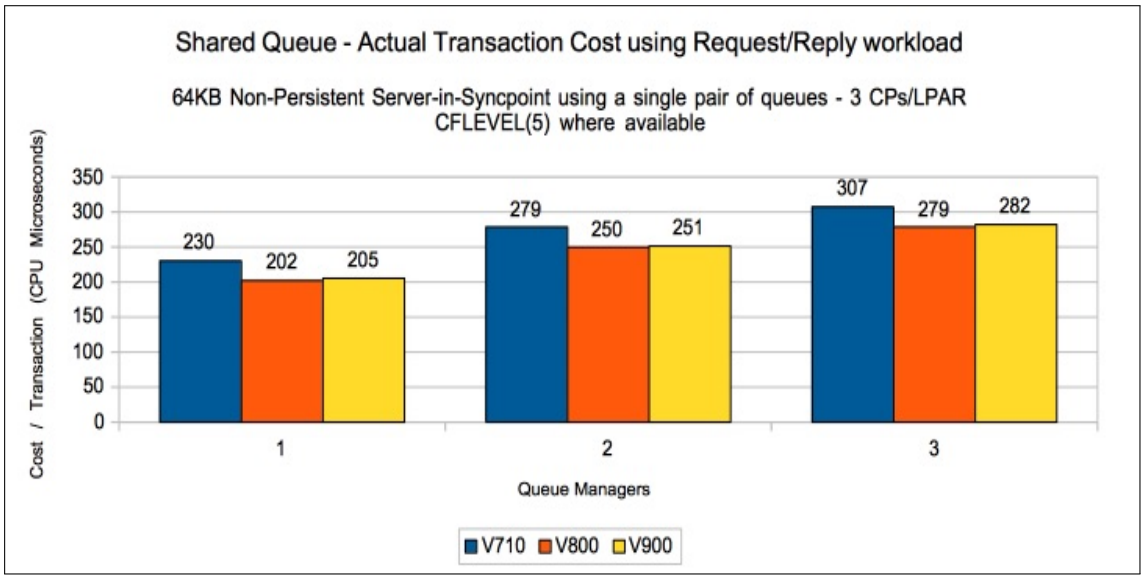


Chart: Transaction rate for non-persistent in syncpoint workload - 4MB

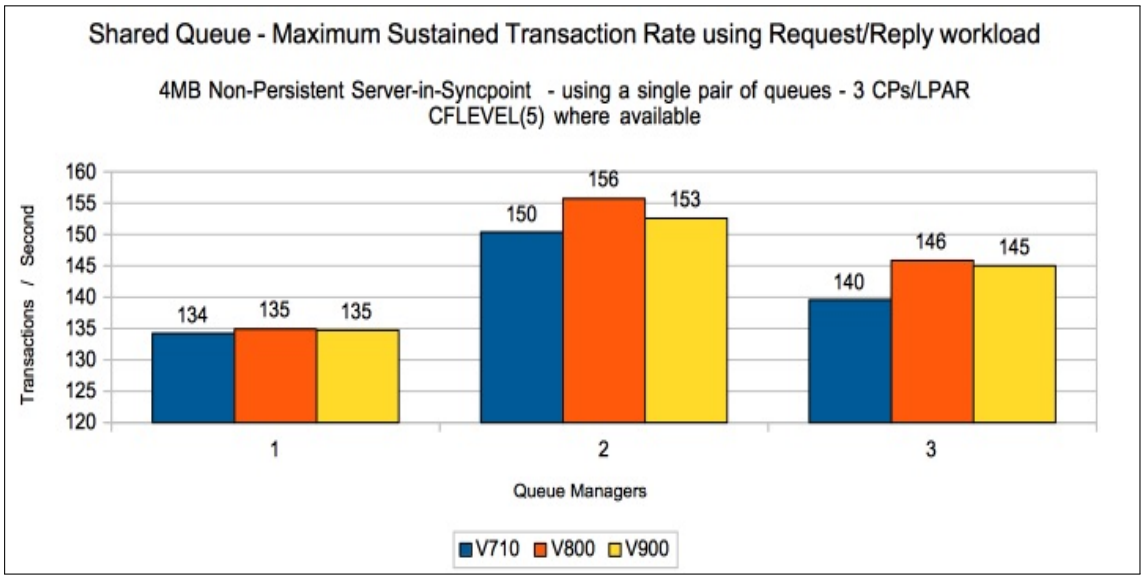
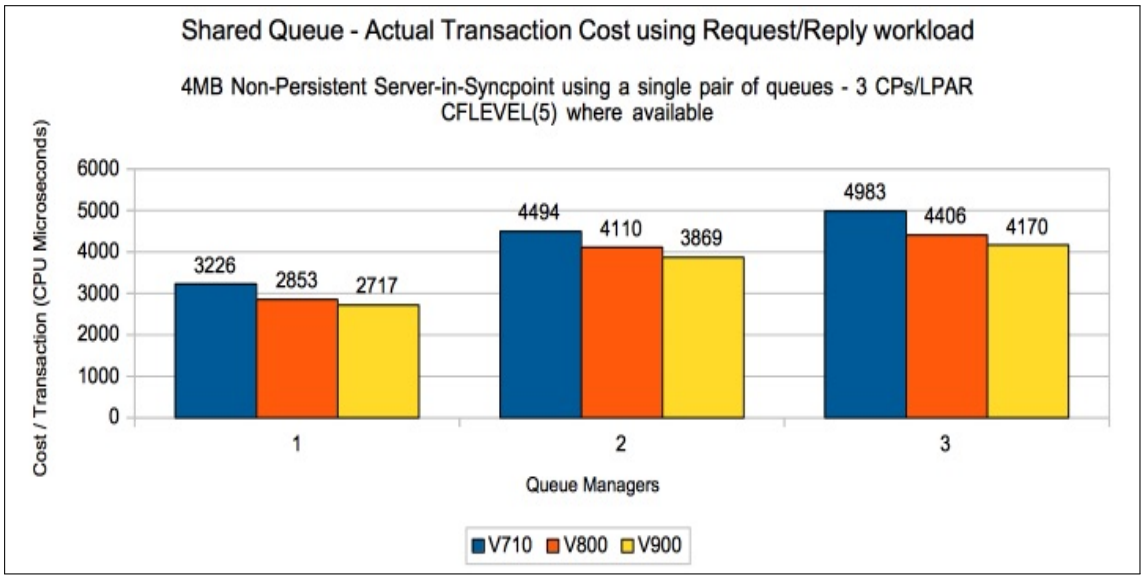


Chart: Transaction cost for non-persistent in syncpoint workload - 4MB



Data sharing non-persistent server in-syncpoint workload

The previous shared queue tests are configured such that any queue manager within the queue sharing group (QSG) can process messages put by any particular requester application. This means that the message may be processed by a server application on any of the available LPARs. Typically the message is processed by a server application on the same LPAR as the requester.

In the following tests, the message can only be processed by a server application on a separate LPAR. This is achieved by the use of multiple pairs of request/reply queues.

The test uses 5 batch requester tasks that each put a message to a common request queue and wait for a specific response on the reply queue. Once they have gotten the message, they put another message to the request queue. The messages are put and got out of syncpoint.

There are 4 batch server tasks that action MQGET-with-wait calls on the request queue, get the message and put a reply to the known (and pre-opened) reply queue and then the application goes back into the MQGET-with-wait call. The messages are got and put in syncpoint with 1 MQGET and 1 MQPUT per commit.

Chart: Transaction rate for data sharing non-persistent in syncpoint workload - 2KB

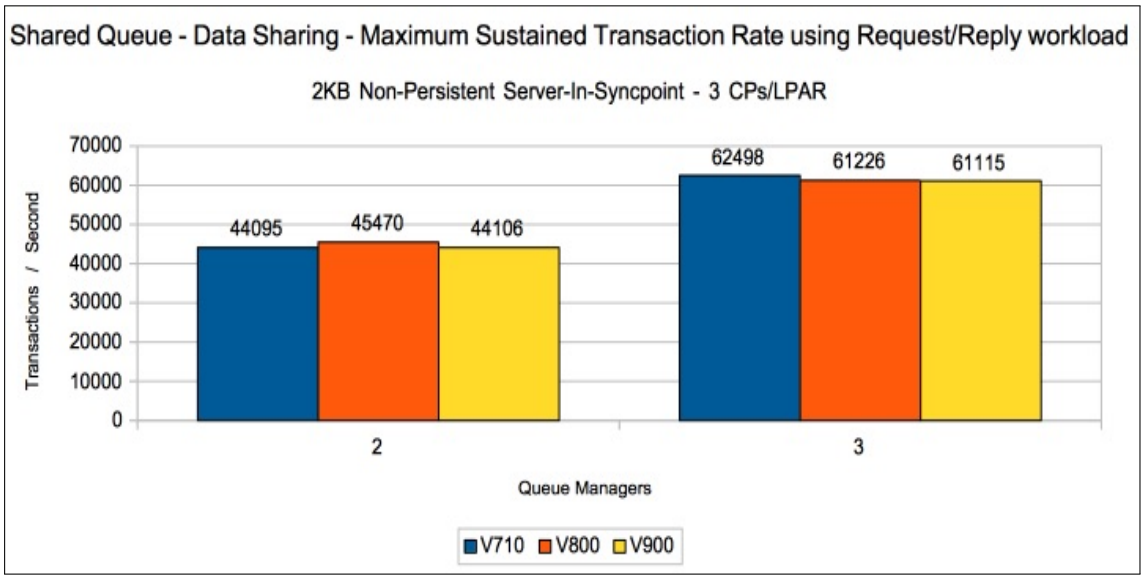


Chart: Transaction cost for data sharing non-persistent in syncpoint workload - 2KB

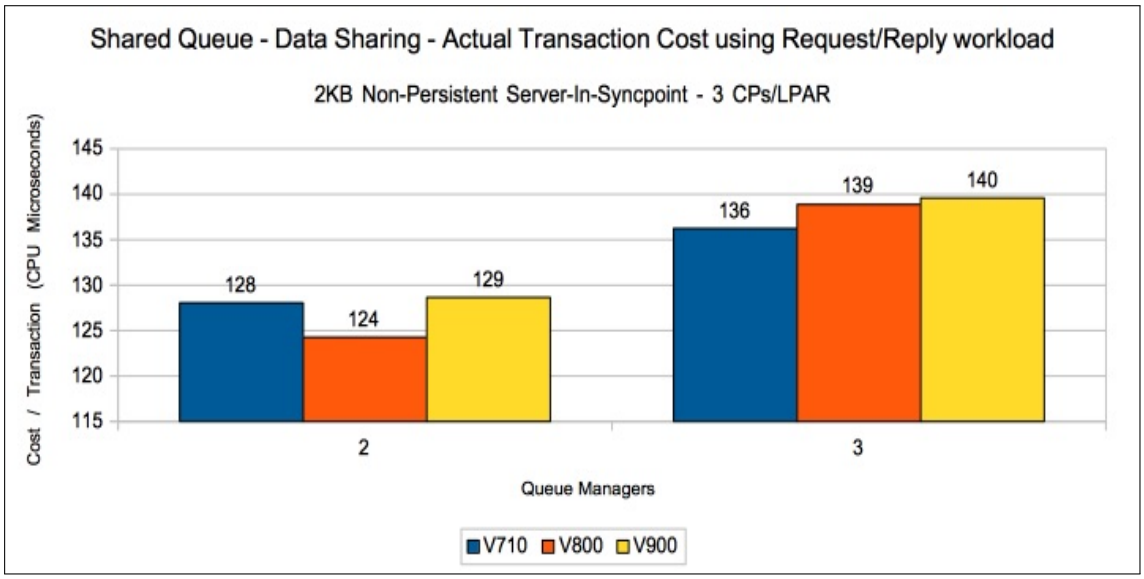


Chart: Transaction rate for data sharing non-persistent in syncpoint workload - 64KB

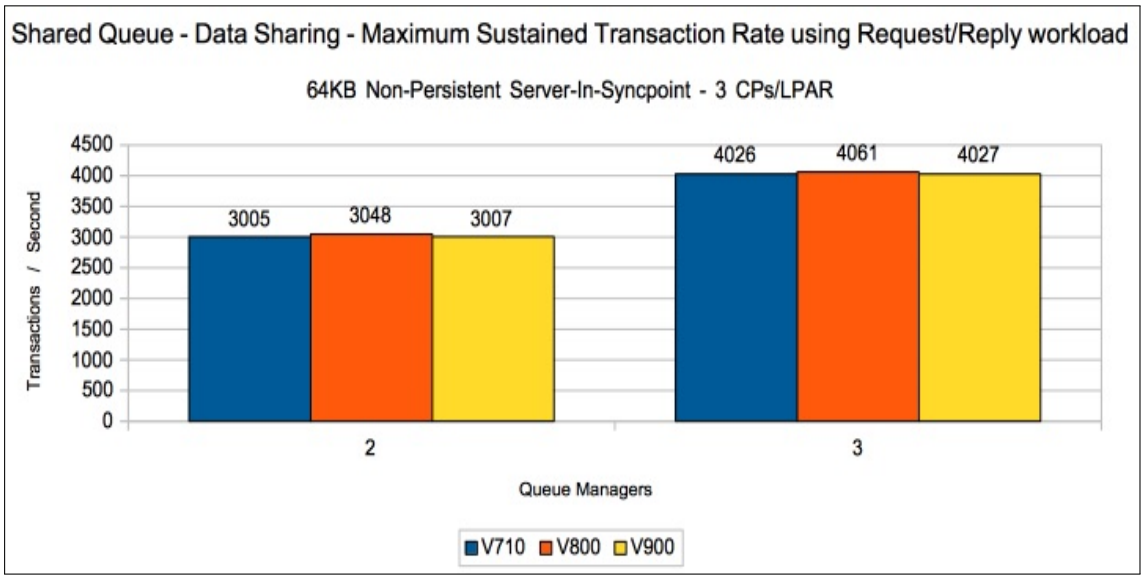
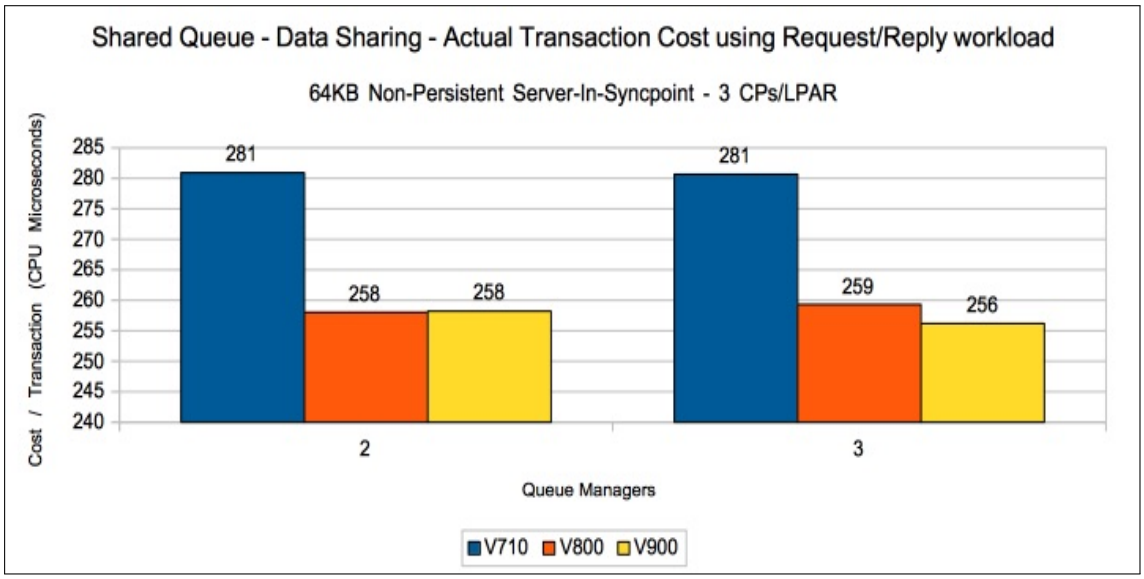


Chart: Transaction cost for data sharing non-persistent in syncpoint workload - 64KB



Moving messages across channels

The regression tests for moving message across channels e.g. sender-receiver channels, are designed to use drive the channel initiator such that the network is the constraining factor. Therefore the tests use non-persistent messages out-of-syncpoint, so that they are not constrained by logging etc.

Within the channel tests there are measurements with small numbers of channels (1 to 4 inbound plus the same number outbound), which was suitable for driving the network to capacity and also tests with up to 50 channels outbound and 50 channels inbound.

For each of the test types, the channels are used both with and without SSL encryption enabled.

The cipher spec "ECDHE_RSA_AES_256_CBC_SHA384" was used for all SSL tests as this is the most secure cipher spec available across the supported releases. The SSLRKEYC attribute was set so that 1MB of data can flow across the channel before renegotiating the key.

In IBM MQ version 8.0.0, the ZLIBFAST compression option was updated to be able to exploit zEDC hardware compression, which is discussed in detail in MP1J "IBM MQ for z/OS version 8.0.0 Performance Report". The compression measurements shown in this document use zEDC hardware compression where possible.

For further guidance on channel tuning and usage, please refer to performance report MP16 "Capacity Planning and Tuning Guide".

The measurements using 1 to 4 channels use batch applications to drive the workload at each end of the channel.

The measurements using 10 to 50 channels use long-lived CICS transactions to drive the workload. This means that each CICS application will put and get thousands of messages before ending. This model means that we are not including the cost of starting a CICS transaction, opening and closing queues and the teardown of the transaction at the end of the workload.

The compression tests use 64KB message of varying compressibility, so there is something to compress, e.g. a message of 64KB that is 80% compressible would reduce to approximately 13KB. These tests are run using 1 outbound and 1 inbound channel so include the compression and inflation costs for the request and reply message.

Non-persistent out-of-syncpoint - 1 to 4 sender receiver channels

Chart: Transaction rate with 2KB messages

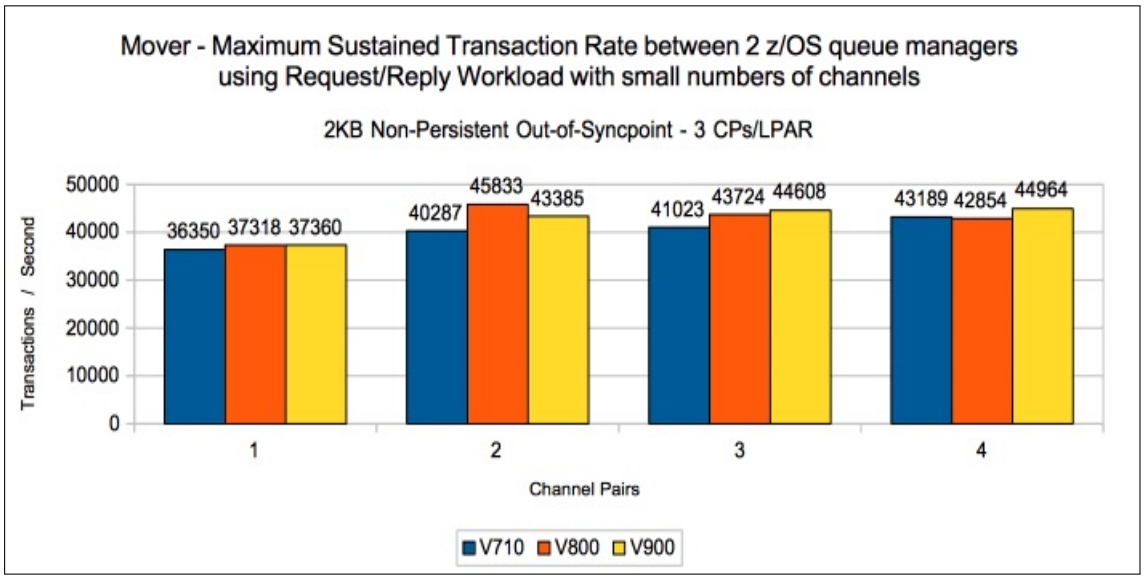


Chart: Transaction cost for 2KB messages

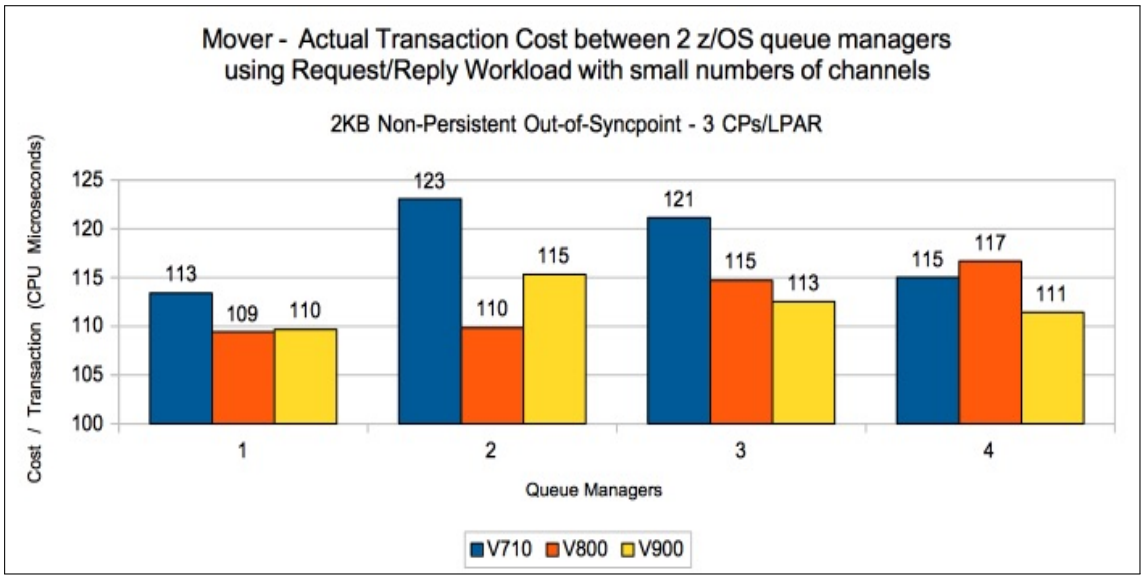


Chart: Transaction rate with 64KB messages

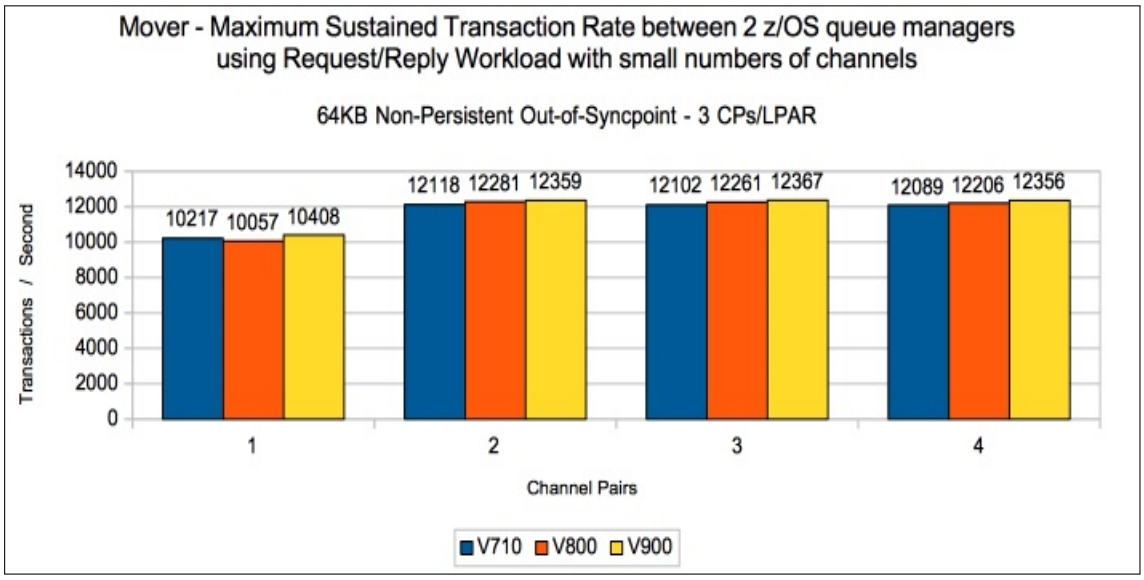


Chart: Transaction cost for 64KB messages

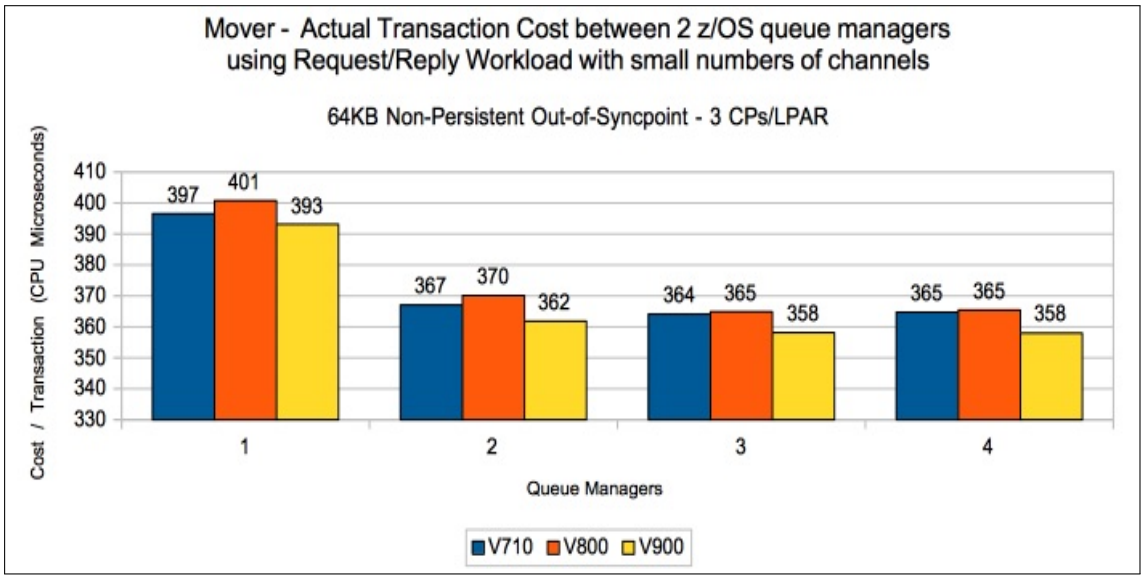


Chart: Transaction rate with 2KB messages over SSL channels

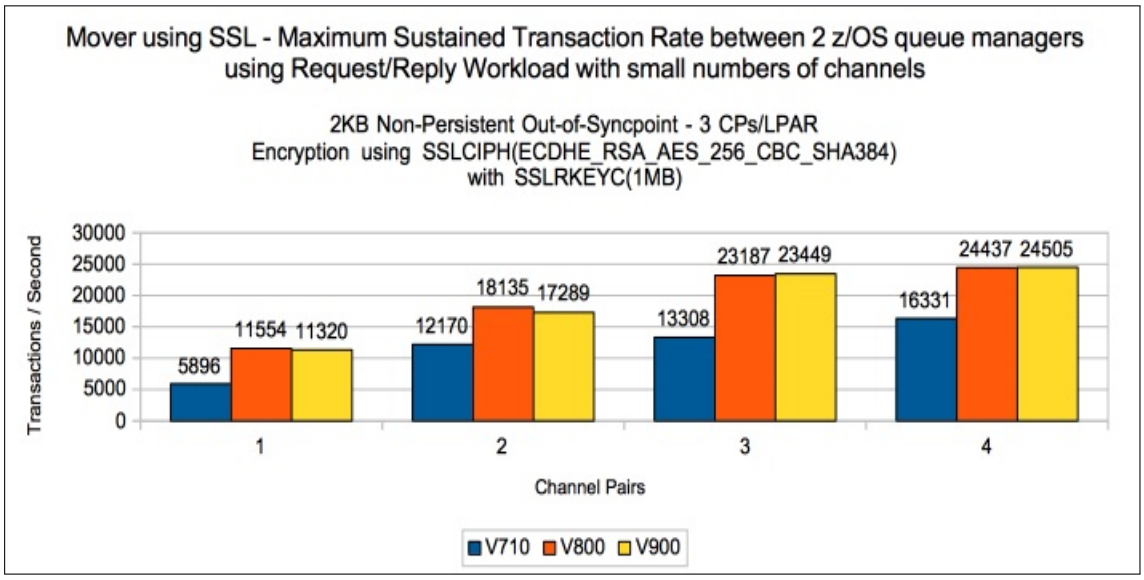


Chart: Transaction cost for 2KB messages over SSL channels

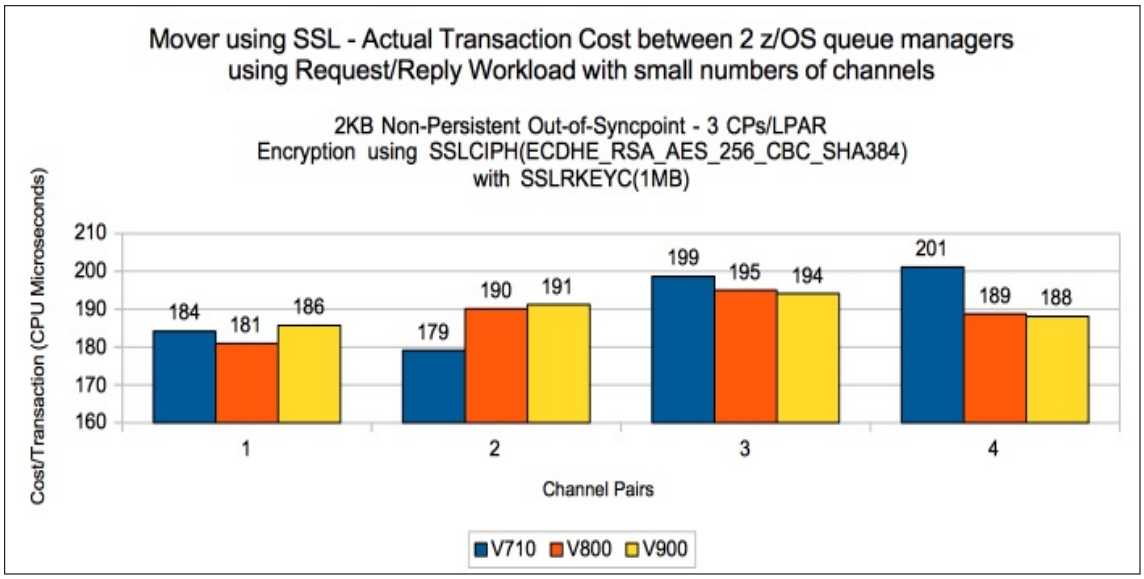


Chart: Transaction rate with 64KB messages over SSL channels

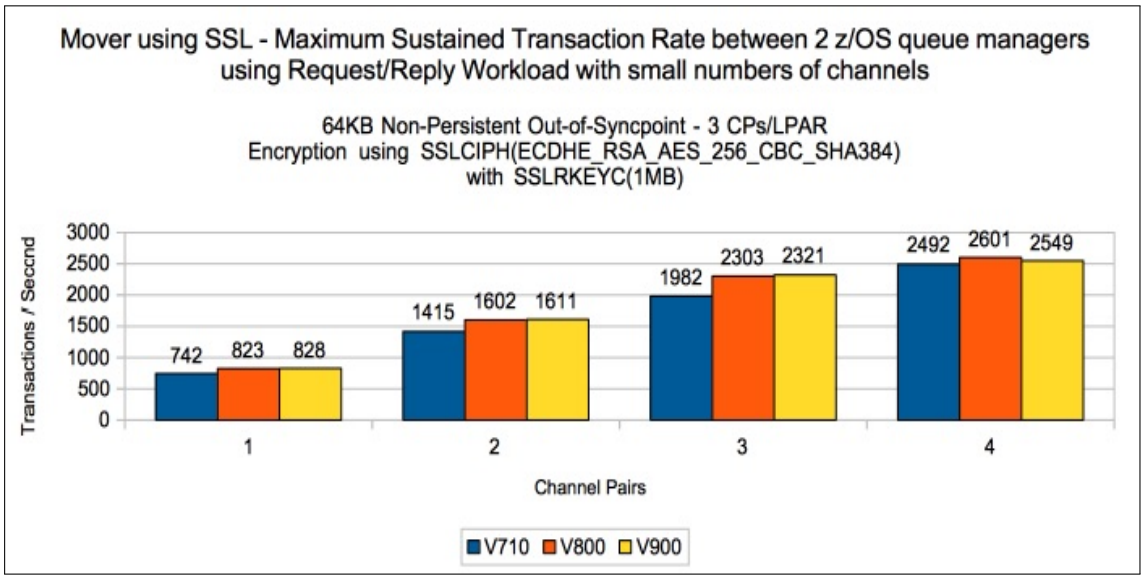
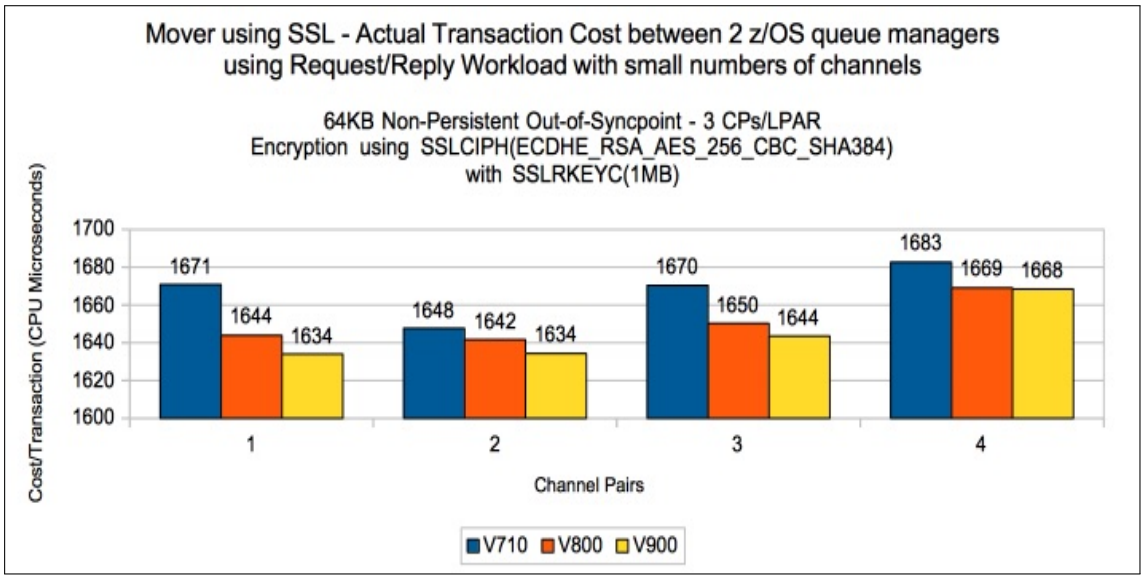


Chart: Transaction cost for 64KB messages over SSL channels



Non-persistent out-of-syncpoint - 10 to 50 sender receiver channels

Chart: Transaction rate with 2KB messages

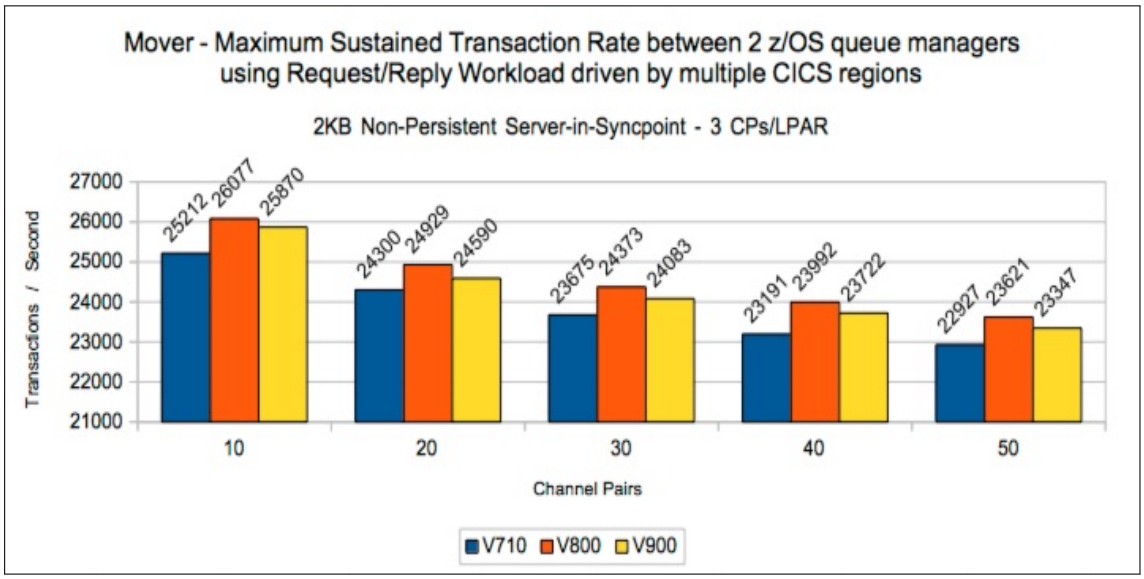


Chart: Transaction cost for 2KB messages

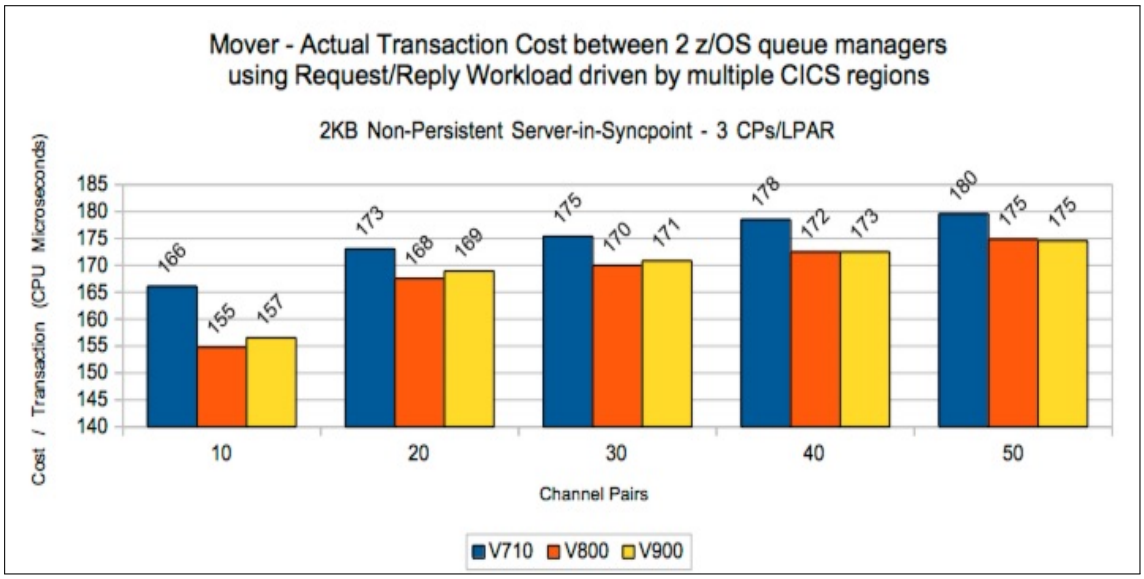


Chart: Transaction rate with 2KB messages over SSL channels

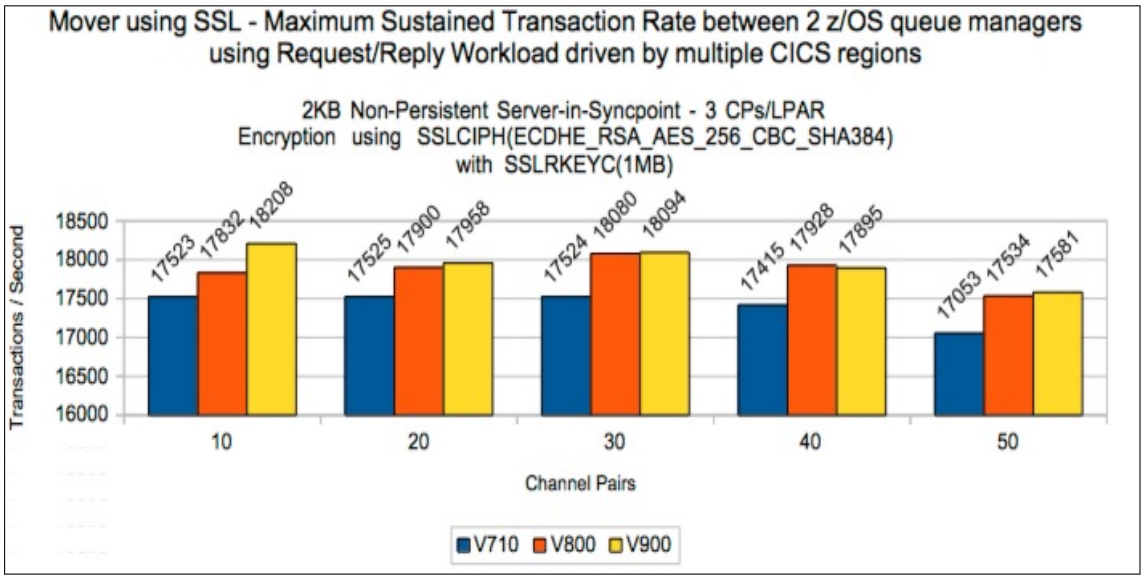
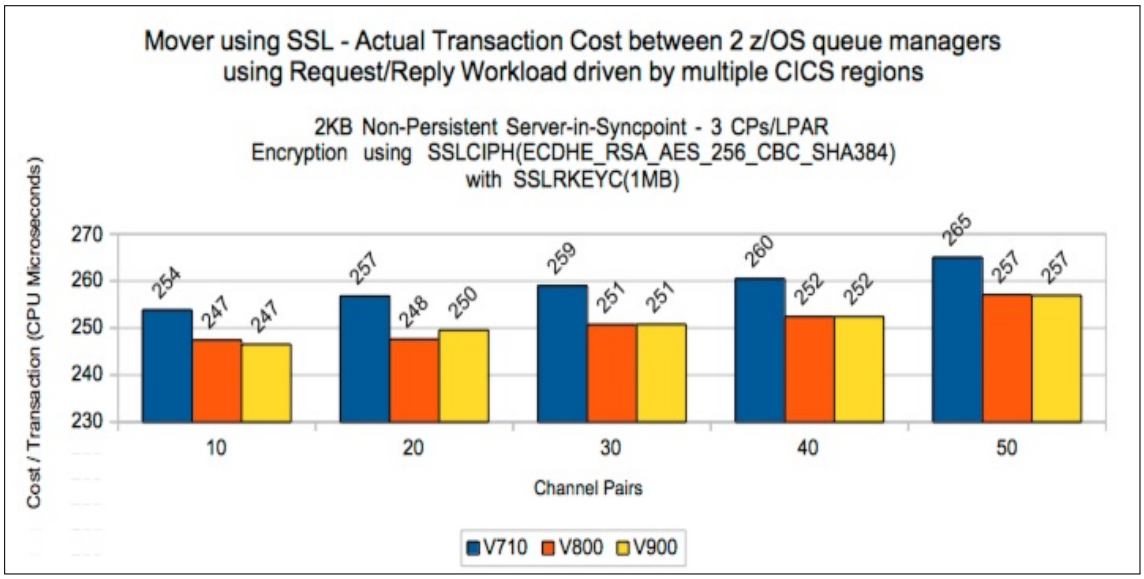


Chart: Transaction cost for 2KB messages over SSL channels



Channel compression using ZLIBFAST

Version 7.1.0 performs compression using software whereas both versions 8.0.0 and 9.0.0 are able to exploit the available zEDC hardware for compression.

Chart: Transaction rate with ZLIBFAST on 64KB messages

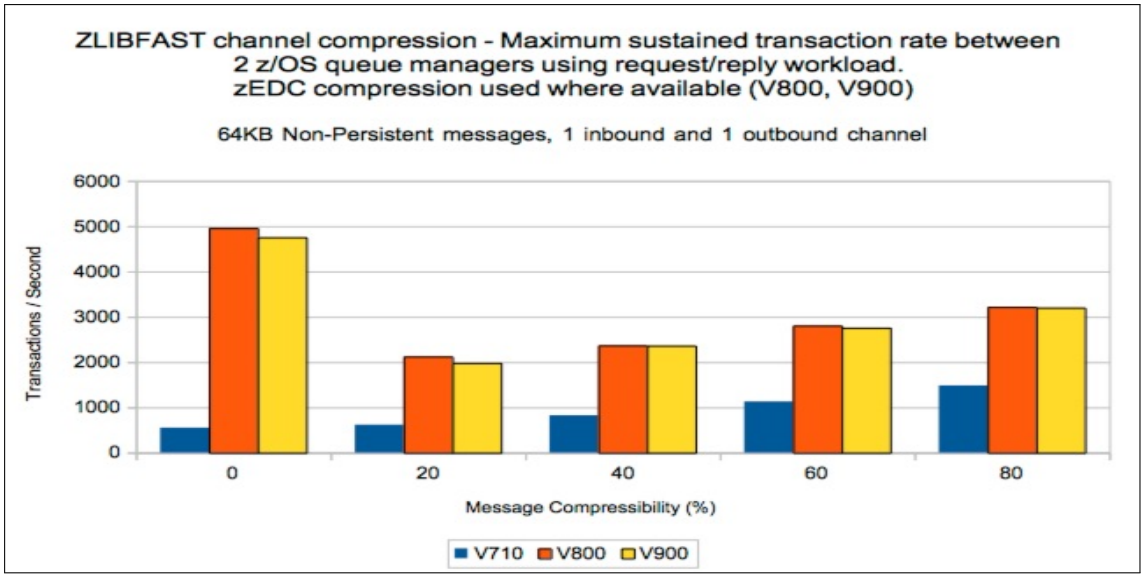
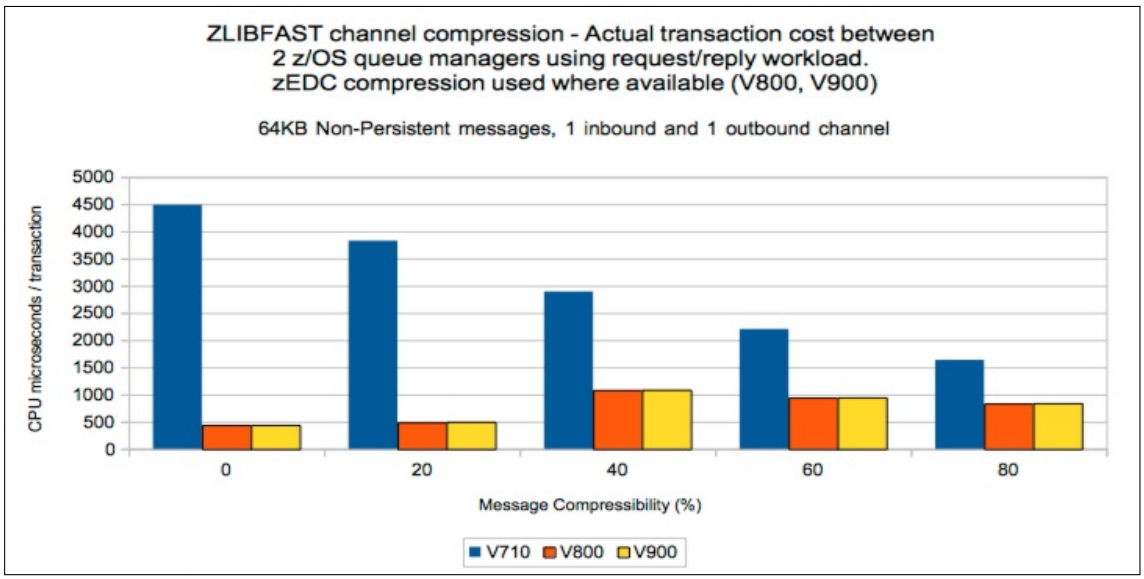


Chart: Transaction cost for ZLIBFAST on 64KB messages



Channel compression using ZLIBHIGH

For ZLIBHIGH we have seen between 5-15% reduction in cost in the channel initiator for versions 8.0.0 and 9.0.0 compared to version 7.1.0.

Chart: Transaction rate with 64KB messages with ZLIBHIGH

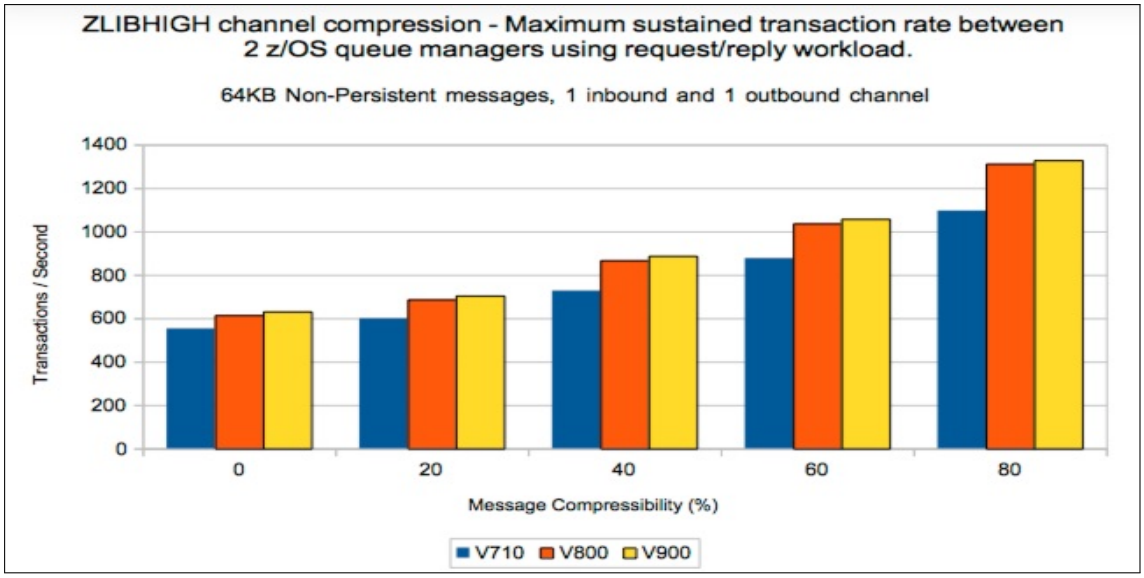
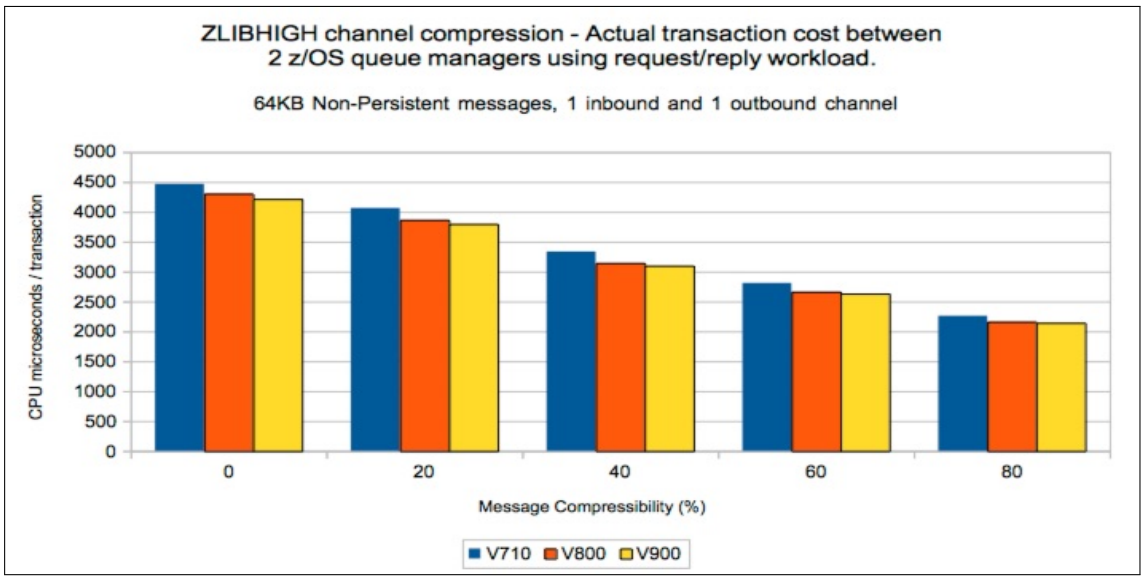


Chart: Transaction cost for 64KB messages with ZLIBHIGH



Channel compression using ZLIBFAST on SSL channels

Version 7.1.0 performs compression using software whereas both versions 8.0.0 and 9.0.0 are able to exploit the available zEDC hardware for compression.

Chart: Transaction rate with 64KB messages with ZLIBFAST on SSL channels

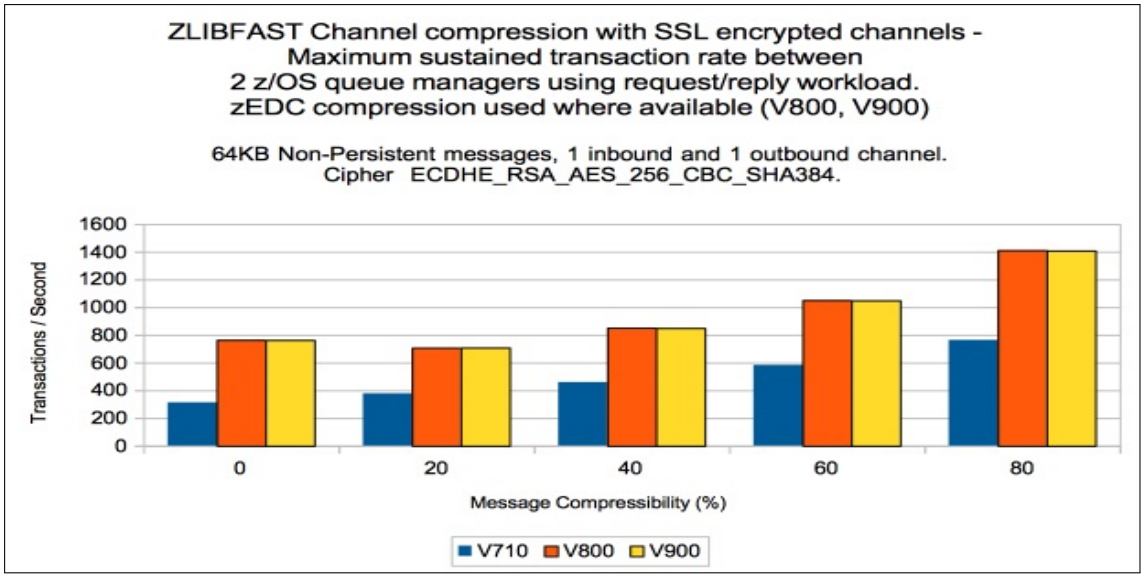
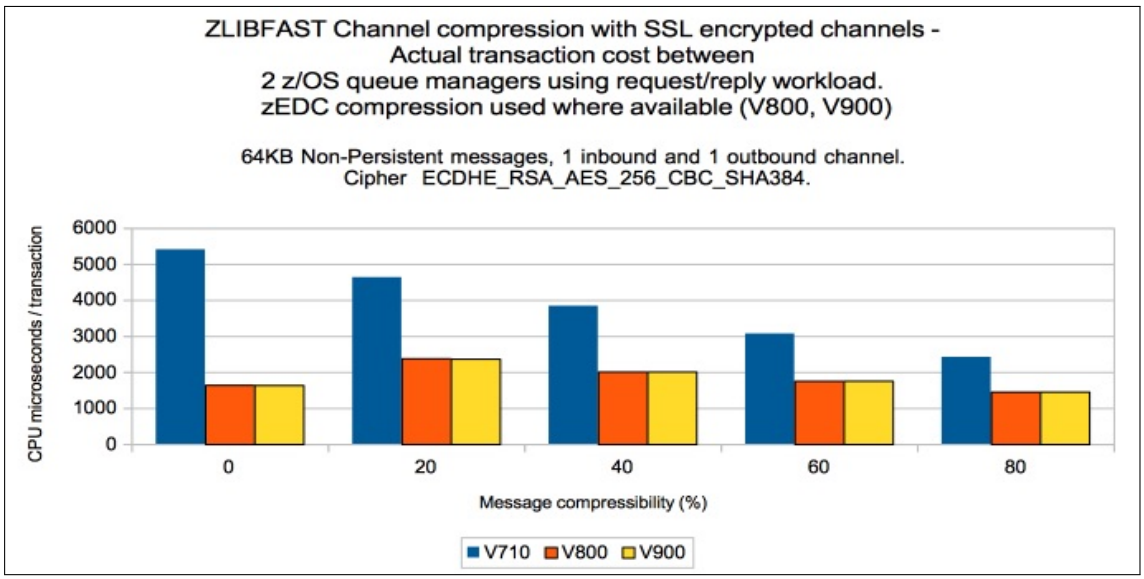


Chart: Transaction cost for 64KB messages with ZLIBFAST on SSL channels



Channel compression using ZLIBHIGH on SSL channels

For ZLIBHIGH we have seen between 10-20% reduction in cost in the channel initiator for versions 8.0.0 and 9.0.0 compared to version 7.1.0.

Chart: Transaction rate with 64KB messages with ZLIBHIGH on SSL channels

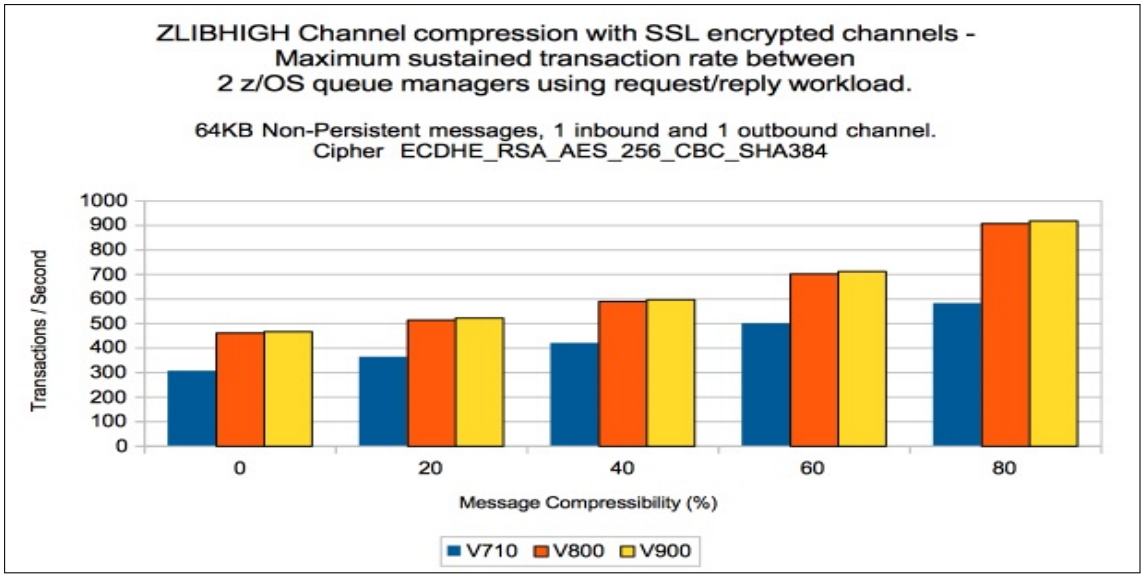
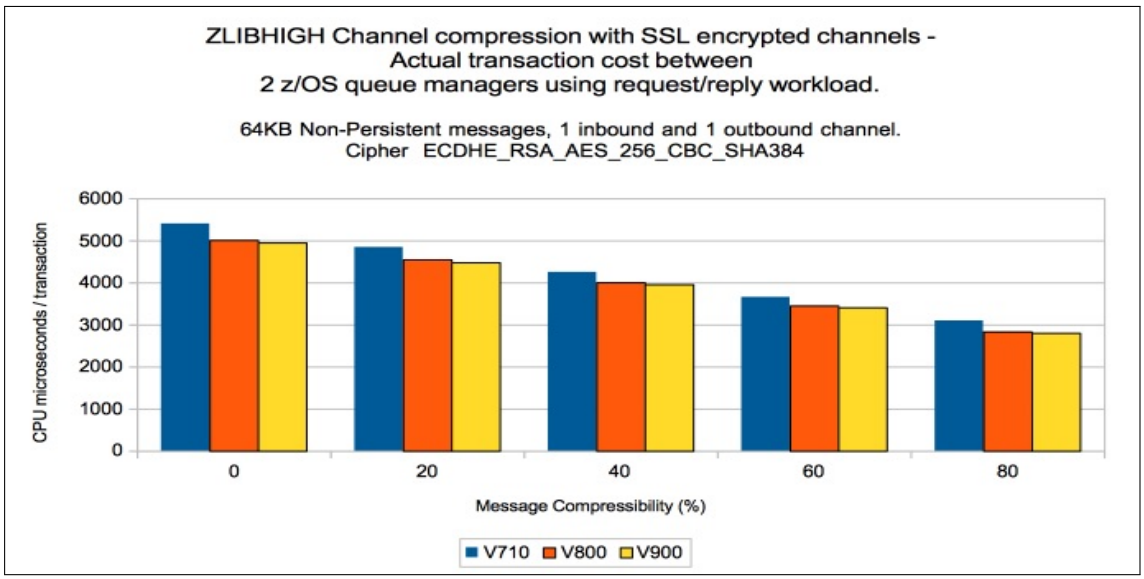


Chart: Transaction cost for 64KB messages with ZLIBHIGH on SSL channels



Moving messages across cluster channels

The regression tests for moving messages across cluster channels are relatively simple, providing multiple destinations for each message put.

The cluster has 3 queue managers - one for the requester workload and two for the server workload. The queue managers hosting the server workload are both full repository queue managers.

A set of requester tasks is run against one queue manager and the application chooses either bind-on-open or bind-not-fixed. The messages flow across the cluster channels to one of the server queue managers to be processed by the server applications, at which point they are returned to the originating requesting applications.

An increasing number of queues (with the corresponding increase in applications) are used.

Bind-on-open

Chart: Bind-on-open transaction rate with 2KB messages

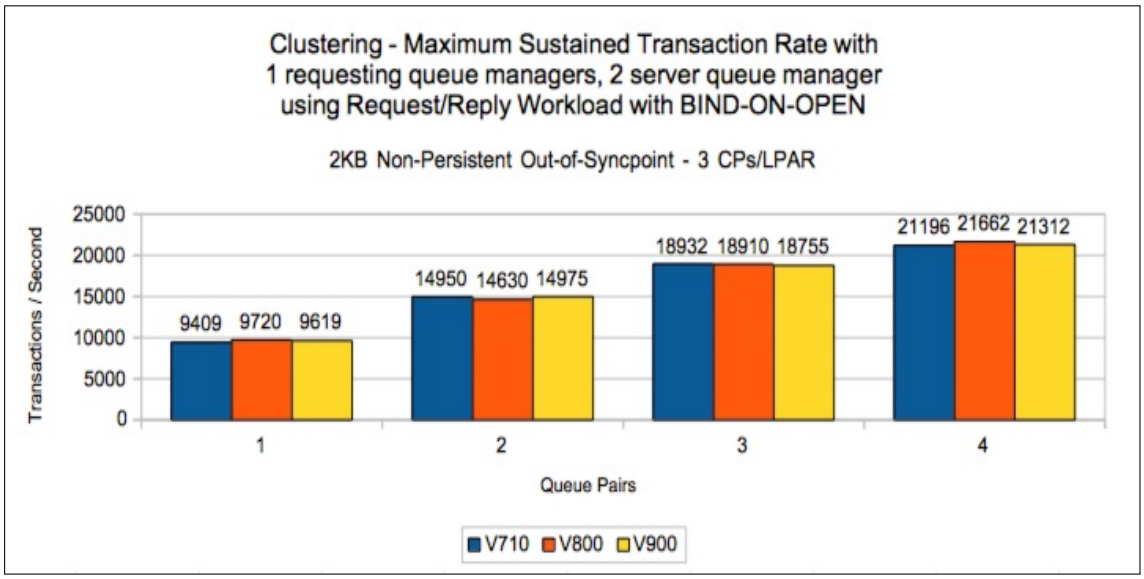


Chart: Bind-on-open transaction cost for 2KB messages

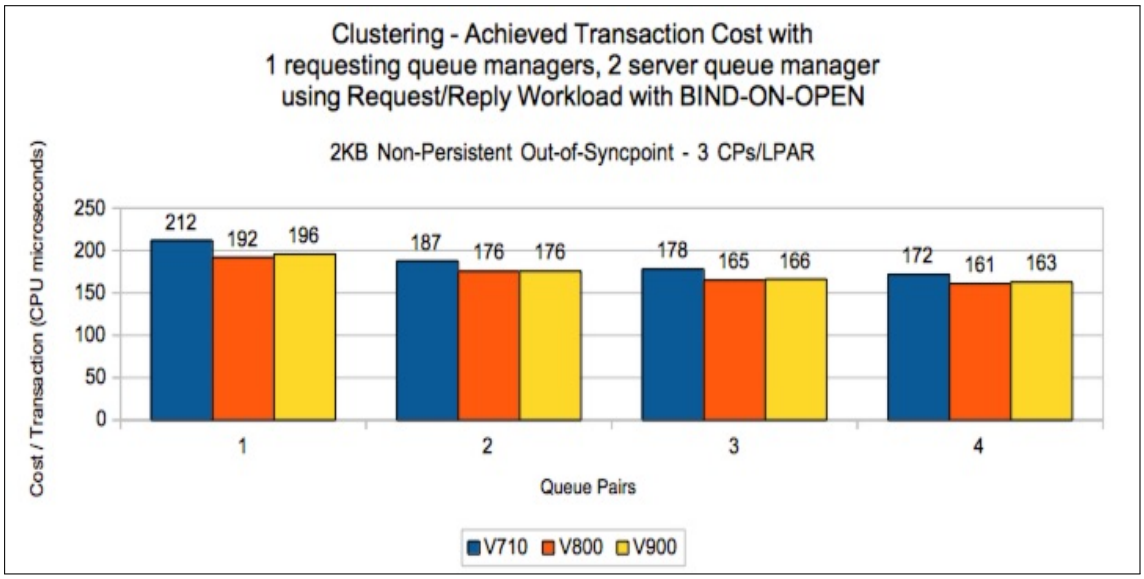


Chart: Bind-on-open transaction rate with 64KB messages

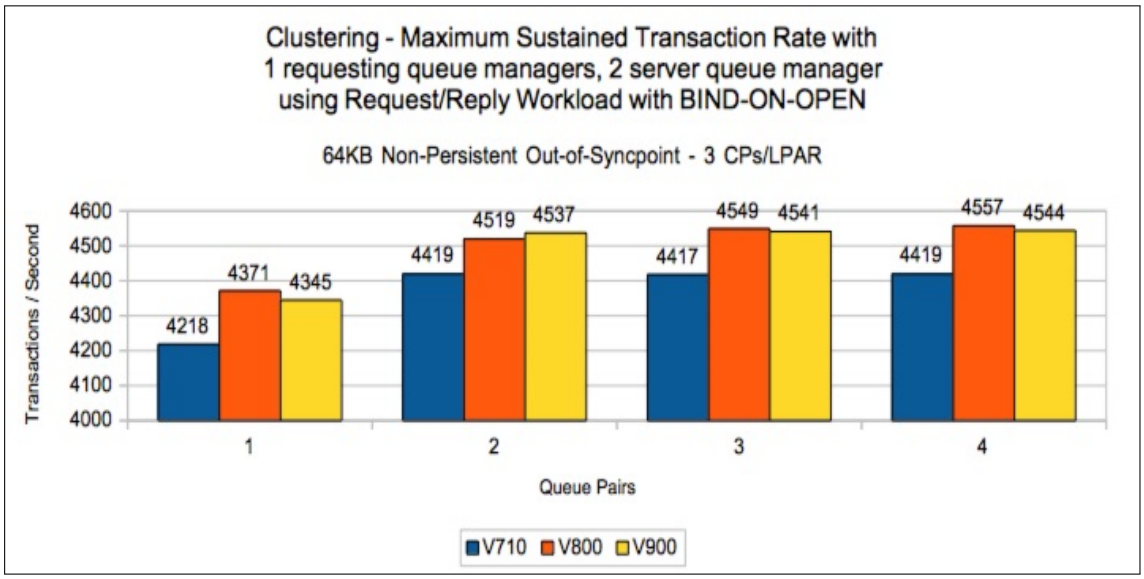
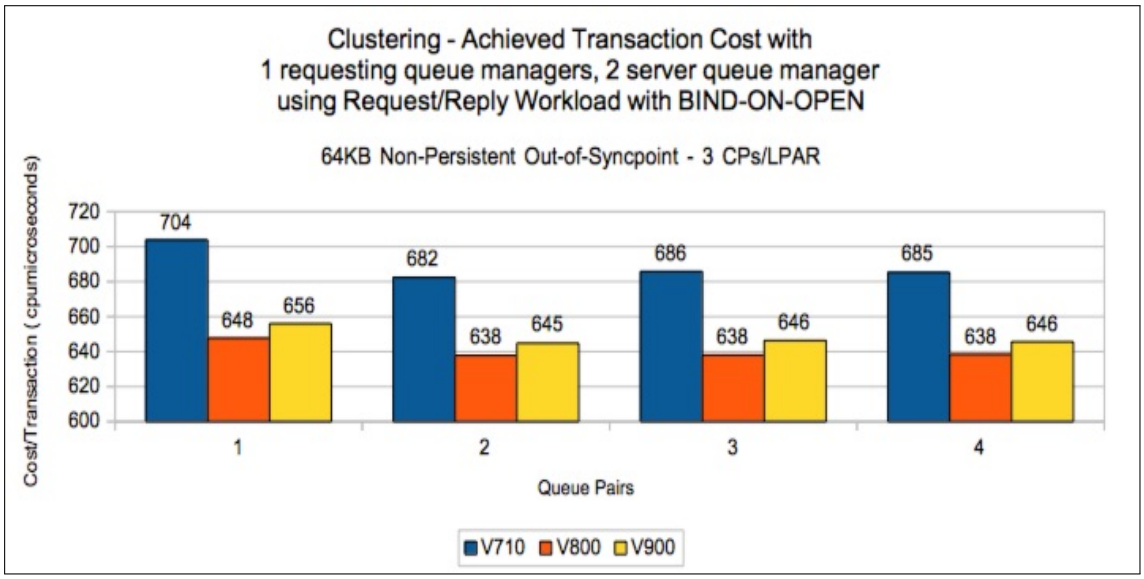


Chart: Bind-on-open transaction cost for 64KB messages



Bind-not-fixed

Chart: Bind-not-fixed transaction rate with 2KB messages

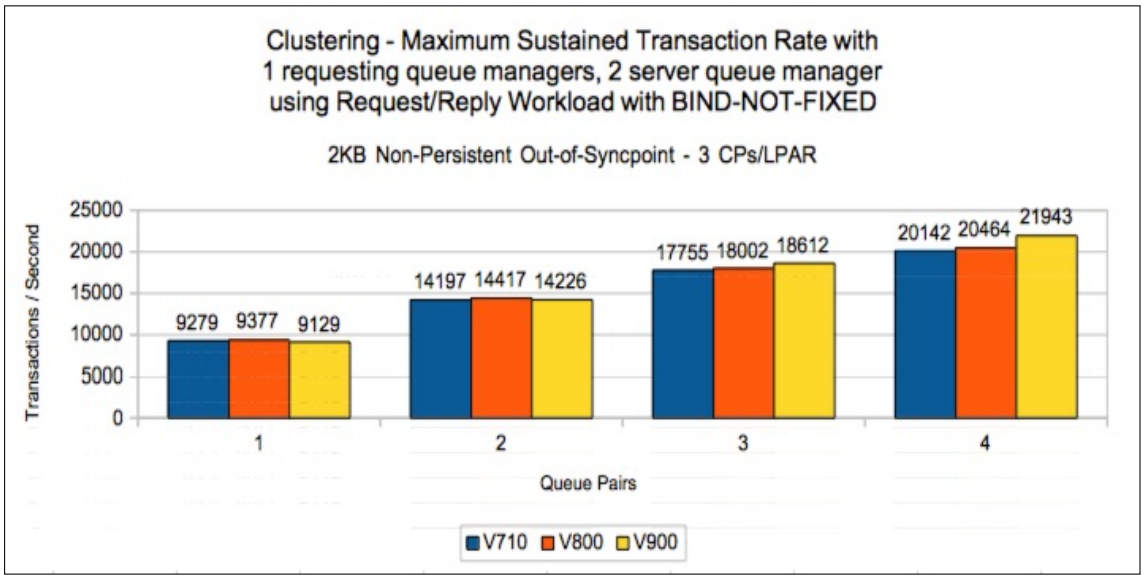


Chart: Bind-not-fixed transaction cost for 2KB messages

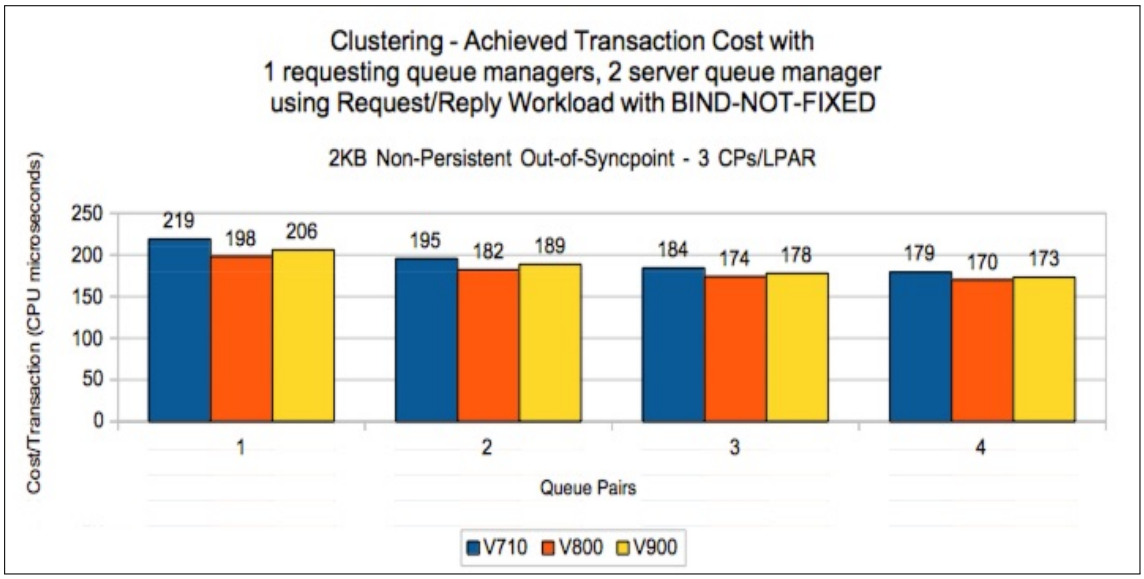


Chart: Bind-not-fixed transaction rate with 64KB messages

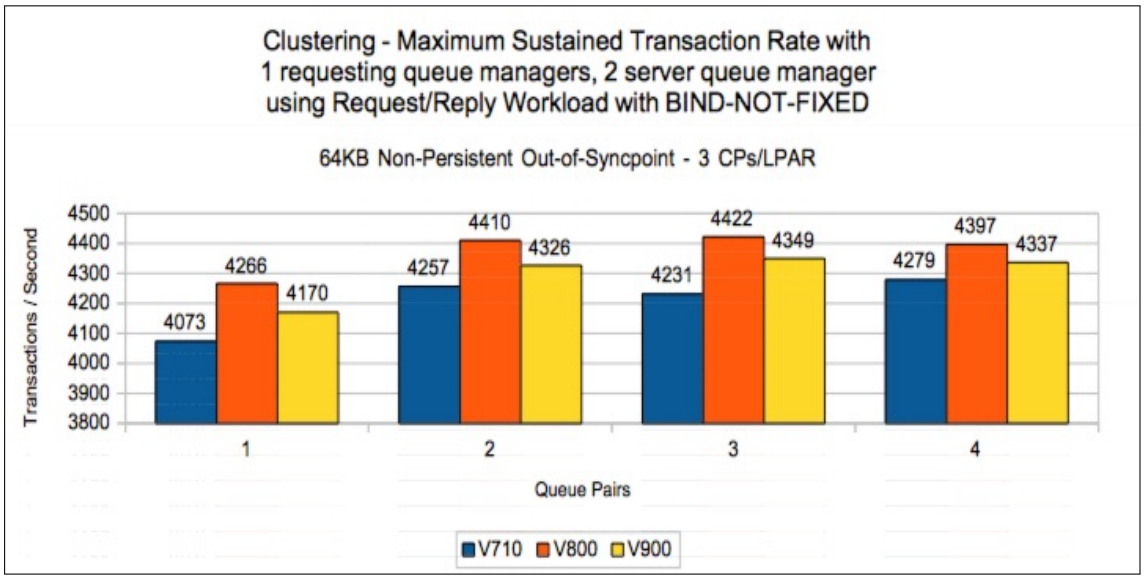
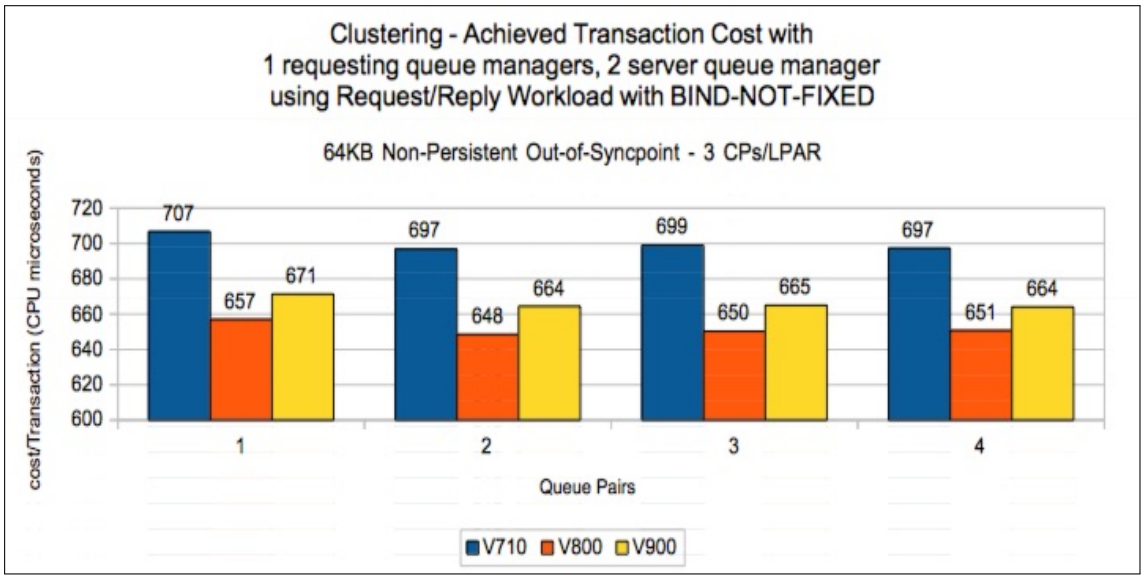


Chart: Bind-not-fixed transaction cost for 64KB messages



Moving messages across SVRCONN channels

The regression tests for moving messages across SVRCONN channels have a pair of client tasks for each queue that is hosted on the z/OS queue manager.

One of the pair of client tasks puts messages to the queue and the other client task gets the messages from the queue. As the test progresses, an increasing number of queues is used, with a corresponding increase in the number of putting and getting client tasks.

Two sets of tests are run, the first uses SHARECNV(0) on the SVRCONN channel to run in a mode comparable to that used pre-version 7.0.0. The second uses SHARECNV(1) so that function such as asynchronous puts and asynchronous gets are used via the DEFPRESP(ASYNC) and DEFREADA(YES) queue options.

Choosing which SHARECNV option is appropriate can make a difference to capacity and the performance which is discussed in more detail in performance report MP16 "Capacity Planning and Tuning Guide" Channel Initiator section.

Note: The rate and costs are based upon the number of MB of data moved per second rather than the number of messages per second.

Client pass through test using SHARECNV(0)

Chart: Throughput rate for client pass through tests with SHARECNV(0)

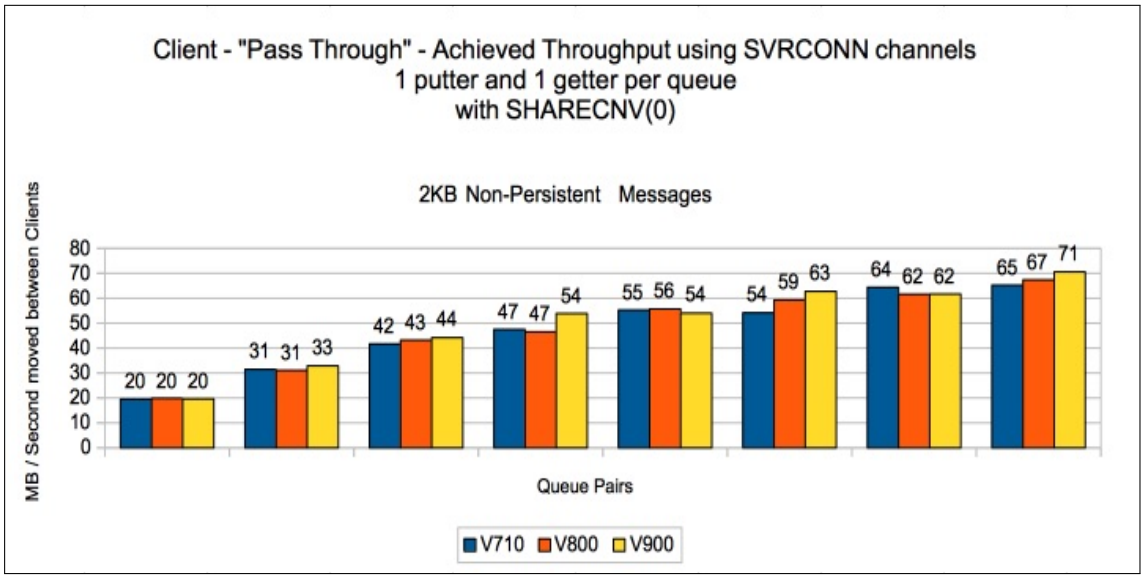
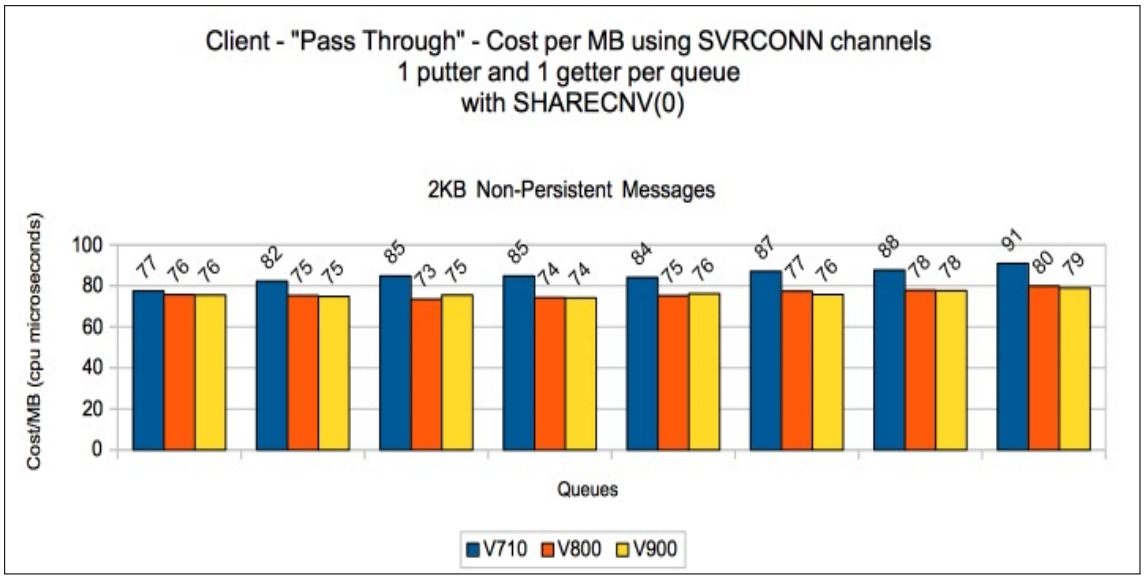


Chart: Cost per MB of client pass through tests with SHARECNV(0)



Client pass through test using SHARECNV(1)

Chart: Throughput rate for client pass through tests with SHARECNV(1)

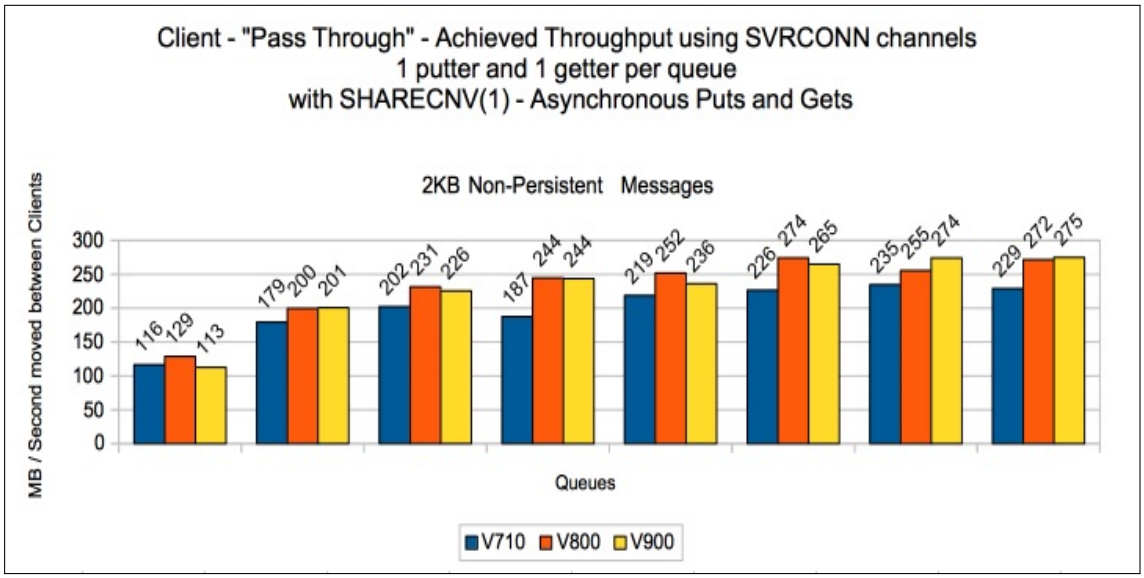
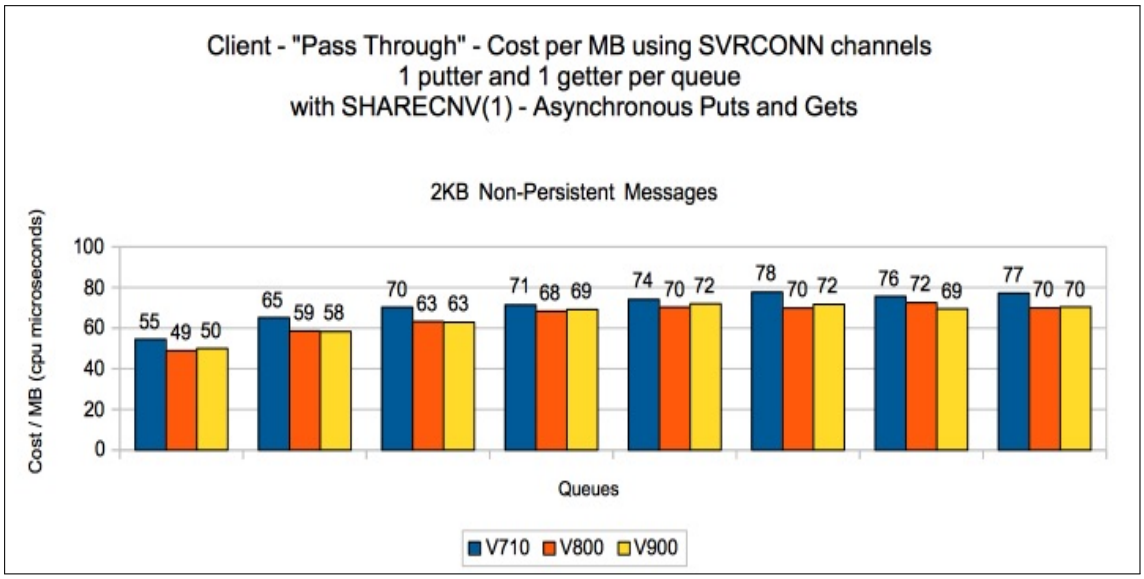


Chart: Cost per MB of client pass through tests with SHARECNV(1)



IMS Bridge

The regression tests used for IMS Bridge use 3 queue managers in a queue sharing group (QSG) each on separate LPARs. A single IMS region is started on 1 LPAR and has 16 Message Processing Regions (MPRs) started to process the MQ workload.

The IMS region has been configured as detailed in performance report MP16 "Capacity Planning and Tuning Guide" using the recommendations in the section "IMS Bridge: Achieving Best Throughput".

Note: 16 MPRs are more than really required for the 1, 2 and 4 TPIPE tests but they are available for consistent configuration across the test suite.

There are 8 queues defined in the QSG that are configured to be used as IMS Bridge queues.

Each queue manager runs a set of batch requester applications that put a 2KB message to one of the bridge queues and waits for a response on a corresponding shared reply queue.

Tests are run using both Commit Mode 0 (Commit-then-Send) and Commit Mode 1 (Send-then-Commit).

Commit mode 0 (commit-then-send)

Chart: IMS Bridge commit mode 0 - Throughput rate

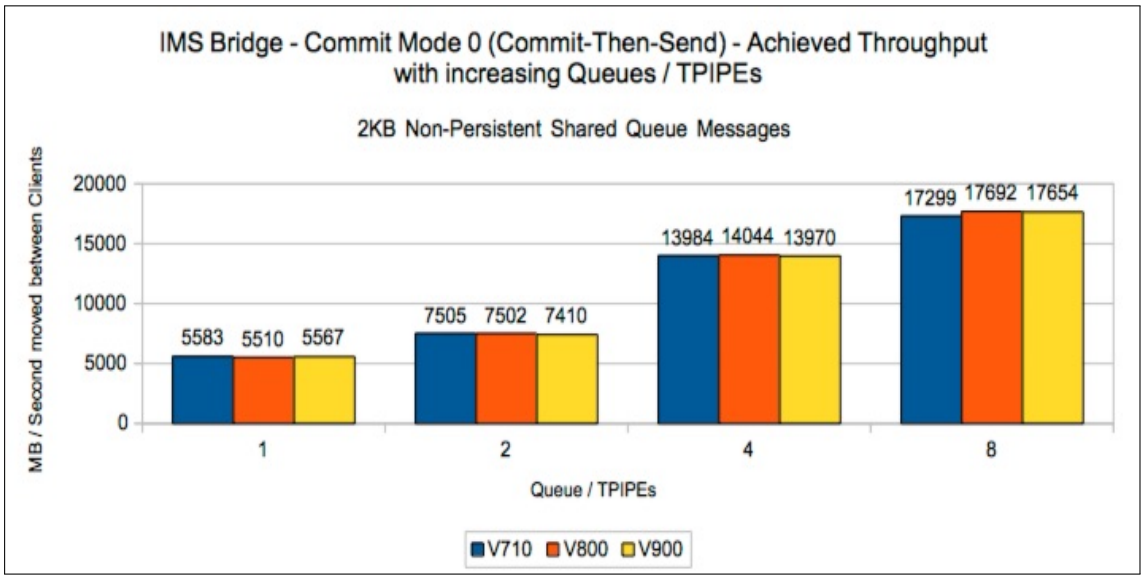
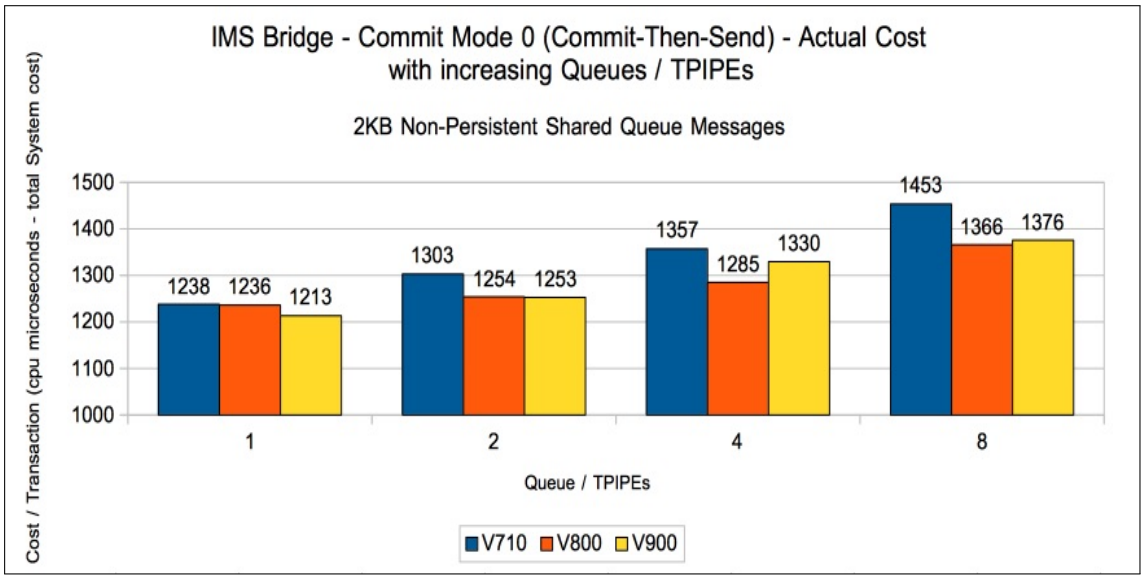


Chart: IMS Bridge commit mode 0 - Transaction cost



Commit mode 1 (send-then-commit)

Chart: IMS Bridge commit mode 1 - Throughput rate

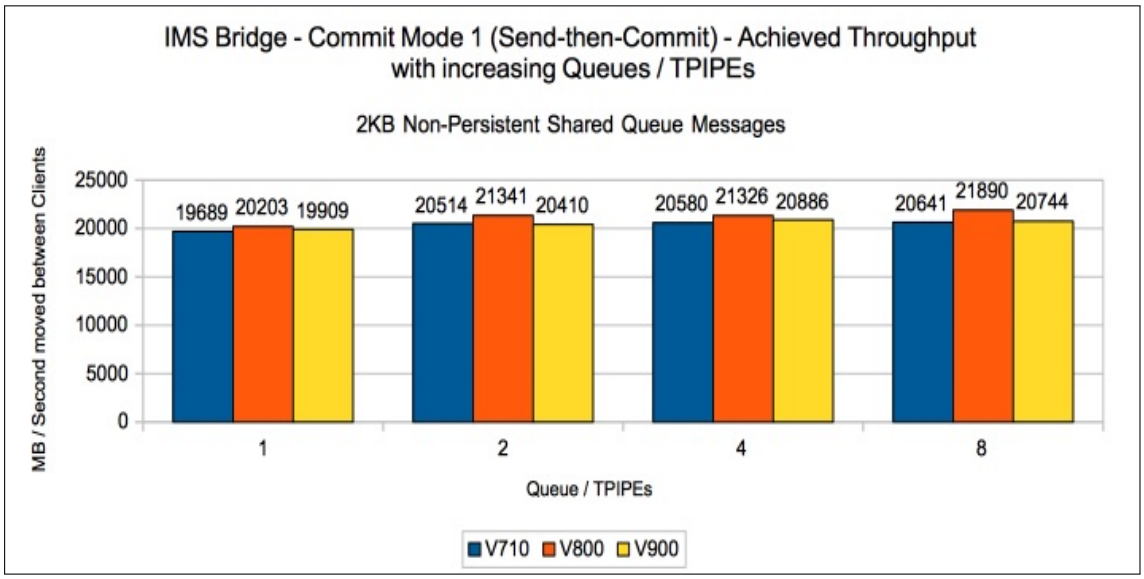
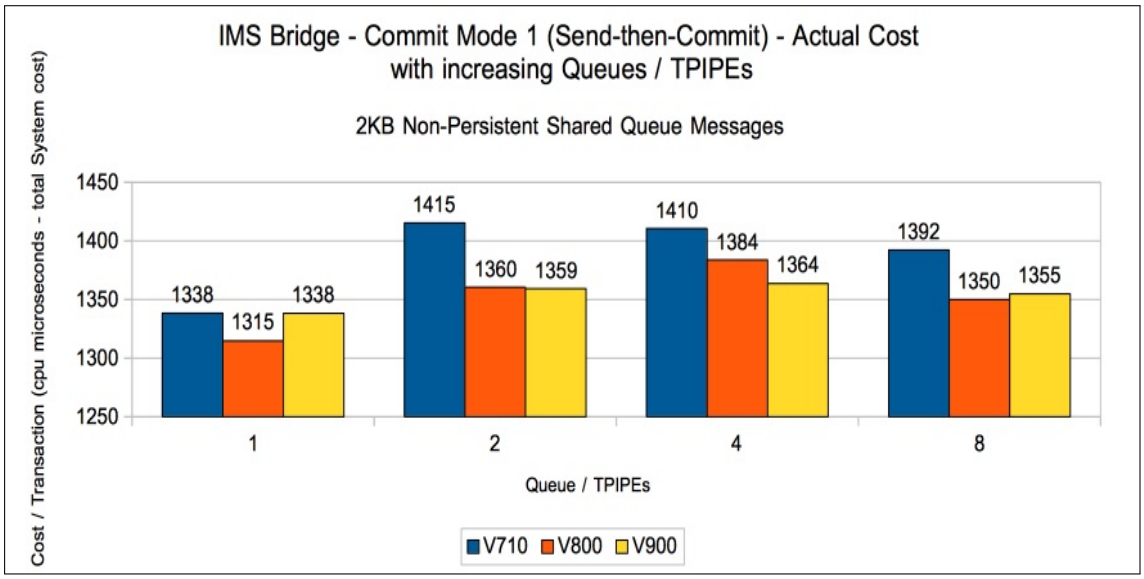


Chart: IMS Bridge commit mode 1 - Transaction cost



Trace

The regression tests for trace cover both queue manager global trace and the channel initiator trace.

Queue manager global trace

The queue manager global trace tests are a variation on the private queue non-persistent 2KB scalability tests with TRACE(G) DEST(RES) enabled.

From version 7.1.0, the global trace uses 64-bit storage which allows the queue manager to store a trace record for each thread and reduces contention on the trace storage. As a result the throughput achieved is significantly improved over previous releases of WebSphere MQ for z/OS.

Chart: Queue manager TRACE(G) DEST(RES) - transaction rate

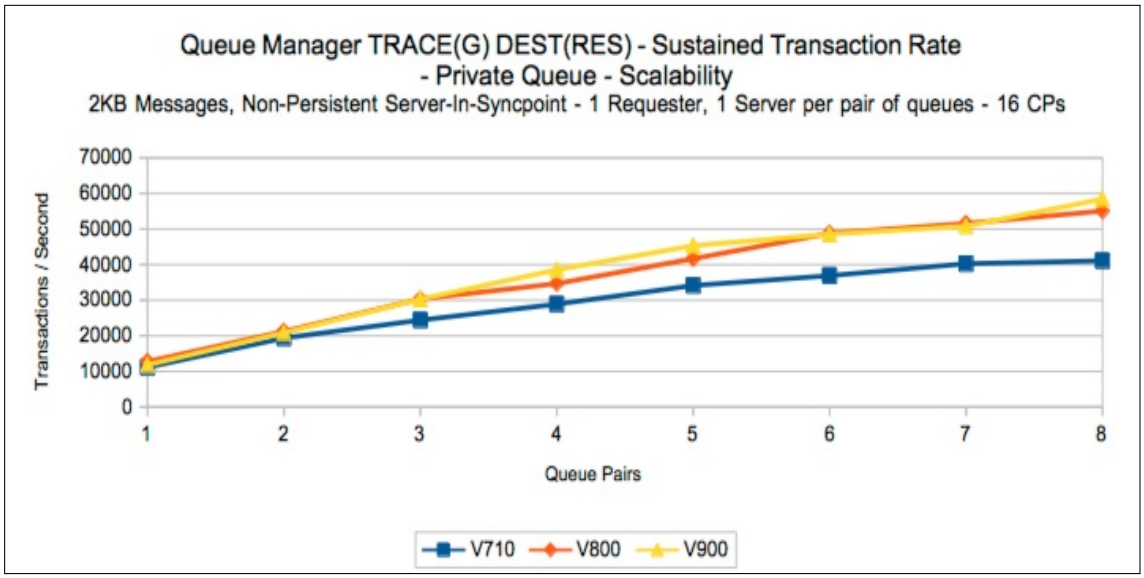
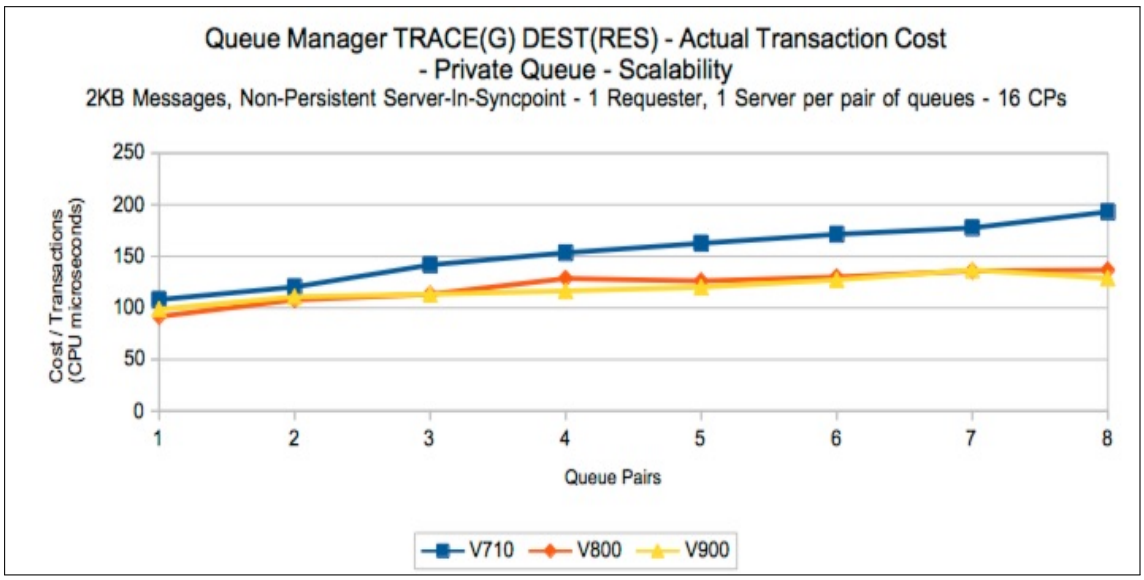


Chart: Queue manager TRACE(G) DEST(RES) - transaction cost



Channel initiator trace

The channel initiator trace tests are a variation on the client pass through tests using SHARECNV(0) with TRACE(CHINIT) enabled.

Chart: Channel initiator TRACE(CHINIT) - throughput rate

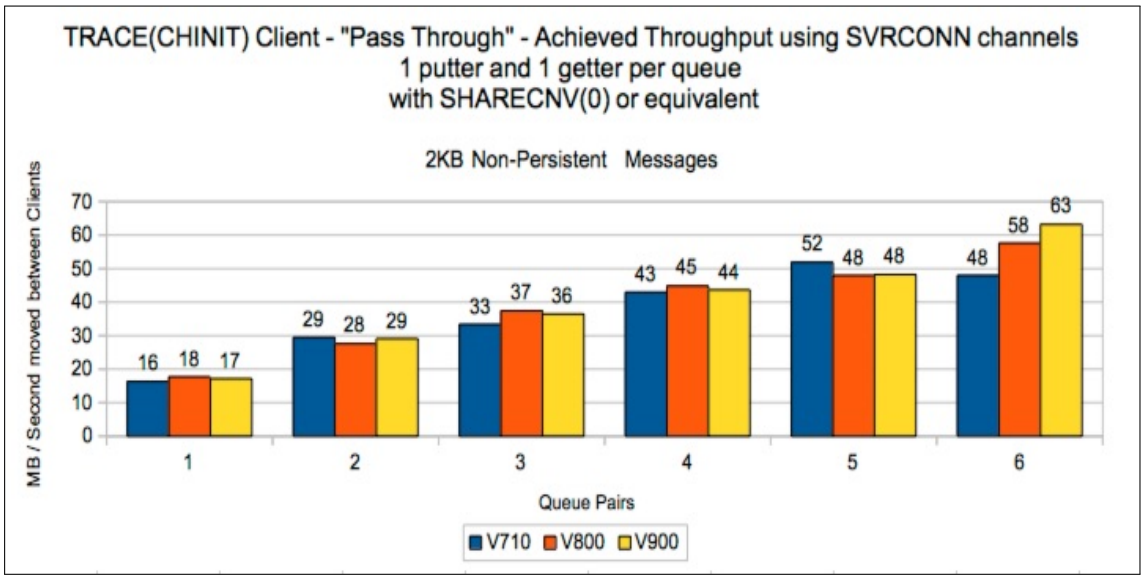
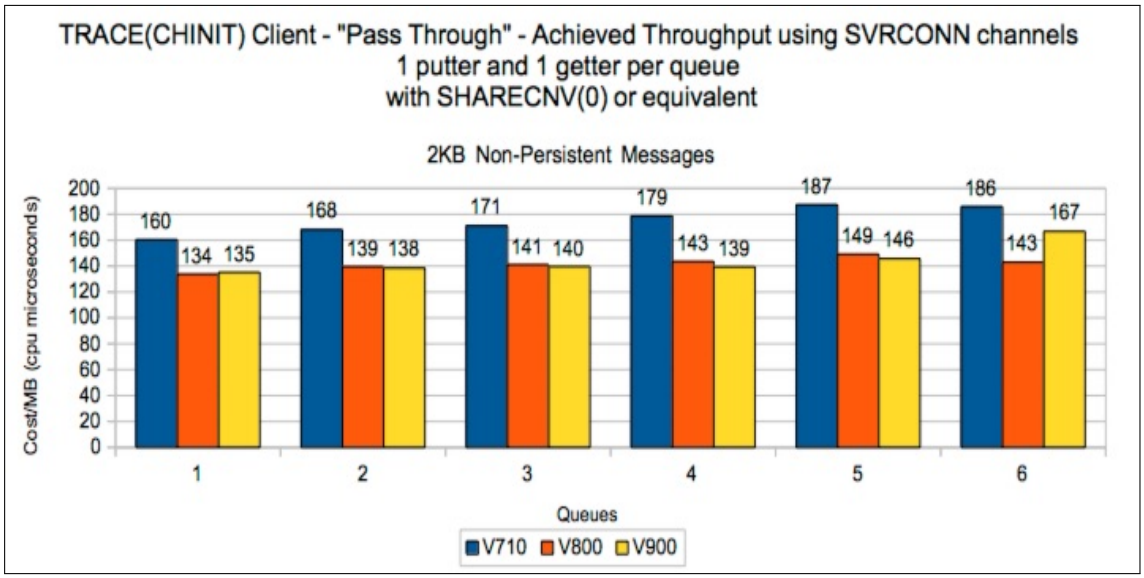


Chart: Channel initiator TRACE(CHINIT) - cost per MB



Appendix B

System configuration

IBM MQ Performance Sysplex running on z13 (2964-NE1) configured thus:

LPAR 1: 1-16 dedicated CP processors, 128GB of real storage.

LPAR 2: 1-3 dedicated CP processors, 32GB of real storage.

LPAR 3: 1-10 dedicated CP processors, plus 1 zIIP, 32GB of real storage.

Default Configuration:

3 dedicated processors on each LPAR, where each LPAR running z/OS v2r1 FMID HBB7790.

Coupling Facility:

- Internal coupling facility with 4 dedicated processors.
- Coupling Facility running CFCC level 21 service level 00.66.
- Dynamic CF Dispatching off.
- 3 x ICP links between each z/OS LPAR and CF

DASD:

- FICON Express 16S connected DS8870.
- 4 dedicated channel paths (shared across sysplex)
- HYPERPAV enabled.

System settings:

- zHPF disabled by default.
- HIPERDISPATCH enabled by default.
- LPARs 1 and 3 configured with different subnets such that tests moving messages over channels send data over 10GbE performance network.
    - SMC-R enabled by default between LPARs 1 and 3.
- zEDC compression available by default - exploited in V800 onwards for ZLIBFAST compression.
- Crypto Express5 card configured thus:
    - 1 x Accelerator, shared between LPARs 1 and 3.

    - 2 x Coprocessor on LPAR1.
    - 2 x Coprocessor on LPAR3.

IBM MQ trace status:

- TRACE(GLOBAL) disabled.
- TRACE(CHINIT) disabled.
- TRACE(S) CLASS(1,3,4) enabled where supported.
- TRACE(A) CLASS(3,4) enabled where supported.

General information:

- Client machines:
    - 2 x IBM SYSTEM X3850 each with 8 x 3.16GhZ Processor, 6GB memory
- Client tests used a 10GbE performance network.
- Other IBM products used:
    - CICS V6.9 CTS5.2 with latest service applied as of May 2016.
    - IMS V13.
    - WebSphere MQ for z/OS version 7.1 with latest service applied as of May 2016.
    - IBM MQ for z/OS version 8.0 with latest service applied as of May 2016.