MQ for z/OS Performance Report

IBM MQ Advanced for z/OS VUE version 9.1
and
IBM MQ for z/OS version 9.1

August 2018

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, August 2018. This edition applies to IBM MQ Advanced for z/OS VUE version 9.1 and IBM MQ for z/OS version 9.1 (and to all subsequent releases and modifications until otherwise indicated in new editions).

© Copyright International Business Machines Corporation 2018.
All rights reserved.

Note to U.S. Government Users Documentation related to restricted rights. Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule contract with IBM Corp.

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 9.1**. 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™
- z14™
- 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
2018	Version 1.0 - Initial Version.

Table of contents

Chapter 1

Performance highlights

This report focuses on performance changes since previous versions (V8.0 and V9.0) 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.

In IBM MQ Advanced for z/OS VUE version 9.1, Advanced Message Security performance is significantly improved over earlier releases, with particular benefits observed in the Confidentiality policy-type. We have seen throughput improve 6 times over comparable AMS Confidentiality measurements in v9.0.

Chapter 2

Existing function

General statement of regression

CPU costs and throughput are not significantly difference in version 9.1 for typical messaging workloads when compared with version 9.0.

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.

Note: From version 9.1, buffer pools allocated in 31-bit storage are being deprecated.

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 v9.1 is similar to both v8.0 and v9.0 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: v8.0, v9.0 and v9.1.

- For local connections, MCA channels and SVRCONN channels with SHARECNV(0), CSA usage is 2.47KB 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.

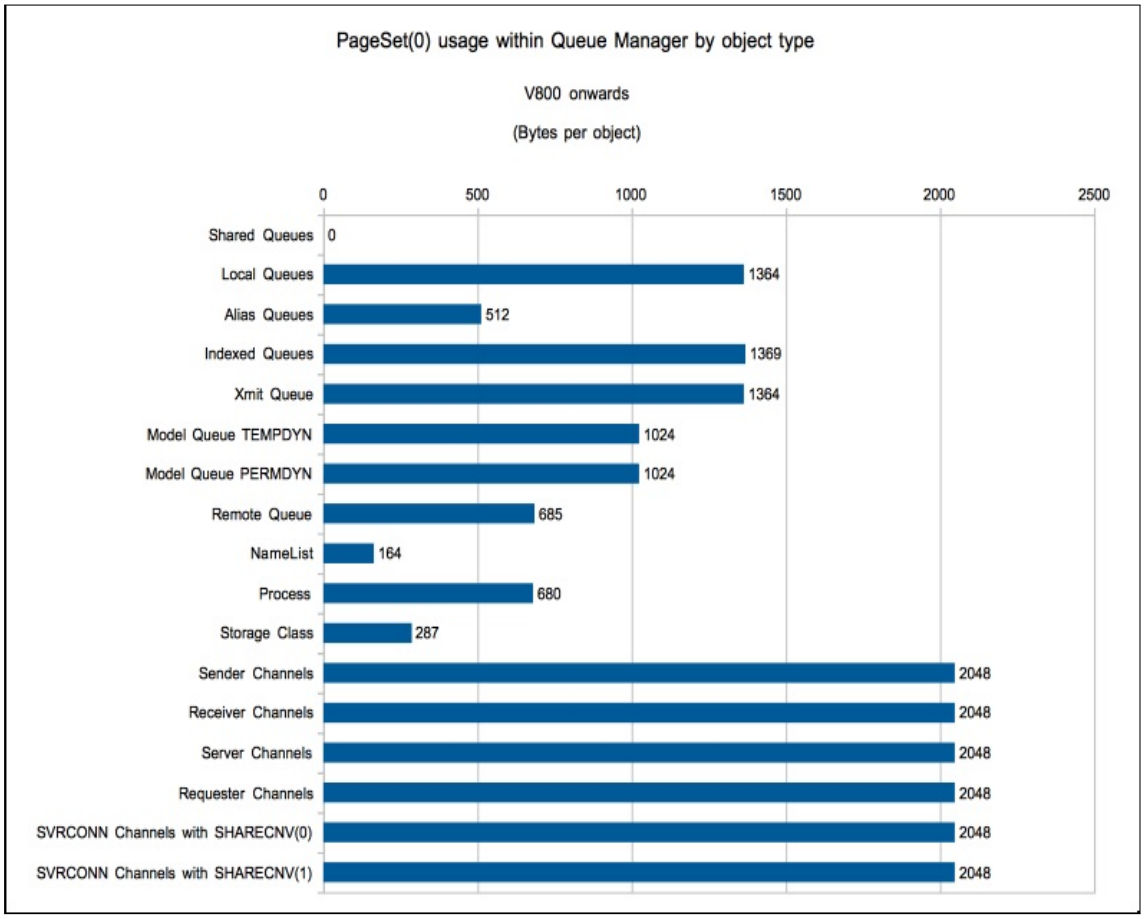
Object Sizes

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.

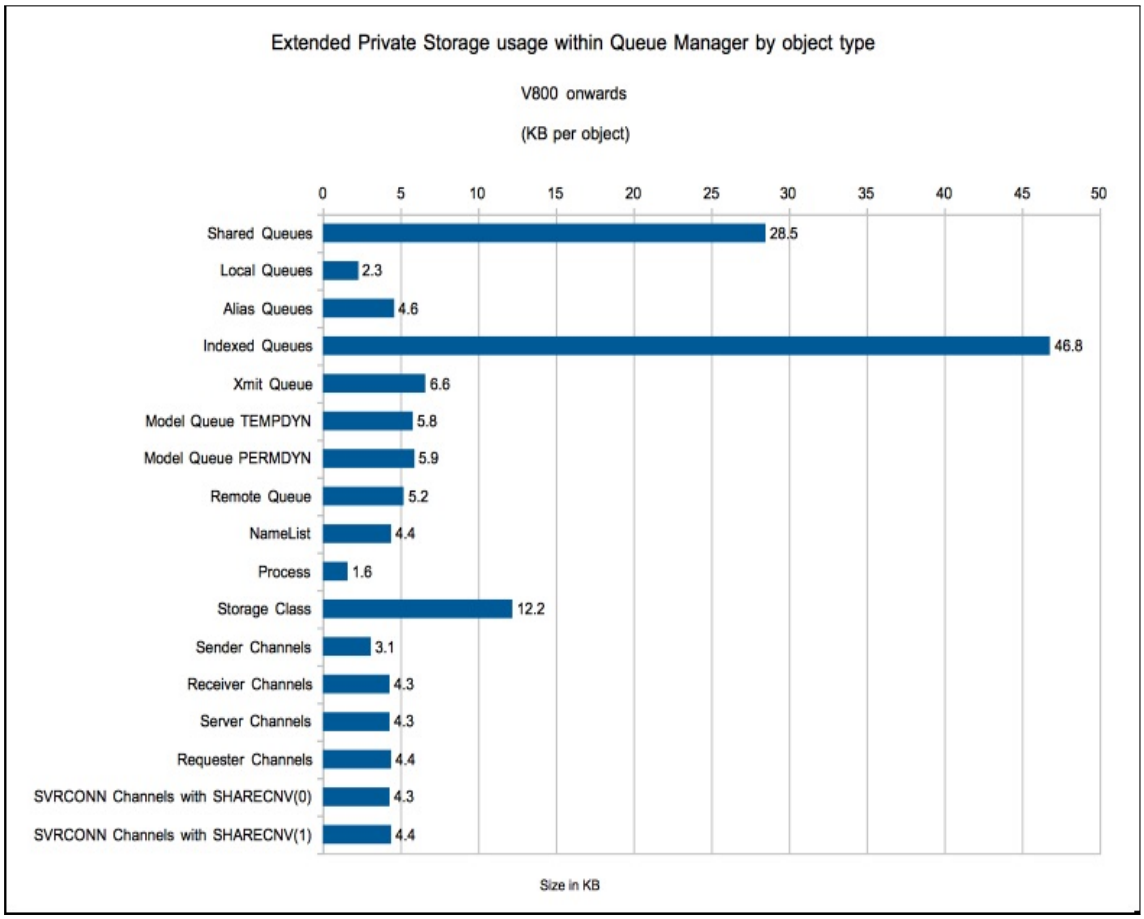
PAGESET(0) Usage

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.

From v8.0-onwards, the typical storage usage is approximately 15KB 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 22KB.
- Where connection is over a SHARECNV(10) channel and has a CURSHCNV of 10, the usage is 15.8KB per connection (158KB 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 28.2KB, giving an additional shared queue overhead of 6.2KB.

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**
v8.0	**0**	114	133	202	235
v9.0	**0**	100	119	187	217
v9.1	**0**	100	119	187	219
v8.0	**1**	184	210	305	377
v9.0	**1**	184	208	304	377
v9.1	**1**	183	209	270	317
v8.0	**10**	258	286	713	1071
v9.0	**10**	263	287	712	1070
v9.1	**10**	258	287	369	436

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(1) SVRCONN channels and using 32KB messages, we could predict that the v9.1 channel initiator could support a maximum of 3,570 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)	91	100	110	1110
Overhead of message size increase on 1KB messages		+9	+19	+1018

Chapter 3

New function for version 9.1

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

- Db2 universal tablespace support
- Advanced Message Security (AMS) enhancements
- Java clients on z/OS

Db2 Universal Table Space support

MQ version 9.1 provides sample jobs for defining the Db2 tablespaces, tables and indexes. Two sets of samples are provided:

- One for compatibility with earlier versions of IBM MQ, although support for traditional configurations is being deprecated in MQ version 9.1.
- One for use with Db2 v12 or later, which exploit Universal Table Spaces (UTS)

Universal Table Space (UTS) support is not specifically a performance feature in terms of MQ's use of Db2, but with Db2 v12 announcing that partitioned non-UTS table spaces are being deprecated, such that they are supported but may be removed in the future, it was necessary to provide the support for when the user is ready to implement UTS.

It should be noted that there is not a direct migration path from the traditional Db2 configuration to Db2 universal table space support, with the suggested path of using the UTS samples when moving to later DB2 versions.

In terms of MQ performance, the preferred option for large shared queue message support is still via Shared Message Data Sets (SMDS) as detailed in the performance report MP1H "WebSphere MQ for z/OS V7.1 Performance Report", but if there are reasons for using Db2 for large shared message support, then UTS may offer some performance benefits for LOB usage over traditional configurations.

To further complicate performance considerations, the supplied samples for the LOB table spaces use the option "GBPCACHE SYSTEM". This is optimised for configuration where messages are potentially processed anywhere in the sysplex.

"GBPCACHE SYSTEM" means that at commit after the insert, the changed LOB page is written to DASD instead of the group buffer pool (CF). Only changed system pages (e.g. space map pages) are written to the group buffer pool (CF). When another member comes to delete the message, it will need to read the page from DASD. In those configurations where data sharing is highly prevalent the "GBPCACHE CHANGED" option may be a more suitable option. For more information on this option, please refer to the Db2 Knowledge Center section "How the GBPCACHE option affects write operations".

Advanced Message Security (AMS) enhancements

IBM MQ Advanced for z/OS VUE version 9.1 only

This section focuses on performance improvements since IBM MQ for z/OS version 9.0, which relate to the Advanced Message Security (AMS) component of the MQ product.

Comparisons in AMS performance are made between versions 8.0, 9.0 and 9.1.

The performance details in this section are broadly similar to those reported in V901 "IBM MQ for z/OS version 9.0.1 CDR Performance Report" but have been updated in-line with the underlying infrastructure changes - namely moving from z13 to z14 and CryptoExpress5S to CryptoExpress6S.

In addition to the performance improvements of the AMS policy types, primarily the AMS Confidentiality policy type which was introduced in IBM MQ for z/OS version 9.0 and has been optimised in IBM MQ Advanced for z/OS VUE version 9.1, changes have also been made to reduce the impact of connecting non-AMS clients to AMS-enabled queue managers.

Performance Highlights

The performance of AMS in V9.1, as demonstrated later in this document, has improved significantly over previous versions. Here are some of the highlights:

Comparing AMS on V9.1 with AMS on V8.0:

- Transactions protected by AMS Integrity policies can see a cost of **32%** of similar transactions run against V8.0.
- Transactions protected by AMS Privacy policies can achieve a cost of **35%** those of similar transactions in V8.0.
- Transactions protected by AMS Confidentiality policies can achieve a transaction rate of **29 times** that of a comparable workload using V800 AMS Privacy protection.

Comparing AMS on V9.1 with AMS on V9.0:

- Transactions protected by AMS Integrity policies can see a cost of **less than half** of similar transactions run against V9.0, with a throughput improvements in **excess of 1.5 times**.
- Transactions protected by AMS Privacy policies can achieve a cost of **40%** those of similar transactions in V9.0, with a throughput improvement in **excess of 30%**.
- Transactions protected by AMS Confidentiality policies can achieve a transaction cost of **17%** those of similar transactions in V9.0, with a throughput improvement in excess of **6 times**.

Comparing AMS on V9.1 with channels protected by TLS cipher specs:

- Small messages workloads (32KB and less) protected using AMS Confidentiality policies can demonstrate a lower transaction cost than channels protected using TLS cipher specs even when the SSLRKEYC settings are such that significantly more data flows between queue managers between AMS key negotiations than the TLS key negotiations.
- Large messages workloads (1MB) protected using AMS Confidentiality policies can show a **20%** reduction in transaction cost over channels running TLS cipher specs where the secret key is re-negotiated at similar intervals.
- AMS policies allow the key to be renegotiated at a queue level, rather than the queue manager level attribute SSLRKEYC (SSL/TLS key reset count, which is the total number of bytes to be sent and received within a TLS conversation before the secret key is renegotiated).

  This means that the key can be renegotiated on an appropriate value for the particular workload. This may be useful if some channels send high volumes of data and some send low-volumes which may leave a secret key unchanged for long periods.
- AMS policy protection means that the message is protected from the time it is put to the queue until it is gotten by the authorised recipient and remains protected regardless of how many queue managers the message transitions through.

  By contrast a message protected by TLS channels is only encrypted for the flow across the network. Furthermore if the message transitions across multiple queue managers each with TLS enabled channels, the cost of encryption and decryption can be incurred for each transition.

  However, when using AMS over MQ channels, it is still advisable to protect the channels using SSL/TLS cipher specs, to prevent "man in the middle" attacks on any data flowing over the channel that are using queues not protected by AMS policies.

Background

IBM MQ Advanced Message Security (IBM MQ AMS) provides a high level of protection for sensitive data flowing through the MQ network with different levels of protection by using a public key cryptography model.

IBM MQ version 9.0.0 supplemented the existing two qualities of protection *Integrity* and *Privacy* with a third quality of protection, namely *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. Symmetric key operations are very fast in comparison to asymmetric key operations, which are CPU intensive and whilst some of the cost may be offloaded to cryptographic hardware such as Crypto Express6S, this can have a significant impact on the cost of protecting large numbers of messages with IBM MQ AMS.

- **Asymmetric cryptographic routines** as used by Integrity and Privacy

  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** as used by Privacy and Confidentiality

  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.

IBM MQ Advanced for z/OS VUE version 9.1 has further optimised the performance of AMS qualities of protection, for all 3 levels, with the most optimal offering from a performance perspective being AMS Confidentiality.

This document aims to demonstrate the improvements in performance, comparing release-on-release improvements as well as comparing the different message protection options offered by MQ.

Scenarios

There are a number of scenarios that were used to gather the data in this report:

- **Request / Reply - Local workload:**

  A request/reply workload is run using pairs of requester and server batch tasks connected to a single MQ queue manager. Each pair of applications uses their own request and reply queues.

  The requester task puts a message to the request queue and waits for a specific response on the reply queue. Once the reply message is gotten, the requester will put another message to the request queue.

  The corresponding server task issues MQGET-with-wait calls on the request queue, gets the message to the known (and pre-opened) reply queue and the application goes back into its MQGET-with-wait call. The messages are got and put in syncpoint with 1 MQGET and 1 MQPUT per commit.

  2KB, 64KB and 4MB non-persistent messages are used.

- **Request / Reply - Mover workload:**

  A request/reply workload is run to move messages between 2 queue managers on separate LPARs of the same performance sysplex. There are sender-receiver channels defined in each direction.

  Comparisons of transaction cost and rate are provided for 3 configurations - no protection, channels protected by TLS encryption and messages protected by AMS policies.

  The TLS configuration uses cipher spec "ECDHE_RSA_AES_256_CBC_SHA384" and the test varies the TLS/SSL key negotiation frequency (SSLRKEYC) using values of 0, 1 and 10MB.

  All AMS policy types are used i.e. Integrity, Privacy and Confidentiality.

  Non-persistent messages of size 32KB are used.

- **Streaming messages between queue managers:**

  An example of streaming messages between queue managers is a scenario such as an InfoSphere Replication Server workload.

  This replication workload simulates moving data from one system to another using MQ channels. The system that sends the data uses a "capture" task to get the data and put to an MQ queue as quickly as possible. At the remote end, there is an "apply" task that gets the messages from the queues and processes them. As the data flows in a single direction, there is the potential for a build up of messages on the transmit queue as the capture task may put messages more quickly than the channel initiator can get and send the messages, for example in the event of a network delay or the apply task being slow.

  Measurements will be run using TLS cipher spec "ECDHE_RSA_AES_256_CBC_SHA384" with a range of key negotiation frequency and compared with queues protected using AMS Confidentiality policies with a range of key reuse values.

  Persistent messages of 10KB and 1MB are used.

All policies are defined such that there is only a single recipient for each message.

The signing algorithm used in the Integrity and Privacy tests is **SHA256**. The encryption algorithm used for Privacy and Confidentiality tests is **AES256**.

The applications in use in all measurements have minimal business logic, which means that the application cost is considerably less than would be expected in a customer transaction. This can make the impact of AMS policies appear more significant than would be observed in a customer environment.

Non-AMS Clients connecting to an AMS-enabled Queue Manager

Prior to MQ v9.1, a client application connecting to a z/OS queue manager that had AMS enabled via the "SPLCAP=YES" option would incur significant additional cost at the time of MQOPEN, regardless of whether a policy was defined for the queue being opened.

As the following table demonstrates, MQ v9.1 shows a reduction in the cost on z/OS of the MQOPEN and MQCLOSE for the MQ queue manager, channel initiator and TCP/IP address spaces in CPU microseconds:

IBM MQ release	SPLCAP=NO	SPLCAP=YES
v9.0	59	190
v9.1	58	60

Note: These costs are based on a single long-running client that opens and closes a private queue 100,000 times.

How much difference does adding end-to-end security to a transaction make?

The simple answer is that it can make a significant amount, but the good news is that the impact is significantly less than on earlier versions of MQ.

In these particular local request/reply scenarios, the costs associated with securing the messages increased the transaction cost between 1.6 and 12 times over the unsecured message workload. It should also be noted that the messaging costs in the baseline measurements do form over 95% of the transaction cost. In cases where the non-messaging processing costs form a larger part of the total transaction cost, for example a short-lived CICS transaction, the relative effect of processing the message with an AMS policy is much smaller.

This overhead is significantly reduced from that detailed in performance report MP1J "IBM MQ for z/OS version 8.0.0 Performance Report", which discusses the impact as being 23 times that of the equivalent unsecured messaging workload.

The following charts provides some indication of the increase in cost and throughout for a set of simple request/reply workloads using non-persistent messages using 2KB, 64KB and 4MB messages.

Note the decrease in transaction rate and increase in transaction cost does not necessarily correlate, as a message protected by an AMS policy will also see time spent in cryptographic processing, where the cost may be offloaded to hardware, and security checking (e.g. RACF).

Chart: The impact of AMS on transaction cost - Request/Reply - Local workload with 2KB messages

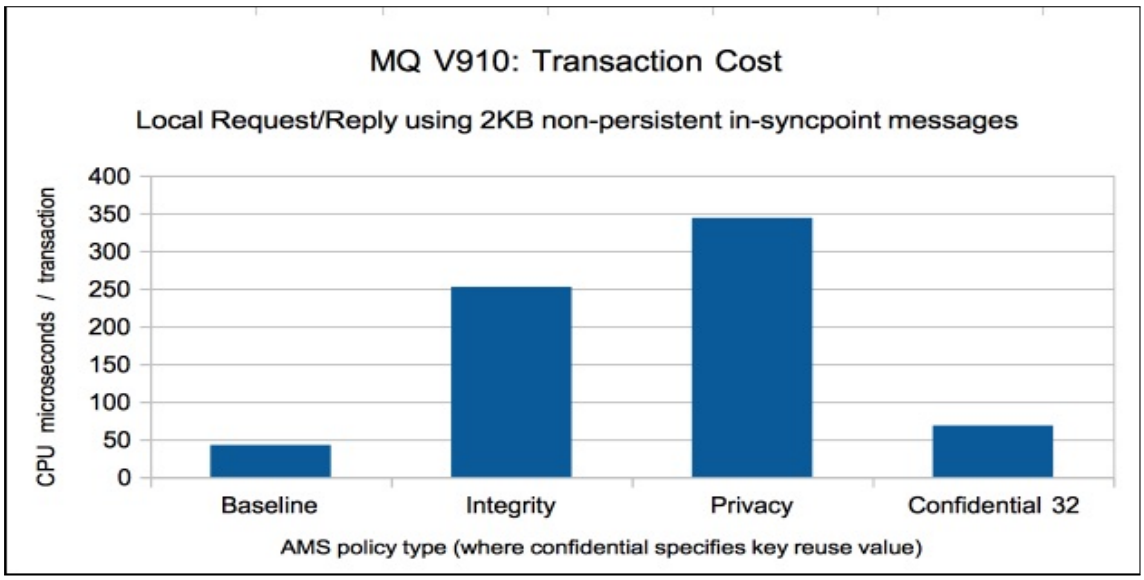


Chart: The impact of AMS on transaction rate - Request/Reply - Local workload with 2KB messages

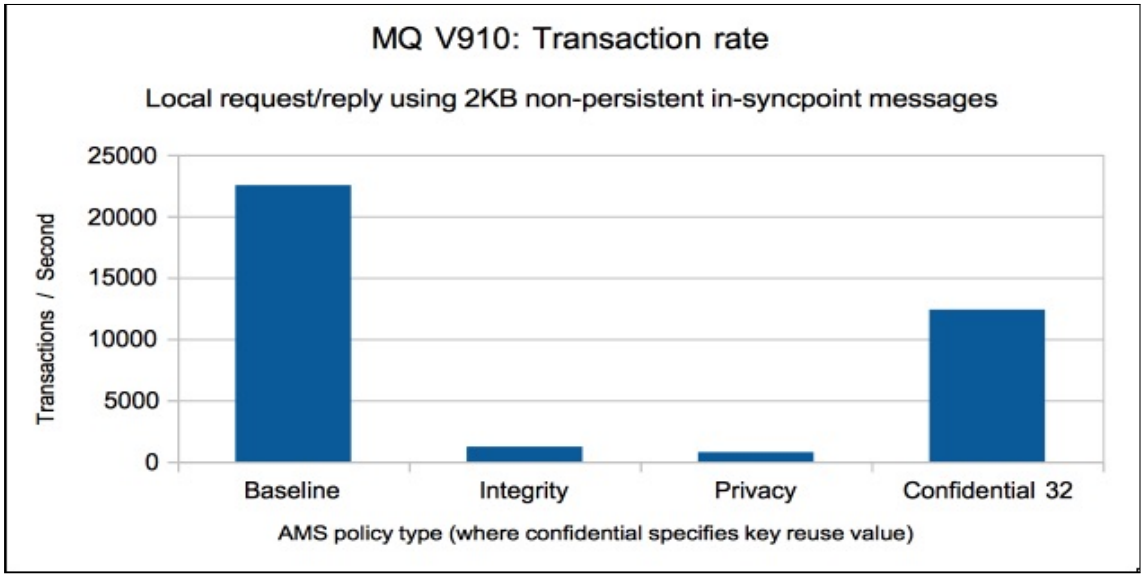


Chart: The impact of AMS on transaction cost - Request/Reply - Local workload with 64KB messages

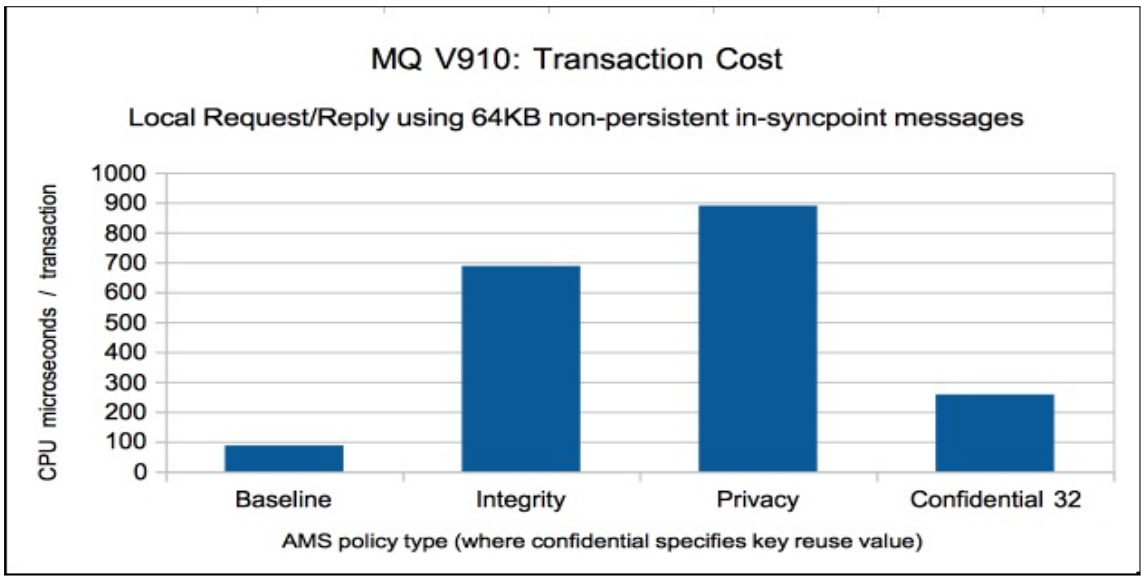


Chart: The impact of AMS on transaction rate - Request/Reply - Local workload with 64KB messages

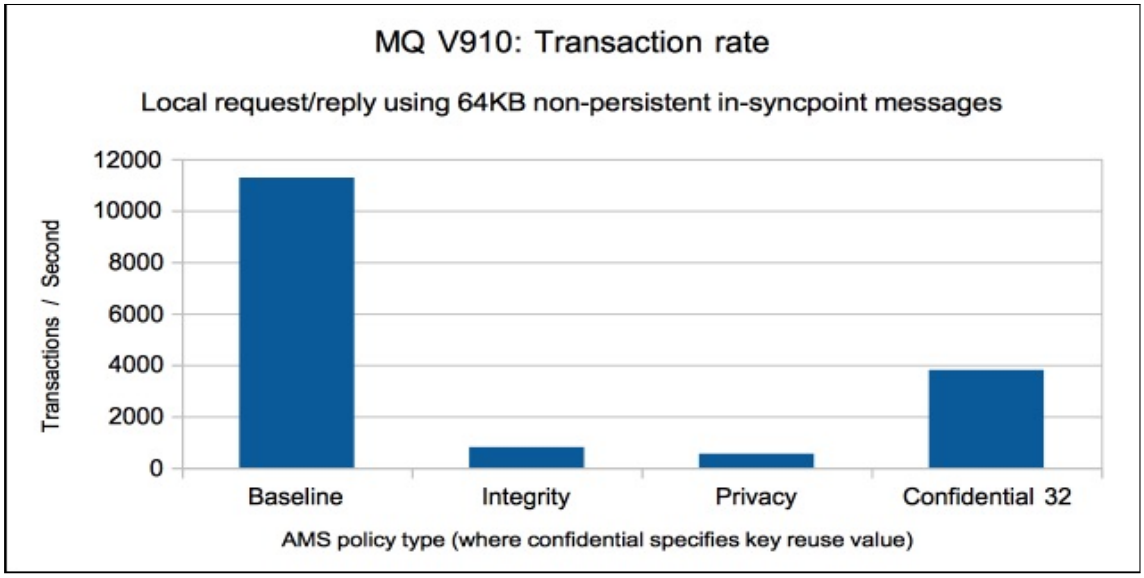


Chart: The impact of AMS on transaction cost - Request/Reply - Local workload with 4MB messages

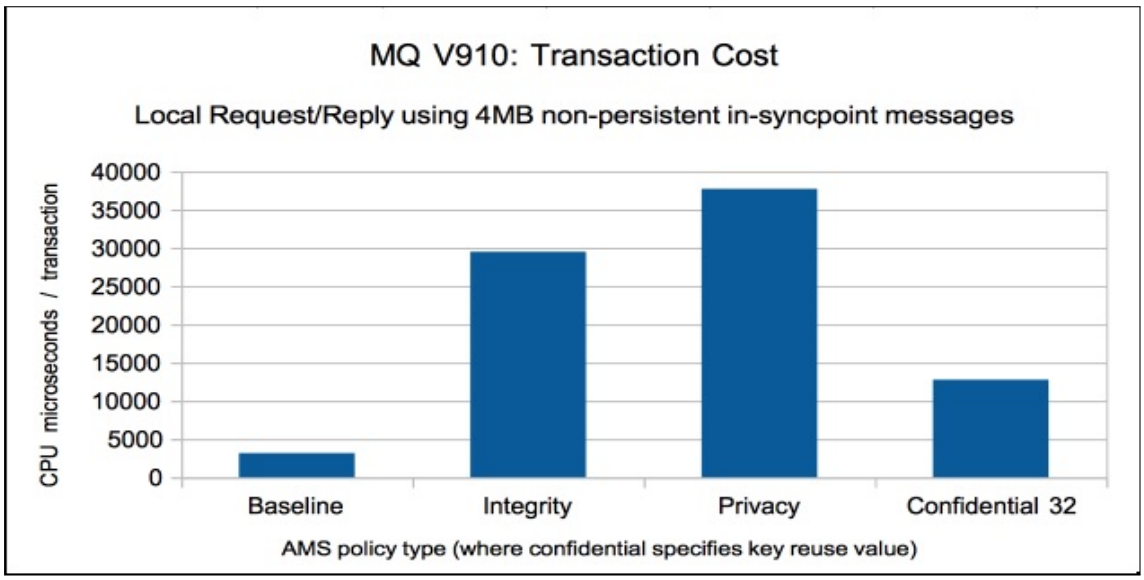
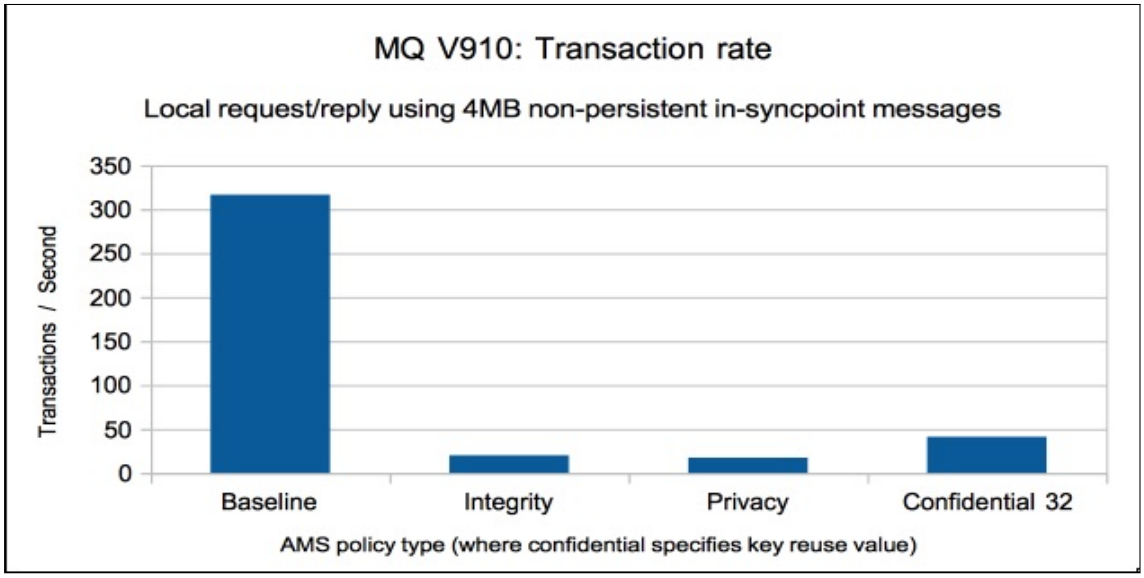


Chart: The impact of AMS on transaction rate - Request/Reply - Local workload with 4MB messages



AMS Measurements

Request / Reply - Local workload

This section demonstrates the improvement in performance of the 3 AMS policy types in v9.1 by comparing the transaction costs and rates of a simple request / reply workload using non-persistent messages.

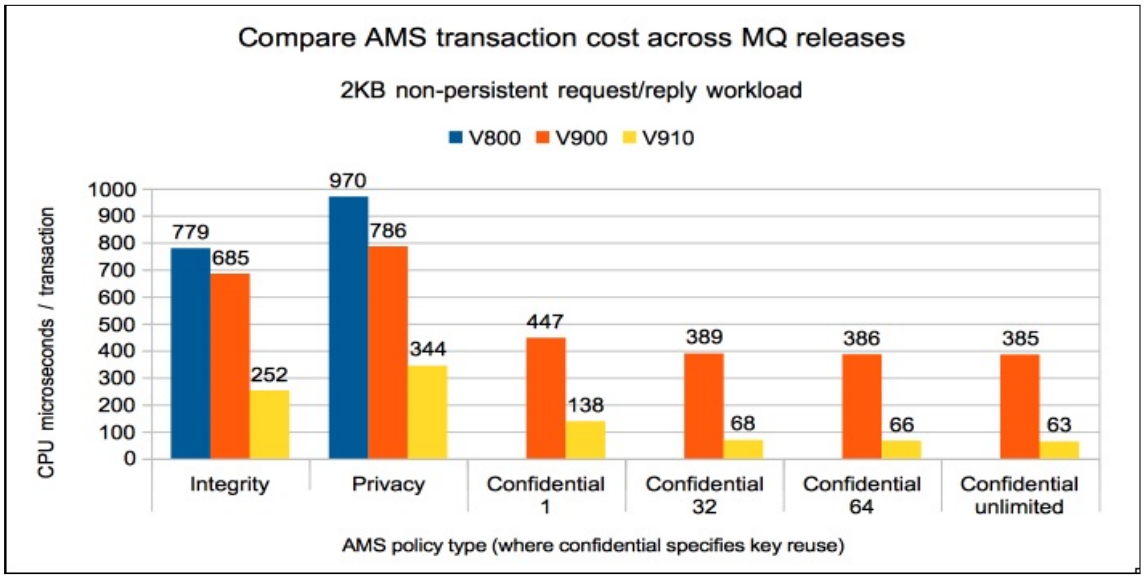
In this most simplistic scenario using small messages, when comparing the total transaction cost for v9.1 against v9.0, there are significant reductions in cost e.g.

- **Integrity**: v9.1 cost was 37% of the equivalent v9.0 measurement.
- **Privacy**: v9.1 cost was 44% of the equivalent v9.0 measurement.
- **Confidentiality**: v9.1 cost was 17-32% of the equivalent v9.0 measurements depending on the key reuse value.

Both Integrity and Privacy policy types showed similar percentage reductions in transaction cost for larger messages.

The improvements in the AMS Confidentiality performance for v9.1 are less marked as the message size increases, but as a guide, with 64KB messages this workload benefits from a 60% decrease in transaction cost over v9.0 with 4MB messages benefitting from a 35% reduction in total transaction cost.

Chart: Transaction Cost - Request/Reply - Local workload



Notes on chart:

- The costs shown are based on the total CPU costs for the queue manager, AMS region and batch application regions, using data from the RMF Workload report, and divided by the number of transactions in the measurement.
- The transaction cost for the Integrity measurements reduced by 12% between v8.0 and v9.0.
- The transaction cost for the Privacy measurements reduced by 20% between v8.0 and v9.0
- The reduction in transaction cost for the Confidentiality measurements depends on the key reuse value selected.
  - Low key reuse values: v9.1 cost is 70% less than v9.0 equivalent
  - Higher key reuse values (32+): v9.1 cost is 83% less than v9.0 equivalent

The 2 charts on the following page show a breakdown of the transaction cost by address space, e.g. how much of the transaction cost is charged to the queue manager, AMS and application address spaces and how this has changed between MQ v9.0 and v9.1.

Chart: v9.0 - Breaking down the transaction cost by address space

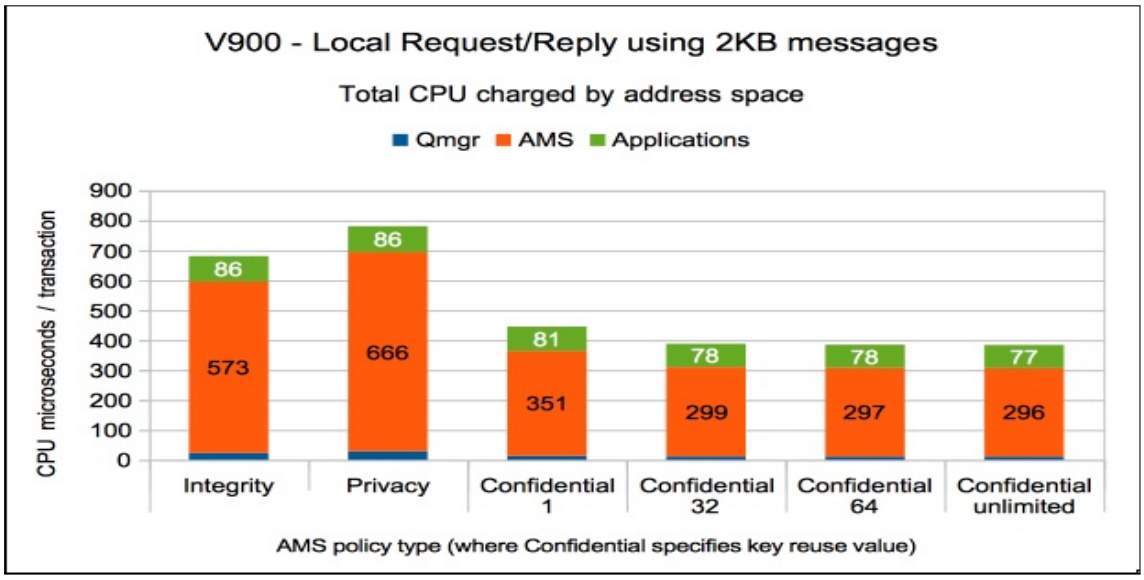
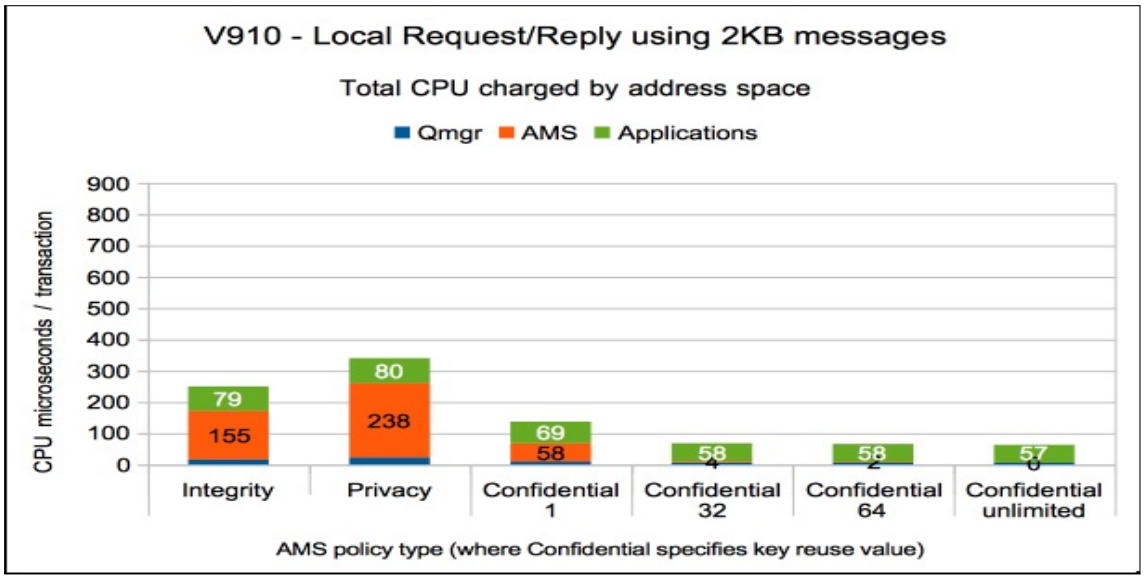


Chart: v9.1 - Breaking down the transaction cost by address space



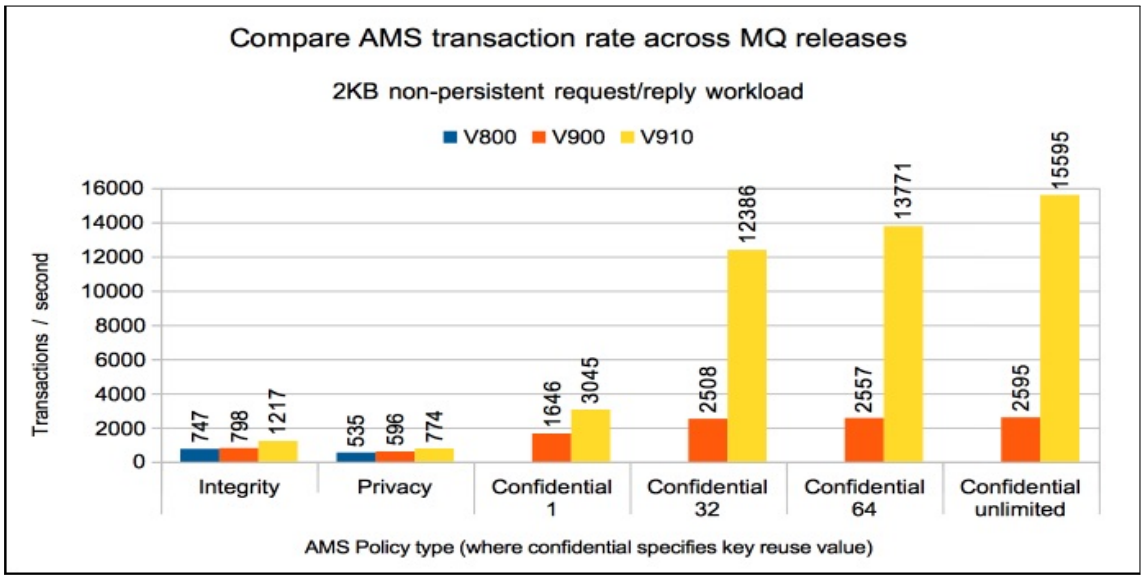
Note that both charts have the same scale on the y-axis (transaction cost).

Both AMS Integrity and Privacy see a much reduced cost in the AMS address space.

The AMS Confidentiality costs remain a significant proportion of the transaction cost in v9.0 even as the key reuse become unlimited, whereas in v9.1 the AMS address space become a much smaller factor in the transaction cost tending towards zero cost, particularly as the secret key is reused for more messages. Note that even with key reuse of "unlimited", there is still an impact to the application address space which is charged for the encryption and decryption of the message.

The final chart for the local request/reply workload shows the transaction rates achieved for the three AMS releases and the different policy types.

Chart: Transaction Rate - Request/Reply - Local workload

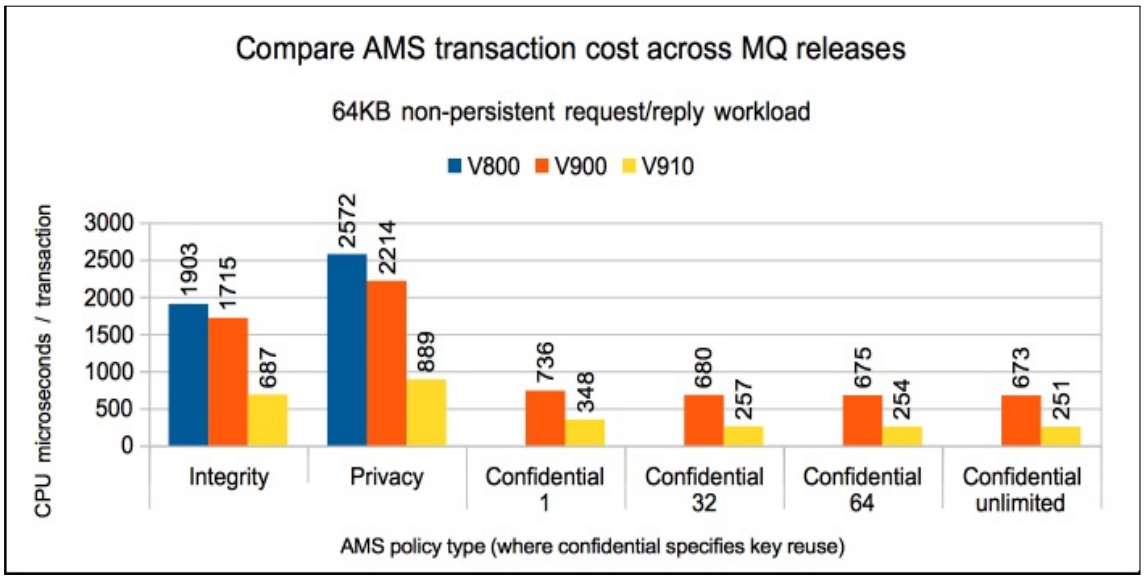


The AMS Confidentiality measurements show an increase in throughput of up to 6 times when moving from v9.0 to v9.1.

Smaller but still significant improvements can be seen with both AMS Integrity and Privacy qualities of protection.

The following 4 charts show the differences in performance for 64KB and 4MB messages.

Chart: Transaction Cost - Request/Reply - Local workload 64KB



Notes on chart:

- The transaction cost for the Integrity measurements reduced by 60% between v9.0 and v9.1.
- The transaction cost for the Privacy measurements reduced by 60% between v9.0 and v9.1.
- The reduction in transaction cost for the Confidentiality measurements depends on the key reuse value selected.
  - Low key reuse values: v9.1 cost is 53% less than v9.0 equivalent
  - Higher key reuse values (32+): v9.1 cost is 63% less than v9.0 equivalent

Chart: Transaction Rate - Request/Reply - Local workload 64KB

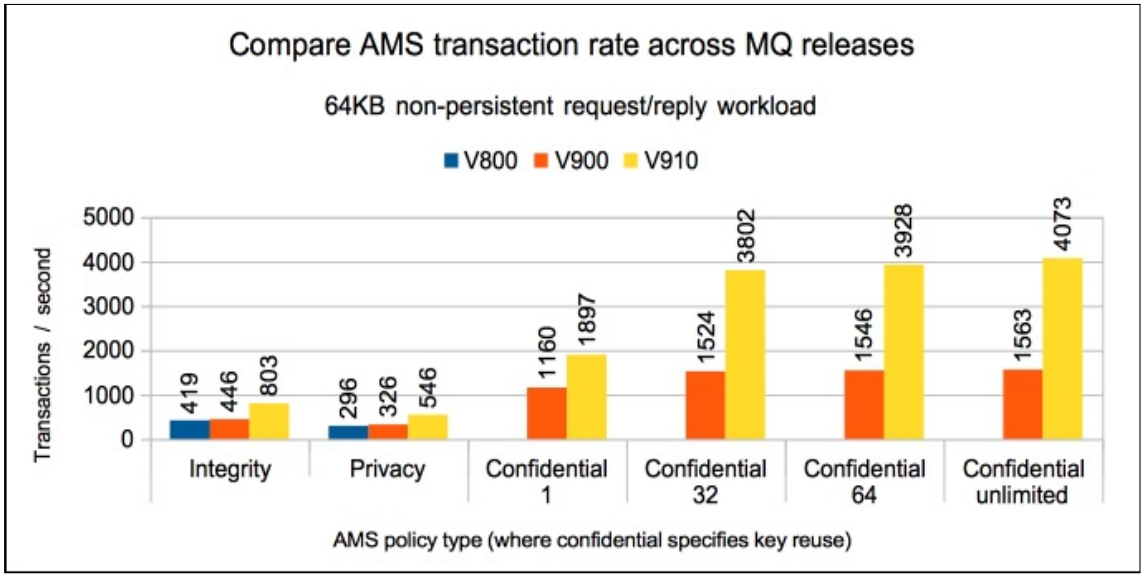
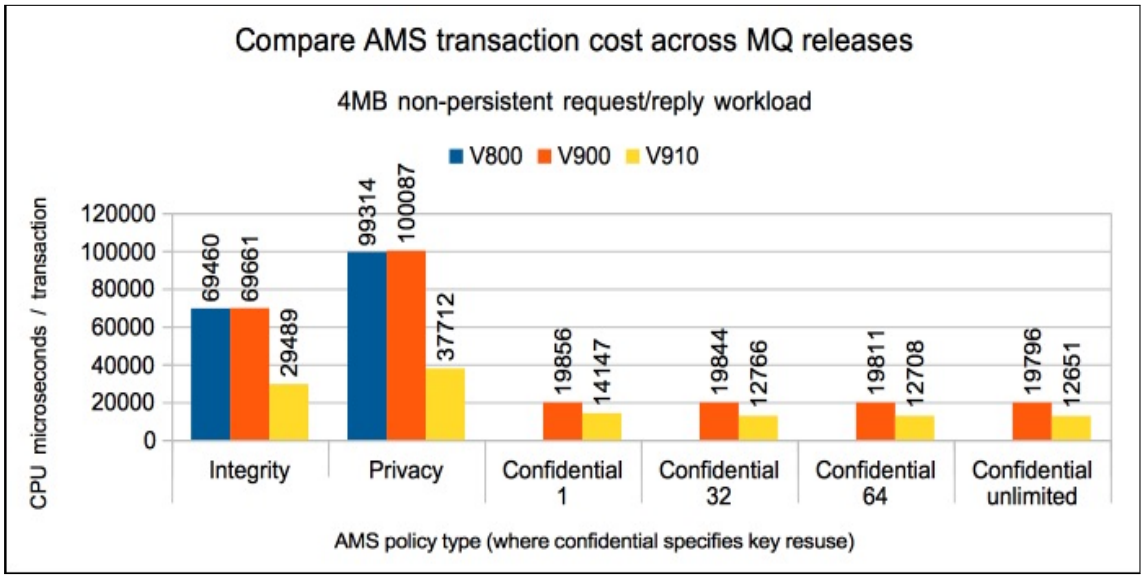


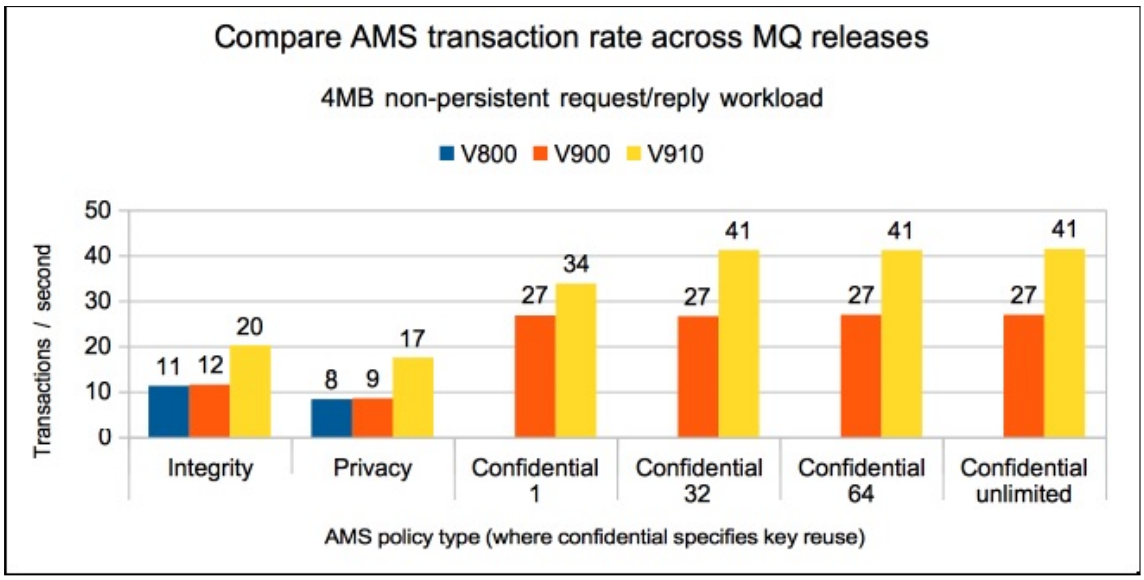
Chart: Transaction Cost - Request/Reply - Local workload 4MB



Notes on chart:

- The transaction cost for the Integrity measurements reduced by 58% between v9.0 and v9.1.
- The transaction cost for the Privacy measurements reduced by 63% between v9.0 and v9.1.
- The reduction in transaction cost for the Confidentiality measurements depends on the key reuse value selected.
  - Low key reuse values: v9.1 cost is 30% less than v9.0 equivalent
  - Higher key reuse values (32+): v9.1 cost is 36% less than v9.0 equivalent

Chart: Transaction Rate - Request/Reply - Local workload 4MB



Request / Reply - Mover workload

This section discusses the performance of the 3 AMS policy types, namely Integrity, Privacy and Confidentiality in a queue manager to queue manager environment. The Confidentiality measurements include key reuse values of 1, 32, 64 and unlimited.

A further comparison is made between AMS Confidentiality and channels that are protected using SSL ciphers where the TLS/SSL secret key negotiation (SSLRKEYC) ranges from 0 (only at channel start), to 1MB and 10MB.

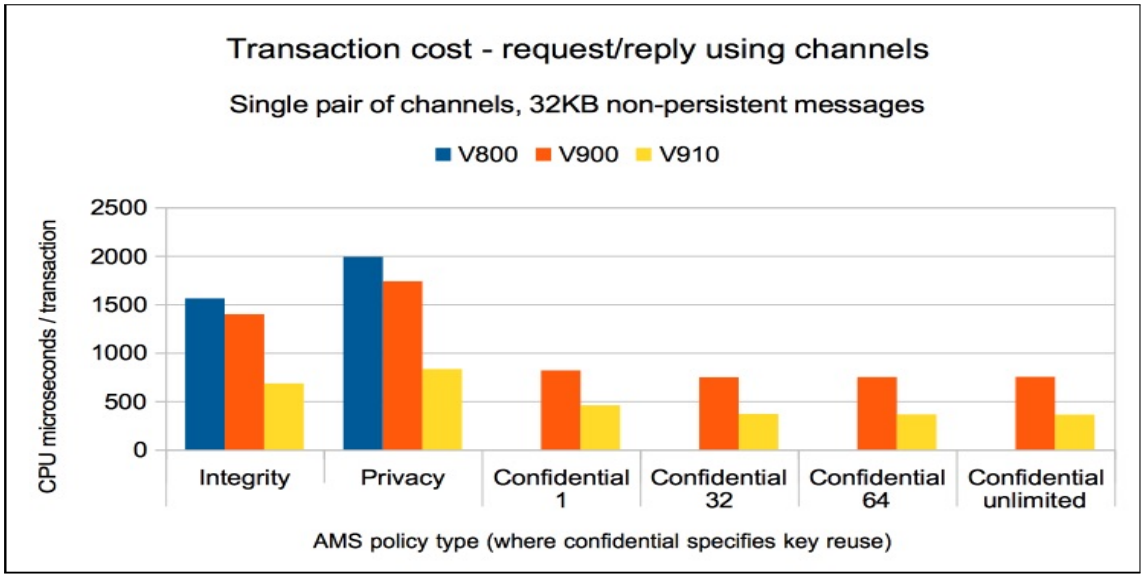
Comparing AMS policy types when moving messages over MQ channels

As was demonstrated in the request/reply local workload section, the improvements are not limited to AMS Confidentiality policies.

The following 2 charts offer a comparison in the performance of a request/reply workload using 32KB non-persistent messages between 2 queue managers on separate LPARs for MQ v8.0, v9.0 and v9.1.

For the purpose of clarity the measurements show the performance of a single outbound and a single inbound channel only.

Chart: Transaction Cost - Request/Reply - Mover workload

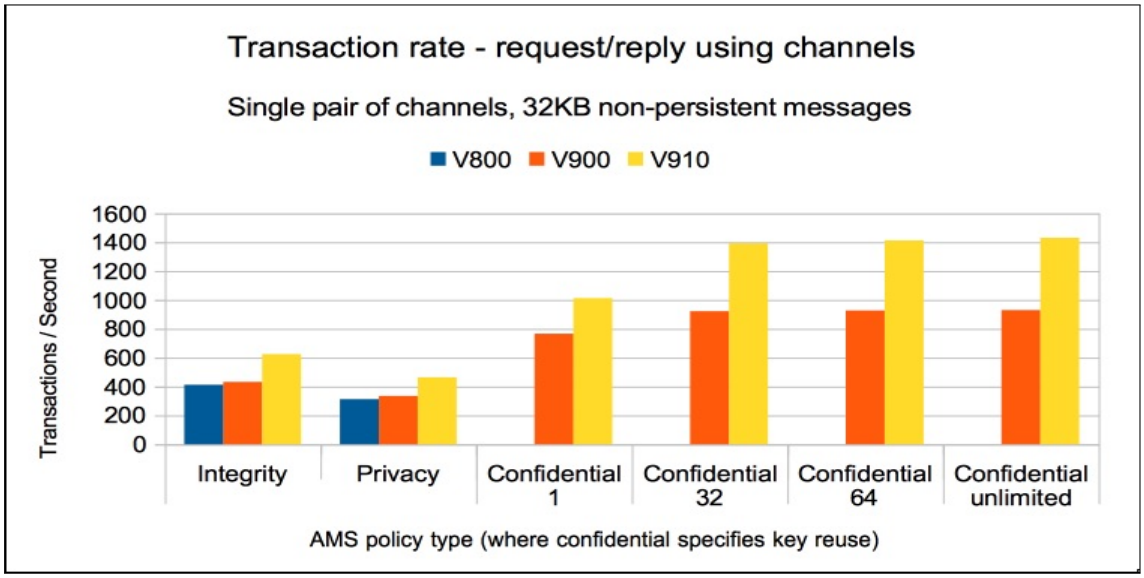


Notes on chart:

- The cost of the transaction include 2 MQPUTs with associated encryption cost and 2 MQGETs with associated decryption cost by the applications plus 2 MQGET and 2 MQPUTs by the channel initiators.
- In v9.1 for this workload, AMS Integrity costs have reduced by 56% compared to the equivalent v8.0 measurement and 51% compared to the equivalent v9.0 measurement.
- In v9.1 for this workload, AMS Privacy costs have reduced by 58% compared to the equivalent v8.0 measurement and 52% compared to the equivalent v9.0 measurement.
- In v9.1 for this workload, AMS Confidentiality costs have reduced by between 44 to 52% compared to the equivalent v9.0 measurement.

- As a guide, the equivalent measurement where no AMS policy is defined for the queues, the transaction cost is 240 microseconds.

Chart: Transaction Rate - Request/Reply - Mover workload



Notes on chart:

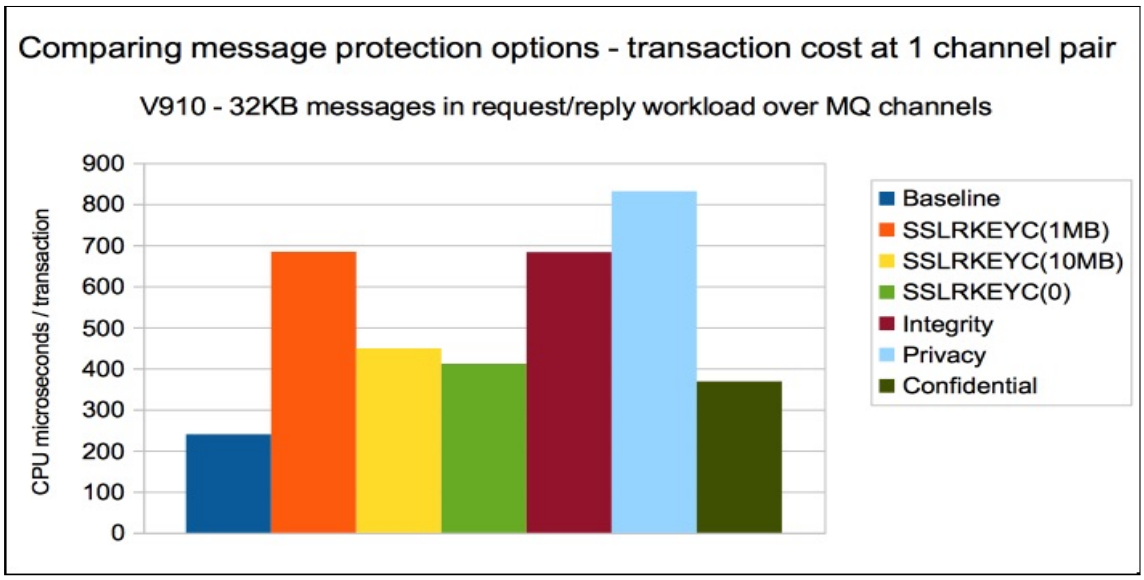
- The chart shows the transaction rate when using a single requesting task and a single server task.
- In v9.1 for this workload, AMS Integrity transaction rates have increased 51% compared to the equivalent v8.0 measurement and 44% compared to the equivalent v9.0 measurement.
- In v9.1 for this workload, AMS Privacy transaction rates have increased by 48% compared to the equivalent v8.0 measurement and 38% compared to the equivalent v9.0 measurement.
- In v9.1 for this workload, AMS Confidentiality transaction rates have increased by between 32 to 54% compared to the equivalent v9.0 measurement.
- As a guide, the equivalent measurement where no AMS policy is defined for the queues, a transaction rate of 1721 per seconds is achieved which is 21% higher than the AMS Confidentiality measurement with a key reuse of 64.

Comparing AMS with SSL ciphers

This section compares the performance of AMS Confidentiality with a key reuse of 32 against the performance of a workload with TLS cipher spec "ECDHE_RSA_AES_256_CBC_SHA384" with a range of secret key negotiation values.

The following charts offer a comparison in the transaction cost and rate of a request/reply workload using 32KB non-persistent messages between 2 queue managers on separate LPARs and compares a baseline (no message protection) with TLS encryption against queues protected using AMS Confidential with a key reuse of 32.

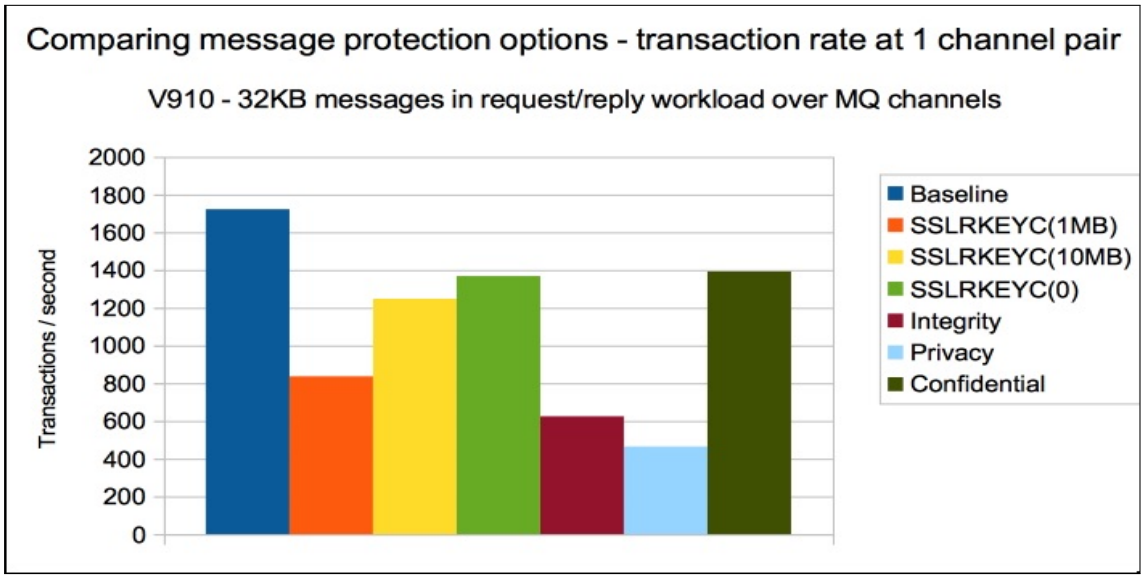
Chart: Transaction Cost - Request/Reply - Mover workload



Notes on chart:

- The chart shows the total transaction cost when using a single requesting task and a single server task.
- In v9.1, the AMS Confidentiality measurement shows transaction costs comparable with TLS measurements where the secret key is negotiated only at channel start (SSLRKEYC(0)).

Chart: Transaction Rate - Request/Reply - Mover workload



Notes on chart:

- The chart shows the achieved transaction rate when using a single requesting task and a single server task.
- In v9.1, the AMS Confidentiality measurement shows transaction costs comparable with TLS measurements where the secret key is negotiated only at channel start (SSLRKEYC(0)).

Protecting messages with AMS policies and TLS/SSL ciphers

This section discusses the impact of enabling TLS/SSL ciphers on MQ channels in conjunction with enabling AMS policies to the queues.

The data shown uses a single pair of sender-receiver channels between 2 queue managers on separate LPARs of the same sysplex. In each case, there are sufficient MQ resources (SSLTASKS, adaptors and dispatchers) and system resources, for example CPU and CryptoExpress features, such that the tests are not delayed for resource. Under increased workload, constraint for resource or contention may introduce additional cost and latency to the measurements.

The SSL cipher spec used is "ECDHE_RSA_AES_256_CBC_SHA384' and the SSLRKEYC secret key negotiation value is set to 0, i.e. only negotiate the secret key at channel start.

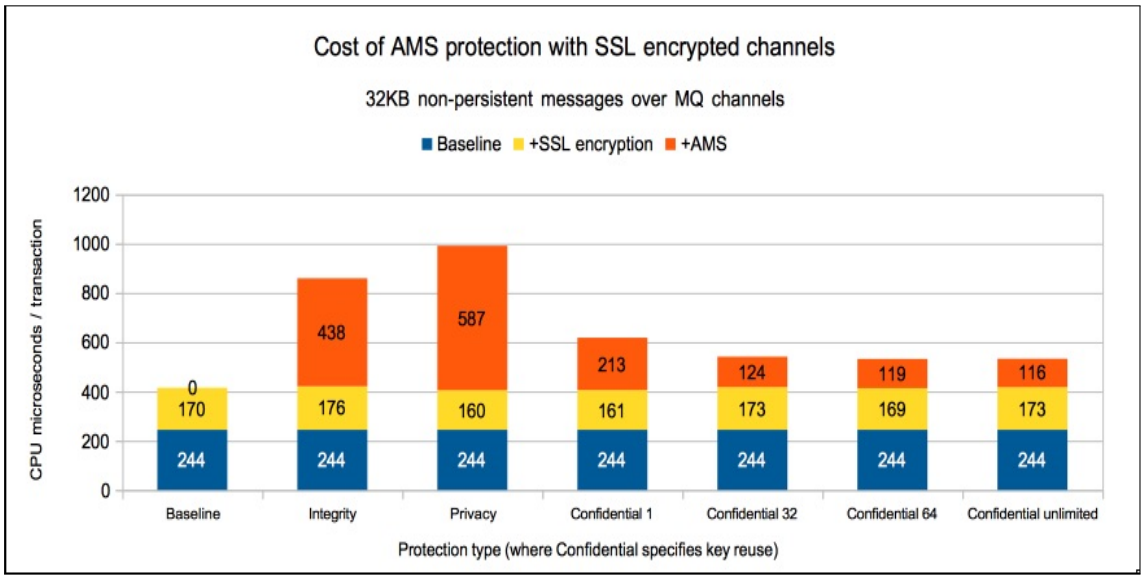
Using SSLRKEYC(0) reduces the load on the available CryptoExpress features, so it should be noted that when re-negotiating the secret key whilst concurrently using AMS Integrity or Privacy-type policies, the load on the CryptoExpress features may impact response time.

Whilst AMS policy-types Integrity, Privacy and Confidentiality can appear to make significant increases to transaction cost, consider a scenario where the data flowing over the channels need to cross (hop) over multiple queue managers, such as a gateway between the main office and a client system.

In this example, the TLS/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. With AMS protection, the message is only encrypted and decrypted once, so if a gateway queue manager is added, there is minimal additional cost.

However, as mentioned previously, if using AMS over MQ channels, it is advisable to protect the channels using SSL/TLS cipher specs to prevent "man in the middle" attacks on any data flowing over the channel that are using queues not protected by AMS policies.

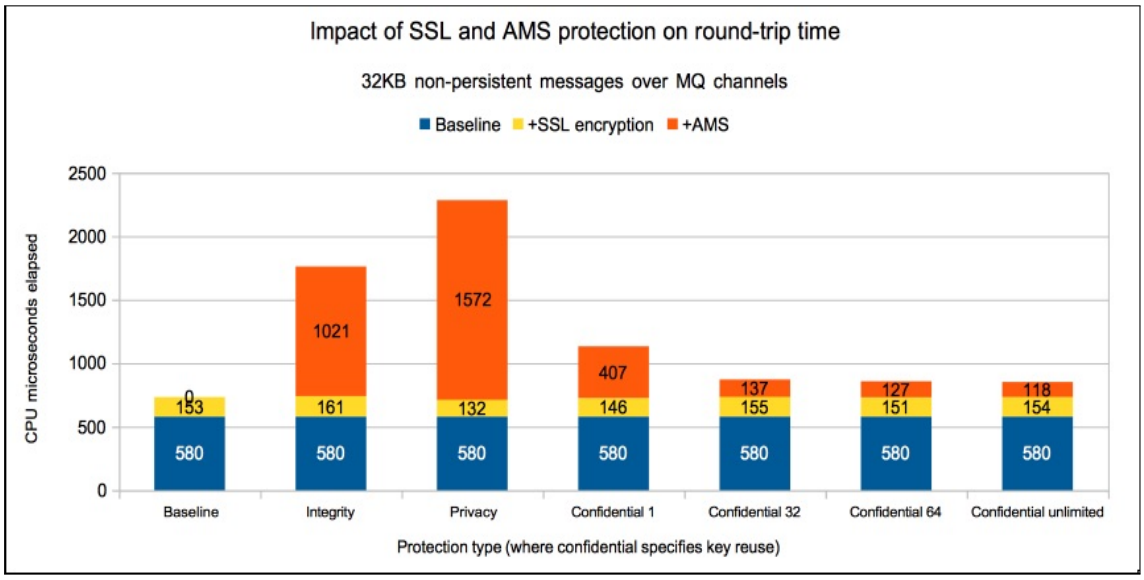
Chart: Impact to transaction cost - TLS and AMS on Mover workload



Notes on chart:

- The chart shows the baseline cost, i.e. running without any protection, as 244 CPU microseconds per transaction.
- The cost of enabling the TLS/SSL cipher with SSLRKEYC(0) adds between 160 to 176 CPU microseconds, for a total of approximately 410 CPU microseconds per transaction.
- The cost of enabling AMS Integrity approximately doubled the cost of the SSL measurement.
- The cost of enabling AMS Privacy increased the transaction cost 2.5 times to 1 CPU millisecond.
- The cost of enabling AMS Confidentiality with a key reuse of 32 or more added 30% to the cost of running with TLS.

Chart: Impact to round-trip time - TLS and AMS on Mover workload



Notes on chart:

- The chart shows the baseline round-trip time, i.e. running without any protection, as 580 CPU microseconds per transaction - which is significantly higher than the transaction cost and is largely due to network latency.
- The impact of enabling the TLS cipher with SSLRKEYC(0) results in the average round-trip time increasing to 740 CPU microseconds per transaction. As there is no key negotiation impacting the round-trip, this is largely due to encrypting and decrypting the messages.
- AMS Integrity adds 1 millisecond to the round-trip time, which is in part due to having to switch to the CryptoExpress feature to perform the digital signature work.
- AMS Privacy adds 1.5 milliseconds to the round-trip time, which is partly due to the digital signing and partly due to encrypting the data with a new key for each message.
- AMS Confidentiality with a key reuse of 32 or more adds between 118 and 137 CPU microseconds to the round-trip time and is largely in-line with the increased cost.

Streaming messages between queue managers

One use of the AMS Confidentiality quality of protection is where data is moved between data centres, such as an IBM InfoSphere Data Replication queue replication scenario.

The channels defined between the queue managers in the two data centres may be protected using TLS/SSL ciphers but it can be less expensive to encrypt the messages using AMS Confidentiality-type policies. However it is still advisable to consider the use of TLS protected channels in addition to AMS policies.

There are a number of considerations to take into account:

- SSL secret key negotiation and the generation of the AMS secret key costs are relatively static.

  What this means is that with a key reuse of 32, the impact of AMS key generation is spread across 32 messages, whether the message is 1 byte or 100MB, which means the impact is greater for small messages. By comparison with SSLRKEYC(32MB), the impact on the message cost is very dependent on the size of the message. For example 978 messages of 10KB could flow between negotiations based on SSLRKEYC(32MB), which is 30 times more messages than if using AMS with key reuse 32.

- Whether the data flows through other queue managers, for example a gateway queue manager.

  In the TLS/SSL configuration, the secret key would be negotiated between the source queue manager and the gateway queue manager, and again between the gateway queue manager and the target queue manager. In addition, the data would be decrypted whilst at rest on the gateway queue manager.

  Using AMS Confidentiality, the message payload would be encrypted once - before it is put to the source queue manager and remain encrypted until it was successfully gotten by the application on the target queue manager.

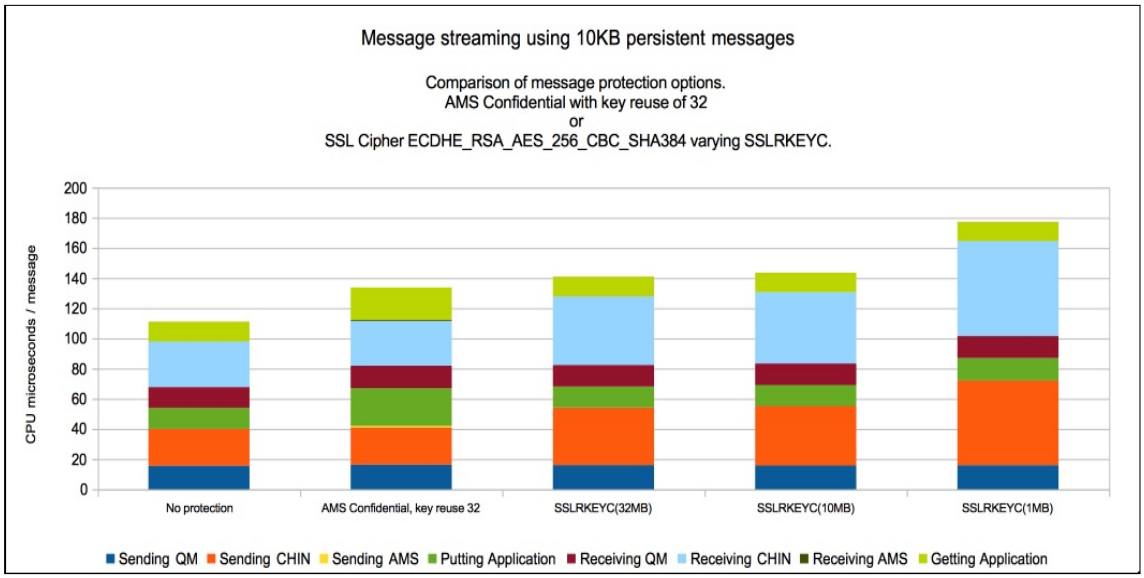
Streaming small (10KB) messages between queue managers The following chart demonstrates in a streaming type environment, the total transaction cost of protecting the messages using AMS Confidentiality is comparable to that when protecting the messages using TLS ciphers.

In the lowest cost measurements of both TLS and AMS Confidentiality, there is a total increase in transaction cost of approximately 20 to 25% over the cost of the workload when no message or channel protection is in place.

When considering the impact of AMS Confidentiality on only the MQ address spaces, there is an increase of 3 to 5% (including only MSTR, CHIN and AMSM). The majority of the transaction cost increase is in the application regions due to encrypting and decrypting the messages.

By contrast, adding TLS/SSL ciphers to the baseline measurement, results in an increase of 35% to the transaction cost in the MQ address spaces with SSLRKEYC(32MB), and significantly more with a higher key negotiation frequency.

Chart: Streaming 10KB messages between 2 queue managers



Notes on chart:

- The chart shows the cost per transaction by address space for a 10KB message streaming-type workload.
- With 10KB messages, the AMS costs are equivalent to those of the TLS protected channels.

  The AMS Confidentiality workload (key reuse 32) is 5% lower cost than the TLS protected channel using SSLRKEYC(32MB), despite TLS transferring more than additional 3200 messages using the same key.
- For the AMS measurements, an impact in transaction cost is seen:
  - primarily in the application address space, which incurs the cost of encrypting and decrypting the data
  - in the AMS address space which diminishes with an increasing key reuse value.
  - in the queue manager and channel initiator, which see a small increase due to the message size increasing due to being protected by AMS policies.
- The TLS measurements see an increase in the channel initiator address space, which includes the cost of encrypting and decrypting the data plus the cost of negotiating the secret key.

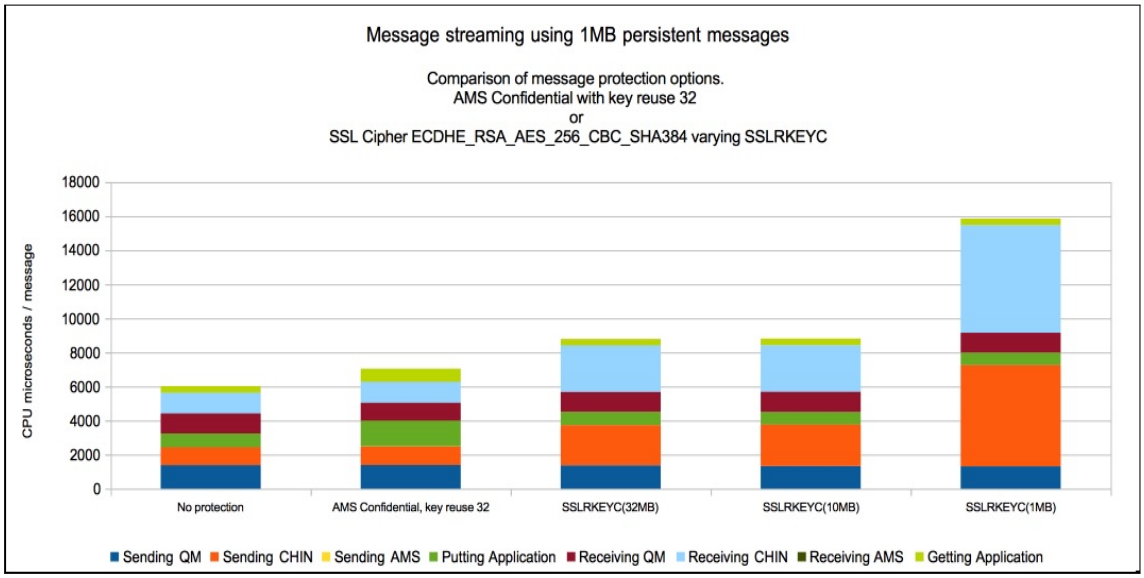
With an increased value in the SSLRKEYC attribute, the impact on each message of negotiating the secret key can be reduced.

Note, that some of the secret key negotiation cost has been offloaded to the available Crypto Express card.

Streaming large (1MB) messages between queue managers The following chart compares the impact of AMS Confidentiality with key reuse of 32, against channels protected using TLS ciphers that negotiate the secret key either every 1MB, 10MB or 32MB.

Where similar numbers of messages flow between key negotiations, namely AMS Confidentiality with key reuse 32 and SSL channels with SSLRKEYC(32MB), the AMS measurement has a transaction cost of approximately 20% less.

Chart: Streaming 1MB messages between 2 queue managers



Notes on chart:

- AMS key reuse 32 shows a 17% cost increase on the "no protection" measurement.
- SSLRKEYC(32MB) shows an additional 24% increase on the AMS Confidentiality measurement, or a 46% increase in the "no protection" measurement.
- AMS Confidentiality shows similar costs in the MSTR and CHIN address spaces to the "no protection" option, but an increase in the application address spaces.
- TLS shows an increase in the CHIN address spaces compared with both the "no protection" and the AMS Confidentiality measurements.

Java clients on z/OS

IBM MQ Advanced for z/OS VUE version 9.1 only

MQ version 9.1 introduces support for Java client applications running on z/OS that connect to IBM MQ Advanced for z/OS VUE version 9.1 queue managers.

This section discusses the impact of running JMS applications in client and bindings mode on z/OS, with regards to the transaction cost and rate.

The reported tests use a simple long-running JMS application that repeatedly put and then get a 2KB non-persistent message from one of a set of queues. Additional instances of the application are added as time progresses. The test is run in both bindings mode, where the applications are running in the same LPAR as the MQ queue manager, and client mode, where the applications are run in a separate LPAR (same sysplex) as the MQ queue manager and connect using SVRCONN channels. Each LPAR is configured with 10 dedicated processors.

What is the impact of using clients on z/OS?

When using clients on z/OS to connect to an MQ queue manager, there will be additional cost incurred in the MQ channel initiator address space, plus there may be additional latency due to the time spent in network transport.

The impact of the increased latency due to network transport may be reduced by using some of the available IBM Z network features such as SMC-R (Shared Memory Communications over RDMA) or SMC-D (Shared Memory Communications - Direct).

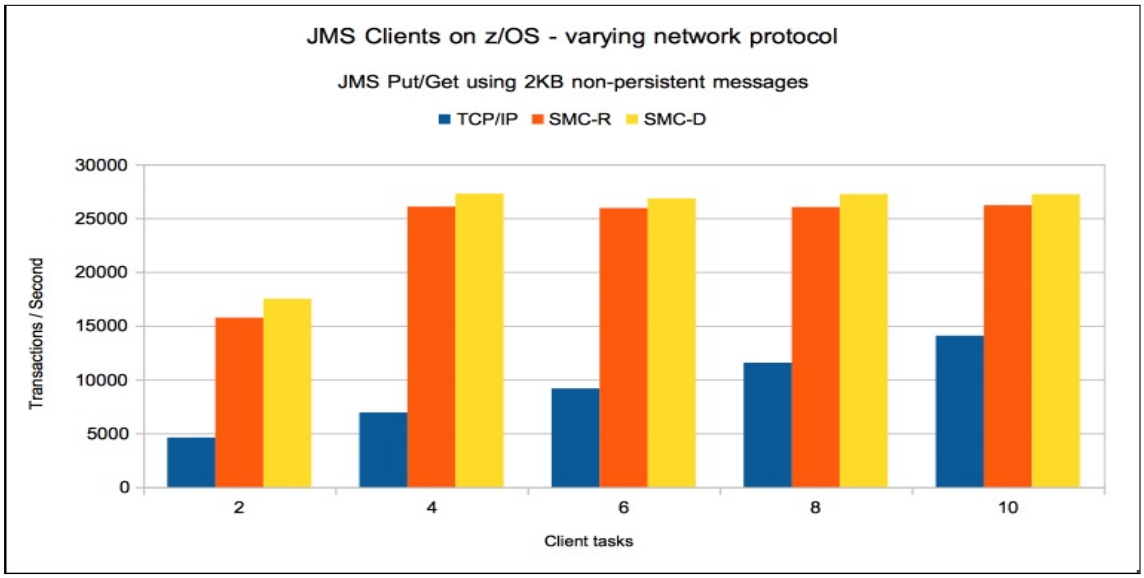
How much difference can the network protocol make?

The use of SMC-R or SMC-D, where appropriate, can significantly reduce the time spent in the network layer.

The following chart shows the achieved transaction rate for 3 configurations

- TCP/IP over a 10GbE link
- SMC-R
- SMC-D

Chart: JMS clients on z/OS - Impact of network transport type



Notes on chart:

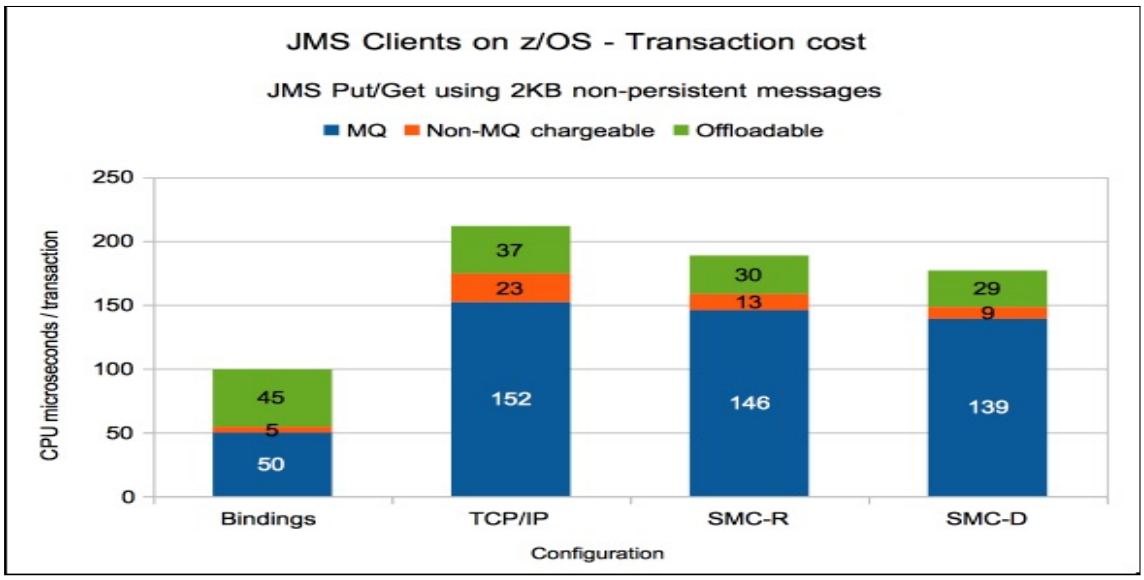
- The peak number of transactions achieved using the 10Gb TCP/IP link reached 14,000 transactions per second, but was still increasing at a rate of approximately 1,100 transactions per second for each application added after the initial workload of 4,600 transactions per second with 2 clients.
- When using SMC-R, the peak rate reached 26,000 transactions per second but then did not increase as more clients were added. Neither LPAR was constrained for CPU.
- When using SMC-D, the peak rate rapidly reached a peak of 27,500 transactions per second, but as with SMC-R, did not increase as more clients were added. Given that neither LPAR was CPU constrained, it is likely that this is the best throughput that this particular workload is able to achieve.

Client or bindings: Transaction cost on z/OS

In the following example we compare the cost of using the same application in bindings and then client mode.

The application used is a simple JMS put/get application, which is either run in the same LPAR as the MQ queue manager, or in a separate LPAR and connecting via a SVRCONN channel. In all cases the application connects and opens the application queues once and then runs many MQPUT and MQGET of a 2KB non-persistent message. This model means that the cost of the MQ connect is relatively small for the workload.

Chart: Comparing the cost of JMS applications on z/OS in bindings and client mode



Notes on chart:

- The MQ cost includes the MQ queue manager and channel initiator cost.
- The non-MQ chargeable cost is largely network costs, which in the client case is included for the LPAR with the client and the LPAR with the MQ queue manager. There is also a small overhead from running z/OS.
- The offloadable costs (to zIIP specialty processors) are largely attributed to the JMS put/get application.

For an indication of the cost of the MQ connect see blog "The cost of connecting to a z/OS queue manager" which discusses the cost of a client connecting to a z/OS queue manager in a range of configurations.

The MQ queue manager and channel initiator costs are similar to those documented in MP16 "Capacity Planning and Tuning Guide", section "Two / Three Tier Configurations".

What about AMS JMS clients on z/OS?

In the addition to the allowing of MQ clients on z/OS in IBM MQ Advanced for z/OS VUE version 9.1, there is also support for JMS clients running on z/OS that access AMS-protected queues.

Unlike JMS (or other language) applications running in bindings mode, that are able to exploit IBM Z cryptographic hardware features, the AMS JMS client support uses the "bouncy castle" jar files to perform the message protection at the client, which results in the additional protection being performed in software. This can have a significant effect on the performance of an application when moving from bindings to client-type connections. However, a large proportion of this increase in cost can be offloaded to zIIP specialty processors.

The following sections compare the performance of the 3 levels of AMS protection in bindings and client mode.

In each case the simple JMS put/get application described earlier in the Clients on z/OS section is reused.

For all measurements in this section, both client LPAR and the LPAR hosting the MQ queue manager are defined with 10 dedicated CPUs.

AMS Integrity in client mode on z/OS: The AMS Integrity measurements use queues protected with digital signature algorithm SHA256. In our measurements, digital signature algorithms MD5 and SHA1 showed similar performance characteristics to SHA256.

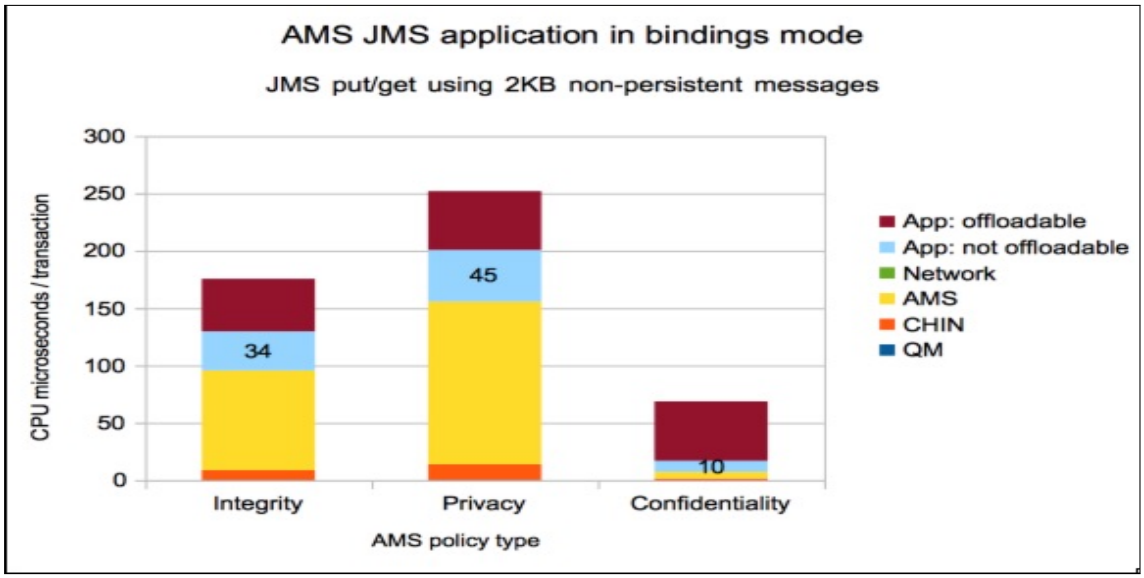
AMS Privacy in client mode on z/OS: The AMS Privacy measurements use queues protected with digital signature algorithm SHA256 and encrypted using AES256. In our bindings measurements, using digital encryption algorithm RC2 gave up 20% worse performance than DES, 3DES, AES128 or AES256, however this was not apparent in the client measurements.

AMS Confidentiality in client mode on z/OS: The AMS Confidentiality measurements use queues protected with digital encryption algorithm AES256 with a key reuse count of 32.

Comparing AMS JMS clients in bindings and client mode

The measurements using bindings connections were constrained by CryptoExpress limits - the tests drove the workload until either the available Accelerator or Co-Processors were running at full capacity. With test using JMS applications to run the workload, there was reasonable scope for offload to zIIP specialty processors.

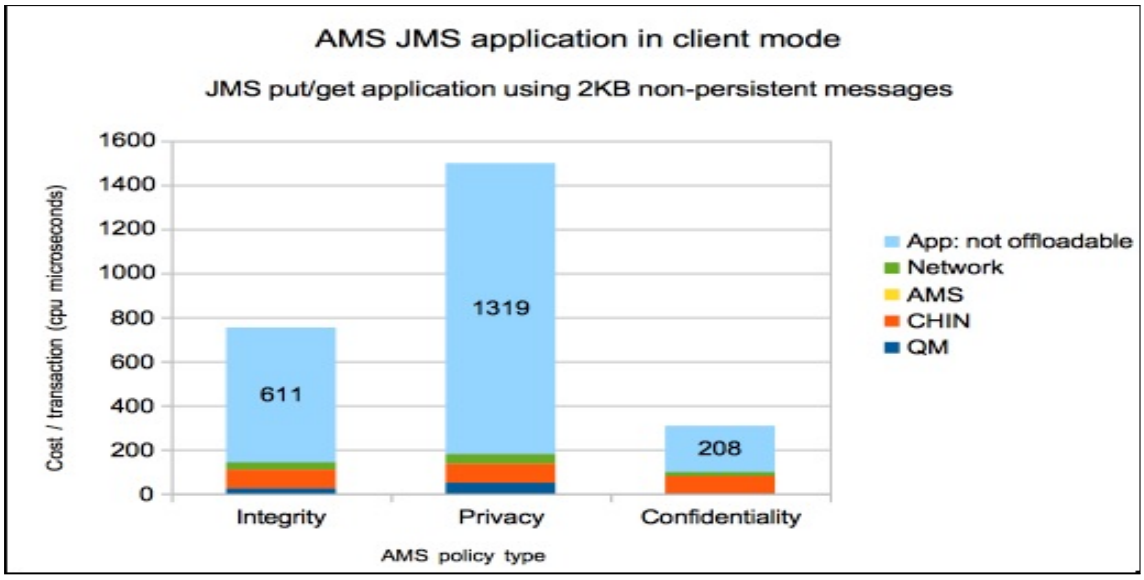
Chart: Cost of AMS JMS bindings application



The measurements using client connections were CPU constrained due to performing the cryptographic work in the software, although the large majority of the cost was eligible for offload to zIIP specialty processors.

The following chart shows a breakdown of the cost of the transaction but excludes the cost of the application that is eligible for offload.

Chart: Cost of AMS JMS client application



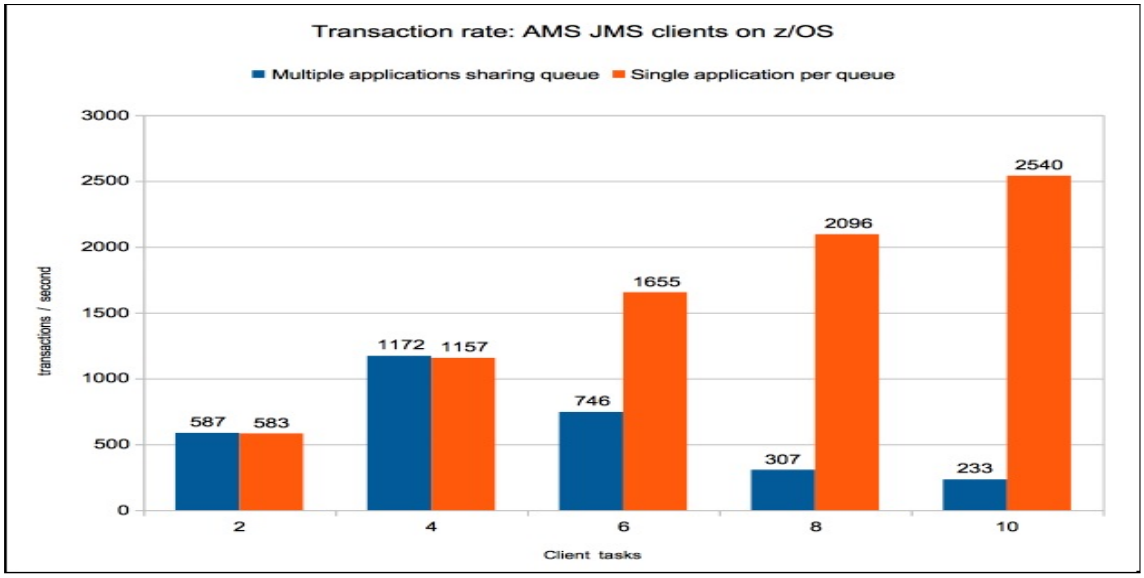
In terms of the offloadable costs for the AMS JMS client measurements, we saw:

- Integrity: 98700 microseconds per transaction.
- Privacy: 197000 microseconds per transaction.
- Confidentiality: 3200 microseconds per transaction.

With regards to transaction rate, the AMS Integrity and Privacy measurements were directly impacted by the high transaction cost, i.e. Integrity was able to achieve 10 transactions per second for each client task, whereas Privacy was able to achieve 5 transactions per second for each client task.

The AMS Confidentiality client performance achieved significantly higher throughput as demonstrated in the following chart.

Chart: Transaction rate for AMS JMS client using queues protected with Confidentiality policy



Notes on chart:

- The data named "Multiple applications sharing queue" used 4 application queues protected by AMS Confidentiality policies. Once more than 4 client tasks were running, queues were being shared, meaning that key reuse was less efficient.
- The data named "Single application per queue" used up to 10 application queues protected by AMS Confidentiality policies, i.e. one distinct queue per task.
- AMS Confidentiality key reuse works best when there is only one putter/getter per queue, as demonstrated by the difference upwards of 4 clients.

Appendix A

Regression

When performance monitoring both IBM MQ Advanced for z/OS VUE version 9.1 and IBM MQ for z/OS version 9.1, 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 V8.0 (V800), V9.0 (V900) and V9.1 (V910) 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 z14 (3906-7E1) 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

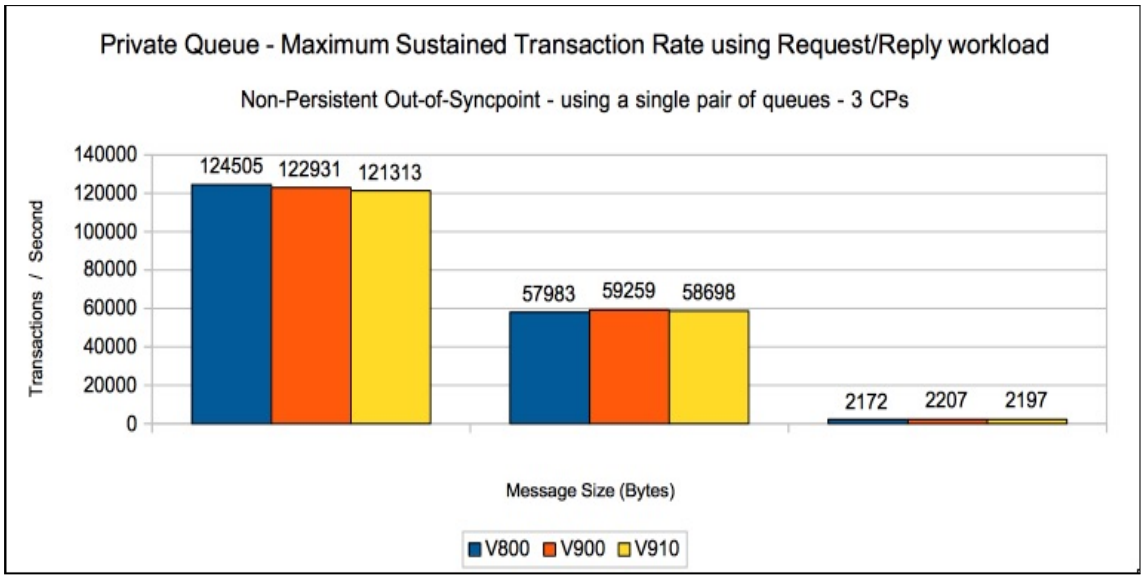
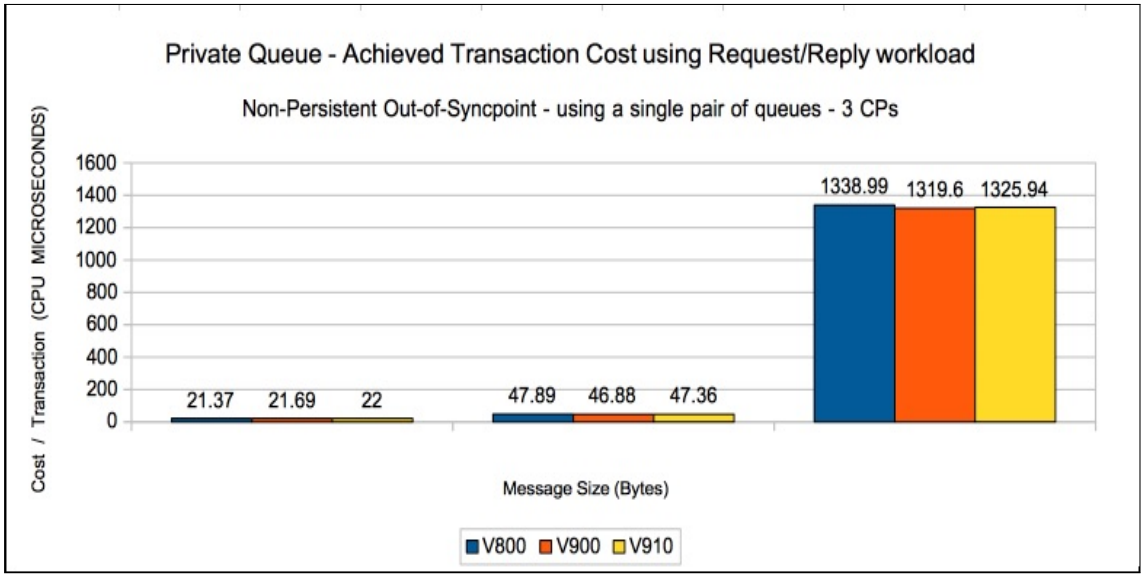


Chart: Transaction cost for non-persistent out-of-syncpoint workload



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 z14 (3906) used for testing.

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

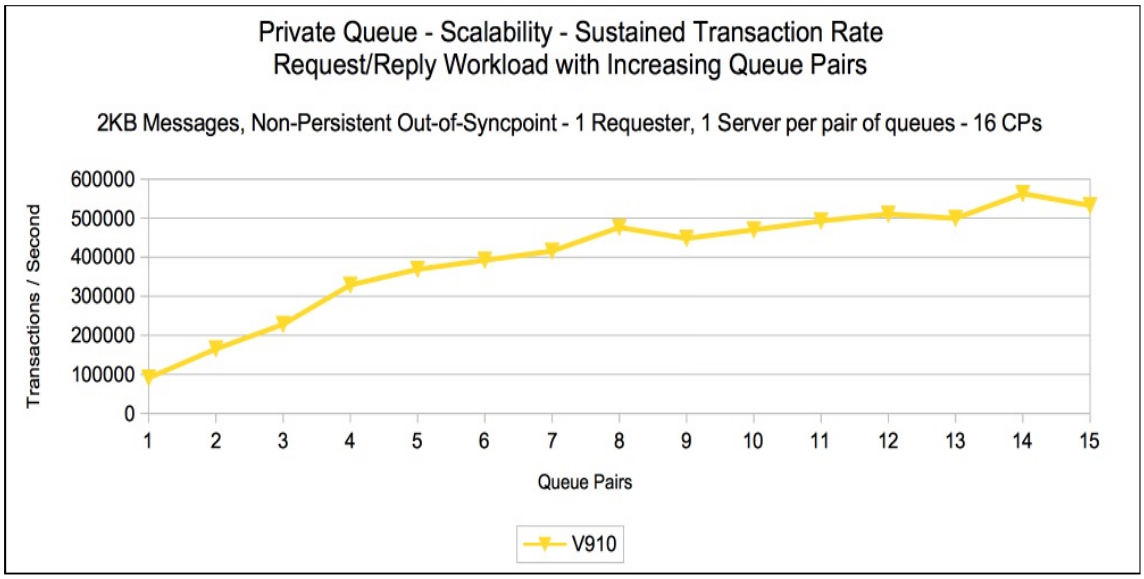
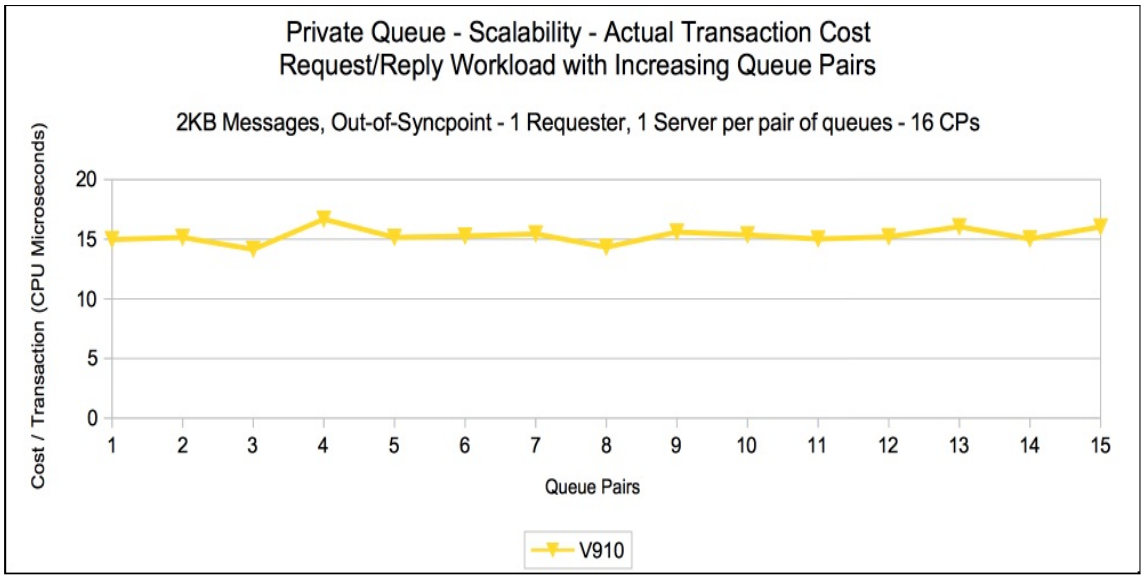


Chart: Transaction cost for non-persistent out-of-syncpoint scalability workload



The preceding 2 charts show that a single queue manager is able to drive in excess of 560,000 transactions per second - or 1,120,000 non-persistent messages per second on a 16-way LPAR.

Note: Since the performance for V800, V900 and V910 is comparable, the charts show V910 data for the purpose of clarity.

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

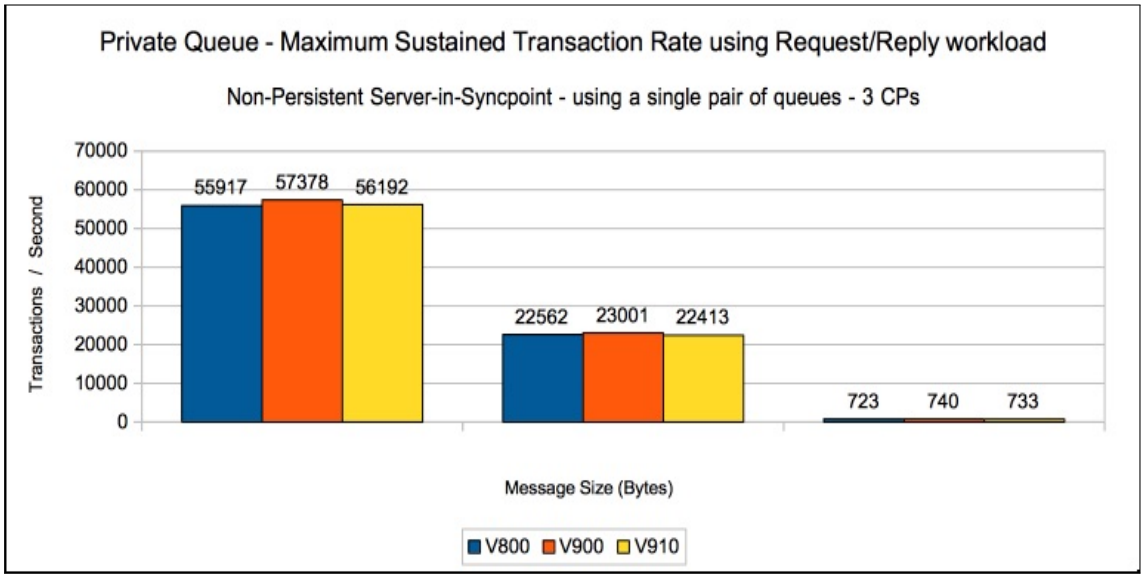
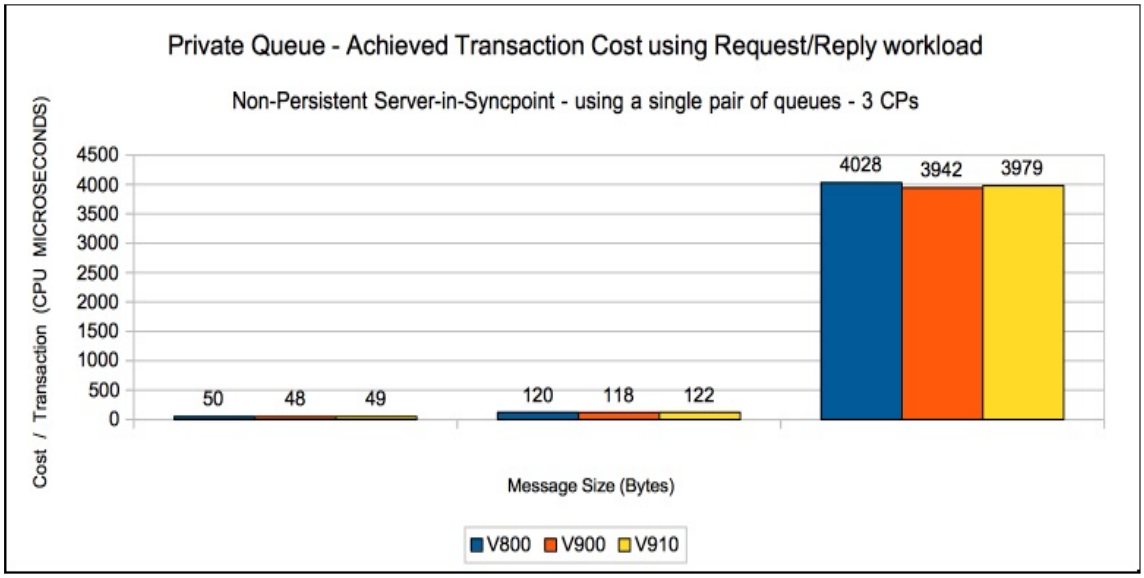


Chart: Transaction cost for non-persistent in syncpoint workload



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 z14 (3906) 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.

Note: Since the performance for V800, V900 and V910 is comparable, the charts show V910 data for the purpose of clarity.

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

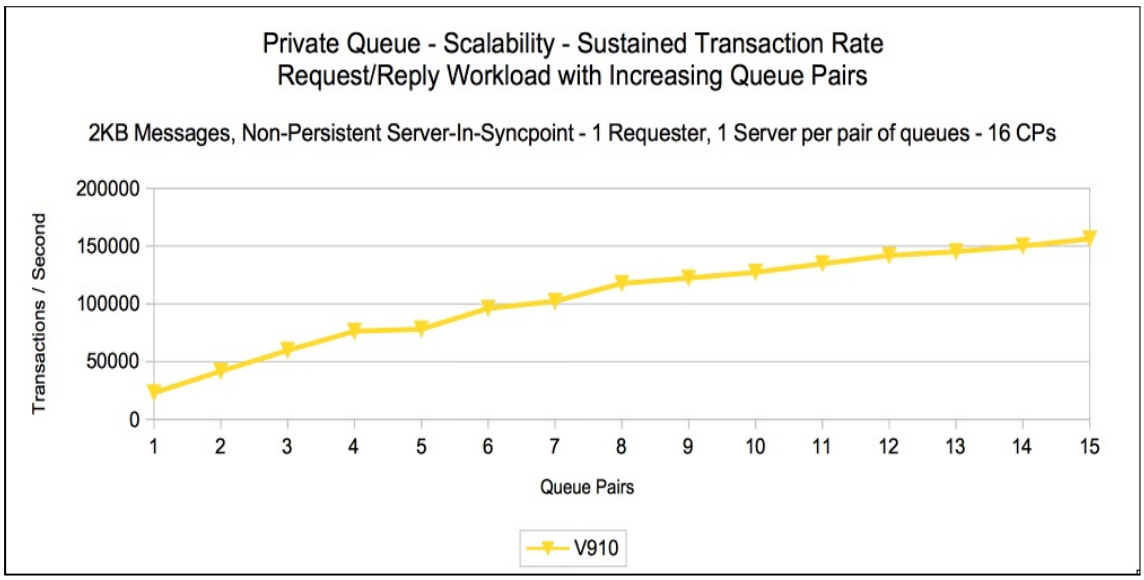


Chart: Transaction cost for non-persistent in syncpoint scalability workload with 2KB messages

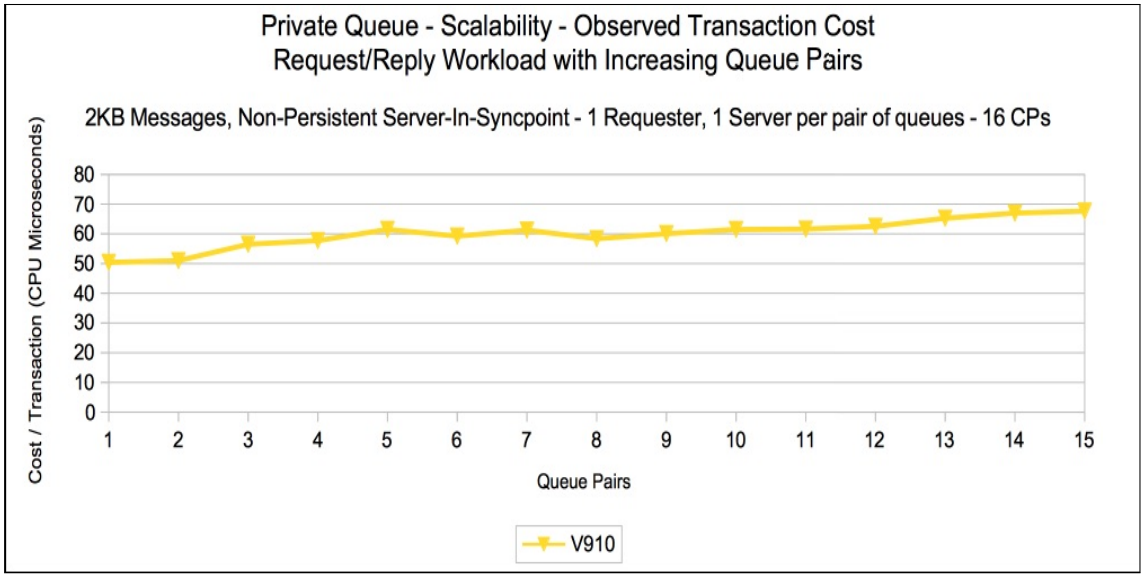


Chart: Transaction rate for non-persistent in syncpoint scalability workload with 64KB messages

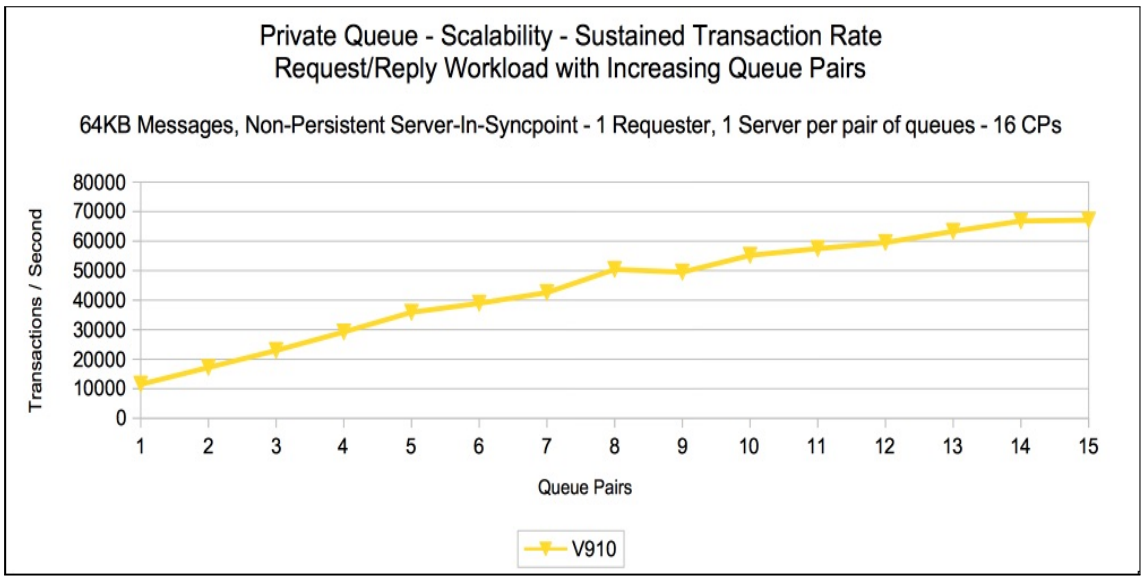


Chart: Transaction cost for non-persistent in syncpoint scalability workload with 64KB messages

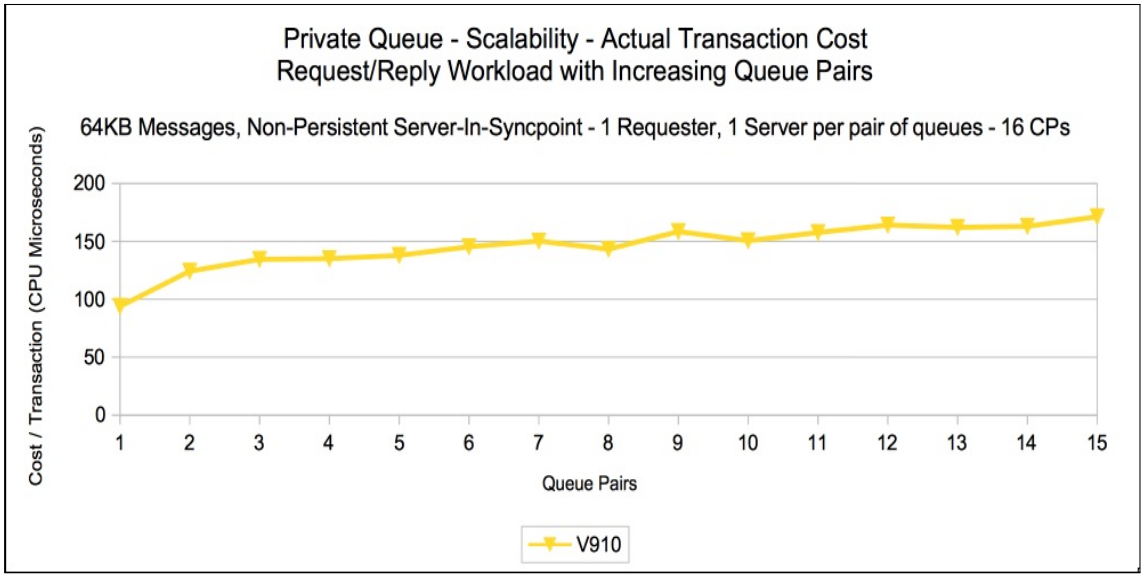


Chart: Transaction rate for non-persistent in syncpoint scalability workload with 4MB messages

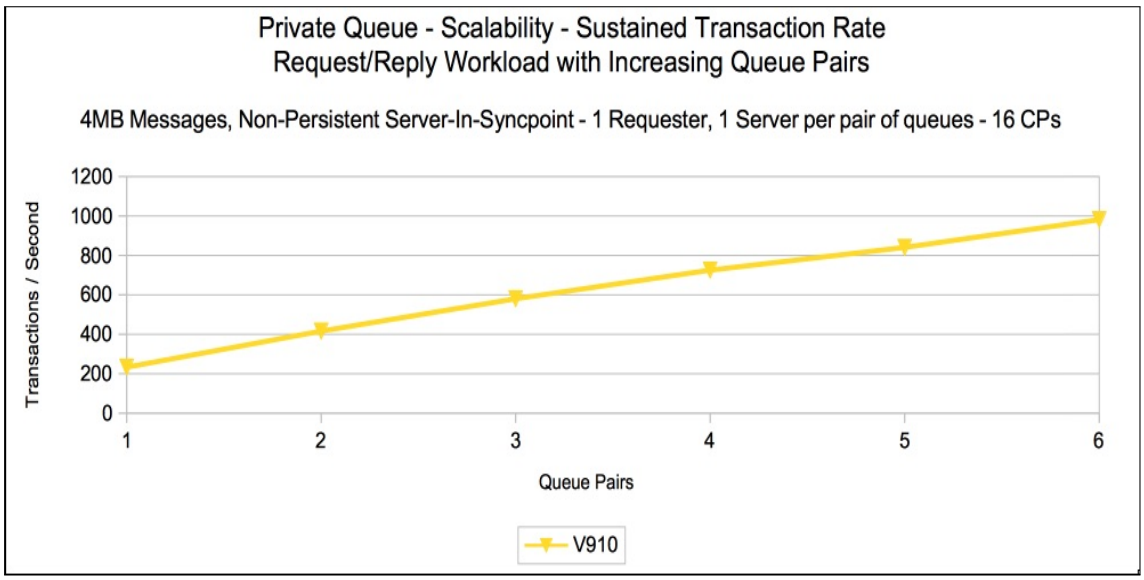
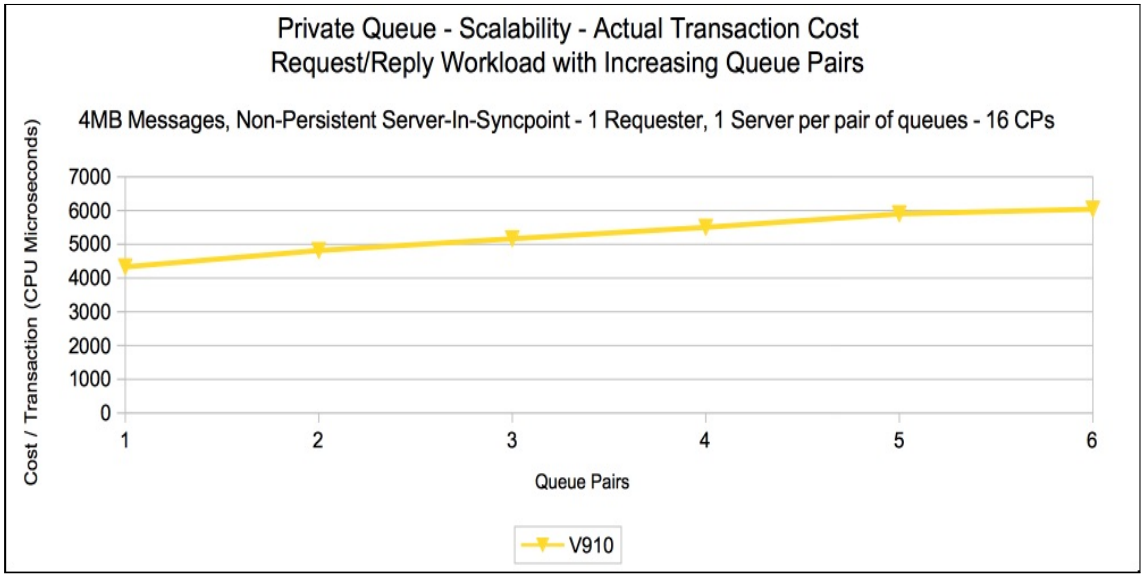


Chart: Transaction cost for non-persistent in syncpoint scalability workload with 4MB messages



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

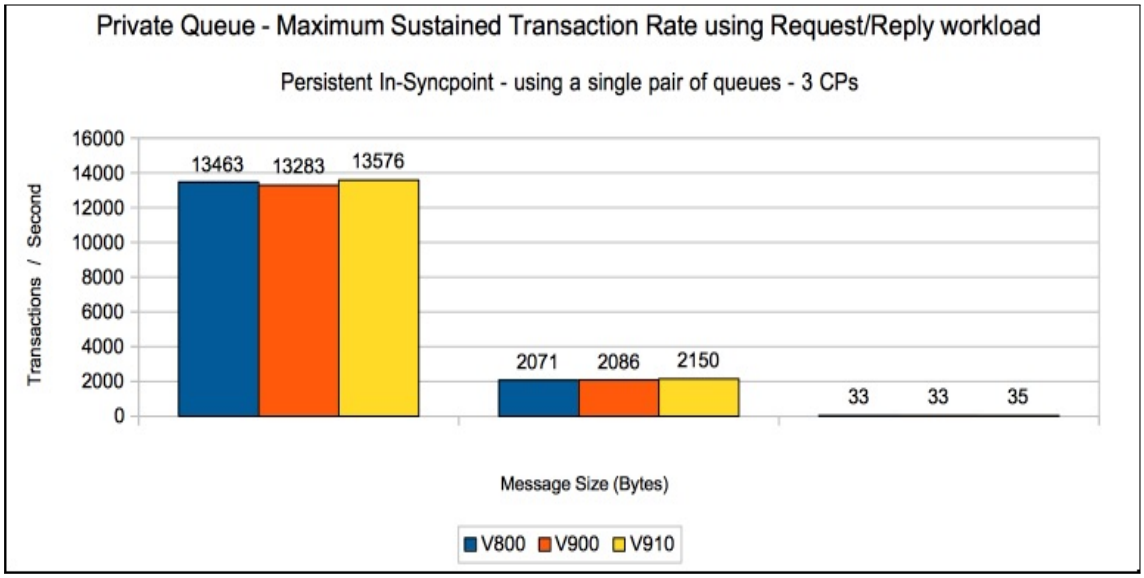
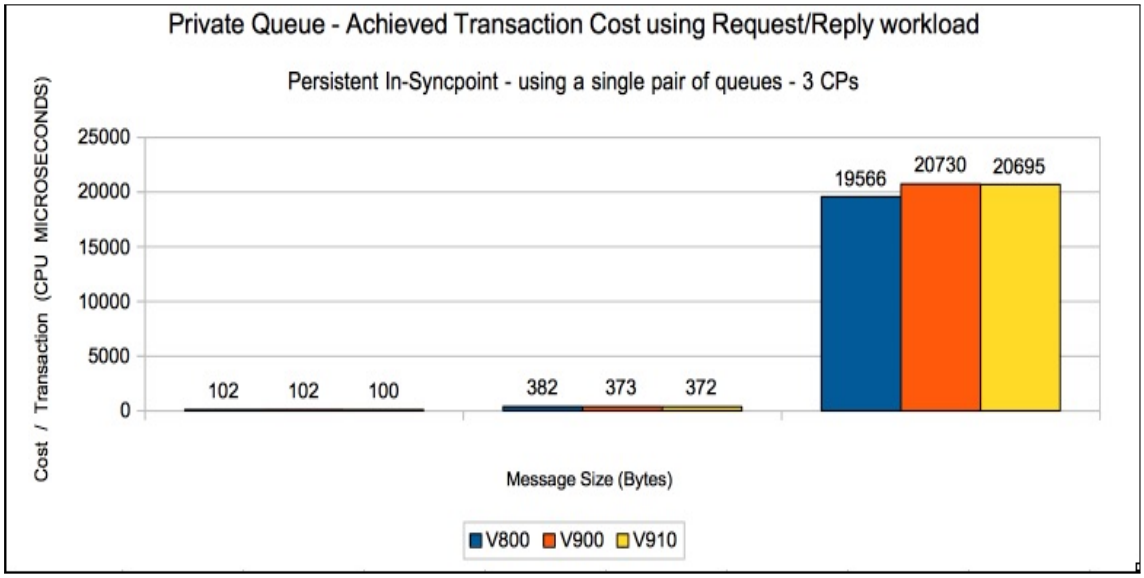


Chart: Transaction cost for persistent in syncpoint workload

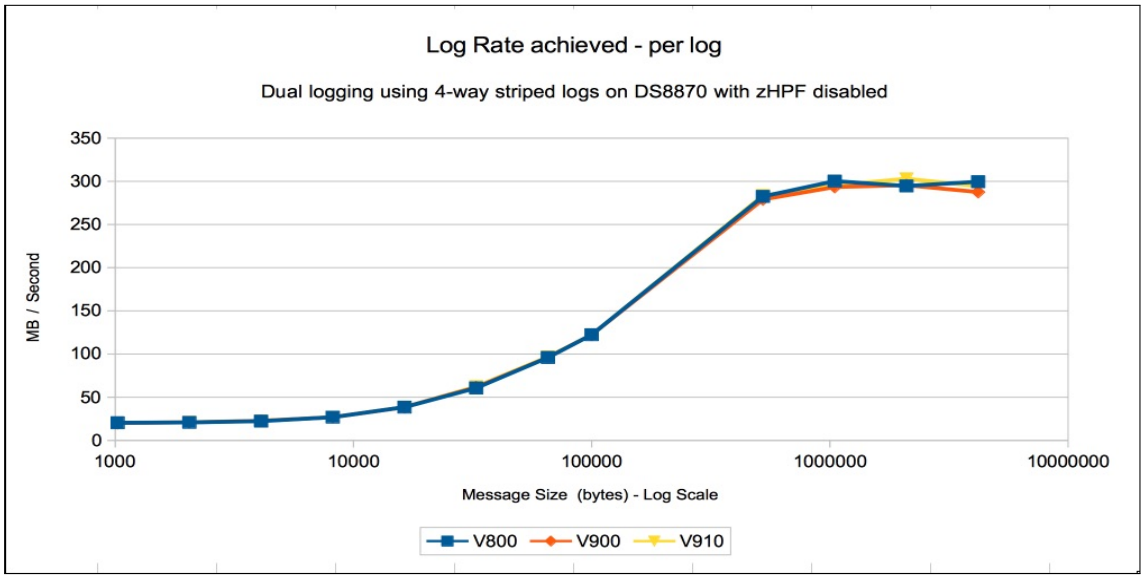


Upper bounds of persistent logging rate

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 z14.

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.

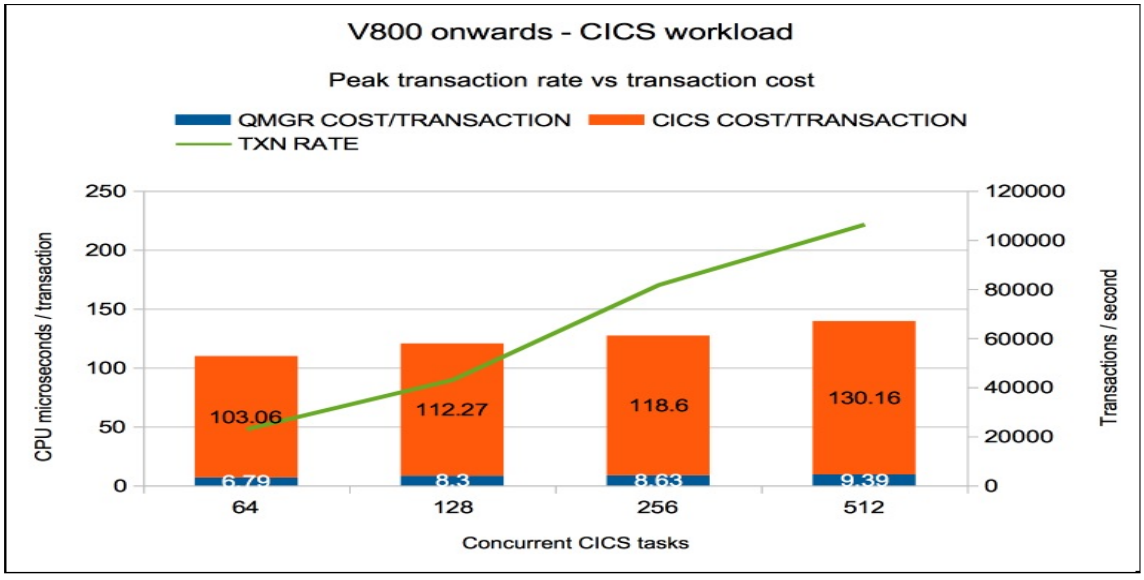
The following chart uses a set of simple CICS transactions that open queues, put a 2KB non-persistent message, get a 2KB non-persistent message, close queues and initiate a new CICS transaction. Running multiple sets of these transactions simulates a number of concurrent transactions across multiple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

In these measurements, the system is running with 16 dedicated processors and becomes CPU constrained with 512 concurrent CICS tasks.

Notes on chart:

- For versions 8.0.0 onward, the queue manager cost associated with the CICS workload is 6.79 CPU microseconds and rises at a similar rate as the workload increases.
- As the V800, V900 and V910 data is similar, only V910 data is shown.

Chart: V910 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.

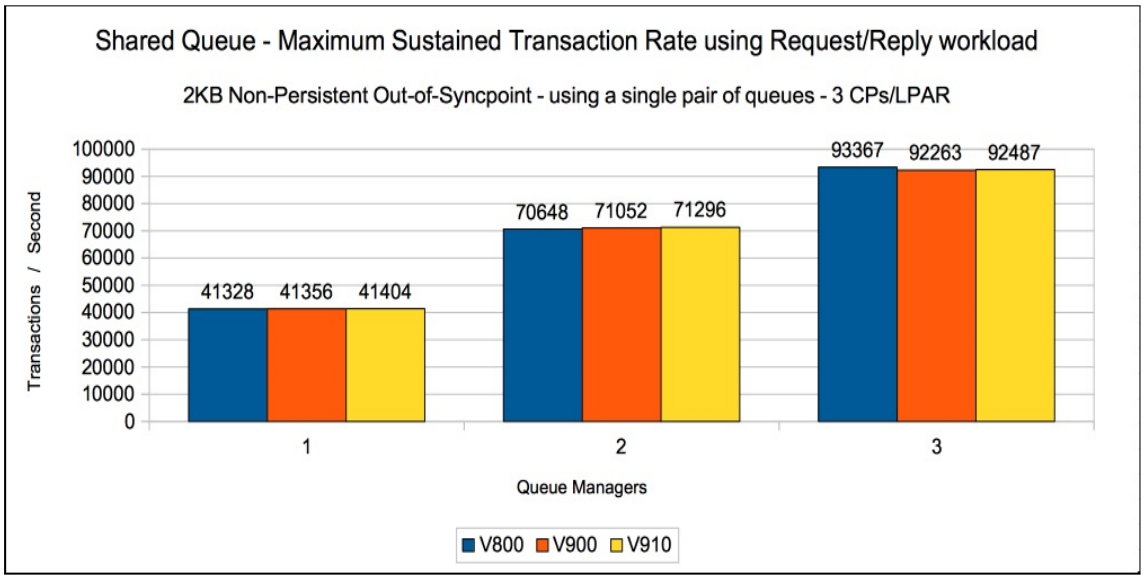


Chart: Transaction cost for non-persistent out-of-syncpoint workload - 2KB messages.

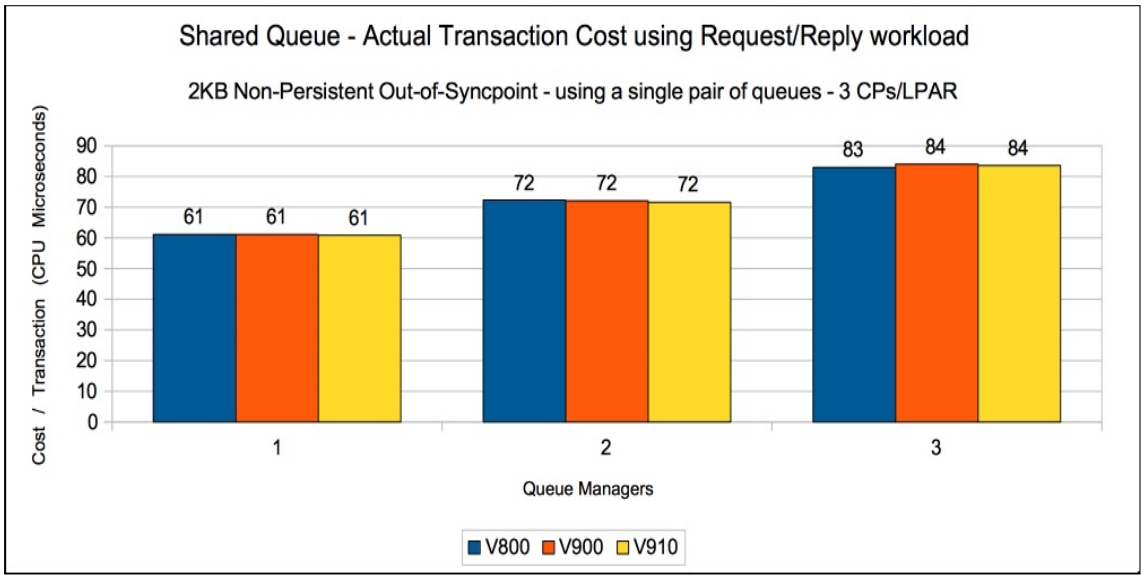


Chart: Transaction rate for non-persistent out-of-syncpoint workload - 64KB messages.

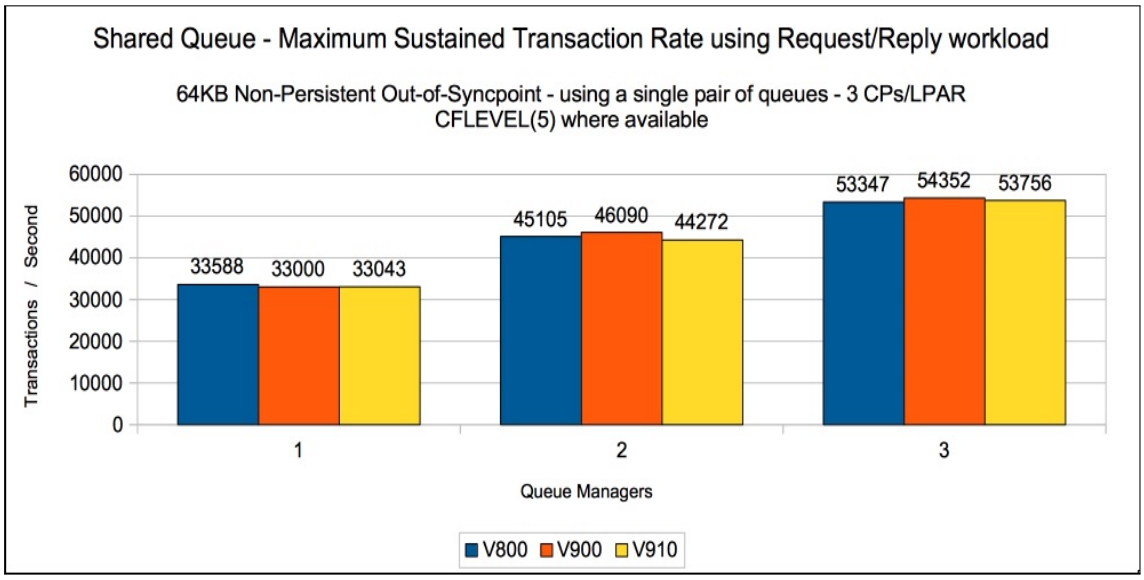


Chart: Transaction cost for non-persistent out-of-syncpoint workload - 64KB messages.

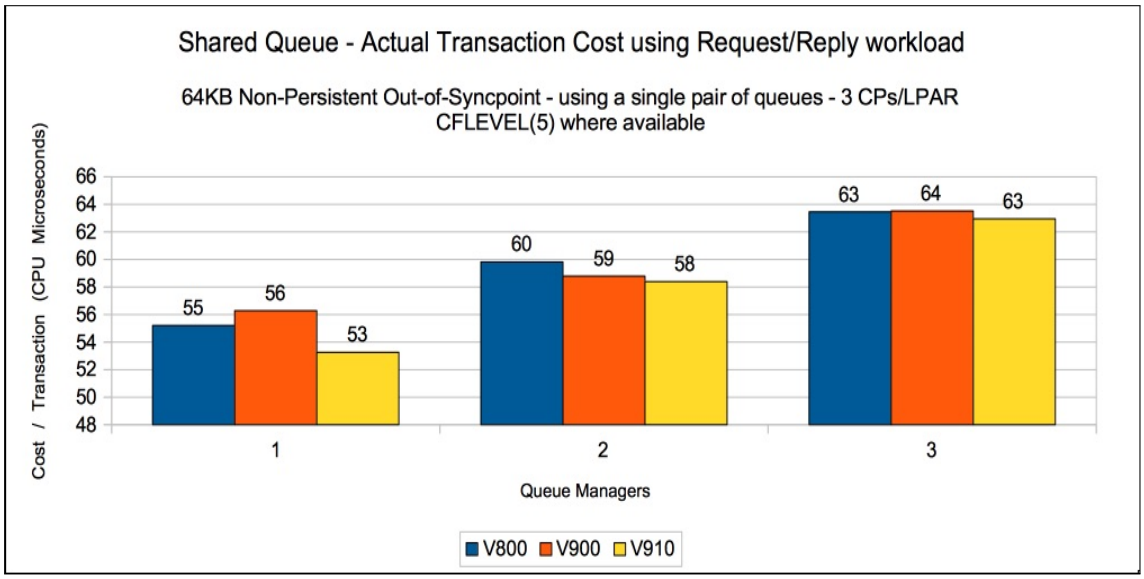


Chart: Transaction rate for non-persistent out-of-syncpoint workload - 4MB messages.

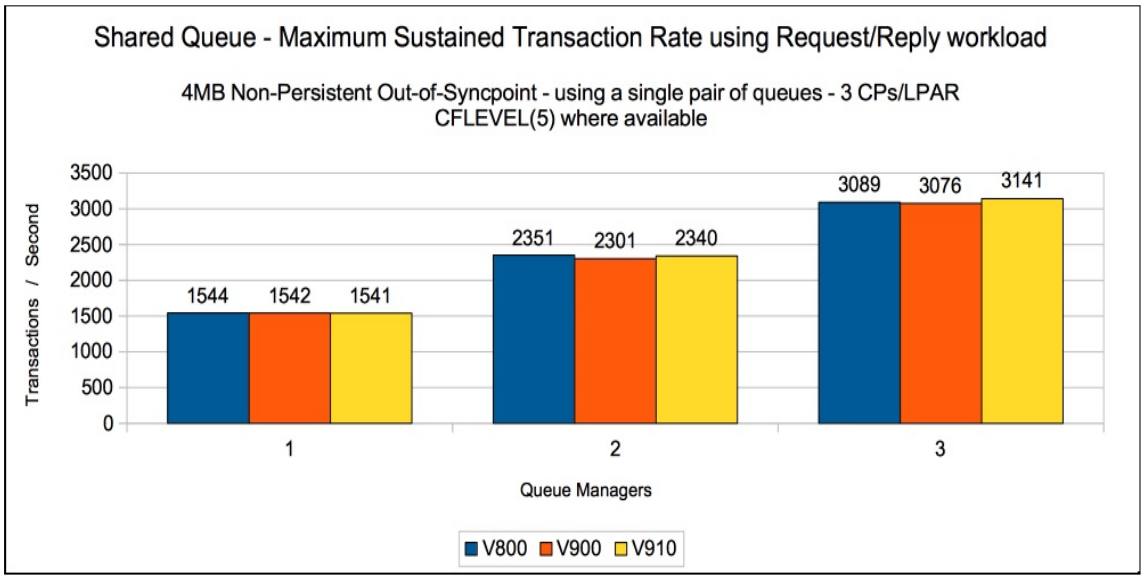
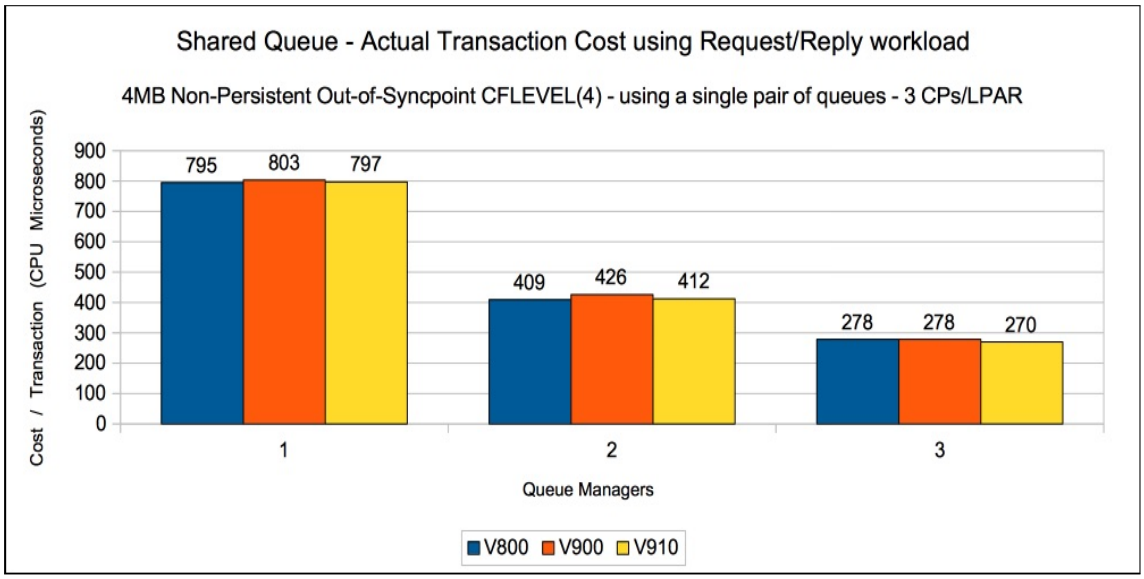


Chart: Transaction cost for non-persistent out-of-syncpoint workload - 4MB messages.



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

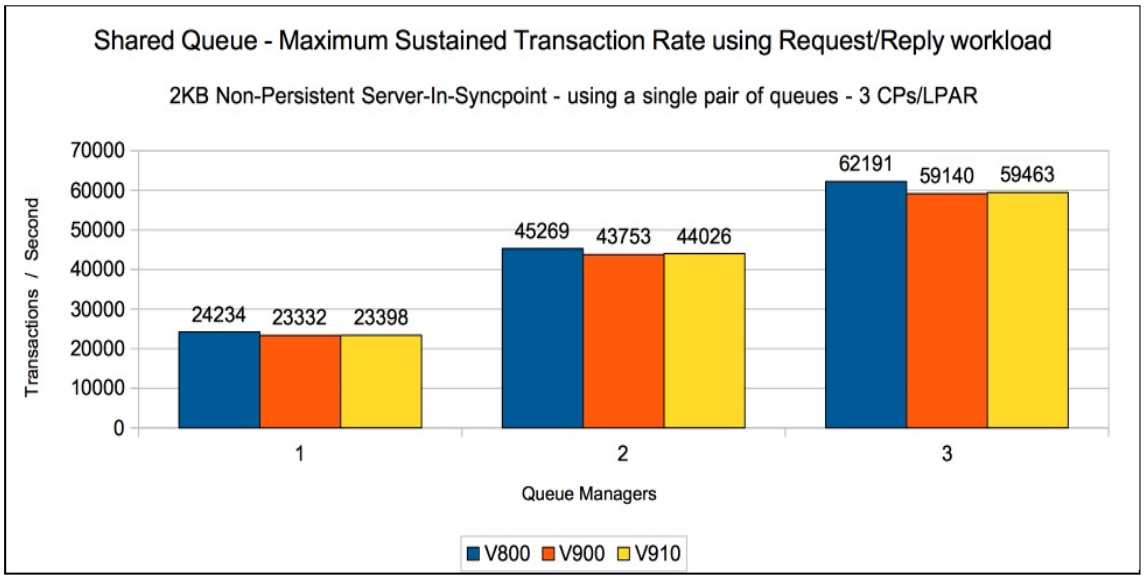


Chart: Transaction cost for non-persistent in syncpoint workload - 2KB

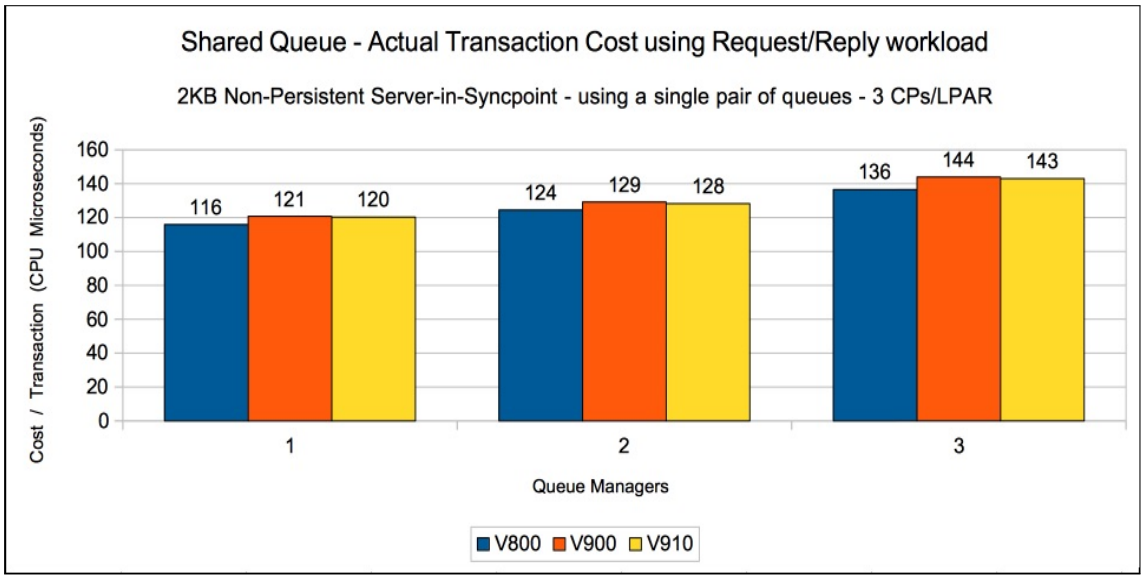


Chart: Transaction rate for non-persistent in syncpoint workload - 64KB

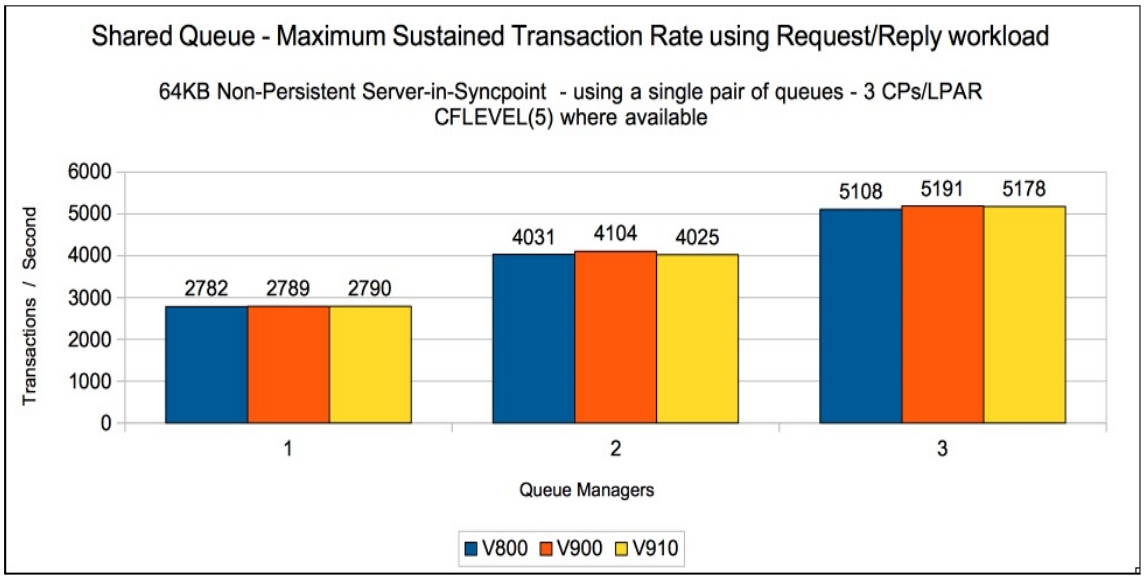


Chart: Transaction cost for non-persistent in syncpoint workload - 64KB

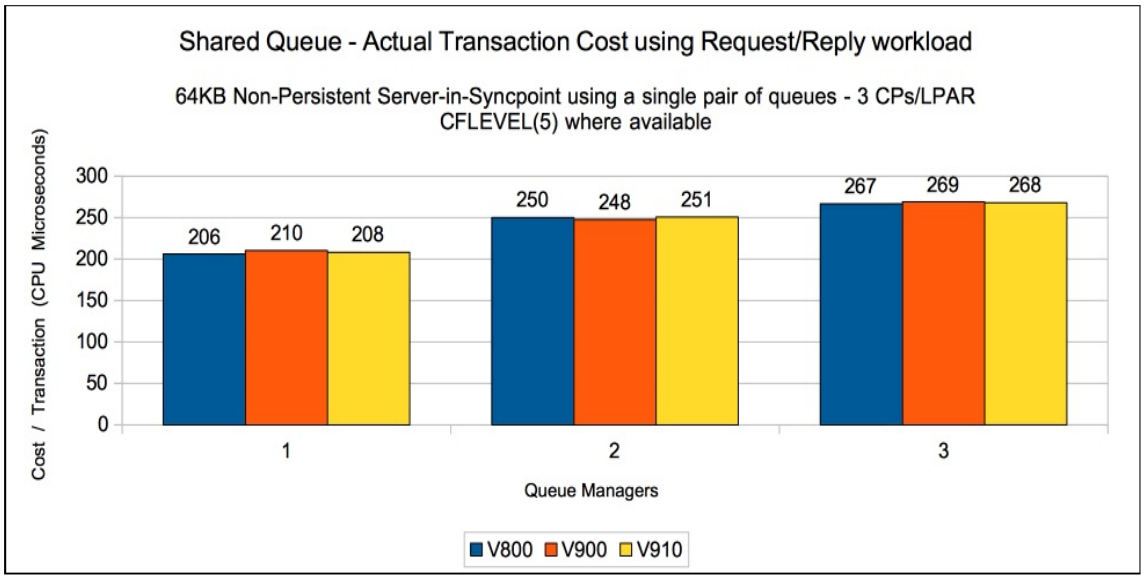


Chart: Transaction rate for non-persistent in syncpoint workload - 4MB

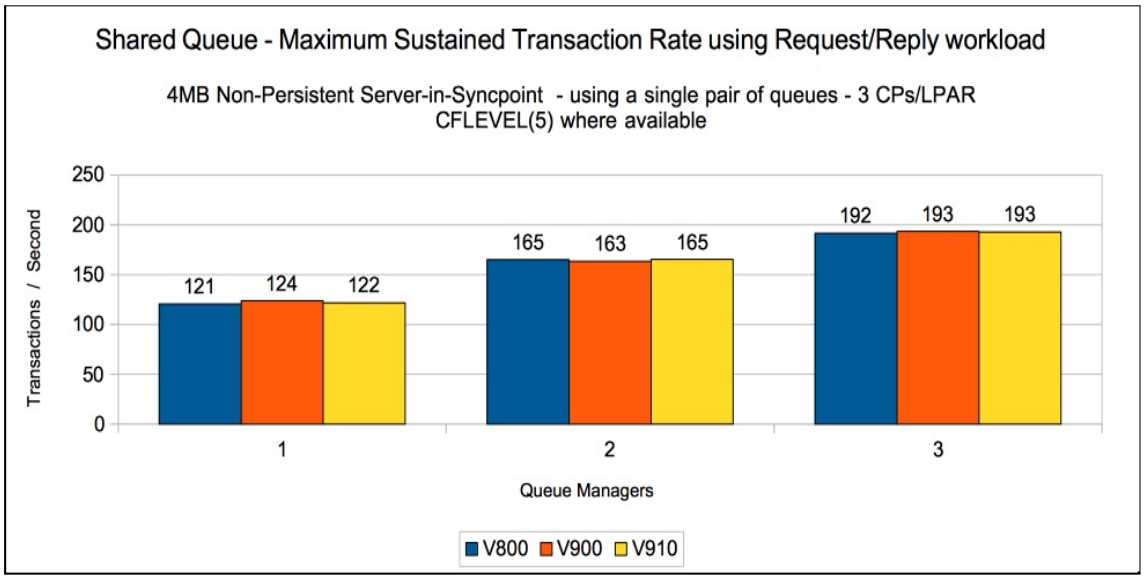
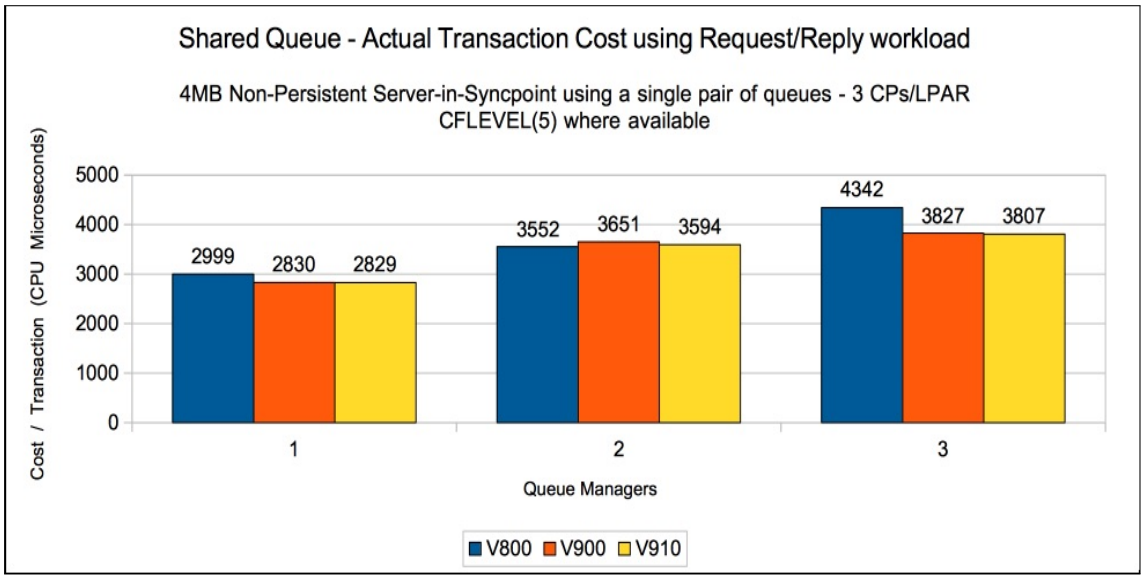


Chart: Transaction cost for non-persistent in syncpoint workload - 4MB



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

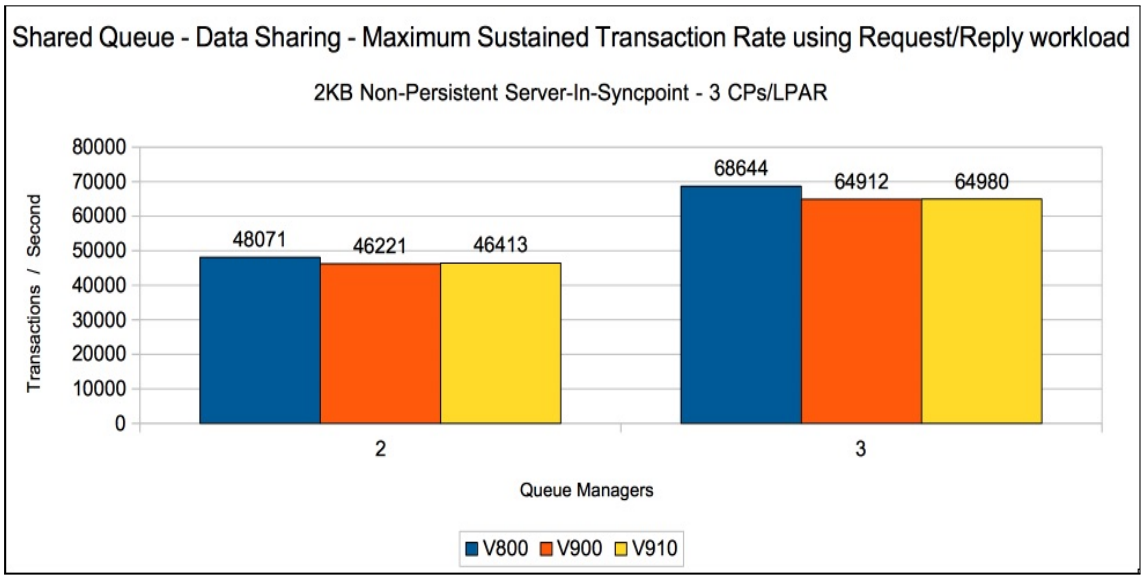


Chart: Transaction cost for data sharing non-persistent in syncpoint workload - 2KB

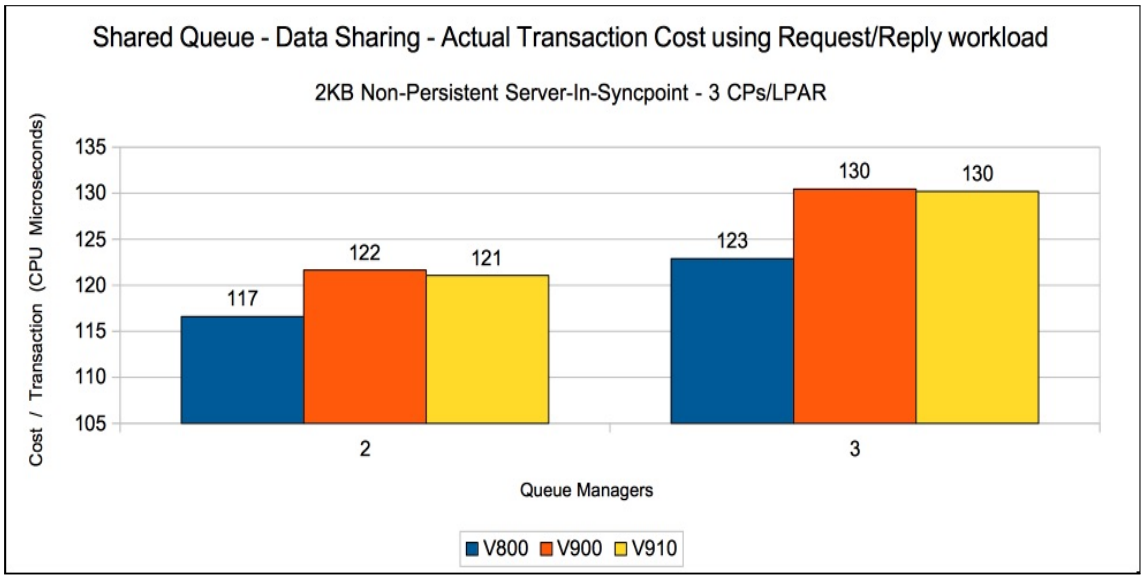


Chart: Transaction rate for data sharing non-persistent in syncpoint workload - 64KB

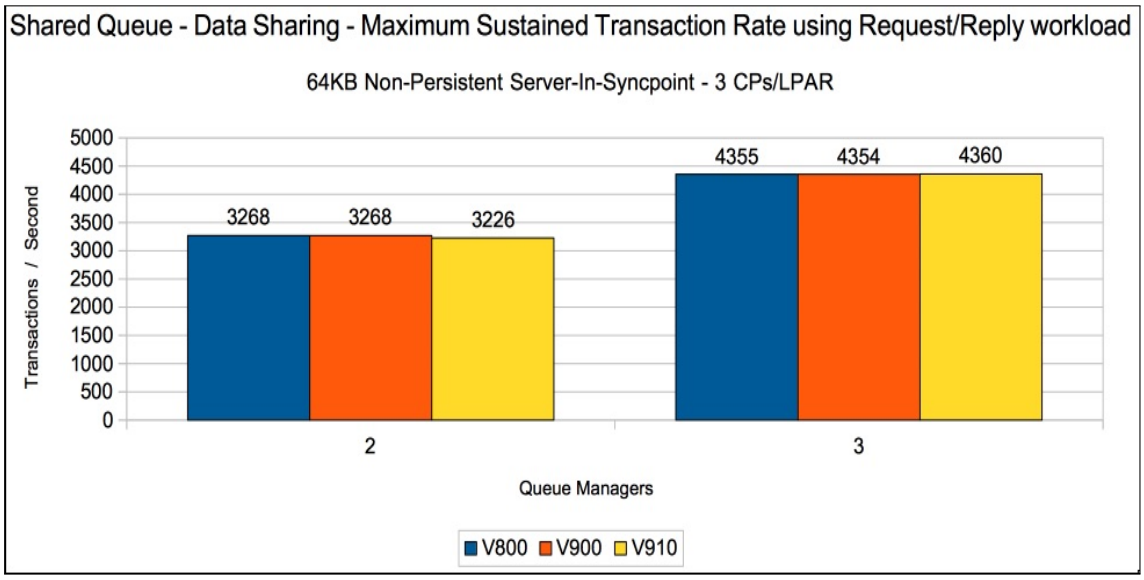
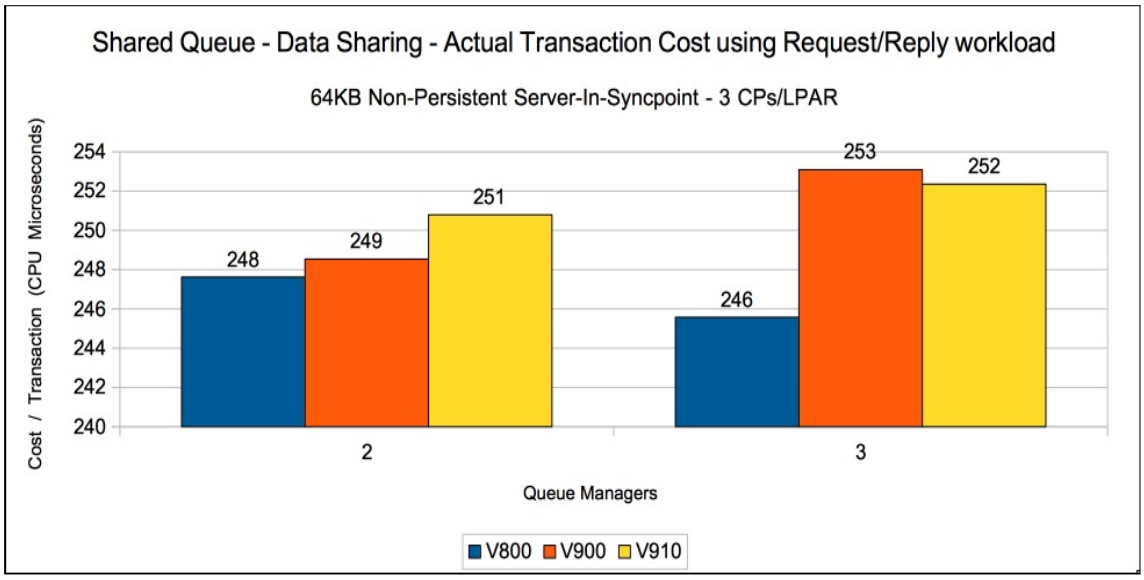


Chart: Transaction cost for data sharing non-persistent in syncpoint workload - 64KB



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 system limits rather than MQ are the reason for any constraints, and typically it is CPU that 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 LPARs to capacity and also tests with up to 50 channels outbound and 50 channels inbound.

Note: Higher throughput can be achieved with additional CPUs but this can increase the transaction cost.

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.

The SSLRKEYC attribute was set so that 1MB of data can flow across the channel before renegotiating the keys.

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".

Measurements:

- 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

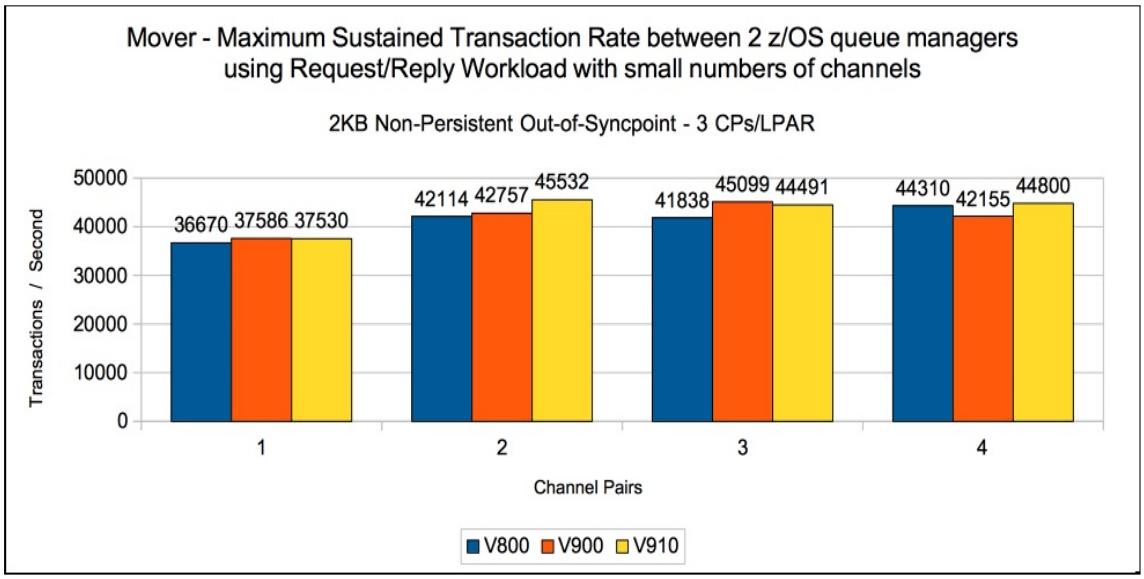


Chart: Transaction cost for 2KB messages

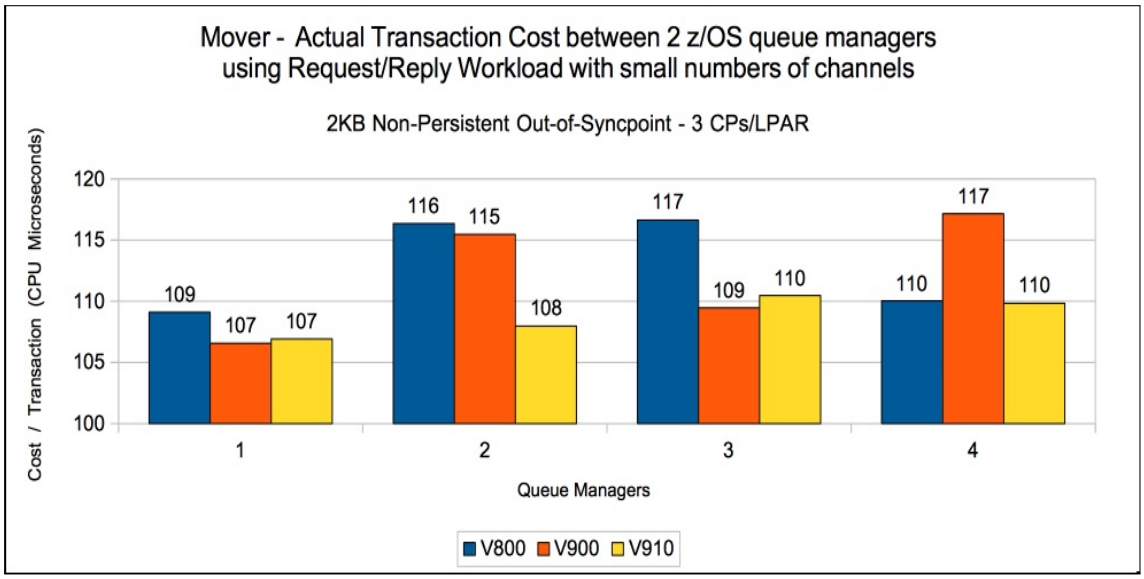


Chart: Transaction rate with 64KB messages

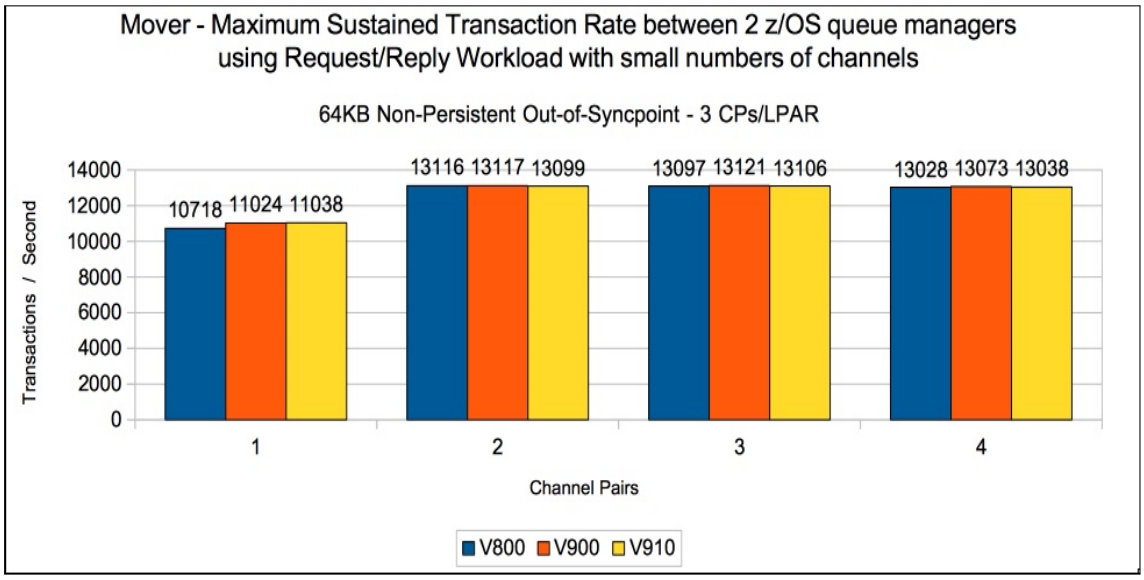


Chart: Transaction cost for 64KB messages

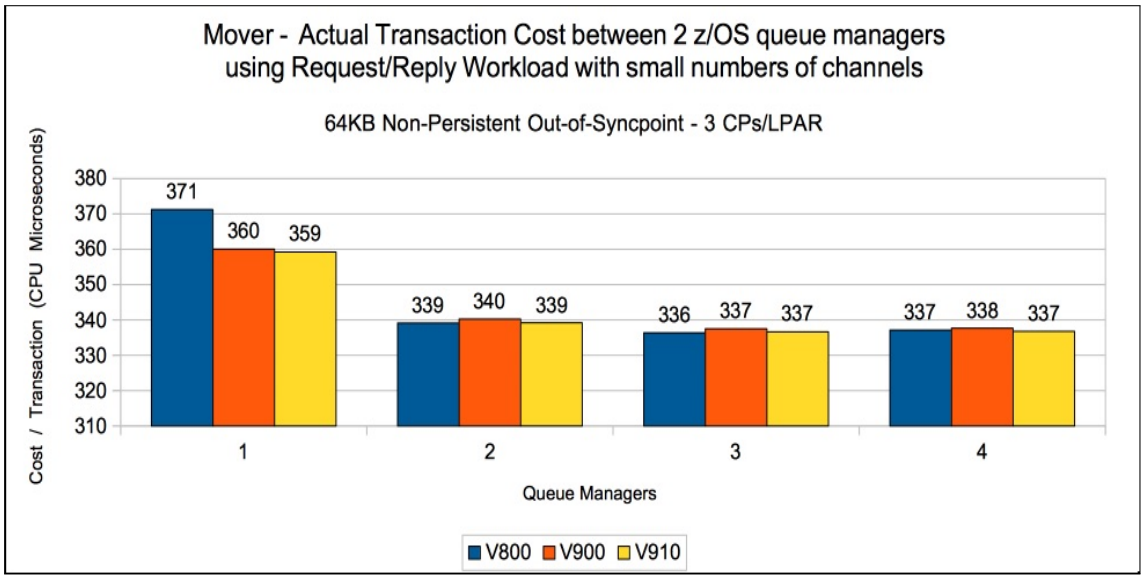


Chart: Transaction rate with 2KB messages over SSL channels

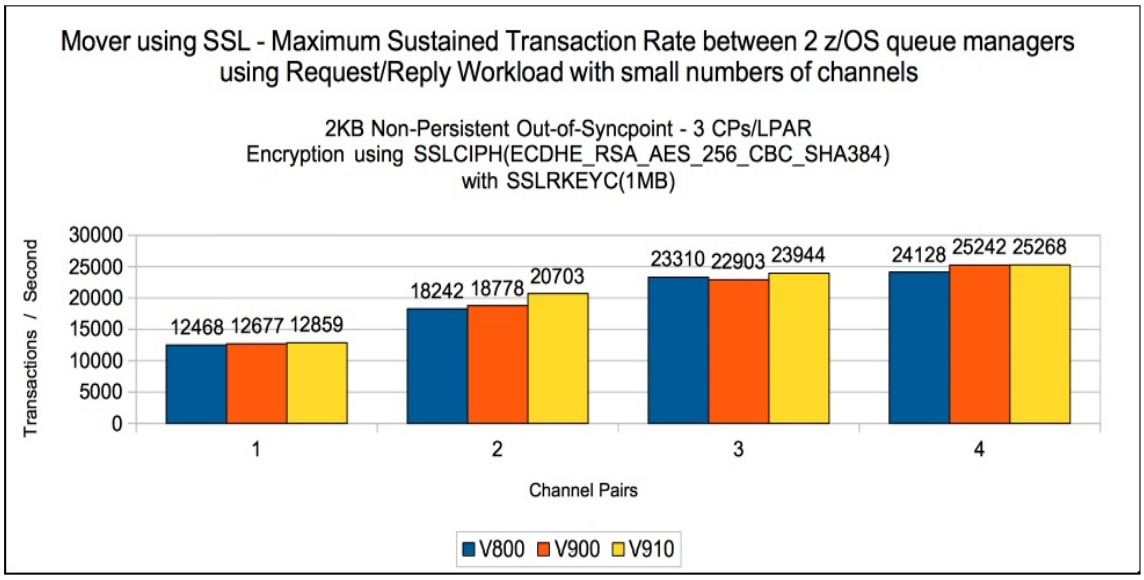


Chart: Transaction cost for 2KB messages over SSL channels

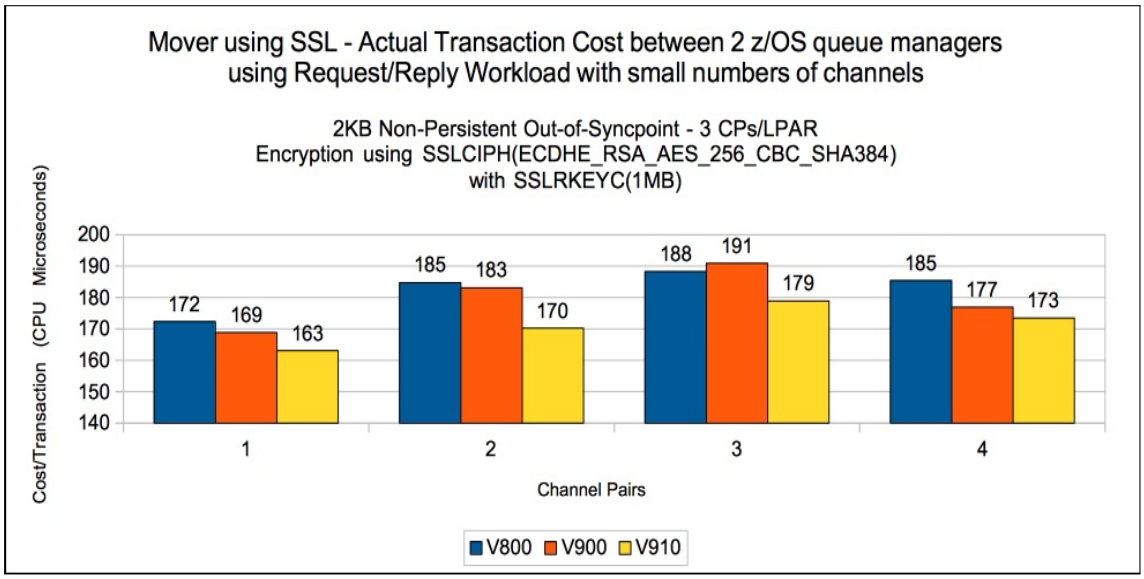


Chart: Transaction rate with 64KB messages over SSL channels

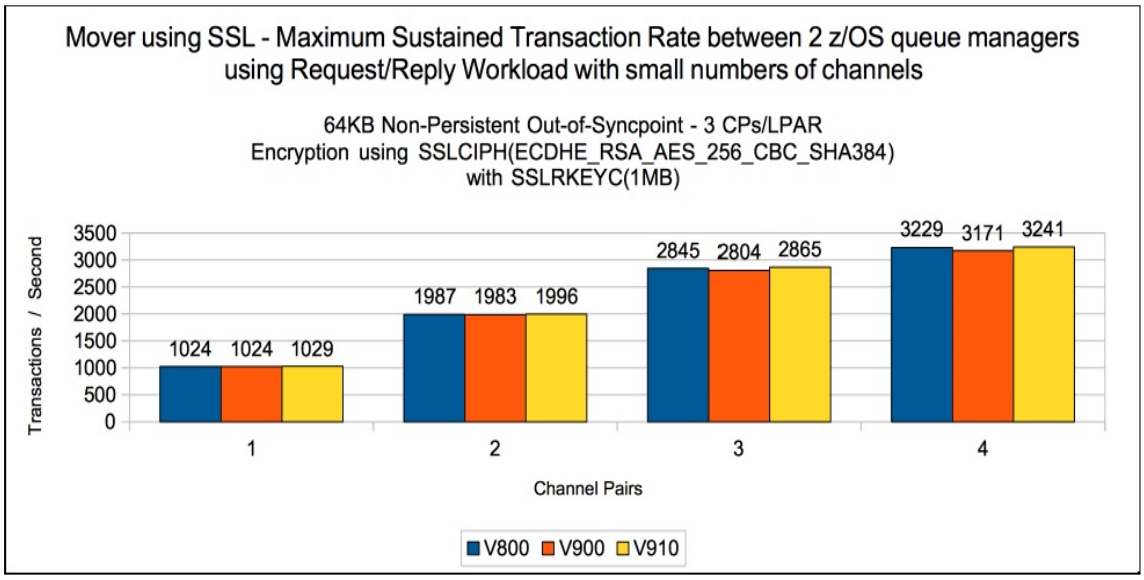
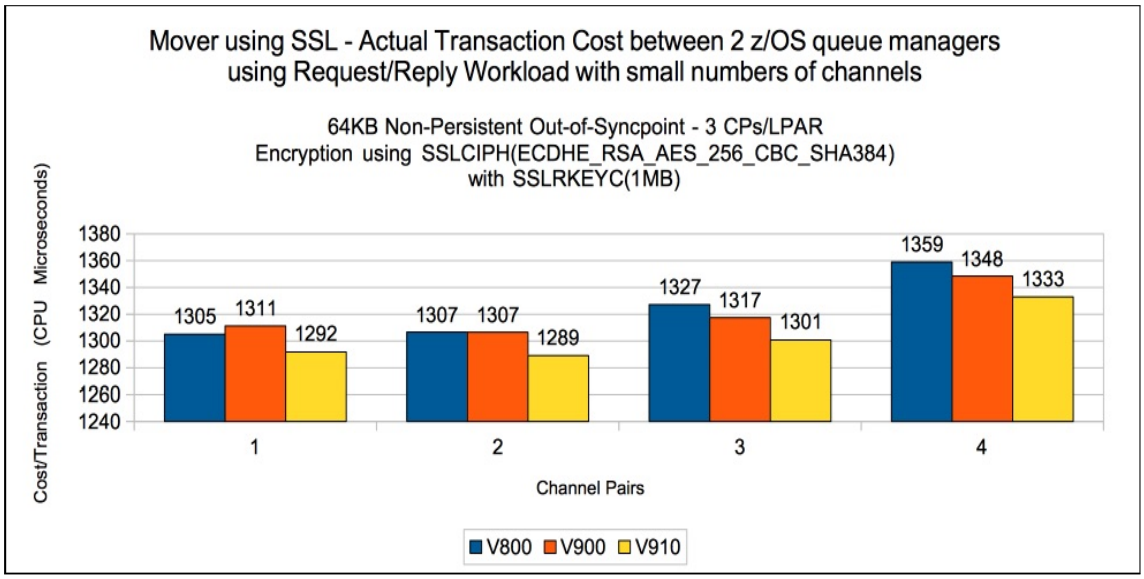


Chart: Transaction cost for 64KB messages over SSL channels



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

Chart: Transaction rate with 2KB messages

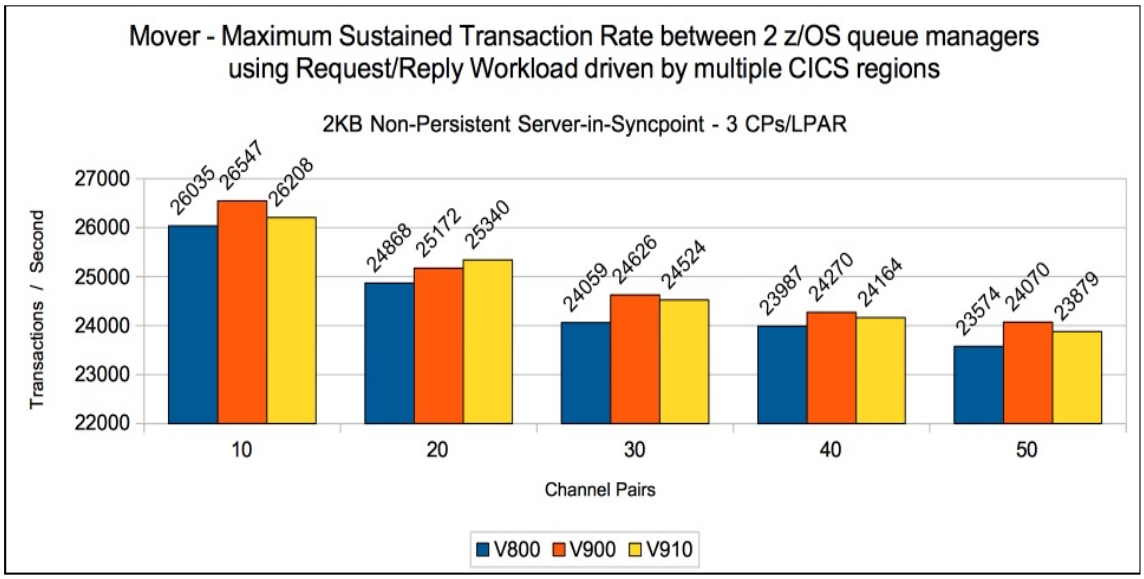


Chart: Transaction cost for 2KB messages

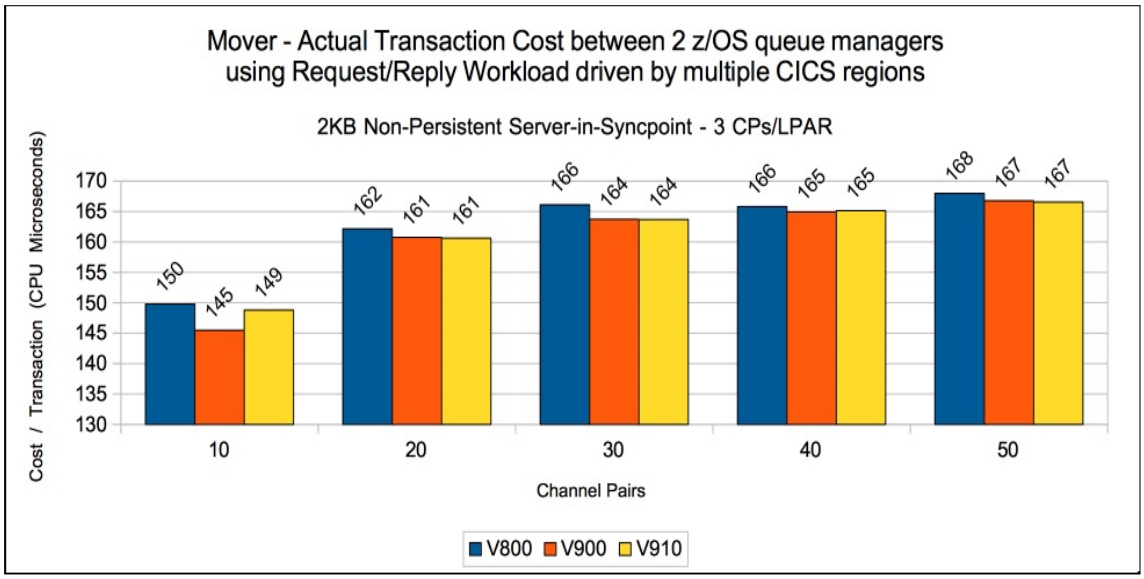


Chart: Transaction rate with 2KB messages over SSL channels

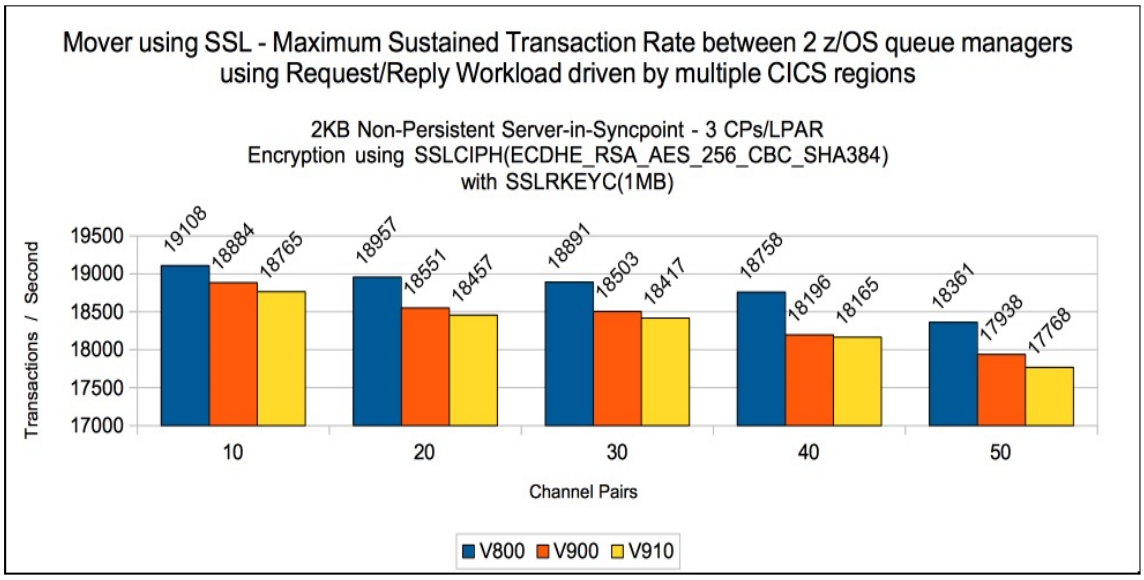
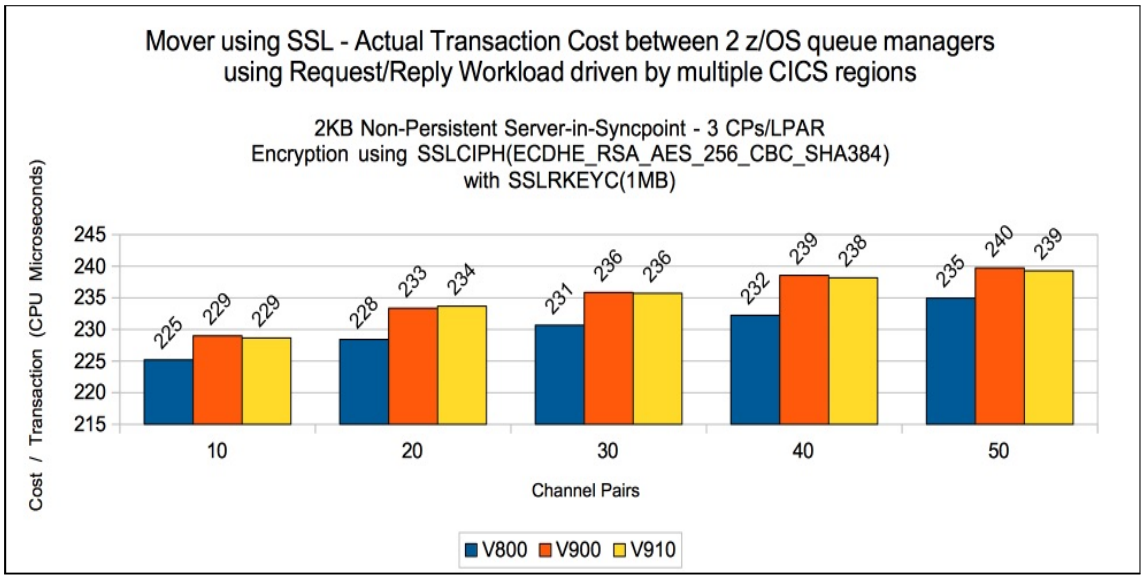


Chart: Transaction cost for 2KB messages over SSL channels



Channel compression using ZLIBFAST

Version 8.0.0 onwards are able to exploit compression using the available zEDC hardware features.

Chart: Transaction rate with ZLIBFAST on 64KB messages

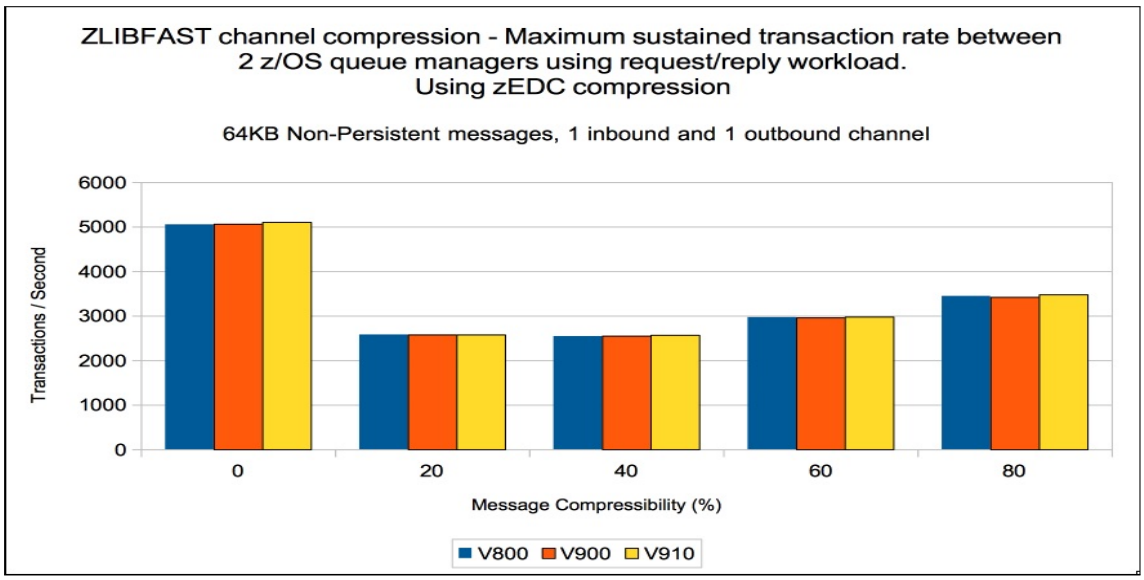
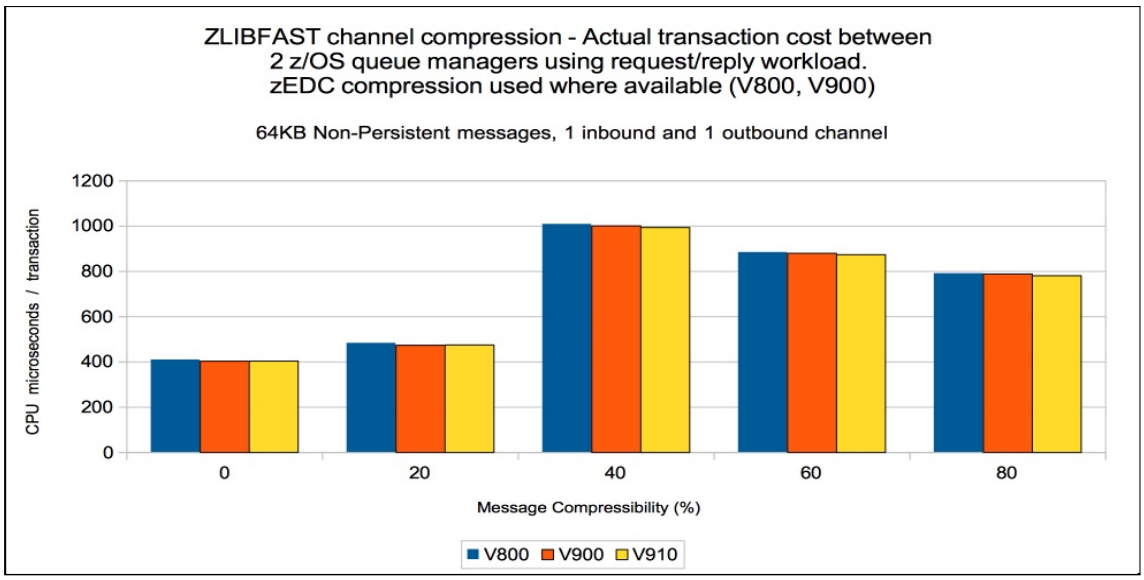


Chart: Transaction cost for ZLIBFAST on 64KB messages



Channel compression using ZLIBHIGH

Chart: Transaction rate with 64KB messages with ZLIBHIGH

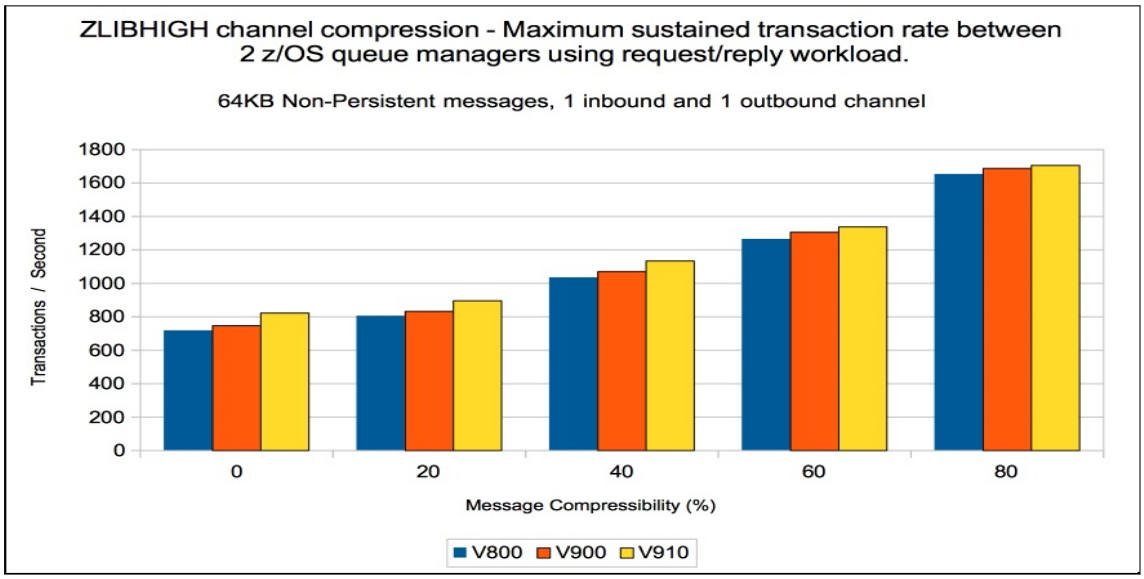
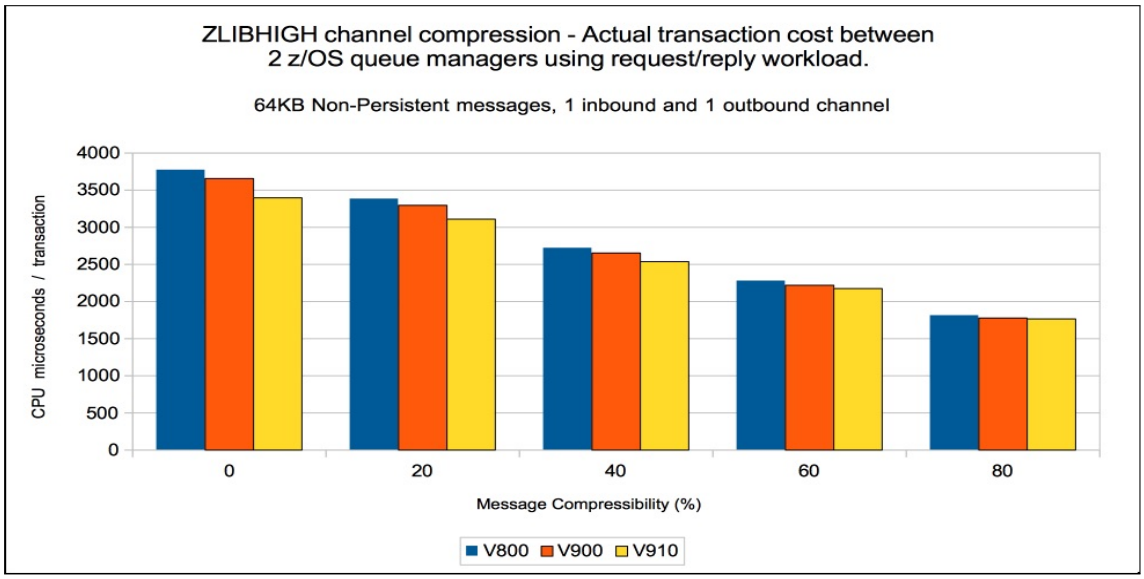


Chart: Transaction cost for 64KB messages with ZLIBHIGH



Channel compression using ZLIBFAST on SSL channels

Version 8.0.0 onwards are able to exploit compression using the available zEDC hardware features.

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

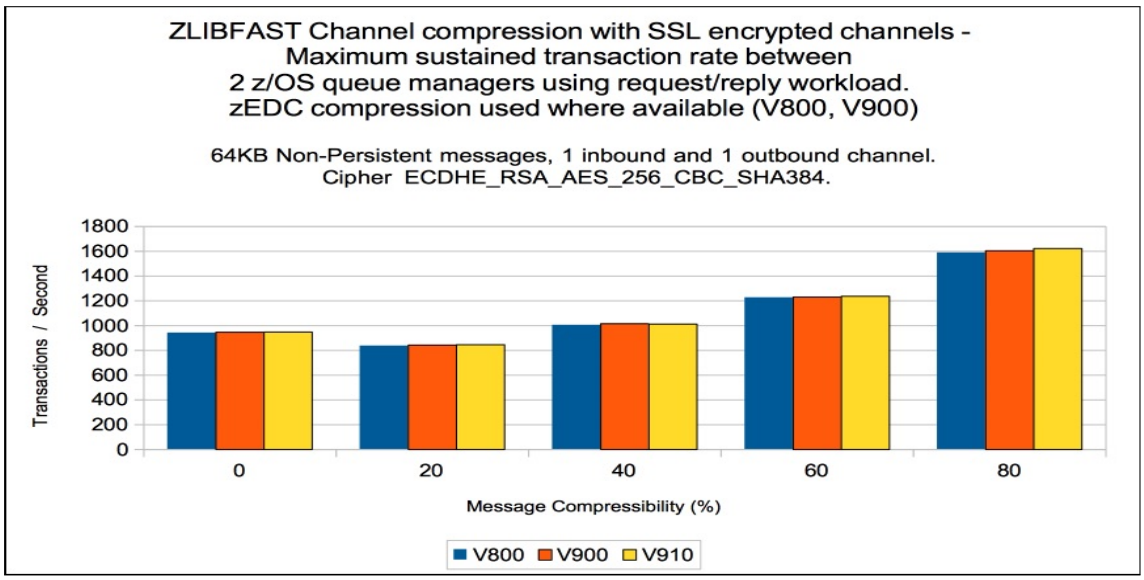
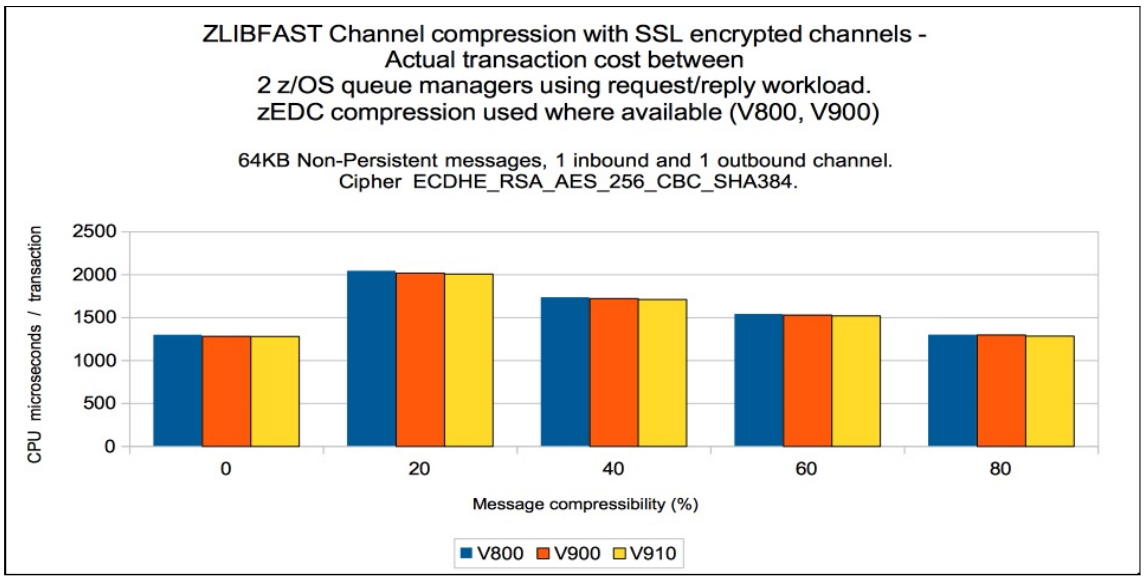


Chart: Transaction cost for 64KB messages with ZLIBFAST on SSL channels



Channel compression using ZLIBHIGH on SSL channels

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

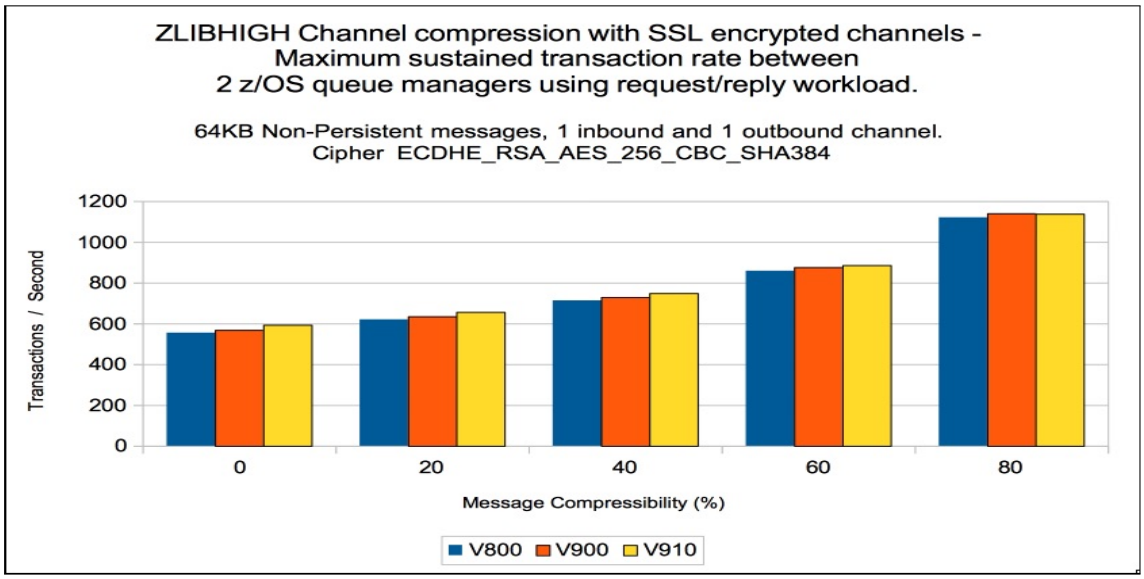
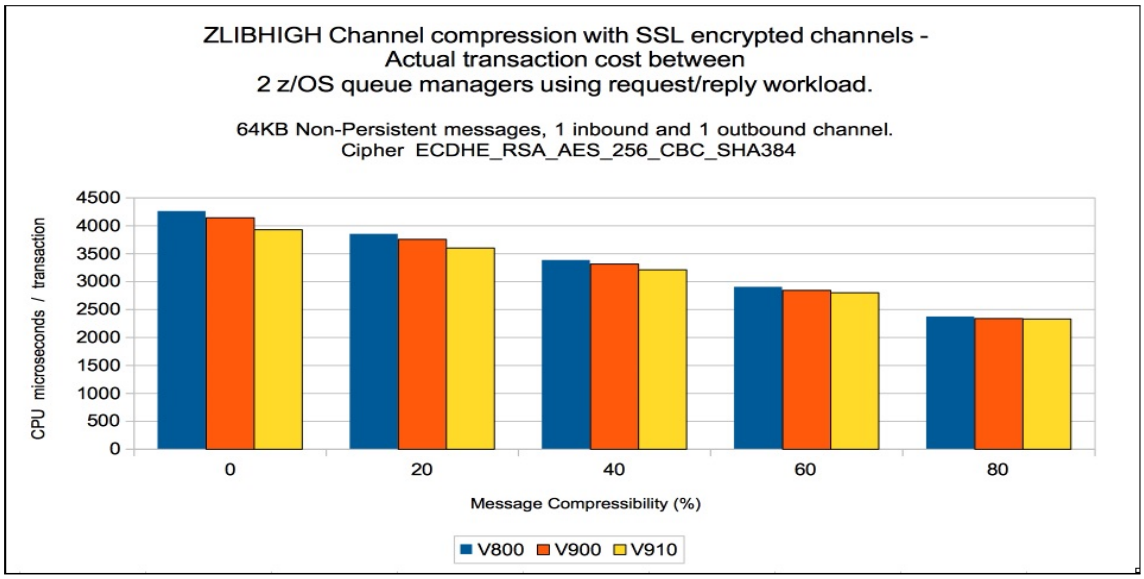


Chart: Transaction cost for 64KB messages with ZLIBHIGH on SSL channels



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

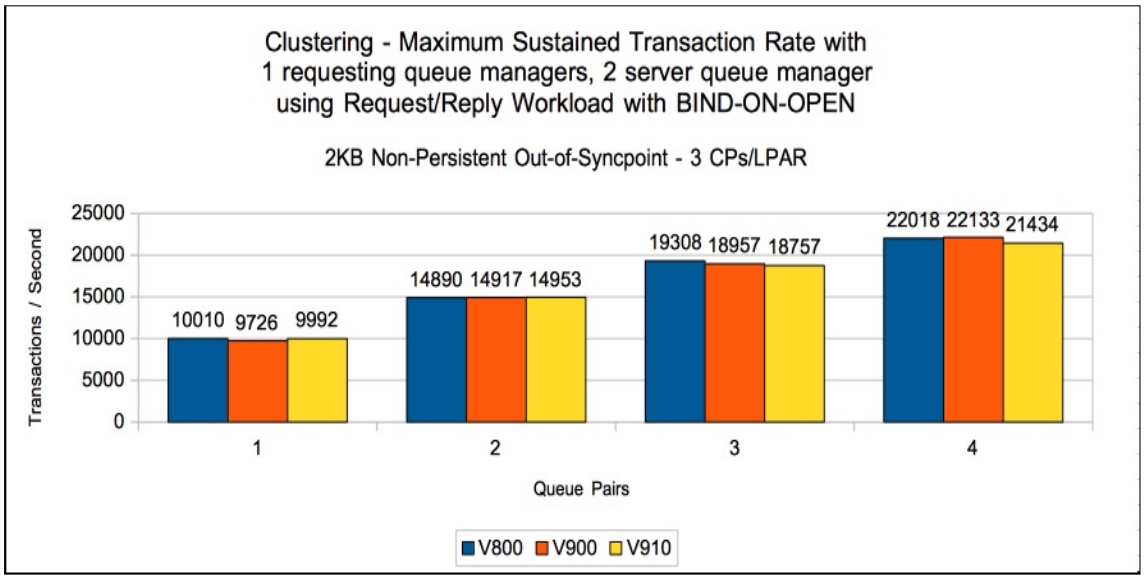


Chart: Bind-on-open transaction cost for 2KB messages

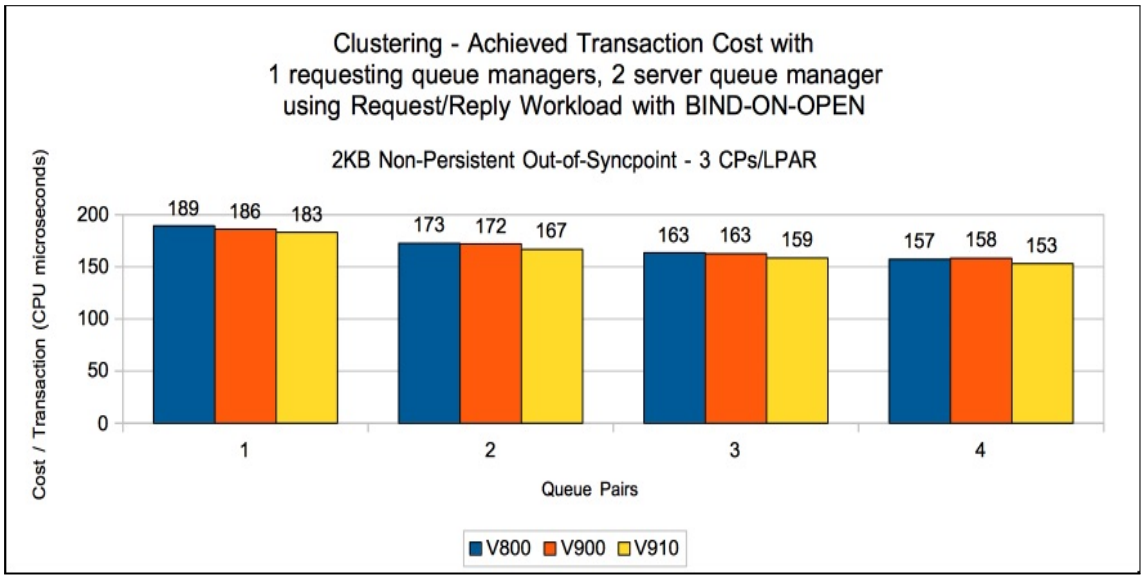


Chart: Bind-on-open transaction rate with 64KB messages

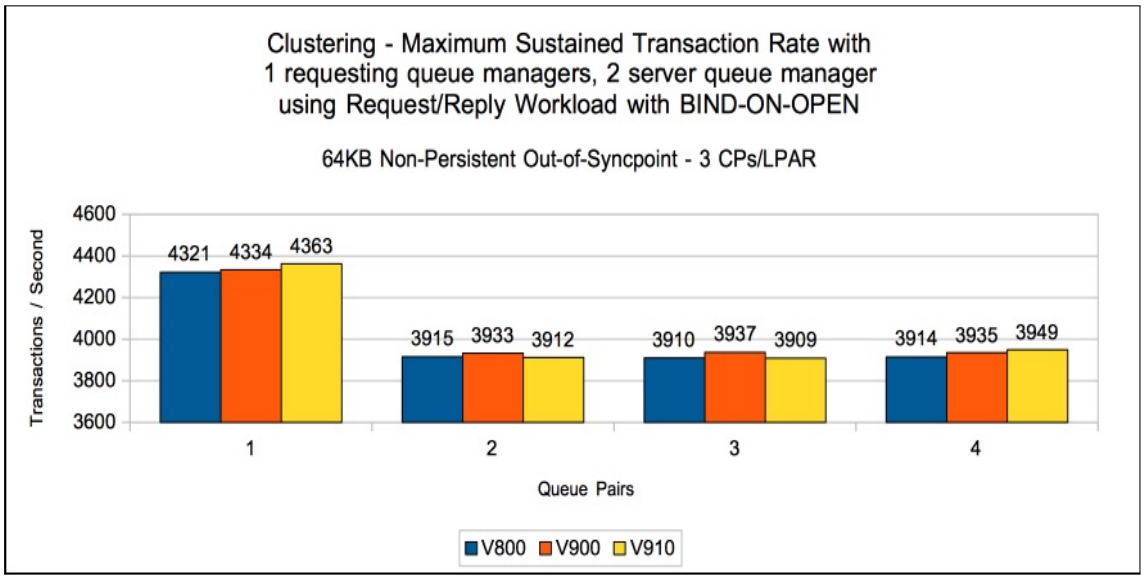
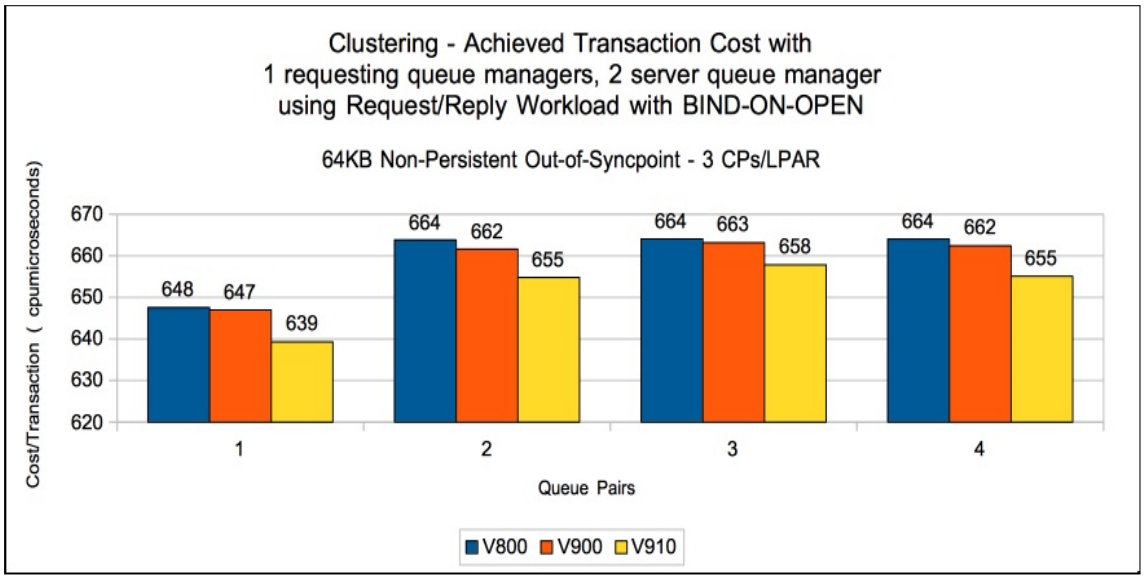


Chart: Bind-on-open transaction cost for 64KB messages



Bind-not-fixed

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

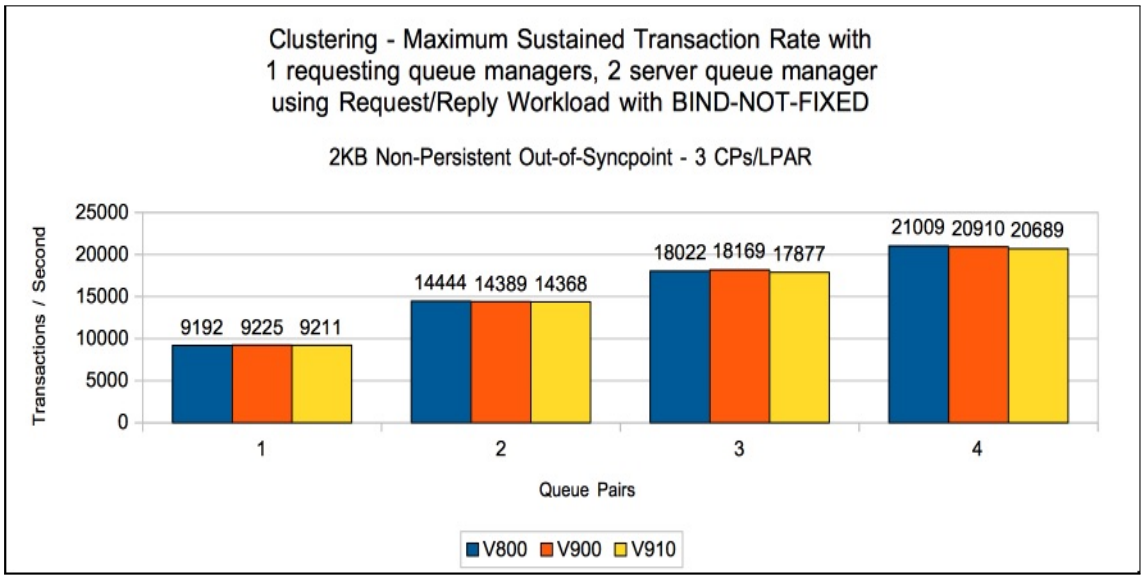


Chart: Bind-not-fixed transaction cost for 2KB messages

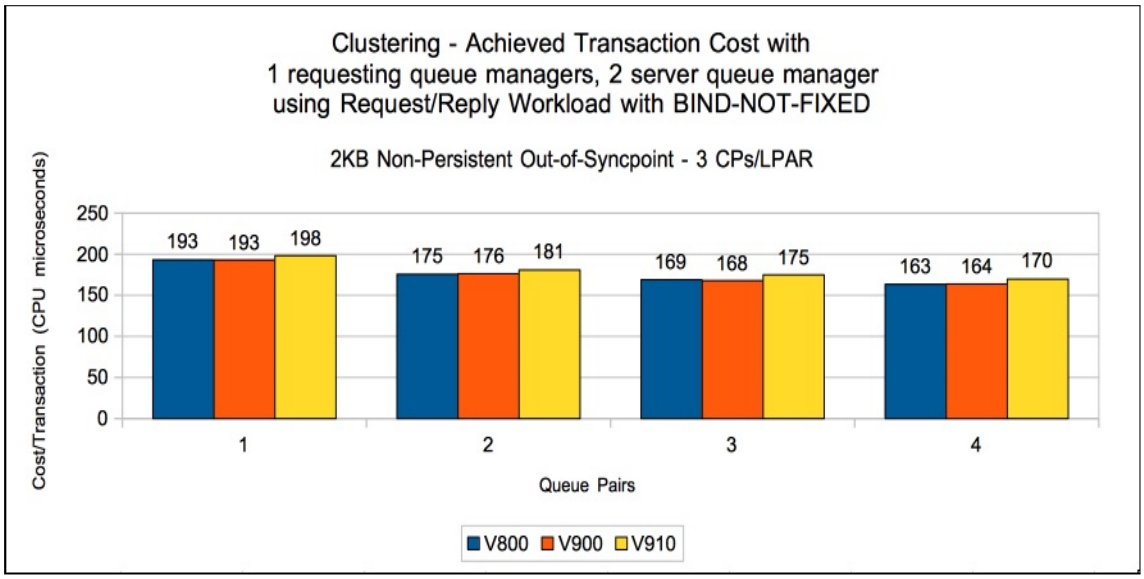


Chart: Bind-not-fixed transaction rate with 64KB messages

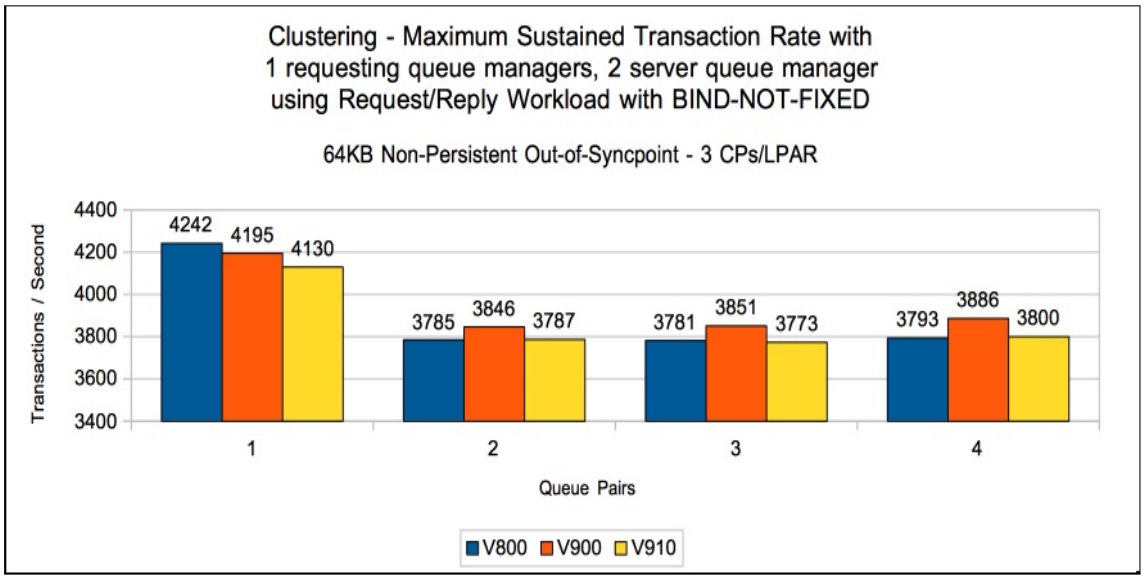
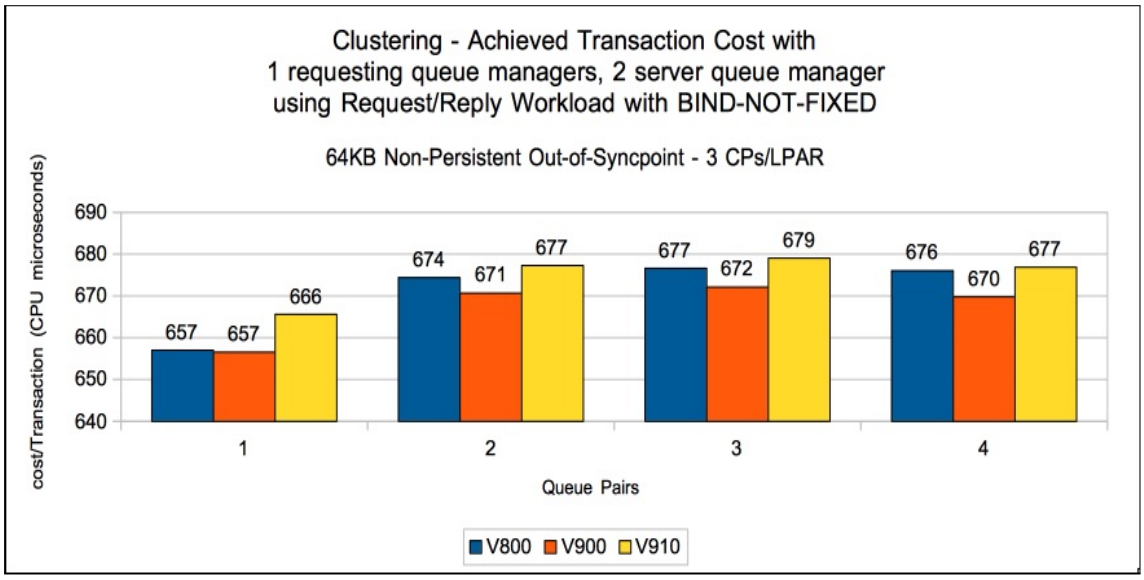


Chart: Bind-not-fixed transaction cost for 64KB messages



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)

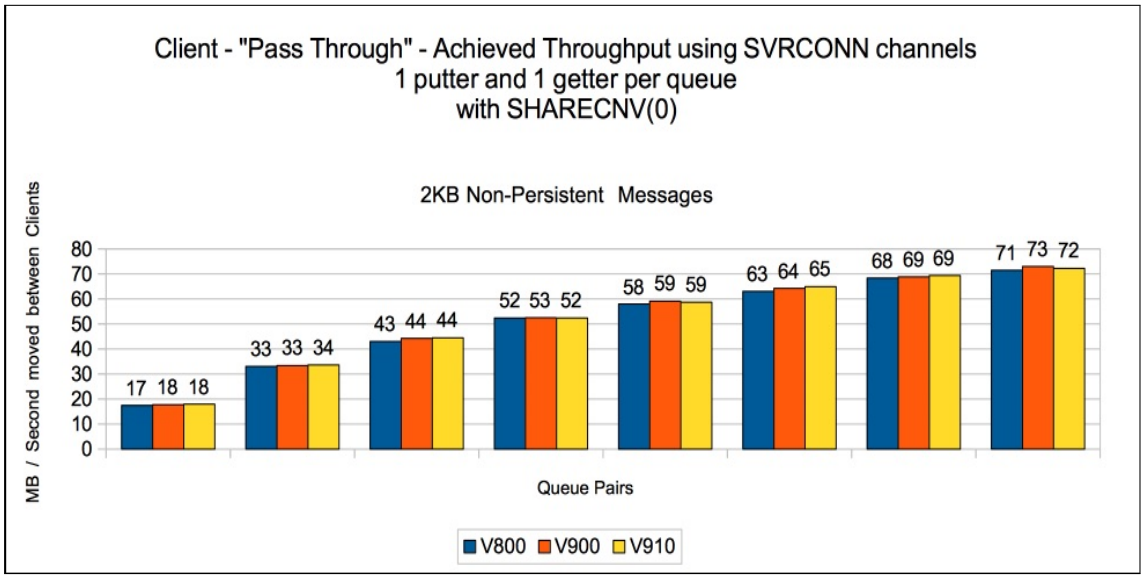
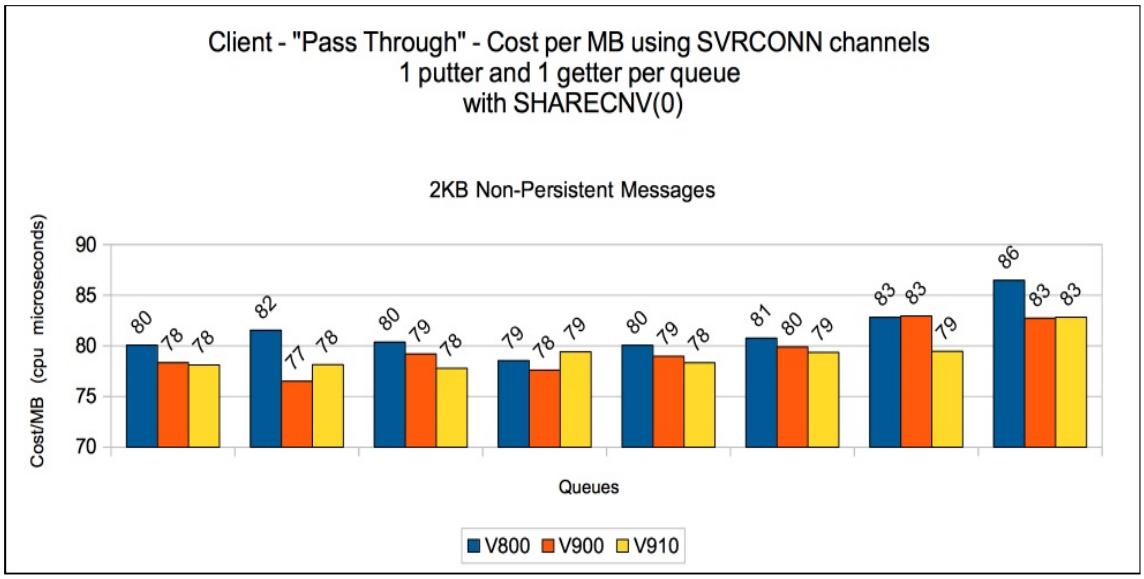


Chart: Cost per MB of client pass through tests with SHARECNV(0)



Client pass through test using SHARECNV(1)

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

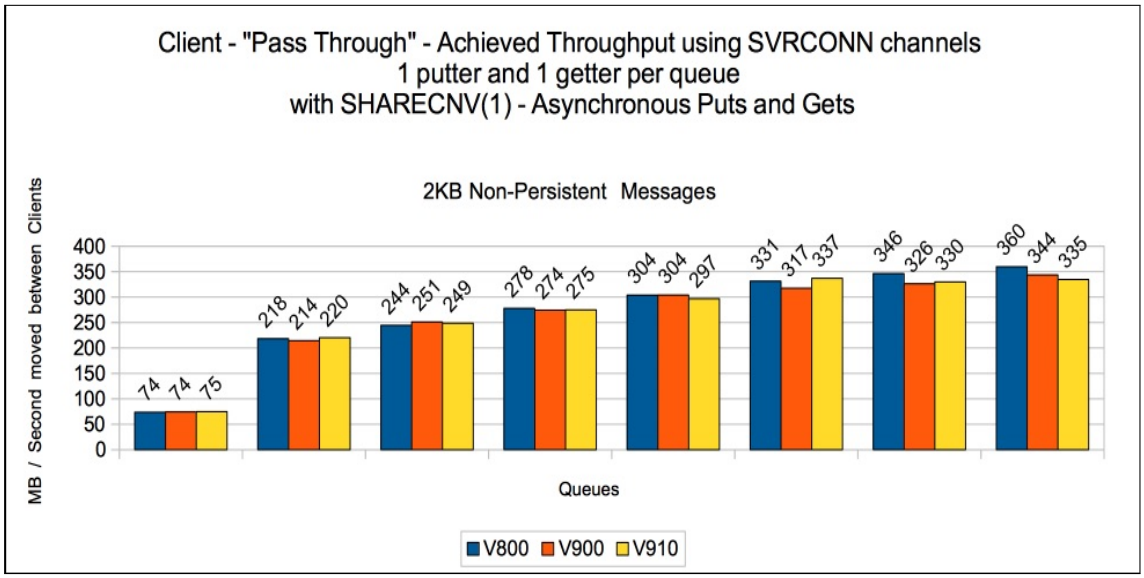
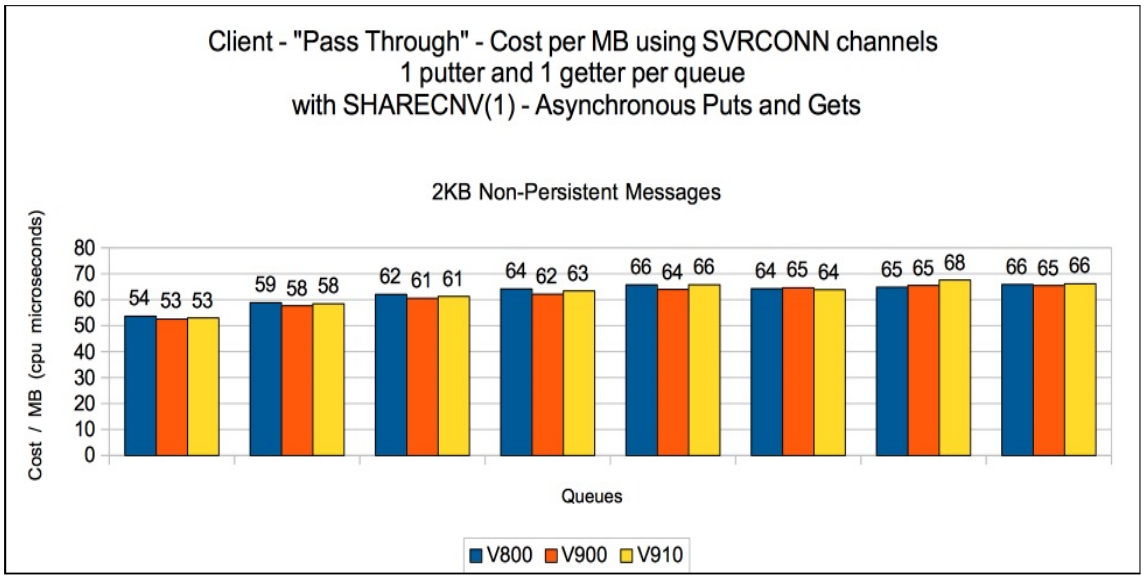


Chart: Cost per MB of client pass through tests with SHARECNV(1)



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

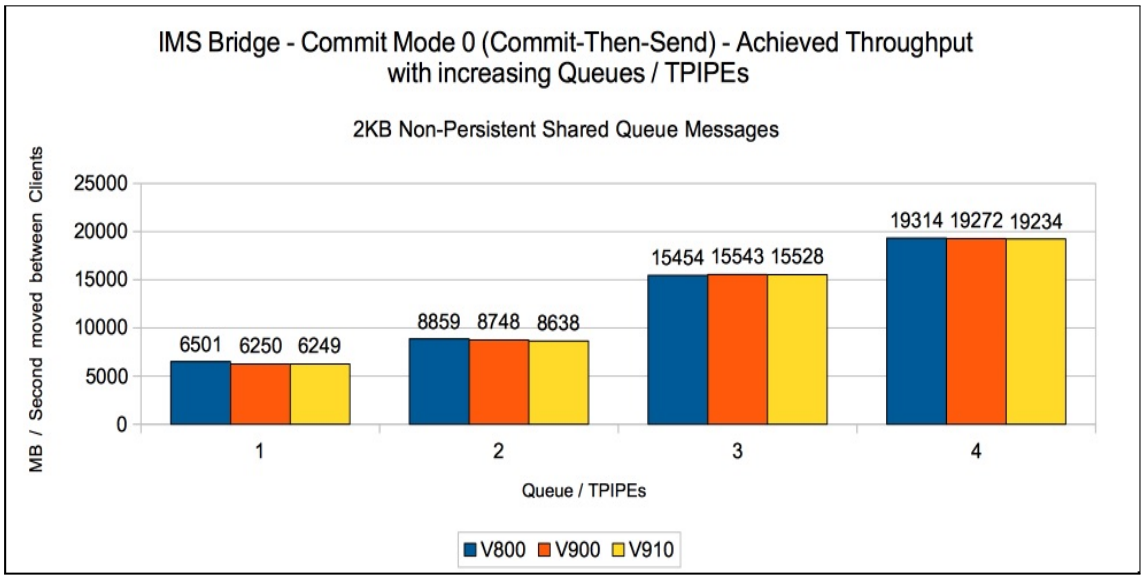
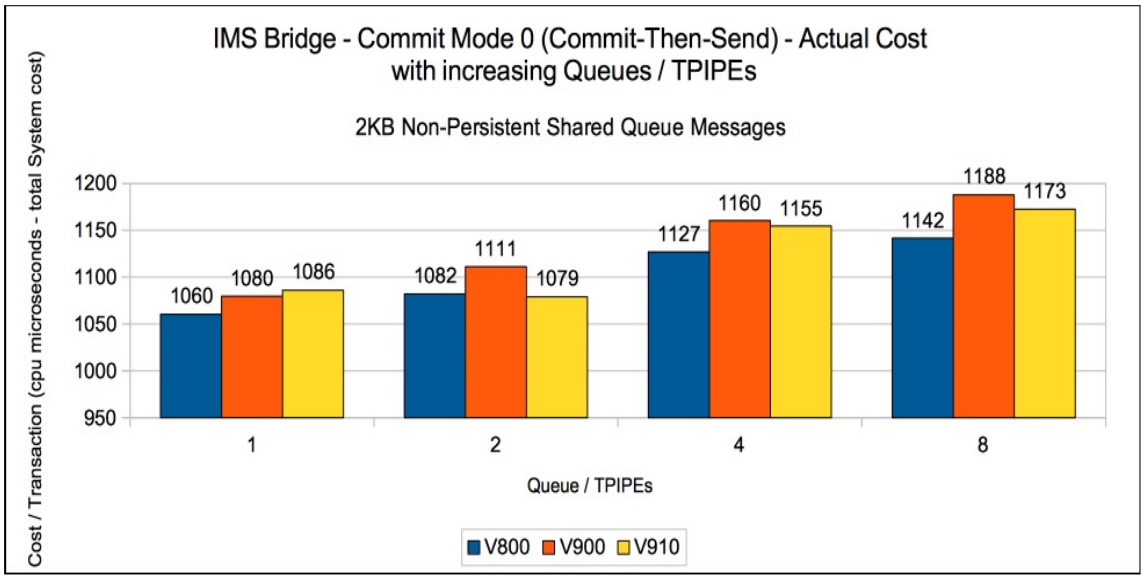


Chart: IMS Bridge commit mode 0 - Transaction cost



Commit mode 1 (send-then-commit)

Chart: IMS Bridge commit mode 1 - Throughput rate

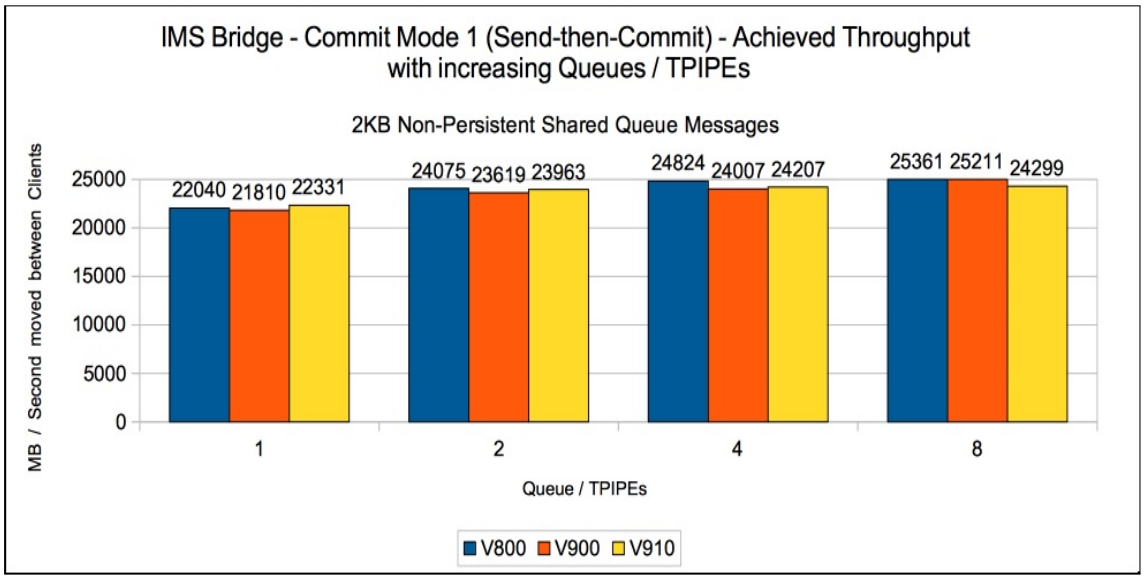
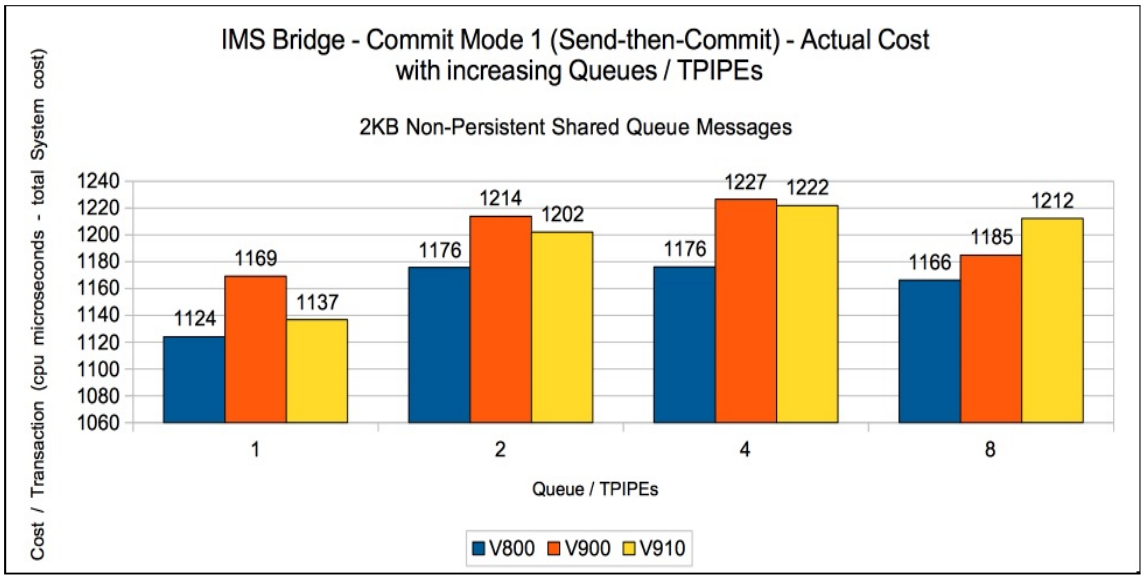


Chart: IMS Bridge commit mode 1 - Transaction cost



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.

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

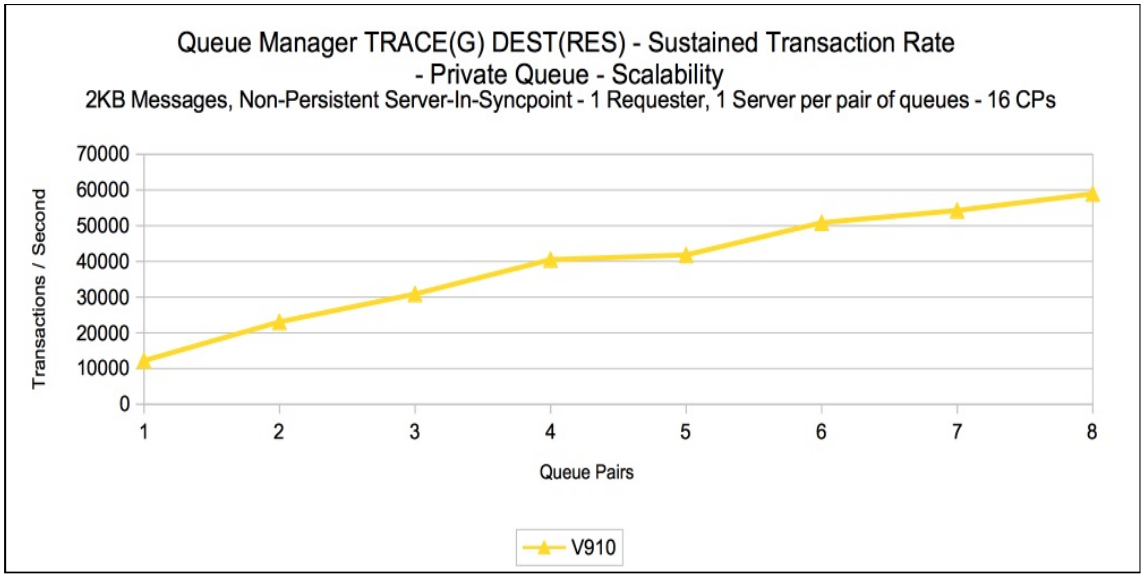
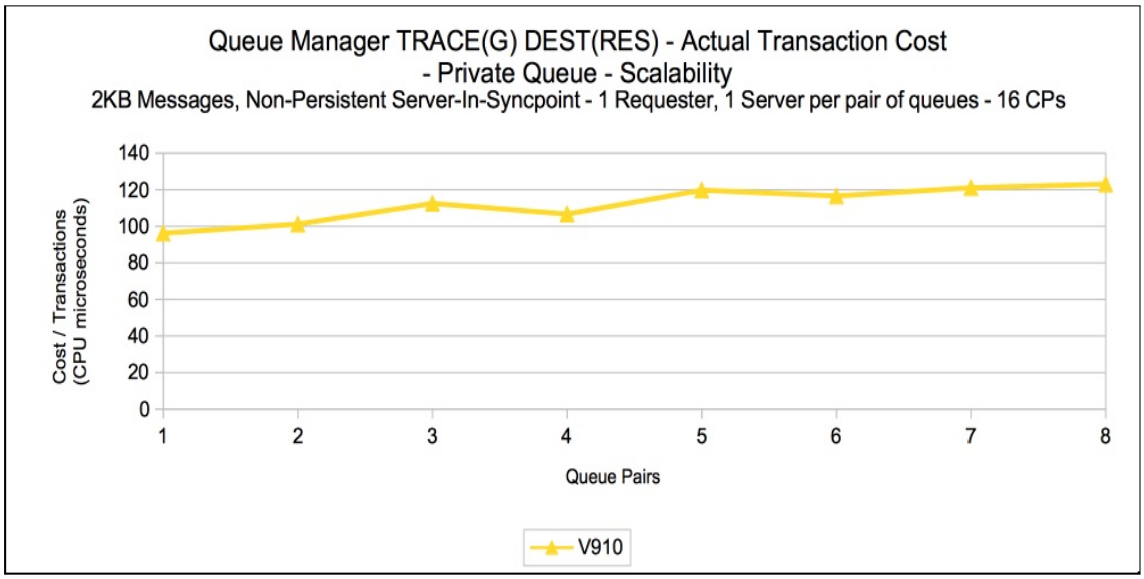


Chart: Queue manager TRACE(G) DEST(RES) - transaction cost



Note: Since the performance for V800, V900 and V910 is comparable, the charts show V910 data for the purpose of clarity.

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

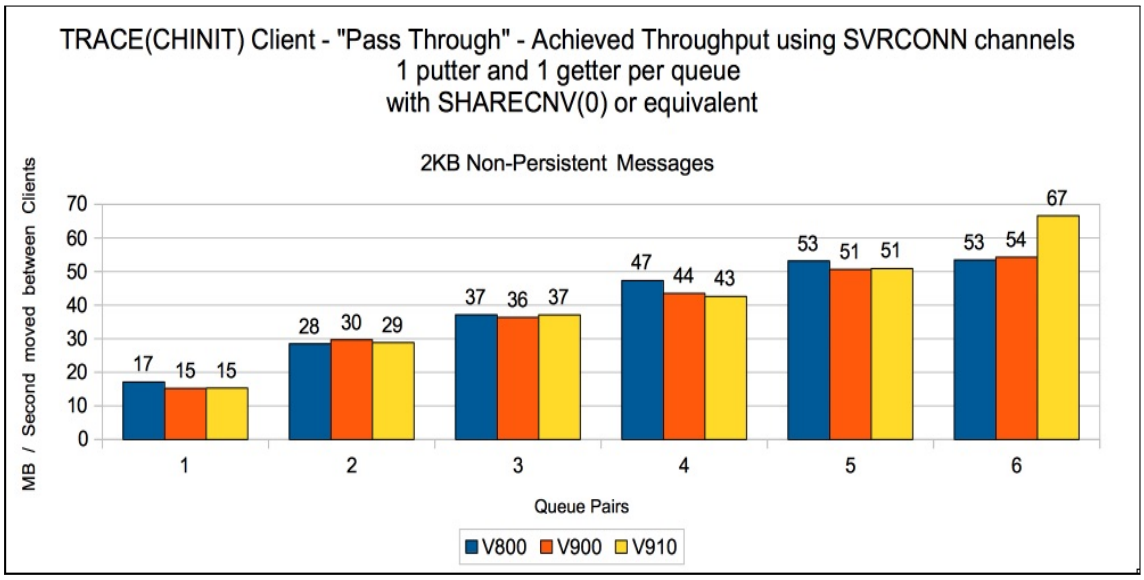
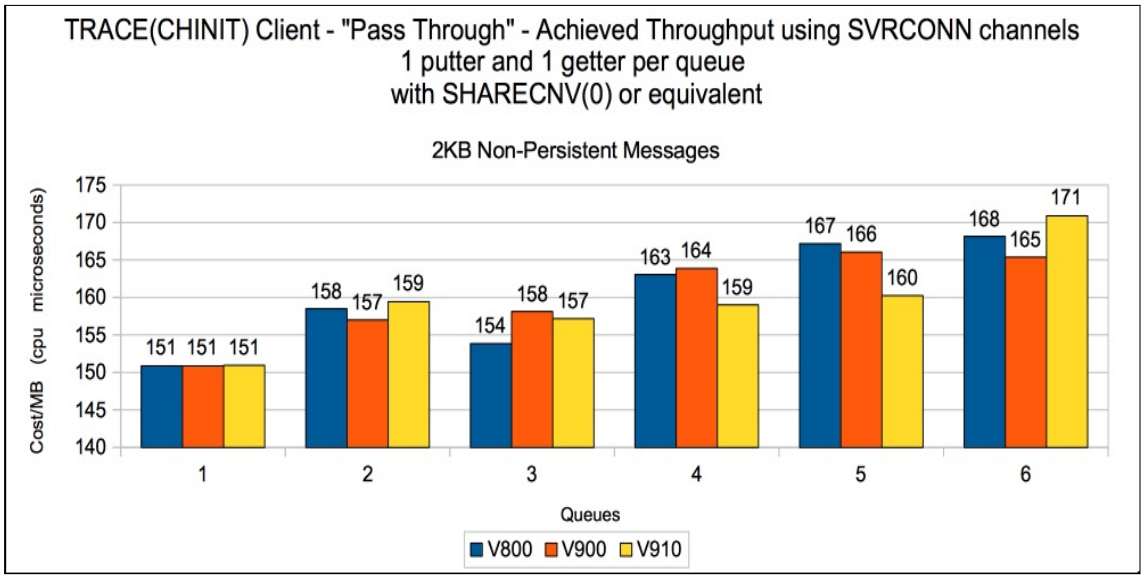


Chart: Channel initiator TRACE(CHINIT) - cost per MB



Appendix B

System configuration

IBM MQ Performance Sysplex running on z14 (3906-7E1) configured thus:

LPAR 1: 1-32 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 v2r3 FMID HBB77B0.

Coupling Facility:

- Internal coupling facility with 4 dedicated processors.
- Coupling Facility running CFCC level 22 service level 00.30.
- 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 Express6 features 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 X5660 each with 12 x 2.6GhZ Processor, 32GB memory
- Client tests used a 10GbE performance network.
- Other IBM products used:
    - IBM CICS TS 5.3.
    - Db2 for z/OS version 12.
    - IMS 15.
    - IBM MQ for z/OS version 8.0 with latest service applied as of May 2018.
    - IBM MQ for z/OS version 9.0 with latest service applied as of May 2018.