MQ for z/OS Performance Report (v1.1)

IBM MQ Advanced for z/OS VUE 9.3
and
IBM MQ for z/OS 9.3

December 2022

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 2022. This edition applies to IBM MQ for z/OS 9.3 and IBM MQ for z/OS 9.3 (and to all subsequent releases and modifications until otherwise indicated in new editions).

Notices

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

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

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

WARRANTY AND LIABILITY EXCLUSION

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

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

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

ERRORS AND OMISSIONS

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

INTENDED AUDIENCE

This report is intended for Architects, Systems Programmers, Analysts and Programmers wanting to understand the performance characteristics of **IBM MQ for z/OS version 9.3**. 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.

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®
- IBM MQ®
- CICS®
- Db2 for z/OS®
- IMS™
- MVS™
- z15™
- z16™
- FICON®
- WebSphere®

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
2022	Version 1.0 - Initial Version.
2022	Version 1.1 - Updates for shared queue support for streaming queue.

Table of contents

Chapter 1

Performance highlights

This report focuses on performance changes since previous versions (9.1 and 9.2) 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 advice and guidance learned as systems increase in power and experience is gained.

In IBM MQ for z/OS 9.3, there are a number of features that can benefit the performance of the MQ queue manager:

SMF enhancements which allows the capture of statistics and accounting data on separate intervals. Statistics data is cheap to capture and small in volume, whereas accounting data is more expensive and can be large in volume. In IBM MQ for z/OS 9.3, you have the ability to define separate intervals, allowing more frequent capture of statistics data without having to capture large volumes of accounting data.

IBM MQ for z/OS 9.3 also adds the ability to collect SMF records at more precise intervals, using values for both minutes and seconds. This not only allows for more precise data collection, but also more frequent collection as values of less than one minute may be specified. This can be particularly useful when analysing performance problems.

The addition of **queue statistics** provide relevant data for each queue without the burden of collecting accounting information, or having to write an application to issue the `DISPLAY QSTATUS` command. This allows the user to better monitor the performance of high-value queues over time, using SMF tools.

IBM MQ for z/OS 9.3 adds the **streaming queue** feature, which allows the configuration of a queue to put a near-identical copy of every message to a second queue. Streaming queues can be useful in certain scenarios, where a copy of the message is required. For example:

- Performing analysis on the data going through the system.
- Storing messages for recovery at a later time,
- Capturing a set of messages to use in development and test systems.
- Consuming IBM MQ event messages from the system event queues, and sending additional copies to other queues or topics.

In IBM MQ for z/OS 9.3, new queue managers have, by default, 64-bit log RBA and are capable of having up to 310 active logs. All measurements run in this report use 64-bit RBA queue managers.

Chapter 2

Existing function

General statement of regression

CPU costs and throughput are not significantly difference in version 9.3 for typical messaging workloads when compared with versions 9.1 and 9.2

The reader 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.

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 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 9.3 is similar to both 9.1 and 9.2 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: 9.1, 9.2 and 9.3.

- 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 3KB 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.

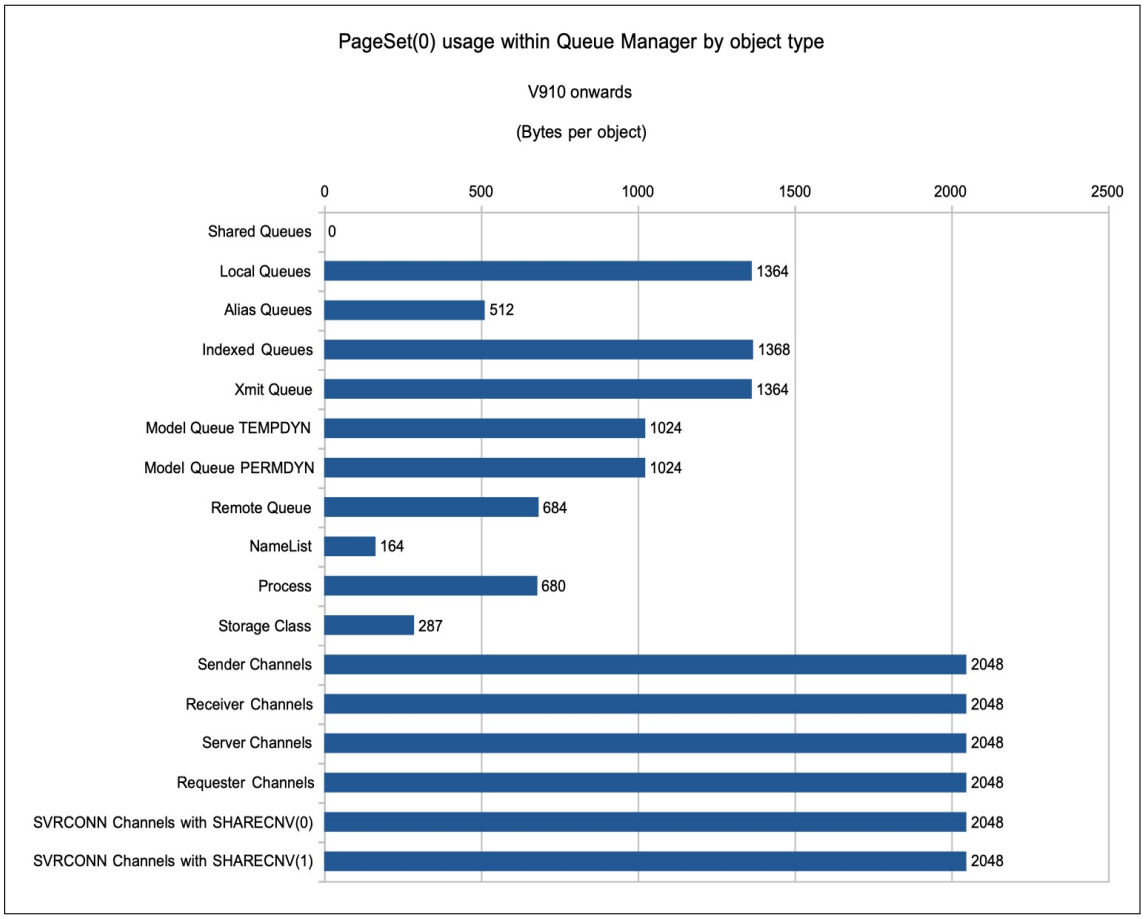
Object Sizes

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.

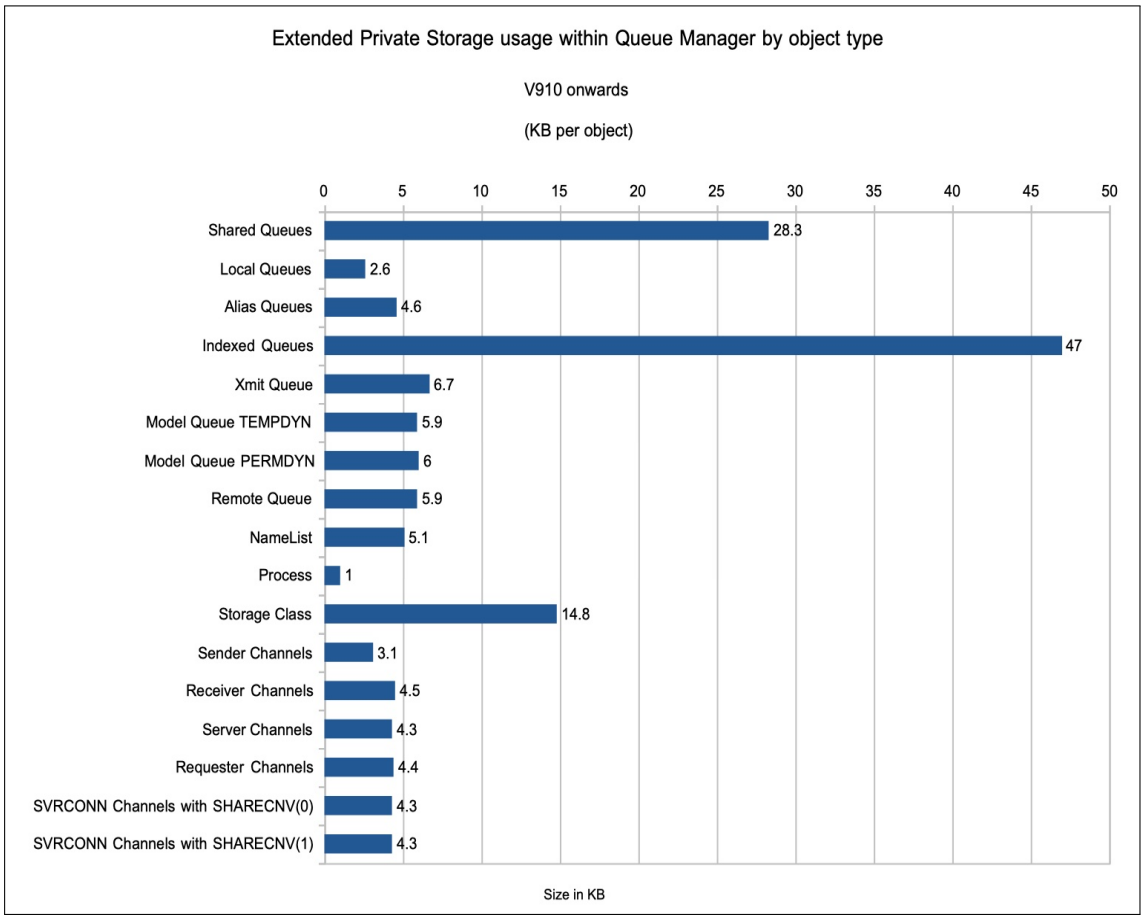
PAGESET(0) Usage

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 v9.0-onwards, the typical storage usage is approximately 16KB 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 24KB.
- Where connection is over a SHARECNV(10) channel and has a CURSHCNV of 10, the usage is 17KB per connection (169.6KB 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 29.5KB, giving an additional shared queue overhead of 5.5KB.

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"[1].

		Channel initiator footprint (KB / SVRCONN channel) Message size (KB)			
IBM MQ release	**SHARECNV**	**1**	**10**	**32**	**64**
9.1	**0**	82	99	161	187
9.2	**0**	82	97	161	187
9.3	**0**	83	99	161	188
9.1	**1**	156	177	234	258
9.2	**1**	157	177	233	264
9.3	**1**	156	178	232	263
9.1	**10**	244	268	347	369
9.2	**10**	244	270	345	383
9.3	**10**	245	268	346	373

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 120 MB of local storage, 1346 MB are free`".

This means that the channel initiator had 1466MB of storage available and has 1346MB 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 1172MB, which in turn means that there is 1052 MB available for channels.

[1] Link will download the performance report as it is no longer available on the IBM SupportPacs site.

If the workload is expected to be clients connecting via SHARECNV(1) SVRCONN channels and using 32KB messages, we could predict that the 9.3 channel initiator could support a maximum of 4,643 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 (KB) per channel (channel initiator)	90	100	109	1125
Overhead of message size increase on 1KB messages		+10	+19	+1035

Chapter 3

New function for 9.3

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

- SMF enhancements are two-fold:
  1. Allowing the data collection interval for accounting data to be set independently of data collection interval for statistics data.
  2. Provide a more granular control of SMF data collection intervals.
- Statistics trace for queues.
- Streaming Queue support for private and shared queues.

Each of these features will be detailed in subsequent chapters.

Chapter 4

SMF enhancements

Queue manager attribute: ACCTIME

Prior to IBM MQ for z/OS 9.3, the queue manager attribute "STATIME" was used to indicate the interval at which MQ's statistics and accounting data was collected.

IBM MQ for z/OS 9.3 introduces the "ACCTIME" queue manager attribute, which allows the data collection interval for accounting data to be set independently of the data collection interval for statistics data.

By separating these two attributes, MQ's SMF data can be collected on different cadences, for example allowing statistics data to be written at a higher frequency than the typically more voluminous accounting data.

Provide more granular control of SMF data collection intervals

Prior to IBM MQ for z/OS 9.3, the statistics and accounting data collection interval was specified in units of minutes, ranging from 1 to 1440. A value of zero means that both the statistics and accounting intervals is collected at the SMF global accounting interval.

With IBM MQ for z/OS 9.3, the data collection periods may be specified in minutes and seconds using the format MMMM.SS, for example `SET SYSTEM ACCTIME(0.05)` will set the accounting data collection interval to 5 seconds, or the format MMMM if values of minutes are sufficiently frequent.

Both STATIME and ACCTIME may be set in three ways:

1. Specifying the values in the CSQ6SYSP system parameter module.
2. Using the SET SYSTEM command.
3. Using PCF SET SYSTEM.

With regards to using the SET SYSTEM command for ACCTIME and STATIME:

- Updated values will not apply until the end of the current interval.
- The old and new values are displayed when using the DISPLAY SYSTEM command.

It is worth noting that the ACCTIME attribute will default to a value of -1, which indicates that the STATIME value will be used for accounting data collection intervals.

When specifying an interval of seconds only, the value must be prefixed with a value of 0.

The smallest valid interval is one second, e.g. `SET SYSTEM ACCTIME(0.01)`.

How frequently should statistics and accounting data be collected?

This is a difficult question to answer in such a way that satifies all users of MQ on z/OS.

Perhaps the question should be, what is a good interval to gather data on, such that:

- The existing performance of your workload is not impacted.
- The data collection is frequent enough to detect trends in response time and general performance.
- The volume of data being generated can be analysed in a timely manner to help and identify system problems.

With the ability to increase data collection frequencies, there are several areas which should be considered before making the changes:

- **CPU**: Impact to existing workload
- **Logging limits**: Data logging rate of MQ SMF data.
- **Storage**: Data management of additional SMF data.
- **Analysis**: Processing of the collected data.

CPU: Impact to existing workload

There is always a cost associated with MQ trace, whether enabling the global queue manager variety or the statistics and accounting trace.

These costs are discussed in the performance report MP16 "Capacity Planning and Tuning Guide" in sections "Queue Manager Trace" and "Channel Initiator Trace" and in much more detail in the blog "MQ and SMF - What, when and how much?".

Typically, the statistics and accounting trace costs are relatively small, but the impact can be more significant depending upon the type of MQ workloads and the trace options set.

For example:

Channel initiator statistics and accounting data (class 4) may only add 1-2% to the cost of the channel initiator address space.

Queue manager statistics costs are also relatively insignificant even when data collection occurs on 1 second intervals.

Queue manager accounting costs can be more significant, particularly when using class(3) accounting trace with high-volume of short-lived tasks, for example a CICS workload, whereas if monitoring a queue manager with long-lived batch tasks, the impact of class(3) accounting trace would be relatively insignificant.

When using class(3) accounting with high-volume short-lived tasks, the SMF interval is largely irrelevant as the task may not span SMF intervals as the SMF data will be written at end of task.

There are several points which are important to make:

1. The changes in IBM MQ for z/OS 9.3 have not altered when the statistics and accounting data is collected. What has changed however, is that there could be more writes to SMF over any given period.

2. Once the data is written to SMF, some of the storage used for data collection for the individual tasks needs to be re-initialized. Since the data is written more often, this re-initialization will also occur more frequently.
3. As a result, data collection cost remains the same as in previous releases but the cost of writing to SMF, as discussed in "MQ and SMF - What, when and how much?", is accrued more frequently.
4. In an environment where there is a high volume of short-lived tasks, changing the ACCTIME to a value of less than 1 minute will have little to no effect on the impact of the accounting trace on either MQ queue manager or application, compared to an environment where ACCTIME is set to 1 minute or higher.
5. In an environment where the workload is primarily long running tasks, for example, but not limited to, batch tasks or long-lived channels, there can be a higher impact from increasing the frequency of class(3) accounting trace to a value of seconds. This is discussed in more detail in "What does this mean to Accounting class(3) costs?"

What this means, is that the costs reported in "MQ and SMF - What, when and how much?" can still be used for estimating the cost of changing the frequency of the statistics and accounting data logging, but there will be a greater impact from the "writing to SMF" phase costs associated with running with MQ statistics and accounting trace.

What does this mean to Accounting class(3) costs?

In terms of cost, the class(3) accounting trace typically has the largest impact to the running costs of MQ compared to any other statistics and accounting traces.

Accounting class(1) trace gathers data about the APIs MQGET, MQPUT and MQPUT1 and only write this data at end of task. As a result, reducing the ACCTIME attribute from a value in minutes to seconds has no impact.

Accounting class(3) records are written at 2 points:

1. For long running tasks - at the end of the SMF accounting interval.
2. At the end of the task.

With an ACCTIME value in seconds rather than minutes, there are many more "end of SMF accounting intervals" In "MQ and SMF - What, when and how much?" we reported the cost of writing accounting class(3) records to SMF as between 4 and 36 microseconds, where the cost depended on the number of queues accessed. In measurements where 2-4 queues were accessed by the task, the costs were of the order of 4 microseconds per write to SMF.

What this means is that moving from an ACCTIME of 1 minute to 15 seconds, would result in 4 times the number of SMF 116 records being written for a particular long running task during a 1 minute period.

Consider the SMF costs of an application that puts 100 messages per second to a queue:

		Cost of write to SMF	
	Data gathering cost	**ACCTIME = 1.00**	**ACCTIME = 0.15**
Application performs 100 puts per second	**0.5 (cost per API) x 100 (APIs / second) x 60 (per minute)**	4 uSecs x 1 write per minute	4 uSecs x4 writes per minute
Total	**3000 uSecs**	4 uSecs	16 uSecs

Costs shown are the **additional cost of class(3) accounting** and are CPU microseconds.

The table shows that for an application that is putting 100 messages a second, the data gathering cost associated with class(3) accounting is 3000 microseconds over a 60 second period.

The cost of writing the data to SMF with `ACCTIME(1.00)` is 4 microseconds, for a total of 3004 microseconds.

By setting `ACCTIME(0.15)` such that data is written every 15 seconds, the cost of writing the data to SMF increases to 16 microseconds, for a total of 3016 microseconds, or an increase 0.4% overall.

Logging limits: Data logging rate of MQ SMF data

When increasing the frequency of data collection, consider that the rate at which data is written to SMF will also increase.

As such, it is imperative that the SMF destination can log the data at the desired rate.

As discussed in blogs "MQ and SMF - Why, which and how?" and "MQ and SMF - How might I process the data?", there are several destinations where SMF can be written - SMF data sets, SMF logstreams and SMF in-memory logstreams.

SMF data sets will be constrained by how quickly the data can be written to them using traditional I/O methods.

If SMF is unable to keep pace with the rate that MQ is generating SMF data, you may see message CSQW133E in the queue manager job log warning of lost SMF data.

Ensure there is sufficient capacity in your disk infrastructure to support additional SMF data without impacting existing workloads.

Additional SMF data logging may result in more frequent log switches, so you may need to provide additional SMF MAN data sets.

SMF logstreams, including compressed logstreams, are typically able to sustain a much higher rate of data collection than SMF data sets. In "IBM MQ and zEDC with SMF" in performance report MP16 "Capacity Planning and Tuning Guide", logstreams allowed twice, while compressed logstreams allowed 7.6 times, the rate of data collection compared to SMF datasets.

SMF logstreams also allow the segregation of SMF types to specific logstreams. For example MQ SMF 115 and 116 records could be written to separate logstreams to those of Db2 SMF records.

Notes:

When using SMF logstreams, if the data collection rate exceeds the available capacity on your system, message IFA786W (SMF data lost - no buffer space available) will be logged to the system log rather than the queue manager job log.

SMF in-memory logstreams are typically used in conjunction with SMF logstreams. It is up to the reading application to continually process the SMF data to avoid exhaustion of SMF buffers.

Storage: Data management of additional SMF data

With a higher rate of data being written to the SMF destination, it is likely that additional storage, whether DASD or tape, will be required to store this additional data.

A simple rule of thumb would be to monitor the volume of MQ SMF data currently being recorded and multiply that by a factor based on the difference between your current STATIME and the proposed value of STATIME and/or ACCTIME.

An indication to the volume of storage required for retaining both statistics and accounting data is provided in "MQ and SMF - What, when and how much?", but it is worth emphasising that increasing the frequency of STATIME, and now ACCTIME, will have a corresponding effect to:

- **All** statistics data classes
- Accounting class(4) data
- Accounting class(3) for long running tasks that span multiple ACCTIME intervals.

Note: Accounting trace using class(1) is not affected by ACCTIME.

For example, consider that you are operating with a STATIME set to 1 minute but you would like to set both STATIME and ACCTIME to 15 seconds. Over an extended period, this will generally result in 4 times the volume of SMF data being written.

If the majority of applications being monitored with accounting trace are short-lived, such as CICS transactions running for less than 1 second, the volume of class(3) accounting trace data generated may not change significantly.

If however, the transactions being monitored are long-running, moving from `ACCTIME(1)` to `ACCTIME(0.15)` would result in up to 4 times the amount of accounting class(3) data being recorded.

For class(3) accounting, it is important to consider the type of applications, short or long-lived, as well as the volume of transactions run over an extended period.

If your environment has large numbers of applications connected to a queue manager or a high volume of short-lived transactions, class(3) accounting trace should only be enabled for short periods. This will allow you to build up a model of the type of transactions using the queue manager without creating too much SMF data.

Analysis: Processing of the collected data

The blog "MQ and SMF - How might I process the data?" offers a number of suggestions for processing MQ statistics and accounting data.

Increasing the frequency of the MQ for z/OS statistics and/or accounting data will increase the amount of data to be analysed. This potentially rules out options that just dump and display the SMF data, such as the sample program `CSQ4SMFD`.

Although applications like those available in MP1B "Interpreting accounting and statistics data" and mq-smf-csv are able to process larger volumes of SMF data, depending on how you intend to use the data, to interpret such volumes of data in a timely fashion to draw or action insights may still be problematic.

With higher volumes of data and a greater need for real-time insights, it is increasingly likely that tools that rely on the IBM Common Data Provider (CDP) to extract and drive analysis, such as IBM Z Anomaly Analytics (ZAA) will become more prevalent.

Benefits of increased granularity of SMF data collection intervals

Despite all of the concerns mentioned relating to increased CPU, data logging and storage quantities, the key benefit is the awareness of what is happening in the MQ queue manager as a result of the data being reported and the ability to extract this data from the queue manager in a more timely fashion.

The ability to report MQ accounting and/or statistics data on a much higher frequency allows change in system performance or workloads to be identified at a much higher granularity and potentially earlier.

For example, with an interval time of 1 minute you might have 100,000 messages being processed. Is that workload evenly spread over the 1 minute, or are there bursts of activity? With an interval time of 1-5 seconds, it is far easier to determine how the workload has been distributed over the same 1 minute period.

Chapter 5

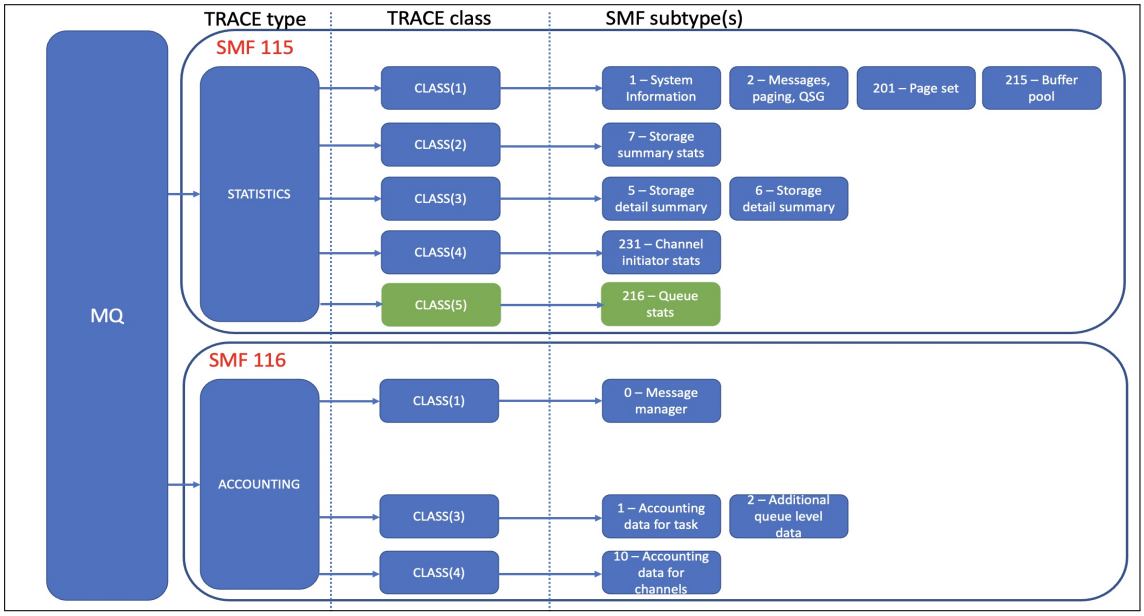
Statistics Trace Class(5)

Queue statistics

IBM MQ for z/OS 9.3, adds the collection of queue statistics to the existing SMF information available at your enterprise. The new queue statistics data provides relevant data for each queue without the burden of collecting accounting information, or having to write an application to issue the `DISPLAY QSTATUS` command. This allows you to better monitor the performance of your queues over time, using your existing SMF tools.

The blog "MQ and SMF - What, when and how much?" discusses what SMF records were available prior to IBM MQ for z/OS 9.3 and includes a diagram showing what SMF records and sub-types are written for each MQ Accounting and Statistics trace. The following updated diagram shows where the new statistics trace fits.

Diagram: MQ Trace type and class with resulting SMF subtype(s)



The new queue statistics records are collected by enabling MQ for z/OS' `TRACE(S) CLASS(5)` and the collected data is detailed in IBM documentation section "Queue data records".

How do I enable queue statistics?

In order to ensure queue statistics data is collected, it is necessary to start `TRACE(S) CLASS(5)` and enable the `STATQ` option on all of the queues for which queue statistics are required.

Queue statistics can be configured either at the queue level using `STATQ(ON | QMGR)`, or if the "QMGR" option is selected then the queue manager must also set `STATQ(ON)`. Note that this will enable queue statistics for **all** queues that have `STATQ(QMGR)`.

It is worth highlighting that the default option for queues is `STATQ(QMGR)`, therefore enabling `STATQ(ON)` at the queue manager may result in all application queues and some `SYSTEM.*` queues collecting queue statistics data.

What is the cost of enabling queue statistics?

The data collected for queue statistics in IBM MQ for z/OS 9.3 is primarily focussed on queue data that is unlikely to change, such as the the page set or the Coupling Facility structure name.

Additionally, MQ collects the queue depth at the time when the SMF data is captured, i.e. at SMF interval end.

To minimise the amount of data written to SMF in IBM MQ for z/OS 9.3, a single SMF 115 subtype 216 record may contain details of up to 341 queues, where each queue statistics record comprises of 84 bytes of data.

This small amount of data collected in conjunction with the frequency of data capture, means that the cost of enabling queue statistics is minimal even when monitoring a thousand queues, of the order of tens of microseconds.

We compared the cost of reporting the queue depth of a set of queues using PCF against the cost of reporting the queue depth using the new queue statistics trace and found that when collecting data for 4 or more queues, the queue statistics was more cost effective than using PCF.

Note: Subsequent releases of MQ will collect additional queue data and this may have the following impact:

- The number of queue statistics records within the SMF 115 subtype 216 record will decrease as more data fields are collected. This will mean that less than 341 queues are written within a single SMF 115 subtype 216 record.

Queue statistics data - example output from MP1B's MQSMF

The application program MQSMF, available as part of MP1B "Interpreting accounting and statistics data", has been updated to support additional output data formats named: `QSTAT` and `QSTATCSV`.

An example of the output from report `QSTAT`:

```
MVAA,VTS1,2022/06/29,21:13:21,VRM:930,
From 2022/06/29,21:13:20 to 2022/06/29,21:13:21, duration 1 seconds.
= Queue Name LQ1001      COMPLETE RECORD    MVAA,VTS1,2022/06/29,21:13:21,VRM:930,
> Disposition Private Pageset ID 1       Bufferpool ID 1
Current Depth 4


MVAA,VTS1,2022/07/01,01:26:23,VRM:930,
From 2022/07/01,01:26:22 to 2022/07/01,01:26:23, duration 1 seconds.
= Queue Name SIXC01      COMPLETE RECORD    MVAA,VKW1,2022/07/01,01:26:23,VRM:930,
> Disposition Shared QSG Name PERF       CF Struct Name APPLICATION1
Current Depth 100
```

Notes on report:

- Queue LQ1001 is a private queue on page set 1, buffer pool 1 and had a message depth of 4 when the SMF record was captured at 21:13:21.
- Queue SIXC01 is a shared queue in QSG "PERF" on structure APPLICATION1 and had a message depth of 100 when the SMF record was captured at 01:26:23.

Chapter 6

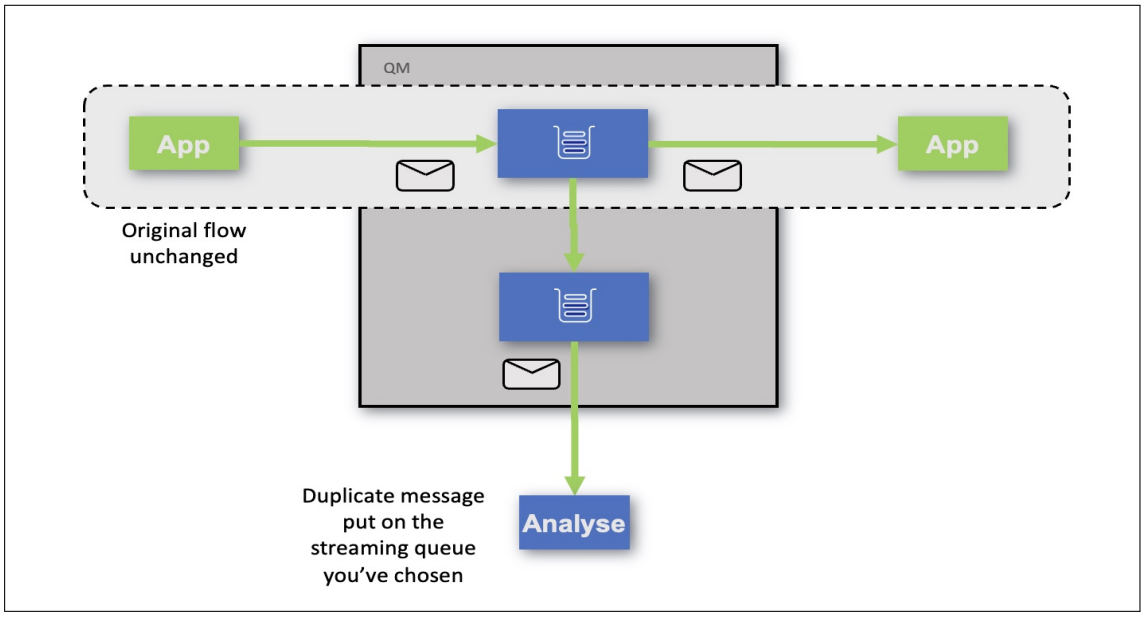
Streaming Queues

Overview

What are Streaming Queues?

IBM MQ for z/OS 9.3 introduces the new Streaming Queues feature. Streaming queues are designed to provide a way for the MQ queue manager to take a copy of messages put to local queues, and deliver the copies to a separate queue or topic of your choosing.

Diagram: Streaming queue overview



With streaming queues configured, when a putting application puts a message to the original queue, a near-identical duplicate message is delivered to the stream queue. They are designed to provide a convenient way of capturing a duplicate stream of messages in order to:

- Stream messages to Apache Kafka using the Kafka Connect source connector for IBM MQ.
- Perform analysis on the data going through the system.
- Store messages for recovery at a later time.
- Capture a set of messages to use in development and test systems.

- Consume IBM MQ event messages from the system event queues, and sending additional copies to others queues or topics.

Because duplicate messages are delivered to the stream queue at put time, messages on the original queue when the stream queue is defined are not sent to the duplicate queue.

The messages sent to the stream queue are almost identical with a small number of exceptions. For example, the following aspects of the stream queue are identical to the original message:

- Message ID
- Correlation ID
- Message properties
- Application data

The following parts of the stream queue message may be different to the original:

- Report options that cause the queue manager to generate new messages are removed.
- Expiry is set to unlimited regardless of the expiry of the original message. If the stream queue has a CAPEXPRY value configured, then that value will be applied to the streamed message.

Configuration

Configuration is done on a queue-by-queue basis. For example if messages are currently delivered to QUEUE1 and it is desired that duplicates of those messages are streamed to QUEUE2, the STREAMQ attribute would be set as follows:

```
ALTER QL(QUEUE1) STREAMQ(QUEUE2) STRMQOS(BESTEF)
```

The `STRMQOS` attribute allows the choice of two streaming queue operating modes:

- **Best Effort (BESTEF)**
  - This option is the default and is the best option for scenarios where you do not want the original application to be affected by the streaming queue feature. Messages will be delivered to the original queue (i.e. QUEUE1) regardless of any problems encountered when streaming messages to QUEUE2. So if QUEUE2 is full when a message is put by the application it will be delivered to QUEUE1 for processing even though the queue manager is not able to stream a copy to QUEUE2.
  - Set using ALTER QL(QUEUE1) STRMQOS(BESTEF)
- **Must duplicate (MUSTDUP)**
  - This option is best for scenarios where it is important that each message delivered to the original queue is also delivered to the stream queue. If there is a problem streaming a message to QUEUE2 then the message will not be delivered to QUEUE1 and the putting application will receive an appropriate error code. The application will need to retry the MQPUT just as it would if QUEUE1 had a problem.
  - Set using ALTER QL(QUEUE1) STRMQOS(MUSTDUP)

Why might I use streaming queues rather than a publish/subscribe model?

Prior to the implementation of streaming queues, if you required a near identical copy of a message, it was possible to configure a pub/sub model using topic alias. This however had several disadvantages over streaming queues, for example:

- Switching from using local queues to alias-to-topic could be complicated.
- Additional MQ objects to manage.
- Loss of message context.

By contrast using streaming queues rather than pub/sub means:

- You have the ablility to leave your original application untouched and still create additional copies of messages.
- No need to stop applications while you reconfigure.
- No need to drain the original queue of messages.
- No need to delete queues and replace with topic aliases.
- Message ID, context etc remain unchanged.

What if I want multiple copies?

Streaming queues allow you to stream the duplicate messages to alias queues as well as local queues. If you define a queue alias to a topic, and configure a queue to stream messages to that topic alias, the duplicate messages are delivered to any subscribers to that topic based on MQ's existing publish/subscribe semantics. Importantly the messages delivered to subscribers will not be identical to the original because MQ's existing publish/subscribe behaviours are retained. One example of this difference is that each message delivered to subscribers will have a new message ID.

It is also worth nothing that if there are no subscribers to the topic the duplicate messages will not be delivered to any applications even when the original queue is set with `STRMQOS(MUSTDUP)`.

Limitations

At launch of IBM MQ for z/OS 9.3, streaming queues was limited to private queues. Subsquently APAR PH49686 provides support for shared queues.

Between IBM MQ for z/OS 9.3 and the availability of APAR PH49686, our performance sysplex was moved onto IBM z16 and as a result, all of the measurements in the streaming queue chapter have been refreshed on the latest available hardware.

What do MQ statistics and accounting traces show?

Statistics trace

Class(1)

The data manager (QIST) record does include a count of messages put to streaming queues.

The message manager (QMST) record includes both the count of opens of streaming queues and the count of messages put to streaming queues.

Since class(1) statistics are generated at the scope of the queue manager, enabling streaming queue may not result in a significant change in the counts reported and as a result there may be little insight gained.

Class(5)

Enabling class(5) statistics in parallel with enabling `STATQ(ON)` on the streaming queue will show the relevant queue statistics data.

Accounting trace

Class(1)

The class(1) accounting records (QMAC) **do not** include counts for messages put to streamed queues, but do include the additional CPU time used on MQPUT and MQPUT1 when using streaming queues.

Since the data reported by class(1) accounting includes the information for tasks *ending* in that interval and therefore may contain information spanning and summarising many hours of activity, the additional cost of enabling streaming queues may not result in a significant change in the CPU time reported for any particular interval.

Using class(1) accounting trace in conjunction with streaming queues is unlikely to provide any significant insight as to the performance of streaming queues on your systems.

Class(3)

The class(3) accounting records (WTID and WTAS) **do not** include counts for messages put to streamed queues, but some insight can be gained when comparing the data from tasks before and after streaming queues are configured.

The following sample data from MQSMF's TASK report, which is part of MP1B "Interpreting accounting and statistics data", shows the differences as a result of enabling streaming queues when the application puts 10 non-persistent messages of 1KB.

Example MQSMF TASK output for batch task performing 10 MQPUTs:

Without streaming queue	With streaming queue
`VTS1 Batch Jobname:R001 Userid:PERFTASK`	`VTS1 Batch Jobname:R001 Userid:PERFTASK`
`...`	`...`
`Pages old 32`	`Pages old 64`
`Pages new 11`	`Pages new 22`
`...`	`...`
`Open name LQ1000`	`Open name LQ1000`

Queue type:	QLocal		LQ1000	Queue type:	QLocal		LQ1000
Queue indexed by	NONE		LQ1000	Queue indexed by	NONE		LQ1000
First opened	2022/07/07,10:42:09		LQ1000	First opened	2022/07/07,10:42:09		LQ1000
Last closed	2022/07/07,10:42:09		LQ1000	Last closed	2022/07/07,10:42:09		LQ1000
Page set ID	3		LQ1000	Page set ID	3		LQ1000
Buffer pool	3		LQ1000	Buffer pool	3		LQ1000
Current opens	0		LQ1000	Current opens	0		LQ1000
Total requests	12		LQ1000	Total requests	12		LQ1000
Open count	1		LQ1000	Open count	1		LQ1000
Open avg elapsed time	39	uS	LQ1000	Open avg elapsed time	44	uS	LQ1000
Open avg CPU time	37	uS	LQ1000	Open avg CPU time	44	uS	LQ1000
Close count	1		LQ1000	Close count	1		LQ1000
Close avg elapsed time	8	uS	LQ1000	Close avg elapsed time	10	uS	LQ1000
Close avg CPU time	8	uS	LQ1000	Close avg CPU time	10	uS	LQ1000
Put count	10		LQ1000	Put count	10		LQ1000
Put avg elapsed time	11	uS	LQ1000	Put avg elapsed time	18	uS	LQ1000
Put avg CPU time	10	uS	LQ1000	Put avg CPU time	18	uS	LQ1000
...				...			
Put + put1 valid count	10		LQ1000	Put + put1 valid count	10		LQ1000
Put size maximum	1024	bytes	LQ1000	Put size maximum	1024	bytes	LQ1000
Put size minimum	1024	bytes	LQ1000	Put size minimum	1024	bytes	LQ1000
Put size average	1024	bytes	LQ1000	Put size average	1024	bytes	LQ1000
...				...			
Total queue elapsed time	159	uS	LQ1000	Total queue elapsed time	236	uS	LQ1000
Total queue CPU used	155	uS	LQ1000	Total queue CPU used	234	uS	LQ1000

Notes on table:

- Pages used - both new and old have doubled as a consequence of configuring a stream queue.
- Open, put and close counts remain the same across configurations.
- Average elapsed and CPU times increase for open, put and close.
- Total queue CPU used increases in-line with the sum of the open, put and close CPU times.

Why don't Accounting classes 1 and 3 report streaming queue data?

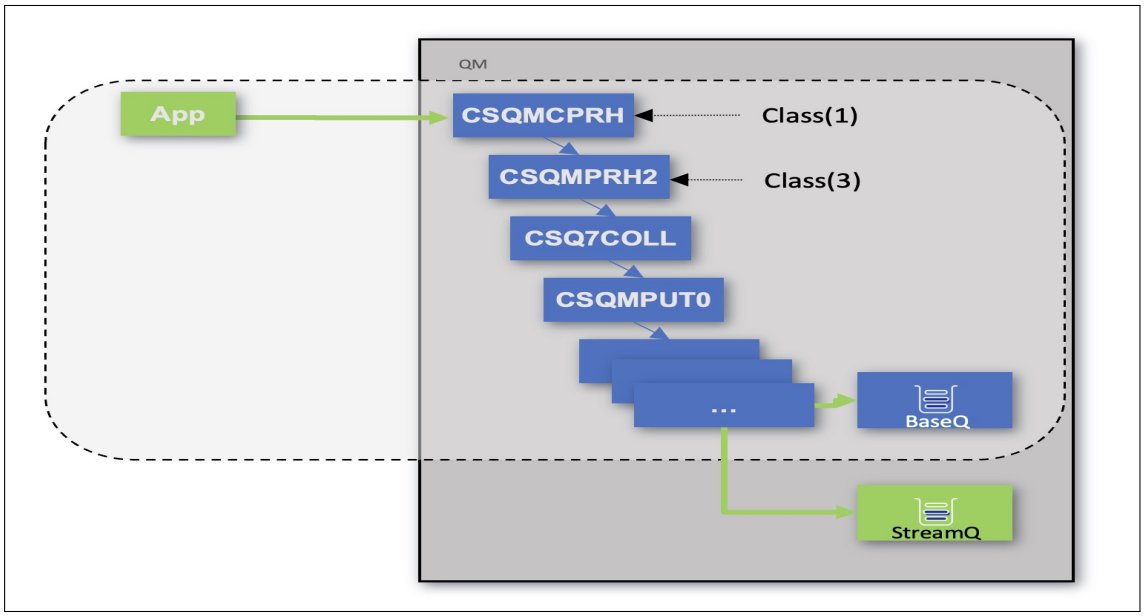
As the previous section states, there is not an obvious indication that streaming queues have been enabled when relying on MQ's accounting trace data, beyond the increased elapsed and CPU time incurred on the MQ APIs.

At the time the class 1 and 3 accounting data is collected, MQ does not have access to determine whether the queue has a streamed queue, at least without additional overhead to the cost of the accounting trace(s).

By contrast the statistics class 1 data is collected much lower down the call stack and therefore is able to include the count of messages put to streaming queues without adding further overhead.

The following diagram shows the initial call stack, based on TRACE(G) data, for an MQPUT and indicates that class 1 and 3 accounting data is collected at a much higher level than when MQ actually performs the put to the streaming queue.

Diagram: Call stack for MQPUT:



Who pays for Streaming Queues?

The putting application is charged for the additional work performed by the configuration of streaming queues.

This may include the channel initiator for messages arriving via MQ channels with a queue destination configured with `STREAMQ`.

The additional time taken for the MQOPEN, MQPUT, MQPUT1 and MQCLOSE APIs will increase the usage of the channel initiator adaptor tasks.

Should you be planning to use streaming queues, it may be beneficial to review your adaptor (`CHIADAPS`) usage using the SMF 115 subtype 231 records generated from enabling MQ's `TRACE(S) CLASS(4)`. This is of particular importance if streaming persistent messages.

When using shared queues for streaming, there will be additional load in both the Coupling Facility (application structure) and the MQ MSTR address space.

Which MQ APIs are affected by Streaming Queues?

Generally, only MQOPEN, MQPUT, MQPUT1, and MQCLOSE are affected by enabling streaming queues, but when using streaming to a shared queue in a separate CF structure to the base queue, there can also be increased cost to the MQ commit.

Primarily this section will concentrate on the costs as reported by class 3 accounting data.

The numbers reported by class 3 accounting data may vary depending on a number of factors - in our test environment, MQ is the focus of the workload whereas in your environment, MQ may be just a small part of the entire workload.

This can affect the reported costs of each particular API - for example in our environment we may be driving the MQOPEN/MQCLOSE APIs at a high rate using a small number of tasks whereas a more real-world configuration might be that many applications are connecting, opening the queue once and performing many MQPUTs and MQGETs. As such, often called MQOPENs in our environment may cost 1-2 CPU microseconds, but the same MQOPEN called occasionally may cost tens of microseconds - which means it is important for the user to look at the values as reported on their own system.

Additionally, and as a side effect of the type of measurements being run for this report, the overhead of streaming queues may be a 1 or 2 microseconds but this may equate to doubling or more the cost of the MQOPEN API. In a different environment where the MQOPEN costs for example 30 microseconds, the impact of streaming queues may be different and as such should be measured on your own system.

Private Queue: Impact of Streaming Queues on MQOPEN and MQCLOSE

It is difficult to separate MQOPEN and MQCLOSE simply because an MQOPEN should be paired with an MQCLOSE, so they will be discussed in parallel.

In our environment the impact of opening a queue where the `STREAMQ` attribute was set, doubled the cost of the MQOPEN. Whilst doubling the cost of the MQOPEN appears high, it is worth noting that this meant the cost increased by 1 CPU microsecond to a total of 2 CPU microseconds.

The following table compares the cost of an MQOPEN of a private queue against a private queue with a streaming queue configured as well as the cost of opening a topic with 0, 1 and 2 subscribers.

MQOPEN	Private queue)	Private queue and streaming queue	Alias-to-topic 0,1,2 subscribers
Cost (CPU microseconds)	1	2	3
Elapsed time (microseconds)	1	2	5

Notes on table:

- Increasing the number of subscribers does not affect the MQOPEN costs of Alias-to-topic.
- Pub/sub also accrues cost under SRB in the MQ queue manager - for MQOPEN this equates to an additional 1.3 CPU microseconds per MQOPEN.

For MQCLOSE the impact was less - the CPU cost was reported as zero whether a `STREAMQ` was configured or not - but the elapsed time increased by 1 microsecond.

Private Queue: Impact of Streaming Queues on MQPUT and MQPUT1

On our performance systems, the impact of streaming queues on MQPUT and MQPUT1 was similar to that of MQOPEN, i.e. in the range of **2 to 2.5 times** cost as reported by class 3 accounting data.

It must be emphasised that this is **not** doubling the application cost by enabling streaming queues - but it is 2-2.5 times the MQ cost of the MQPUT when putting to a queue configured with a streaming queue.

Private Queue: How much does message size affect streaming costs?

Message size does affect the overhead of streaming queues to a certain extent.

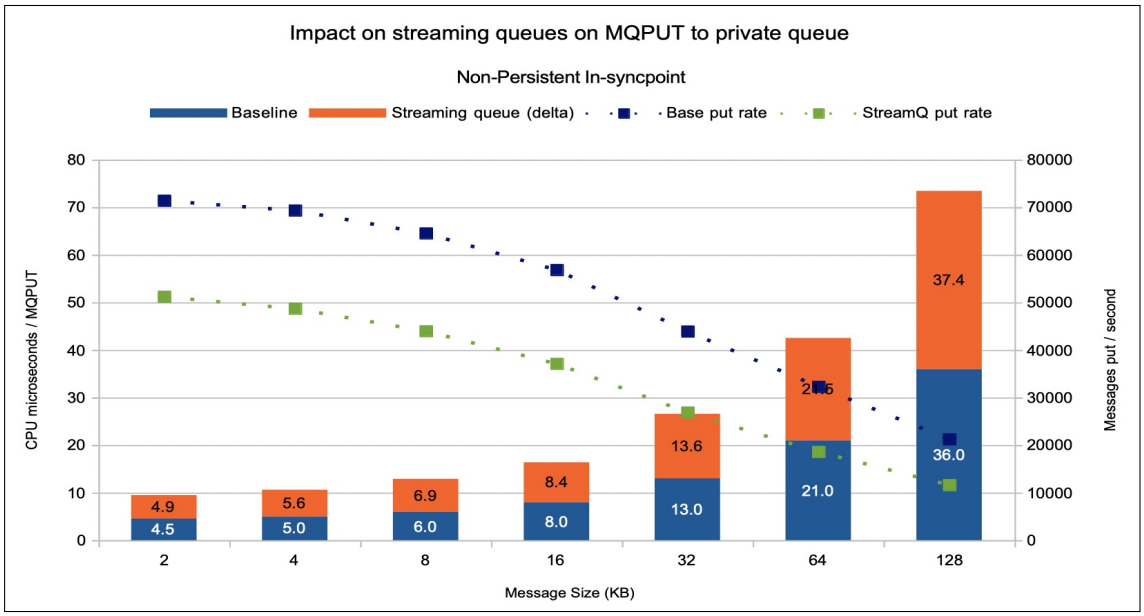
The following table shows the cost of a batch application putting messages of increasing sizes to both a private queue and then to a private queue where a streaming queue is configured. The values reported are class 3 accounting data based on thousands of messages being put to queues where the buffer pool is sufficiently large to avoid the MQPUT being delayed whilst offloading to page set.

Message size	Private queue)	Private queue and streaming queue	Overhead of streaming queue
2KB	4.5	9.5	5
4KB	5	10.6	5.6
8KB	6	12.9	6.9
16KB	8	16.4	8.4
32KB	13	26.6	13.3
64KB	21	42.5	21.5
128KB	36	73.4	37.4

Whilst the overhead of streaming queues grows from 5 to 37 microseconds when increasing messages from 2KB to 128KB, the factor increase remains approximately double that of the private queue without streaming.

The following chart repeats the data in the table, whilst including the rate at which the single batch application was able to put the messages. The application uses a model of [put-then-commit] repeated thousands of times.

Chart: Impact of streaming queues on MQPUT



Notes on chart:

- The stacked chart shows that for 128KB messages, the cost of putting to a private queue cost 36 microseconds - and configuring a streaming queue added 37.4 microseconds to the MQPUT for a total of 73.4 microseconds.
- The achieved rate of the streaming queue configuration was between 45 and 28% lower than the baseline configuration. This is because the batch application is in a tight loop of MQPUT-then-Commit and does not have any business logic. As a result, any impact to MQAPIs causes a significant impact to the achievable performance. Applications with less intensive MQ operations, i.e where MQ does not account for 95% of the application cost, would see a smaller impact from streaming queue.

Shared Queue: Impact of Streaming Queues on MQOPEN and MQCLOSE

The impact of streaming queue on shared queues can be far more significant than for private queues, particularly for the MQOPEN verb.

In our measurements, a typical MQOPEN and MQCLOSE where the queue is shared, costs of the order of 18 CPU microseconds.

With the `STREAMQ` attribute configured to use a shared queue, the cost increased to 32 CPU microseconds. This was regardless of whether the streaming queue was on the same or a different application structure in the same geographically located and configured Coupling Facility.

The "first-open and last-close" effects as discussed in MP16 "Capacity Planning and Tuning Guide" section "Frequent opening of shared queues" can make a significant difference to the cost of both the MQOPEN and MQCLOSE when accessing shared queues.

Avoiding CF access for the open and close of a queue configured *without* a streaming queue reduces the cost from 18 to 5.5 CPU microseconds.

Avoiding CF access for the open and close of a queue configured *with* a streaming queue reduces the cost from 32 CPU microseconds, but the scale of the impact will depend on how that CF is avoided.

The following table illustrates the impact of avoiding first-open and last-close effects on the queue referenced by the `STREAMQ` attribute.

Configuration	Effect on streaming queue	Cost of MQOPEN and MQCLOSE (CPU microseconds)
First-open and last-close effect	CF Access every time	32
Hold base queue open for input wrong type	CF Access every time	20
Hold base queue open before streamQ configured	CF Access every time	20
Hold base queue open when streamQ in separate structure	CF Access every time	20
Hold only streamQ open	No CF Access	20
Hold base queue open with streamQ in same structure	No CF Access	7.5
Hold base queue and stream queue open (different structures)	No CF Access	7.5

Shared Queue: Impact of Streaming Queues on MQPUT and MQPUT1

On our performance systems, the impact of using shared queues when streaming on MQPUT results in the costs increasing by approximately **2 to 3 times**, as reported by class 3 accounting data. The impact of streaming queues on MQPUT1 can be more significant, with the costs as reported by class 3 accounting data increasing 3 times.

As with private queues, it must be emphasised that the impact of streaming queues on MQPUTs to shared queues is not necessarily doubling the application cost.

Shared Queue: How much does the type of MQPUT affect the cost?

With shared queues, the impact of adding a streaming queue can vary depending on a number of factors, including whether the message is put in syncpoint. The following table offers example class 3 accounting costs for MQPUT and MQPUT1 in a range of configurations when putting a 1KB non-persistent message.

Configuration	STRMQOS	Baseline put cost (CPU microseconds)	StreamQ put cost (CPU microseconds)
MQPUT out-of-syncpoint	BESTEF	7	15
MQPUT out-of-syncpoint	MUSTDUP	7	22
MQPUT in-syncpoint	Either	7	15
MQPUT in-syncpoint with StreamQ on separate structure	Either	7	14
MQPUT1 in-syncpoint	Either	10	32

Notes on table:

- The high cost of MUSTDUP on the MQPUT out-of-syncpoint measurement is due to effectively enforcing an MQ commit such that the put is successful for both or neither queue. The blog "MQ for z/OS - CF statistics" discusses the additional CF statistics data included as part of MQ's task records, and reviewing this data shows that the additional cost arises from MQ having to issue 2 additional "New", 1 "Write" and 1 "MoveEnt" calls to the Coupling Facility.
- The three-times cost increase incurred when using MQPUT1 with shared queues is due to 1 additional "new" and 2 "write" requests.

Shared Queue: How much does message size affect streaming costs?

Message size does affect the overhead of streaming queues to a certain extent.

The following table shows the cost of a batch application putting messages of increasing sizes to both a shared queue and then to a shared queue where a streaming queue is configured. The values reported are class 3 accounting data based on thousands of messages being put to queues.

Message size	Shared queue	Shared queue and streaming queue	Overhead of streaming queue
2KB	7.3	15.5	8.2
4KB	9	18	9
8KB	11	23	12
16KB	15.4	32.3	16.9
32KB	23.4	51.9	28.5
64KB	25.4	51.9	26.5
128KB	36.9	74.6	37.7

Whilst the overhead of streaming queues grows from 8.2 to 37.7 microseconds when increasing messages from 2KB to 128KB, the factor increase remains approximately double that of the shared queue without streaming.

Shared Queue: Impact of Streaming Queues on MQCMIT

When the `STREAMQ` is configured on a separate application structure to the base queue, there is additional cost in both the z/OS LPAR and the Coupling Facility, and this is due to the impact of additional CF work at the time of the commit.

The following table compares an MQPUT and MQCMIT when using streaming queues, where in one instance the streaming queue uses the same application structure as the base queue and in the second instance the streaming queue uses a separate application structure.

Once more, we are using the CF statistics as discussing in the blog "MQ for z/OS - CF statistics".

	Queues on same structure	**Queues on separate structures**
MQPUT to base and streaming queue	2 x "New"	2 x "New"
MQCMIT	1 x "New" 1 x "Write" 1 x "MoveEnt"	2 x "New" 1 x "Write" 2 x "MoveEnt"
z/OS cost per MQPUT	14 CPU microseconds	15 CPU microseconds
z/OS cost per MQCMIT	8 CPU microseconds	18 CPU microseconds
CF cost per MQPUT	8 CPU microseconds	8 CPU microseconds
CF cost per MQCMIT	11 CPU microseconds	29 CPU microseconds

Notes on table:

- The MQPUT performance is not notably impacted by the streaming queue being on a separate application structure.
- The MQCMIT performance is affected by having to commit across the two structures, with additional calls to the CF in the form of 1 "New" and 1 "MoveEnt". These additional CF calls result in both increased load on the z/OS LPAR and the CF.

How much will streaming queues cost?

Throughout this chapter, we have reported that according to class 3 accounting data, the impact of using queues with `STREAMQ` configured is to effectively double the cost of the affected MQ APIs.

How much enabling streaming queues will affect your workload will very much depend on how you measure the cost of the MQ workload. For example, if the costs are measured using the RMF Workload report and based on the CPU used by the MQ queue manager and the applications, the impact may be much less than observed from the class 3 accounting data.

Similarly if using class 1 accounting data, the impact may be much less than reported by class 3 accounting data.

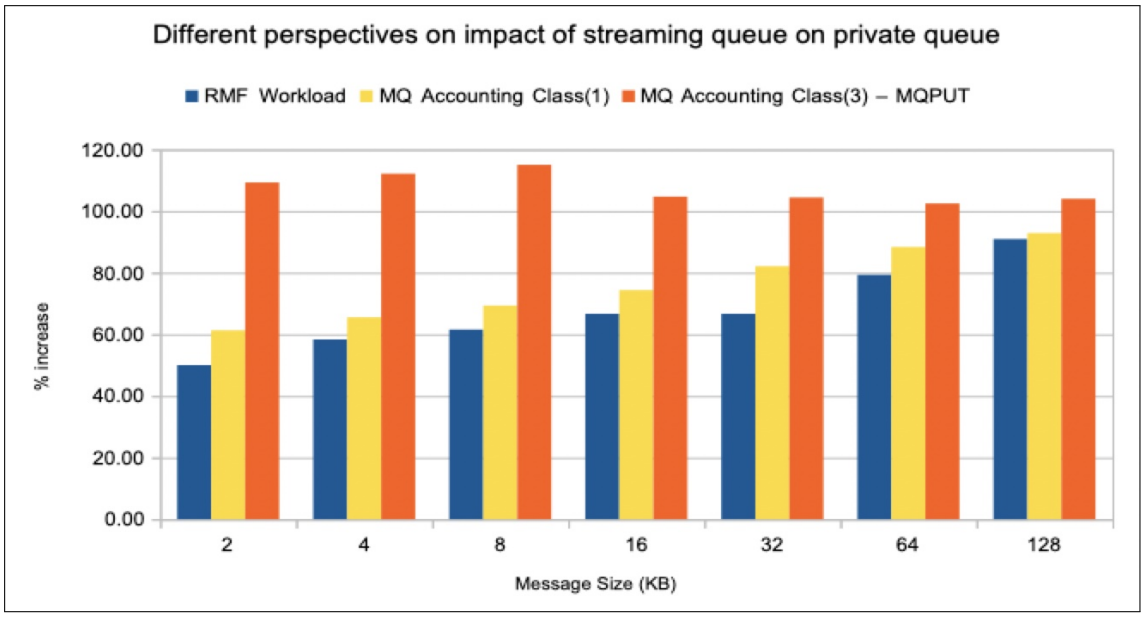
In these measurements, we use a simple batch application that opens the queue and then repeatedly put messages to the base MQ queue, before closing the queue and exiting. The measurements are then repeated with streaming queues configured. A more complex application that performed MQGETs, and performed database requests or file I/O would see a much smaller impact from streaming queues as the proportion of MQPUT work compared to the total workload would be much less.

The measurements compare the impact of streaming queues based on comparing the costs from the:

- RMF Workload report - calculating the transaction cost based on the CPU used by MQ MSTR address space and the application.
- MQ class 1 accounting data - based on the total cost of an MQOPEN, multiple MQPUT and MQCMIT calls and MQCLOSE, divided by the number of MQPUTs.
- MQ class 3 accounting data - based on the average cost of the MQPUT.

The first chart shows that the impact of streaming queues on a private queue workload can vary significantly depending on which data is compared, and the second chart presents the data from the test using shared queues.

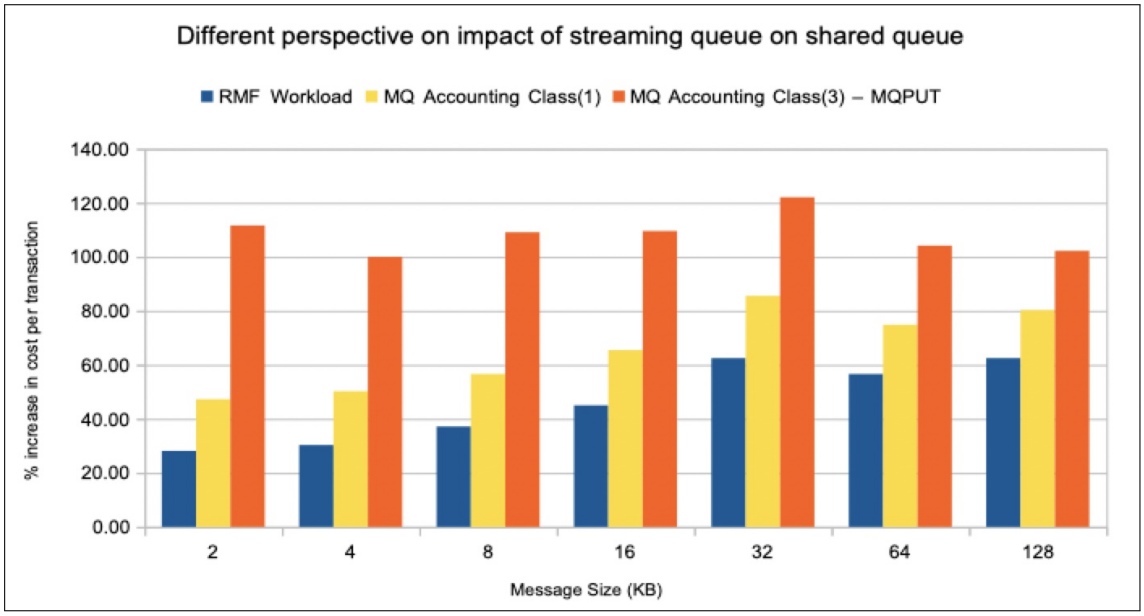
Chart: Impact of streaming queues on private queue - comparing different sources of cost



Notes on chart:

- With a very simple application primarily calling MQ APIs, the impact of streaming queues according to the RMF Workload report is to add between 50 and 90%.
- Class 1 accounting data shows an impact of between an additional 60 to 90%.
- Class 3 accounting data shows the impact to be approximately doubling the MQPUT cost.

Chart: Impact of streaming queues on shared queues - comparing different sources of cost



Notes on chart:

- With a very simple application primarily calling MQ APIs, the impact of using shared queues for streaming according to the RMF Workload report is to add between 25 and 60%.
- Class 1 accounting data shows an impact of between an additional 45 to 80%.
- Class 3 accounting data shows the impact to be approximately doubling the MQPUT cost.

Must Duplicate or Best Efforts?

As discussed in the configuration section, the **must duplicate** option ensures that either both the original message and the streaming queue message are delivered or neither are delivered.

By contrast, the **best effort** option will allow the original message to be put, even if the streaming queue message is not able to be delivered.

Should an error occur when putting the streaming message, regardless of streaming quality of service (QoS), a message will be written to the queue manager job log in the form:

```
CSQM587E VTS1 CSQMPUT Unable to put to streaming queue STRMQ000 for queue LQ1000, mqrc=2192 (MQRC_PAGESET_FULL)
```

This message should only be logged once for the life time of the application handle, unless the error changes.

In the event of the `CSQM587E` message being logged, the MQPUT API may get a non-zero return code, depending on the `STRMQOS` value.

Consider the following:

- Queue manager has queue LQ1000 defined on page set with 2 million pages.
- Queue manager has queue STRMQ000 defined on page set with 20,157 pages and `EXPAND(NONE)`.
- Queue LQ1000 configured to use STREAMQ(STRMQ000) with STRMQOS configured as either BESTEF or MUSTDUP.

A single batch application is configured to put 500,000 non-persistent messages of 1KB to queue LQ1000.

The following is an example of the different outcomes based on the value of `STRMQOS`.

STRMQOS(BESTEF):

- Application reports 500,000 messages put to queue LQ1000 and exits with return code 0.
- Queue manager logs `CSQM587E` with return code 2192 (MQRC_PAGESET_FULL).
- LQ1000 depth reported as 500,000.
- STRMQ000 depth reported as 19,980.

STRMQOS(MUSTDUP):

- Application reports MQPUT failed with mqrc=2192 MQRC_PAGESET_FULL, on iteration 19,881.
- Queue manager logs `CSQM587E` with return code 2192 (MQRC_PAGESET_FULL).
- LQ1000 depth reported as 19,980.
- STRMQ000 depth reported as 19,980.

What else differentiates MUSTDUP and BESTEF?

Setting MUSTDUP as the QoS on a stream queue has a small performance impact as there is additional work carried out to ensure that the original message is rolled back if the MQPUT of the duplicate failed.

How much does MUSTDUP affect performance?

The impact of MUSTDUP is relatively small when compared to BESTEF, and a comparison is offered in the following scenario.

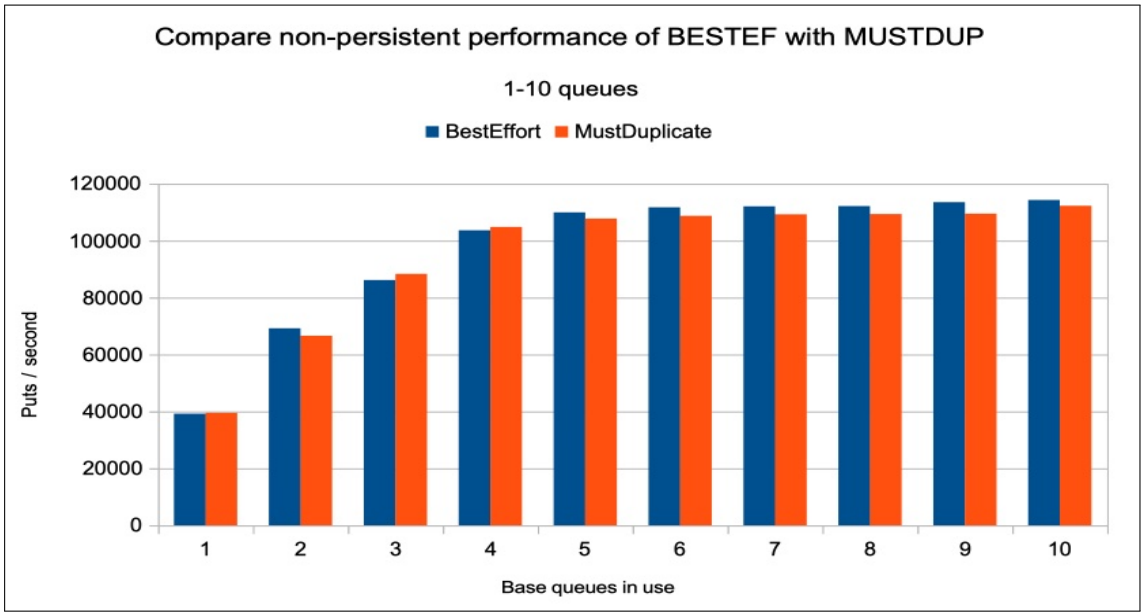
The test shown uses a set of applications - one putting task and two getting tasks per queue. One of the getting tasks will additionally reply to the putting task, whereas the second task will perform gets-only.

As time progresses, the test adds further sets of applications until there are 10 queues in use, with a total of 10 putting tasks and 20 getting tasks.

The data is reported as the average transaction rate over multiple 15 second SMF intervals per queue count.

This chart compares the performance of streaming queues using BESTEF against MUSTDUP and shows a small impact to the performance due to the additional processing required to ensure that puts to both the base and streaming queue are completed with the same outcome to both queues.

Chart: Non-Persistent message comparison of Streaming Queues BESTEF against MUSTDUP



Notes on chart:

- The BESTEF streaming queues configuration achieved a 16% improvement in transaction rate over the equivalent MUSTDUP streaming queue configuration.
- MQ queue manager costs were comparable when using streaming queues with MUSTDUP and BESTEF.
- Put application costs increased by 1-2 CPU microseconds when using MUSTDUP rather than BESTEF.
- Get application costs were similar in both configurations.

Streaming Queues and Message Expiry

One use for streaming queues is to create duplicate messages which will be stored for a short period of time as a contingency measure. In this case, the `CAPEXPRY` custom property is set on the streaming queue to set a time limit on messages, for example to set the messages to expire after 20 minutes:

```
ALTER QL(STRMQ000) CUSTOM('CAPEXPRY(12000)')
```

IBM MQ for z/OS schedules an expiry task run at an interval specified by the queue manager `EXPIRYINT` attribute, for example:

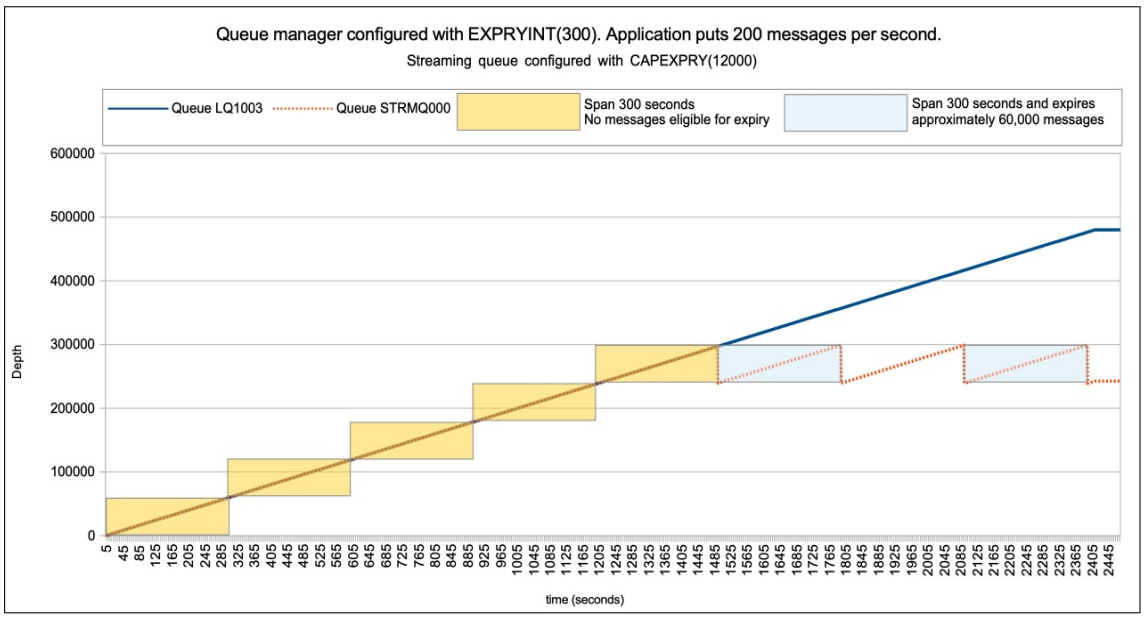
`ALTER QMGR EXPRYINT(300)` will set the expiry timer to run every 300 seconds.

This expiry task deletes any messages that are older than the `CAPEXPRY` value set on it. `CAPEXPRY` has queue scope, and only affects messages arriving after it has been set, similarly amending the `CAPEXPRY` value will only affect subsequent messages.

If the messages on a streaming queue with `CAPEXPRY` set are not being consumed, then the queue will grow until the expiry task is scheduled and finds messages that are old enough to be deleted.

The following diagram shows the depth of the base queue (LQ1003) and the streaming queue (STRMQ000) as MQ's expiry task is run every 300 seconds and messages are expired from the streaming queue after 20 minutes.

Diagram: EXPRYINT and CAPEXPRY example:



Notes:

In this example the messages are arriving at 200/second on a single queue. The plot shows the depth of the base and streaming queue, where the streaming queue has `CAPEXPRY` is set to 20 minutes and the queue manager has `EXPRYINT` set to the value of 300 seconds. Both queues grow in depth until the expiry task encounters messages that are 20 minutes or more old. At that point, each time the expiry task is run, it will find another 5 minutes worth of messages that can be deleted, creating the saw-tooth effect above.

When using message expiry with streaming queues, you should consider the depth that the queues will grow to and ensure that MQ page sets are configured such that streaming queues with `STRMQOS(MUSTDUP)` do not cause failures in the primary applications.

EXPRYINT - why is the default value "OFF"?

The expiry task will only expire messages that are eligible for expiry based on their `CAPEXPRY` value.

Typically the cost of the MQ task identifying and subsequently deleting eligible messages for expiry is relatively small but is not zero.

For **private queues**, we have measured the costs as approximately 4 CPU microseconds per message expired when the messages are exclusively held in buffers. This cost increases to 7 microseconds when the task needs to access the data on the MQ page set, however MQ does not need to read the full message back into buffer pools, which does reduce the amount of page set I/O.

As the process to identify eligible messages is de-coupled from the process to delete the messages, the expiry task is able to identify and delete up to 250,000 messages per CPU second for expiring.

For **shared** queues, the impact of expiry processing is more significant.

MP16 "Capacity Planning and Tuning Guide" states:

- Each defined shared queue must be processed.
- A single queue manager, of those with non-zero `EXPRYINT` in the QSG, will take responsibility.
- ***All*** queues, including those where expiry is not set, will be scanned.

What this means is that if `EXPRYINT` is set on a queue manager that is part of a QSG, even if the intention is to expire messages solely on private queues, at the expiry interval the queue manager will also scan all shared queues, regardless of whether `CAPEXPRY` has been set.

In our system where we have a local CF which is non-duplexed with the fastest available links and sufficient dedicated processors, scanning shared queues with combined depths of 225,000 messages where there were no queues configured to use message expiry, resulted in 0.877 CPU seconds of CF. In this instance there were 225,000 synchronous requests to the CF with an average time of 3.9 microseconds.

When message expiry was set on the shared queues holding 225,000 messages, the impact to the CF increased due to 450,000 synchronous requests with an average time of 3.1 microseconds. This meant that to expire 225,000 messages, the CF used 1.4 CPU seconds.

If the `EXPRYINT` is set to a small interval on systems with less responsive CF's or more messages on queues, there is a risk that the expiry task will be overwhelmed (*unable to complete before the next interval begins*).

Due to the risk of increased cost of scanning all shared queues for messages eligible for expiry, even on queues where `CAPEXPRY` is not set, the detault setting for `EXPRYINT` is `OFF`.

Performance of Streaming Queues vs Pub/Sub on private queues

In this section, the performance of streaming queues is compared directly with a pub/sub configuration where there are 2 subscribers. This gives the best comparison point when two copies of a message are required.

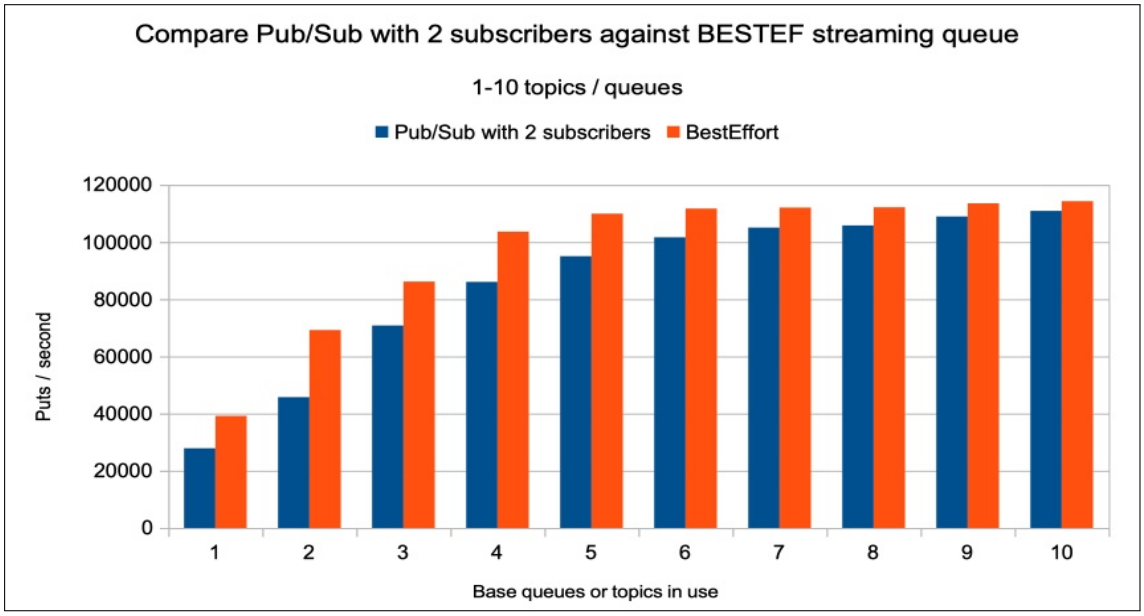
The test shown uses a set of applications - one putting task and two getting tasks per queue / topic alias. One of the getting tasks will additionally reply to the putting task, whereas the second task will perform gets-only.

As time progresses, the test adds further sets of applications until there are 10 queues (or topic aliases) in use, with a total of 10 putting tasks and 20 getting tasks.

The data is reported as the average transaction rate over multiple 15 second SMF intervals per queue count.

Performance of Non-Persistent messages

Chart: Non-Persistent message comparison of Pub/Sub with Streaming Queues (BESTEF)



Notes on chart:

- The streaming queues configuration is achieving an 18% improvement in transaction rate over the equivalent pub/sub configuration.
- MQ queue manager costs were comparable per transaction when using streaming queues compared to the equivalent pub/sub configuration.
- Put application costs were 30% lower per transaction when using streaming queues compared to the equivalent pub/sub configuration.
- Get application costs were similar in both configurations.

Comparing the performance of MUSTDUP against pub/sub with 2 subscribers can be summarised as:

- The MUSTDUP streaming queues configuration is achieving up to 16% improvement in transaction rate over the equivalent pub/sub configuration.

- MQ queue manager costs per transaction were comparable when using MUSTDUP streaming queues compared to the equivalent pub/sub configuration.
- Put application costs were 27% lower per transaction when using MUSTDUP streaming queues compared to the equivalent pub/sub configuration.
- Get application costs were similar in both configurations.

Performance of Persistent messaging

When enabling streaming queues on workloads that use persistent messages, it is important to recognise that there will be additional load on your MQ logging capacity.

Putting persistent messages to queues configured with streaming queues will effectively result in **twice** the amount of data being logged for those transactions.

This additional load may impact your system in a number of ways.

If the queue manager is already running at the maximum capacity of the MQ logger task, the additional load may result in a decrease in transaction rate of the putting tasks - or impact other workloads that also rely on persistent messaging.

If the queue manager is not running at capacity, or else there is additional capacity in the MQ logger task but not for your existing workload(s), enabling streaming queues with persistent messages will increase the API time of the applications opening, putting and closing the queues with streaming queues configured as well as increasing the time to write the data to the MQ log data sets(s).

Note, to clarify the statement "there is existing capacity in the MQ logger task but not for your existing workload(s)", is where the MQ SMF log manager records indicate that the logger task is fully utilised for the entire SMF interval but the pages per I/O is less than 128. This can be determined from MP1B's MQSMF program using the `LOG` report, as below:

```
...
Pages written per I/O:   11
...
Log write rate:          220MB/s per copy
Logger I/O busy:         94.66%
Logger task busy:        99.9%
```

In this instance, the `LOG` report shows that the logger task was 99.9% busy throughout the interval, therefore can be regarded as fully utilised and the time spent waiting for I/O accounts for 94.66% of the interval. The simplest, but not necessarily the least costly, way to increase the capacity of this queue manager whilst writing 11 pages per I/O would be to have more responsive DASD and faster links to the DASD.

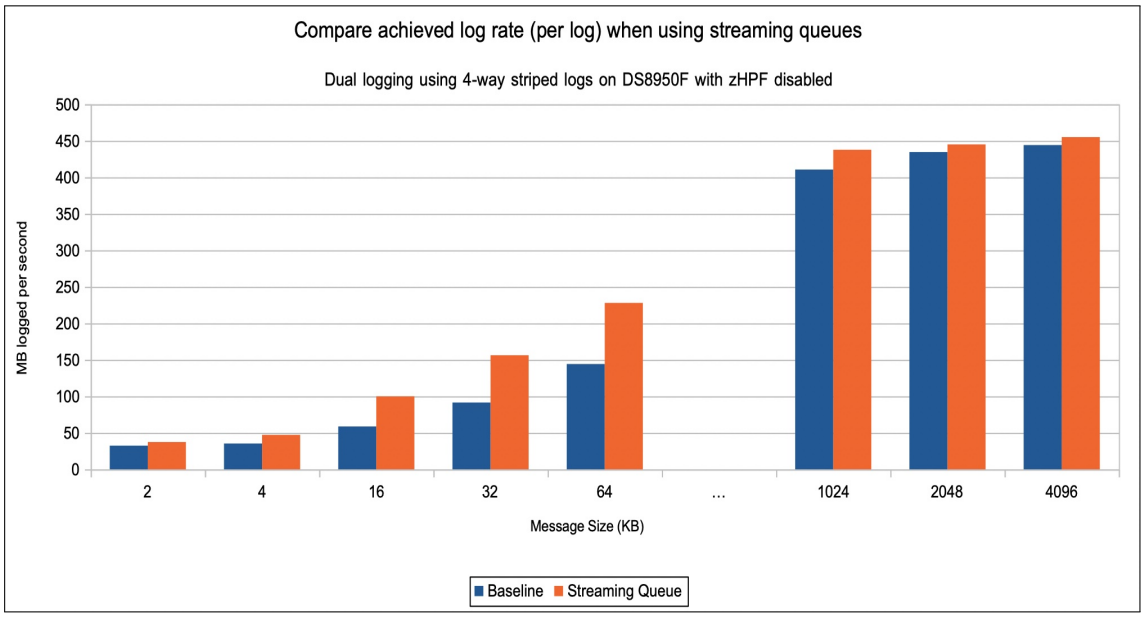
Additionally there were 11 pages written per I/O (i.e. 44KB per I/O) and the log task was able to sustain 220MB per second per log copy.

In this example we can regard the log task as fully busy *for the current workload*, but this does not mean that the queue manager does not have capacity for more persistent work. Potentially the log task could write 128 pages per I/O request, which is a significant step up from the 11 pages it is currently writing. It is worth noting that writing 128 pages in a single I/O is likely to take longer than writing 11 pages per I/O - and therefore the *existing* workloads may see a drop in performance.

To illustrate the impact of streaming queues on persistent messaging, we can compare the performance with the upper bounds of persistent logging rate tests in the regression section of this report.

This particular workload uses just 3 batch tasks to drive the MQ log task to 100% busy. One of the tasks uses a 1KB persistent message that is put and got repeatedly from a queue. The remaining two tasks vary the size of their message as per the x-axis but also repeatedly perform MQPUT and MQGETs.

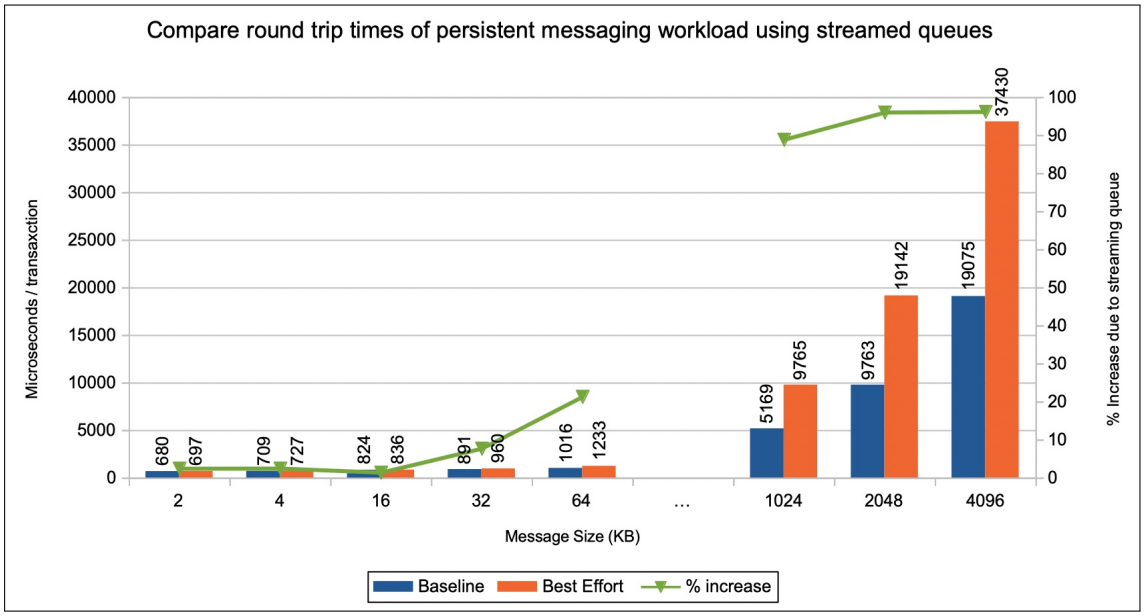
Chart: Achieved log rate when using streaming queues.



For each message size used, the baseline queue tests reported the log task as being fully busy throughout the SMF intervals. Despite this, for the message sizes up to 64KB, adding streaming queues resulted in the log write rate increasing by up to 70%. For messages of 1MB and larger, where the log task was already writing 100+ pages per I/O, the log task was already close to capacity, and adding streaming queues only increased the maximum log rate by 2-6%.

The following chart shows the impact that applying a streaming queue to the workload, so that all messages put by the 3 batch tasks are duplicated. The chart shows the round-trip times, i.e. the time to complete a transaction consisting of the MQPUT and the MQGET for the tasks using the message size as per the x-axis.

Chart: Impact to transaction round-trip times when enable streaming queue on persistent workload.



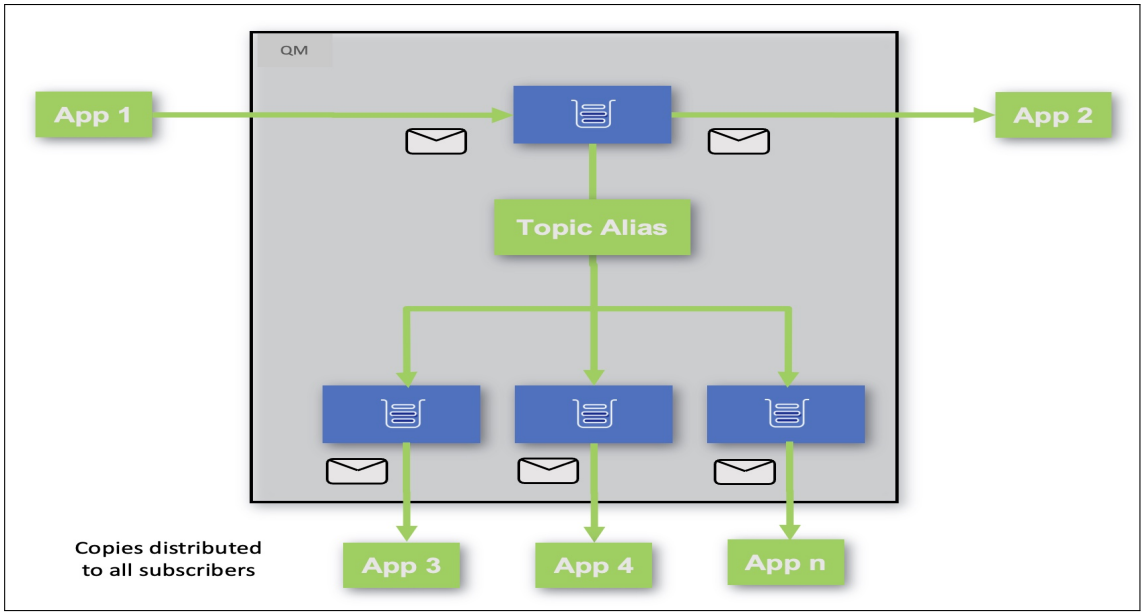
Notes on chart:

- For messages up to 32KB, the round-trip times increase by less than 10% as despite the log task reported 100% utilisation, there were low numbers of pages per I/O.
- For 64KB messages, the round-trip times have increased by 20% even though the overall log rate has also increased by 70%
- For 1-4MB messages the round-trip times have nearly doubled due to the log task being fully utilised even without log streaming being enabled.

In this example, we see the overall log rate has increased as a result of enabling streaming queues, but the transaction rate has decreased due to log limits.

Streaming queue with multiple copies

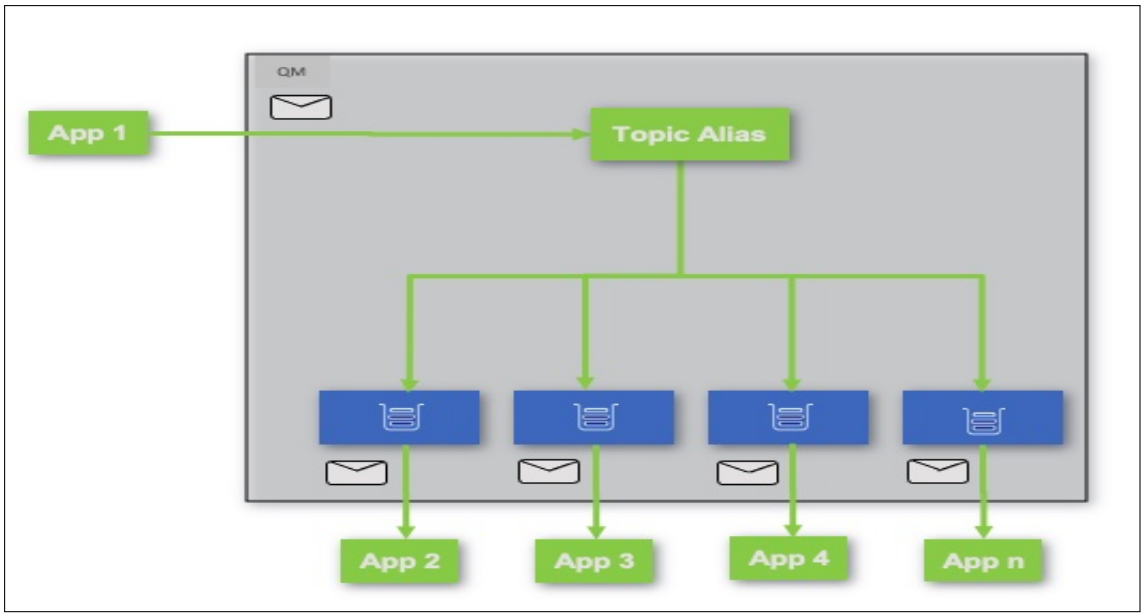
Diagram: Topology for 1 to n duplicate messages using streaming queues with Pub/Sub.



The above diagram shows how multiple duplicates of messages being put onto a queue can be generated by setting the `STREAMQ` attribute to a topic alias and using Pub/Sub to distribute messages to multiple subscribers. The advantages of this solution is that App 2 does not need to be switched to a new subscriber queue and the existing applications do not need to be stopped to make the necessary queue manager changes.

The following diagram shows the topology of using Pub/Sub to generate multiple messages. If this approach is used to create duplicate messages for an existing application, the App 2 needs to be changed to consume from a subscription, rather than getting from the original queue.

Diagram: Topology for 1 to n duplicate messages using Pub/Sub.



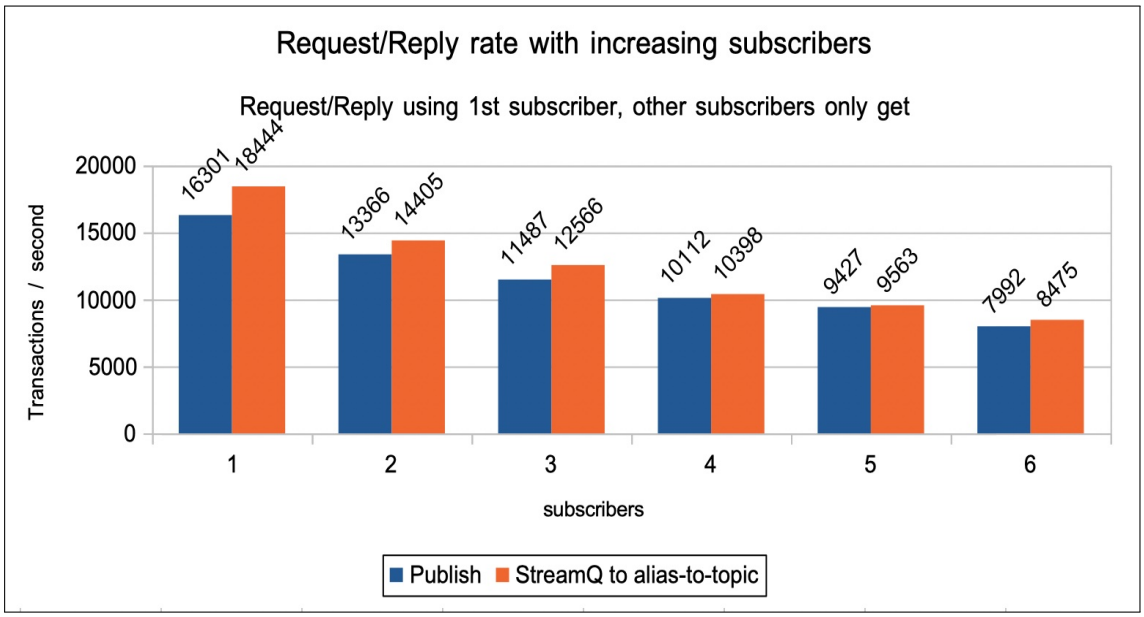
In this section, the performance of using streaming queues configured with the `STREAMQ` attribute referencing a topic alias where there are increasing numbers of subscribers is compared to using Pub/Sub to generate multiple messages.

The Pub/Sub tests are configured with 1 to 6 subscribers, whereas the streaming queue tests are configured with 0 to 5 subscribers in addition to the original getting application. In both configurations the test ends with 6 applications getting messages from a single putting application.

In each case, App 1 puts a message and waits for a reply from App 2. The remaining applications are configured to only get the published messages and ensure that queue depths remain low.

Workload uses 1KB non-persistent messages.

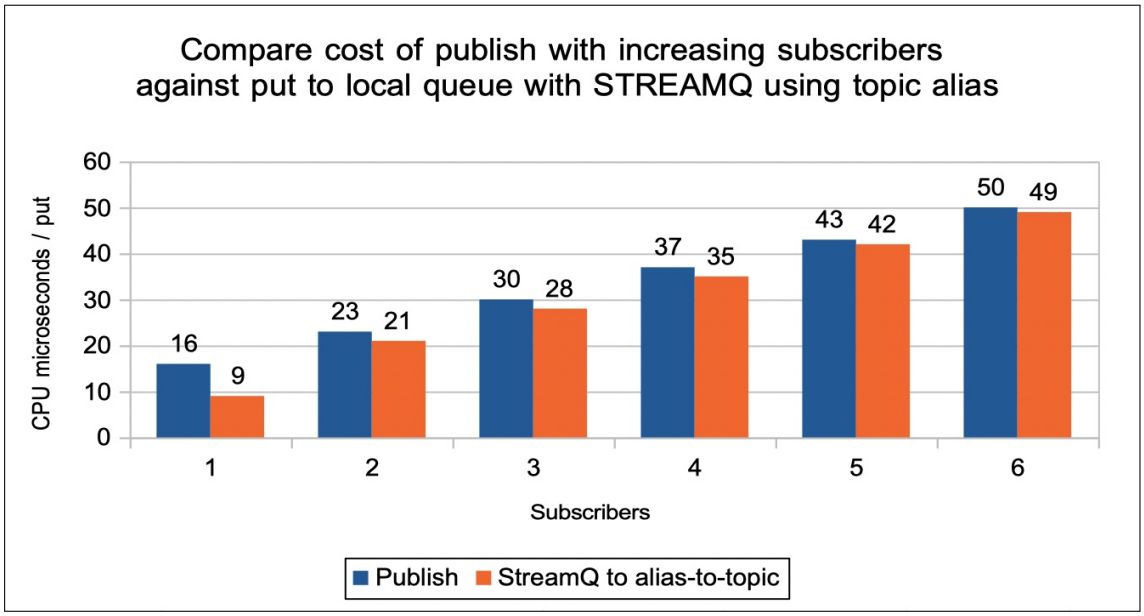
Chart: Compare performance of STREAMQ to topic alias with increasing subscribers against performance of Pub/Sub to topic alias.



Notes on chart:

- The achieved transaction rate for the request/reply workload using the `STREAMQ` configuration is between 2 to 13% higher than the equivalent Pub/Sub configuration.
- The throughput is limited by round-trip times, rather than CPU.
- For applications that were only getting messages, queue depths remained low.

Chart: Compare cost of STREAMQ to topic alias with increasing subscribers against performance of Pub/Sub to topic alias.



Notes on chart:

- Costs shown the MQPUT costs of the driving application. Costs of the server applications, whether just getting messages or getting and putting messages are comparable across configurations.
- Costs shown are as reported by Accounting Class(3) and include CPU time and Additional SRB time, which is used by the Pub/Sub processing.
- For streaming queue to topic alias with 1 subscriber measurement, the topic has 0 subscribers and as a result the cost os the Pub/Sub processing is much less than when there is at least one subscriber..
- Costs are relatively similar between the two configurations except when there are 0 subscribers in the `STREAMQ` configuration.

Pub/Sub and Accounting Class(3) data

When using `STREAMQ` to reference a topic alias whether there are **0, 1 or many subscribers**, there is additional cost of Pub/Sub that may affect the cost of the MQOPEN, MQPUT, MQPUT1 and MQCLOSE.

This cost is reported in Accounting Class(3)'s WQSTAT fields `TOPICOPENSRB`, `TOPICPutSRB`, `TOPICPut1SRB` and `TOPICCLOSESRB`.

Accounting class(3) data reports the number of messages published using the `WQSTATS PublishedN` field.

In a Pub/Sub environment, this value can be divided by the number of puts to give an indication of the number of subscribers over the interval.

For example, where Pub/Sub is configured with 2 subscribers, the put count would equal 1 and the publish count would equal 2 - and a total of 2 messages would be put to queues.

In a streaming queue environment where the `STREAMQ` references a topic alias, the published count reports the number of published messages and does not include the original put. In this environment where there are 2 subscribers, the put count would equal 1 and the publish count would equal 2, but in this instance **3** messages would have been put to queues.

Appendix A

Regression

When performance monitoring both IBM MQ Advanced for z/OS VUE 9.3 and IBM MQ for z/OS 9.3, 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)
  - Streaming messages
- Moving messages using cluster channels
- Client
- Bridges and adaptors
- Trace
- AMS message protection

These tests are run against version V9.1 (V910), V9.2 (V920) and V9.3 (V930) 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 z15 (8561-7J1) 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 between runs.

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

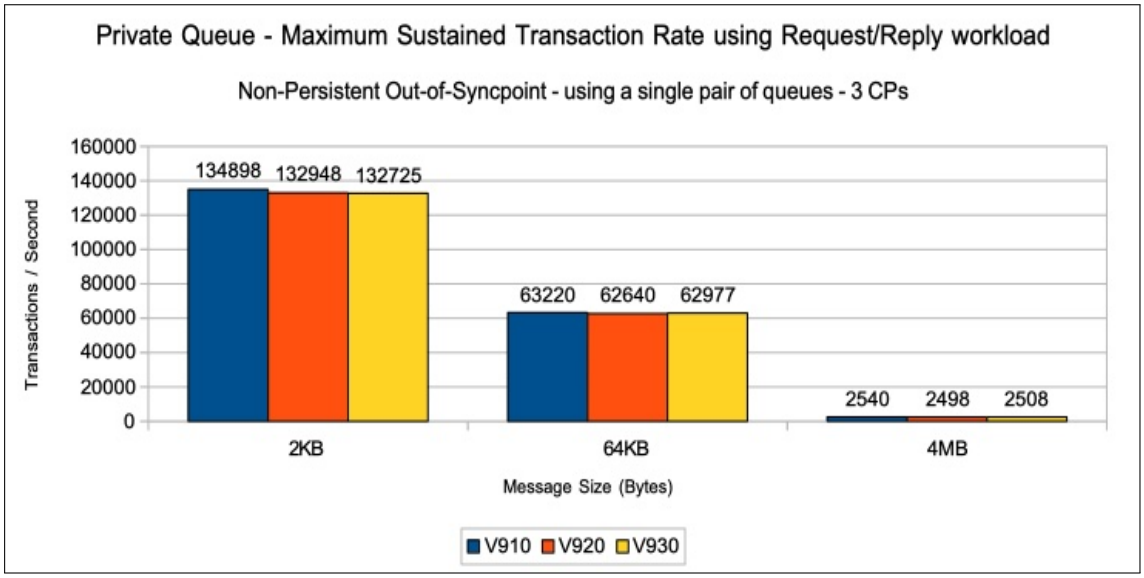
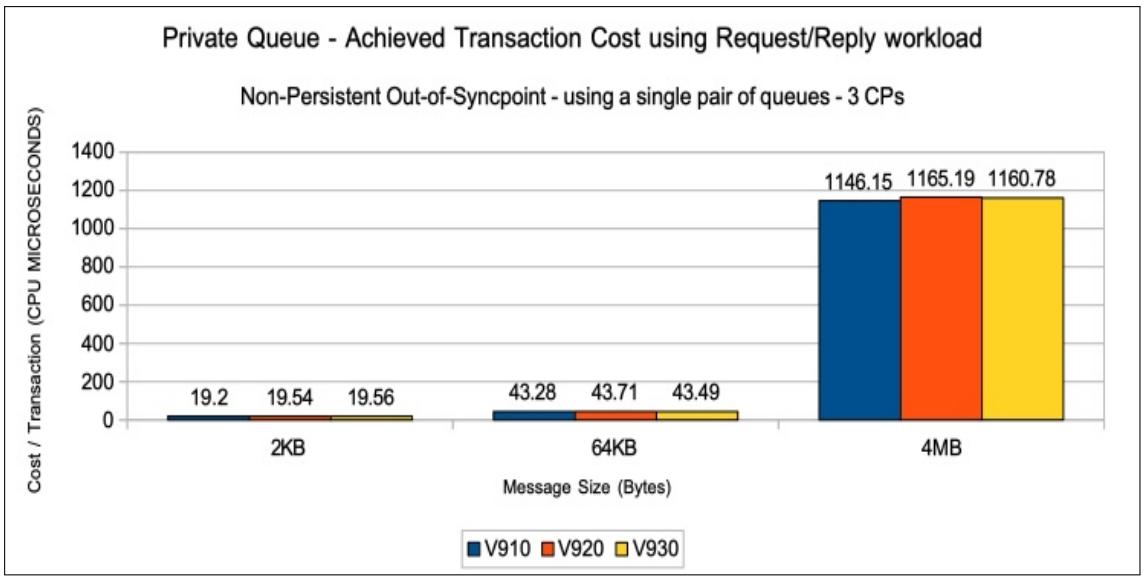


Chart: Transaction cost for non-persistent out-of-syncpoint workload



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 z15 (8561) used for testing.

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

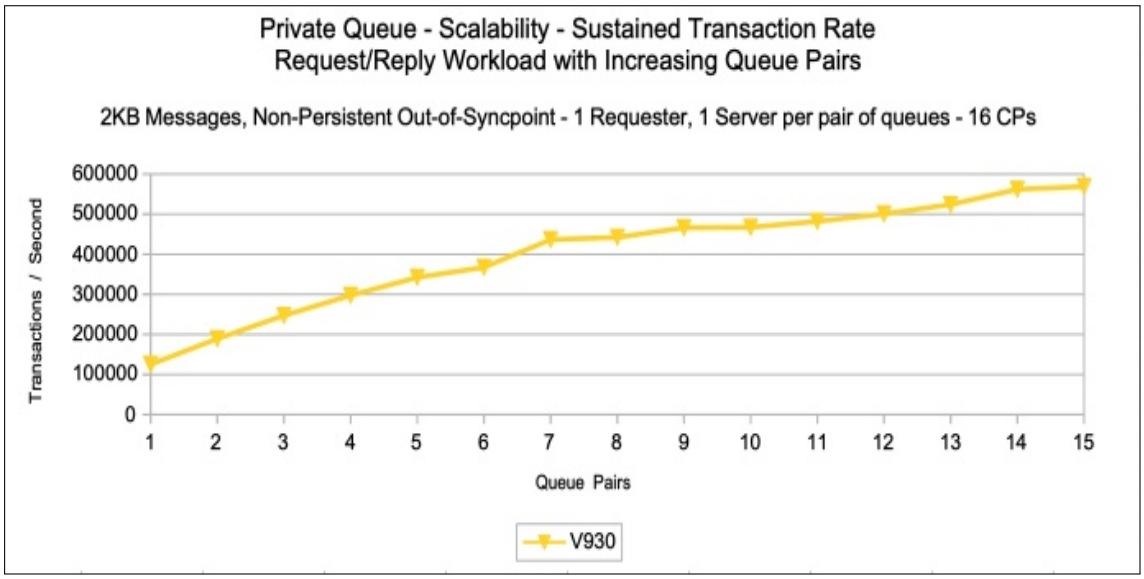
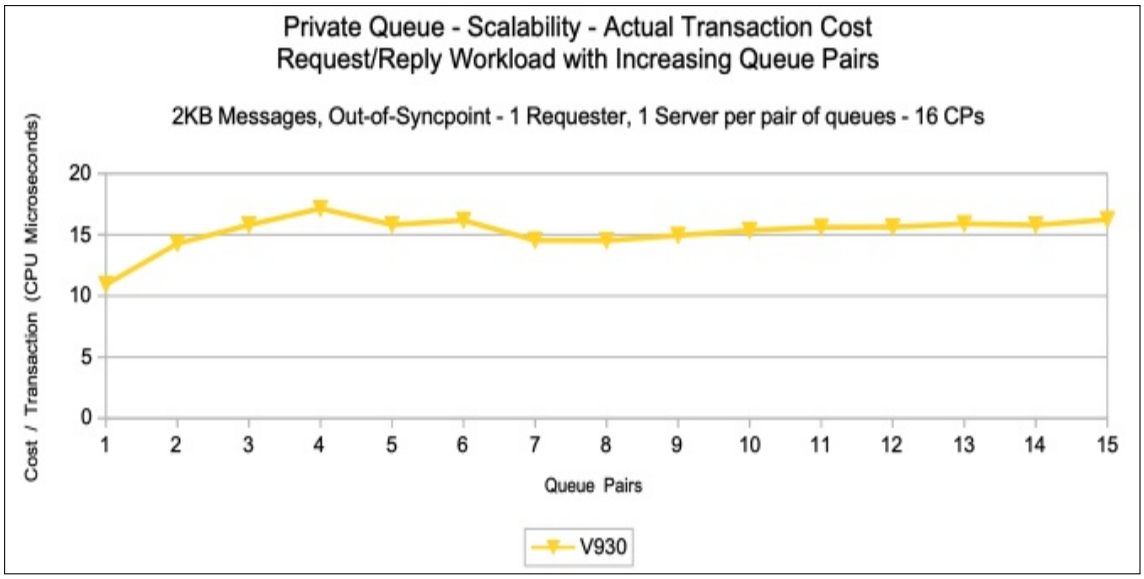


Chart: Transaction cost for non-persistent out-of-syncpoint scalability workload



The preceding 2 charts show that a single queue manager is able to drive in excess of 575,000

transactions per second - or 1,150,000 non-persistent messages per second on a 16-way LPAR.

Note: Since the performance for 9.1, 9.2 and 9.3 is comparable, the charts show only 9.3 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

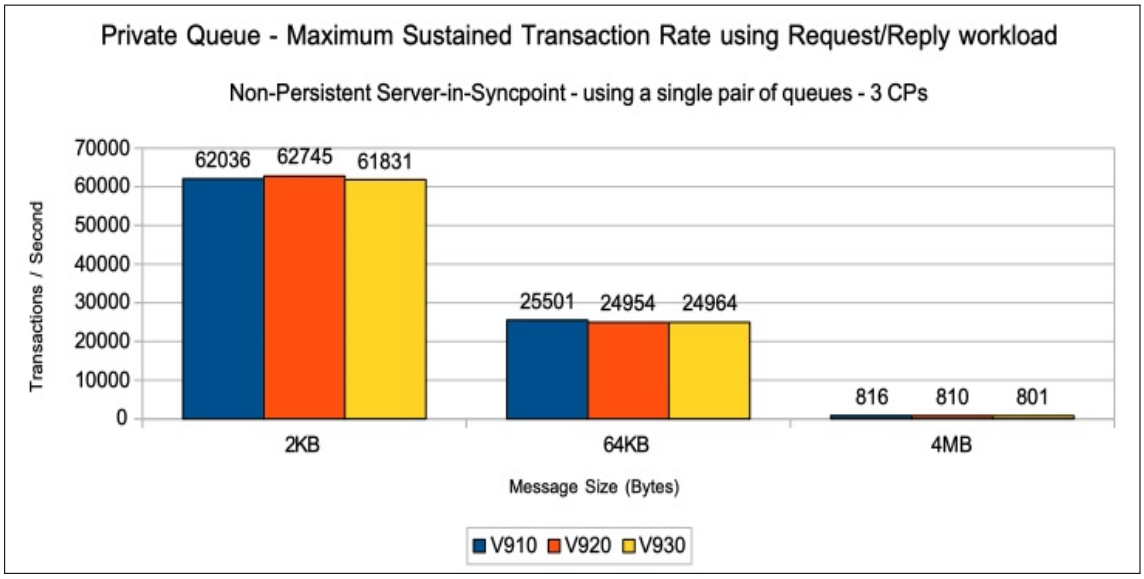
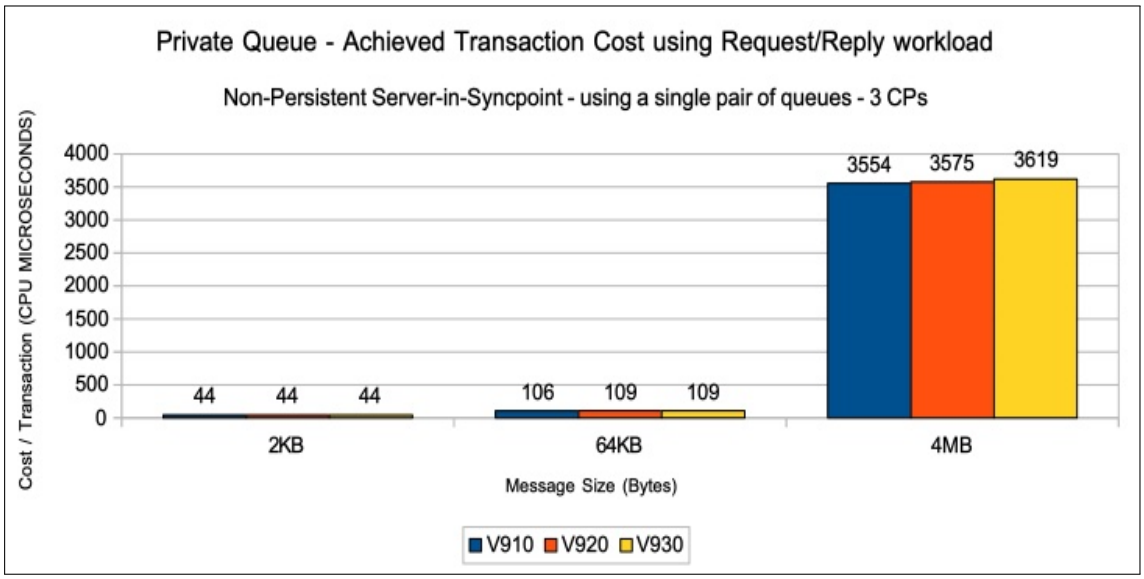


Chart: Transaction cost for non-persistent in syncpoint workload



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 z15 (8561) 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 9.1, 9.2 and 9.3 is comparable, the charts show only 9.3 data for the purpose of clarity.

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

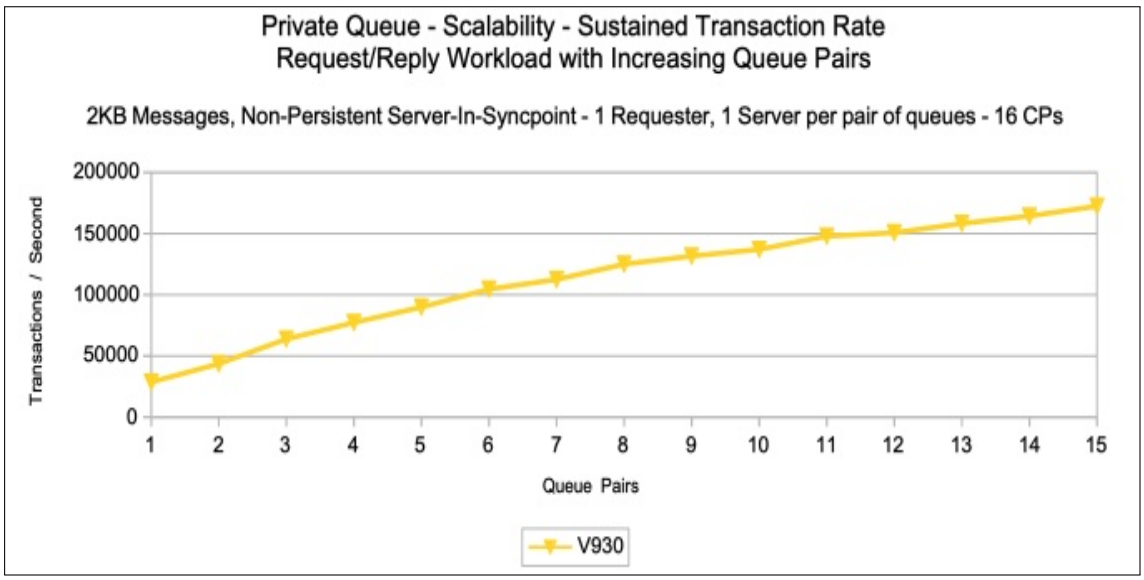


Chart: Transaction cost for non-persistent in syncpoint scalability workload with 2KB messages

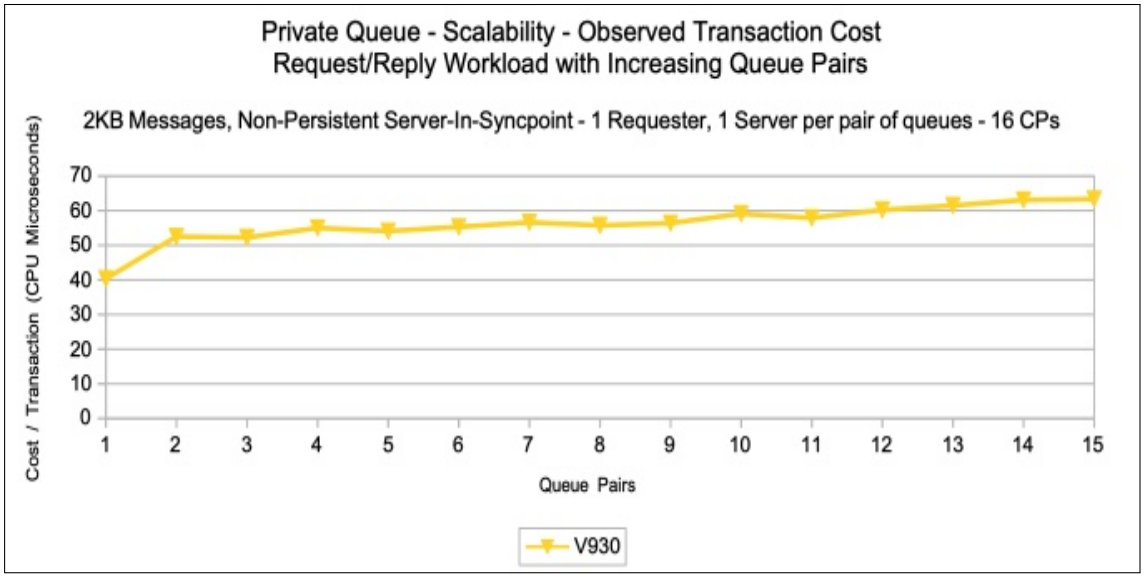


Chart: Transaction rate for non-persistent in syncpoint scalability workload with 64KB messages

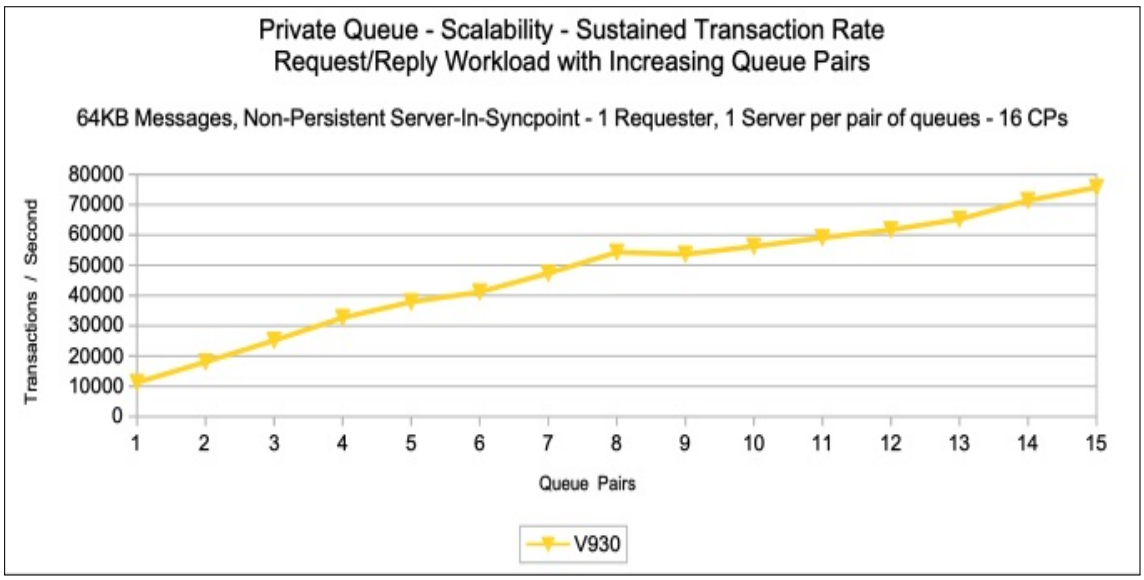


Chart: Transaction cost for non-persistent in syncpoint scalability workload with 64KB messages

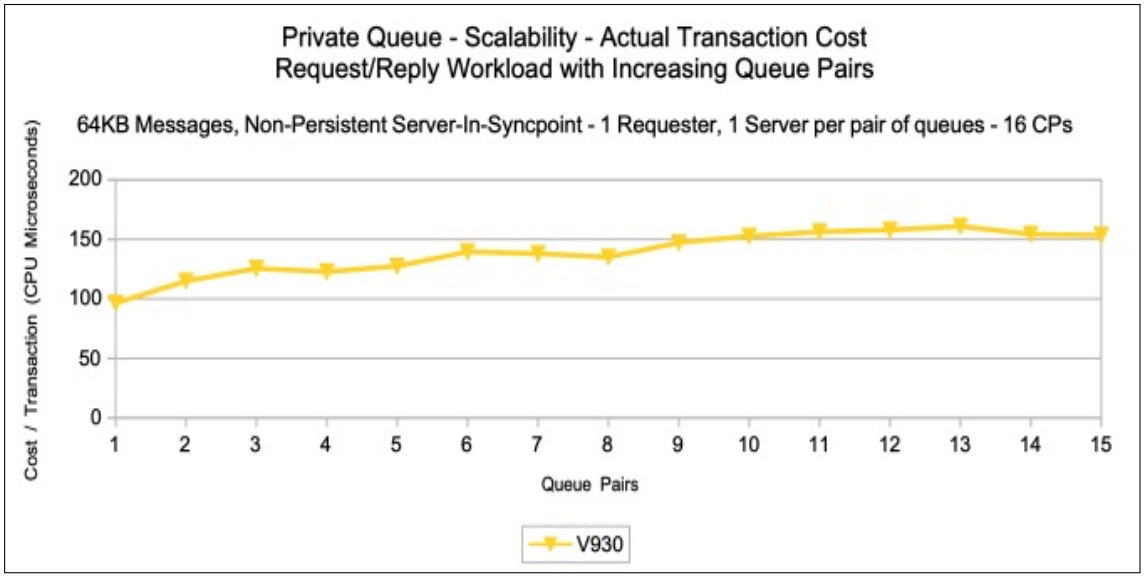


Chart: Transaction rate for non-persistent in syncpoint scalability workload with 4MB messages

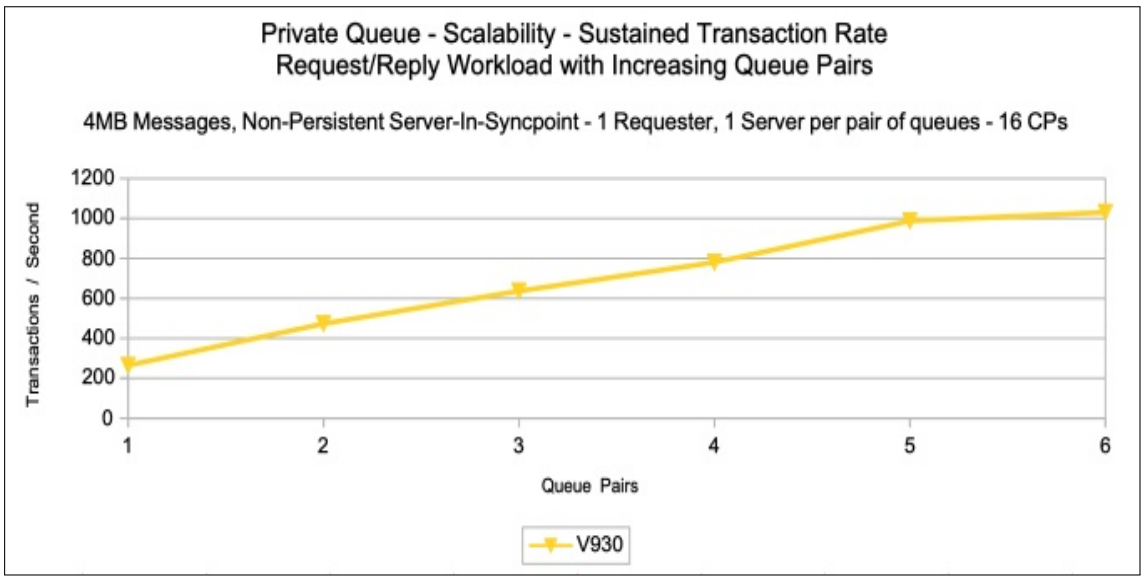
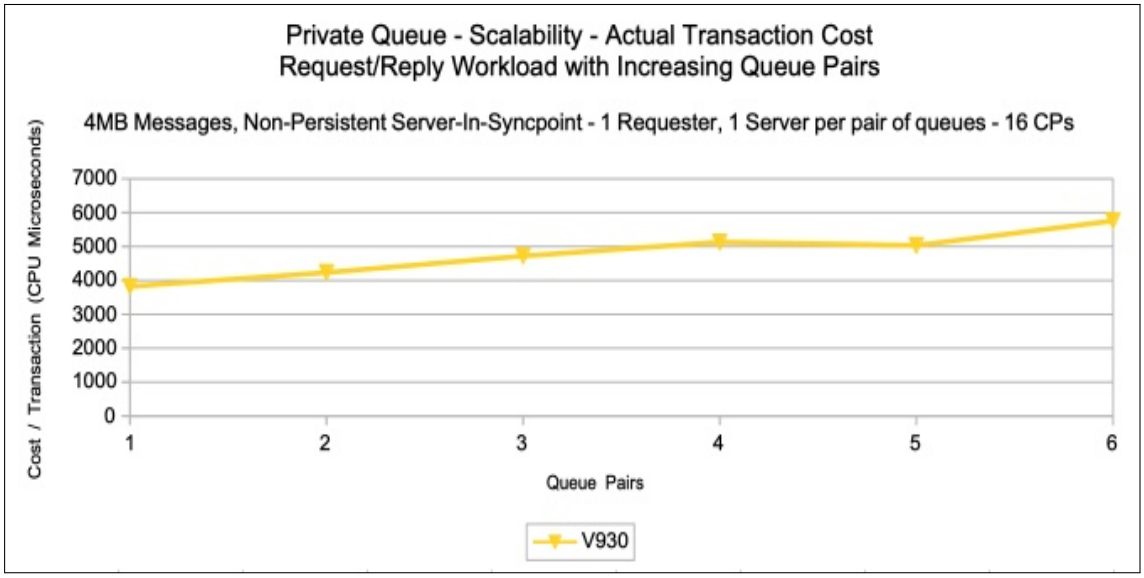


Chart: Transaction cost for non-persistent in syncpoint scalability workload with 4MB messages



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

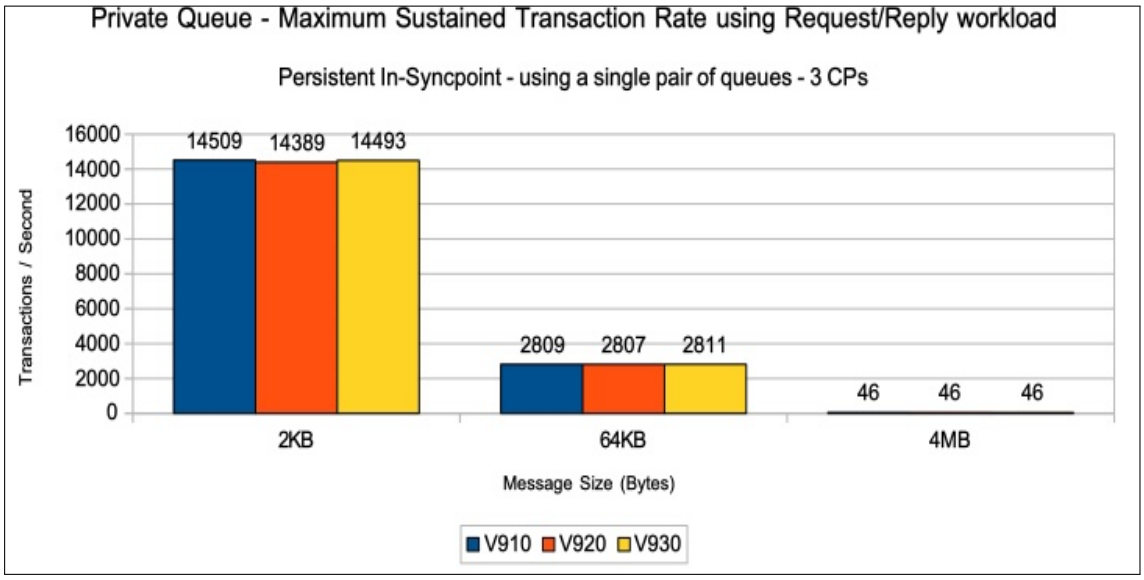
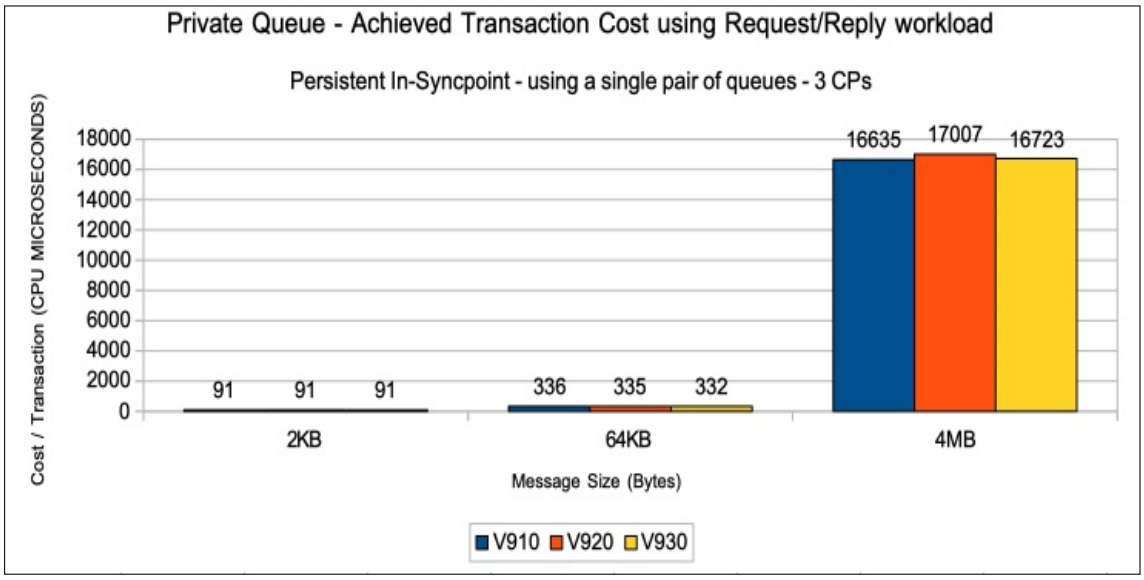


Chart: Transaction cost for persistent in syncpoint workload

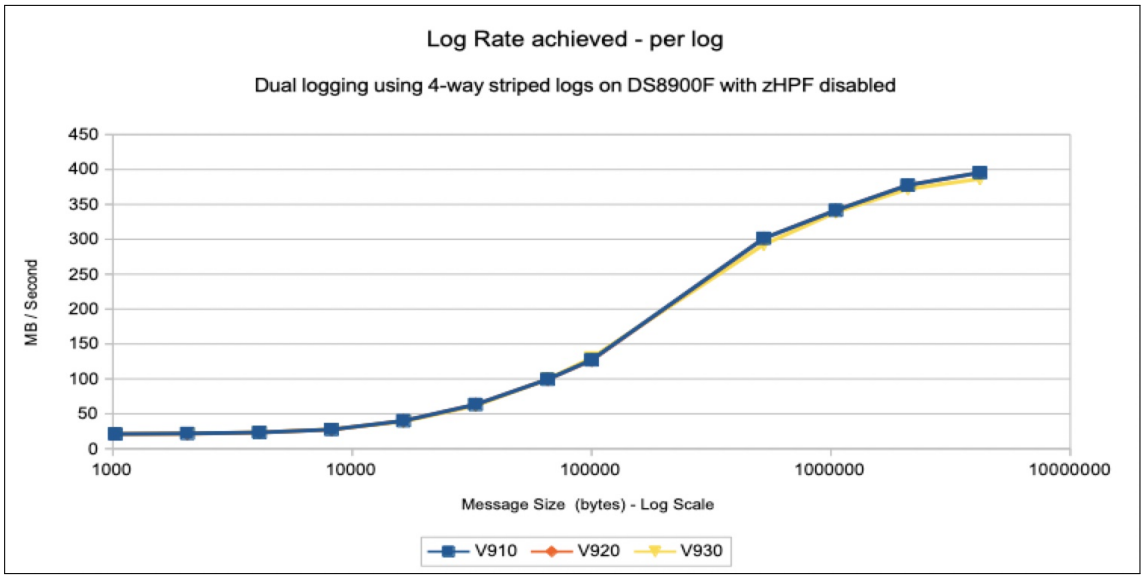


Upper bounds of persistent logging rate

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 z15.

Chart: Peak log rate achieved during persistent message workload



Notes:

The peak log rate in excess of 380MB per second is for each copy of the dual logs, i.e. there is more than 760 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.

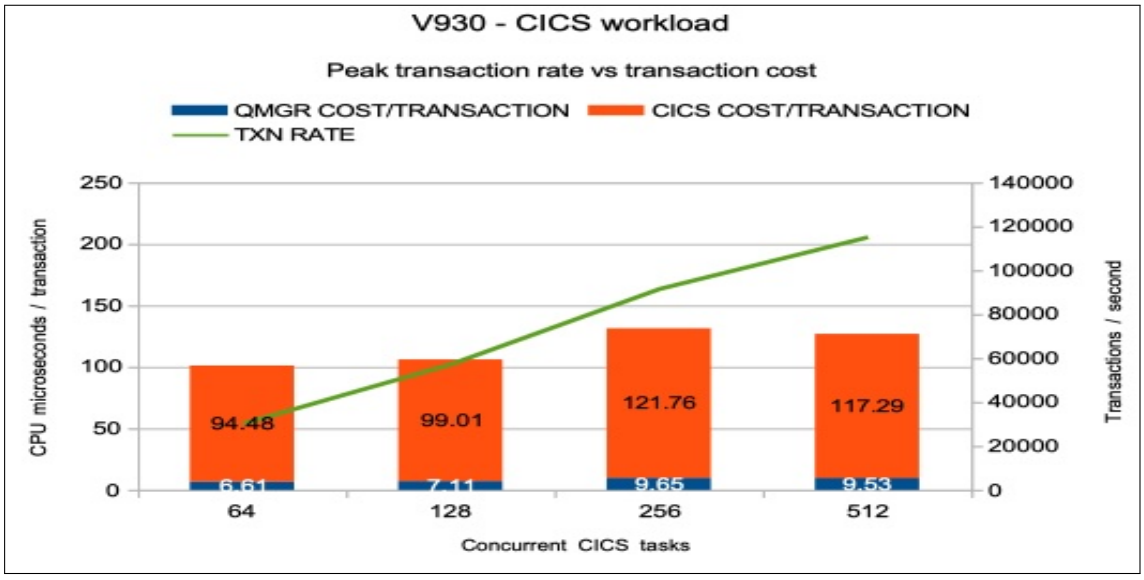
The following chart uses a set of simple CICS transactions that open queues, put a 2KB non-persistent message, get a 2KB non-persistent message, close queues and initiate a new CICS transaction. Running multiple sets of these transactions simulates a number of concurrent transactions across multiple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

- As the 9.1, 9.2 and 9.3 data is similar, only 9.3 data is shown.

Chart: CICS workload with MQ 9.3



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"[1].

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.

[1] Link will download the performance report as it is no longer available on the IBM SupportPacs site.

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

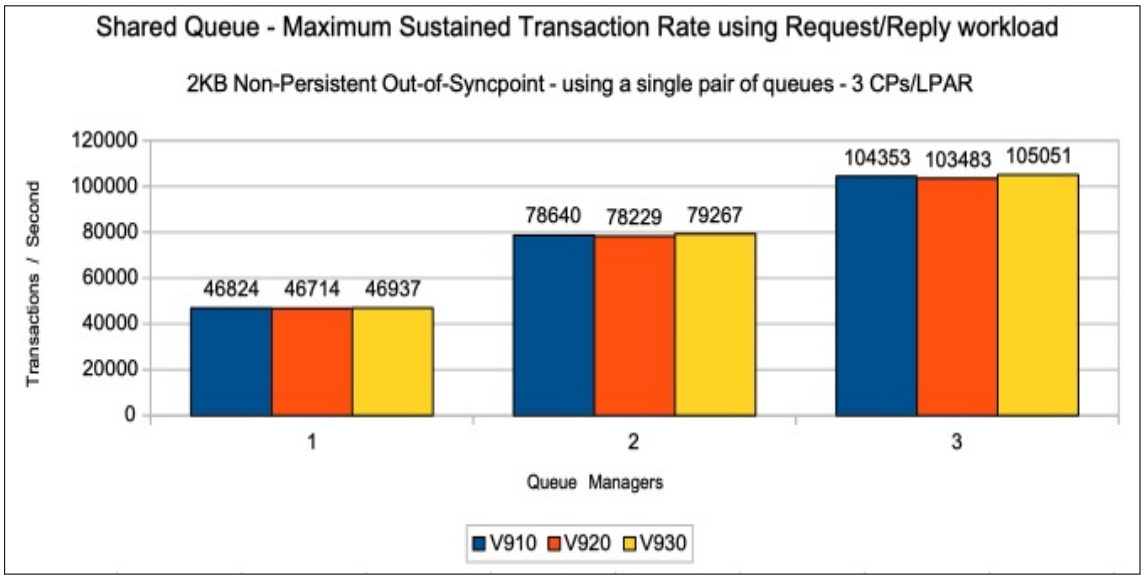


Chart: Transaction cost for non-persistent out-of-syncpoint workload - 2KB messages.

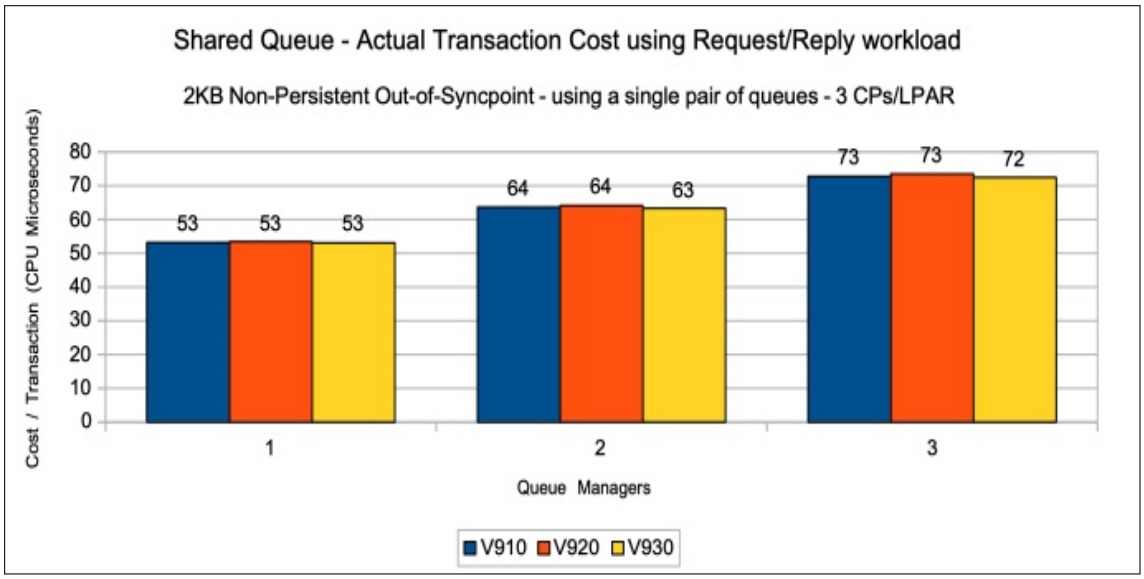


Chart: Transaction rate for non-persistent out-of-syncpoint workload - 64KB messages.

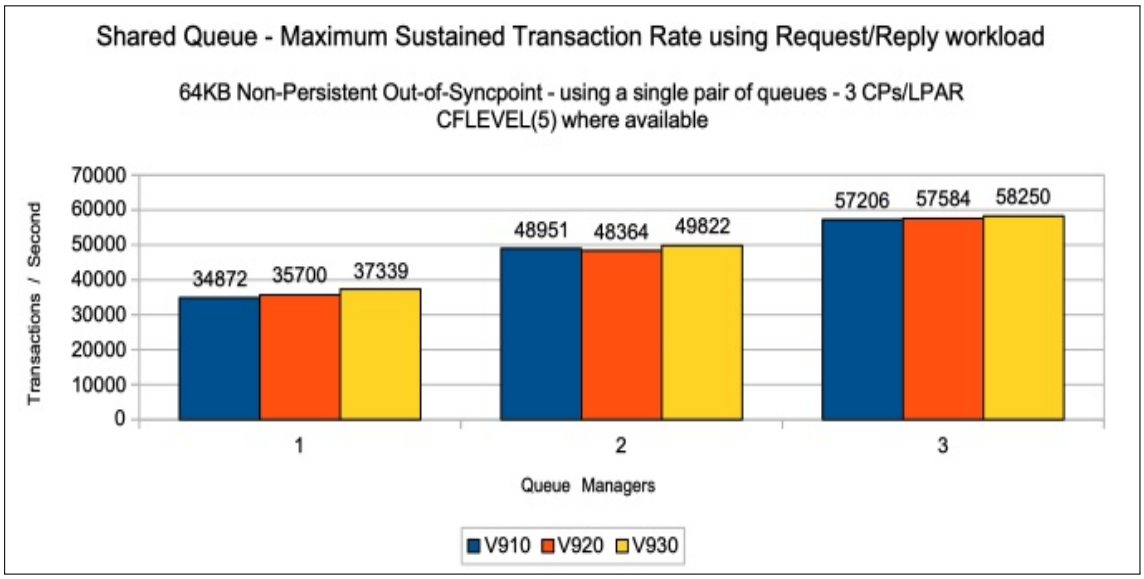


Chart: Transaction cost for non-persistent out-of-syncpoint workload - 64KB messages.

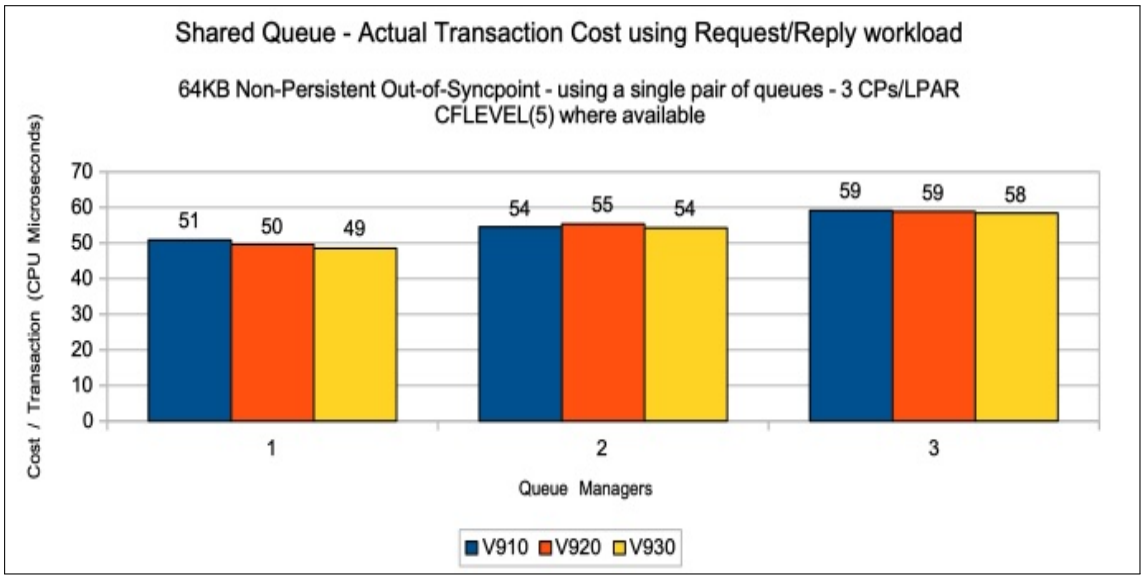


Chart: Transaction rate for non-persistent out-of-syncpoint workload - 4MB messages.

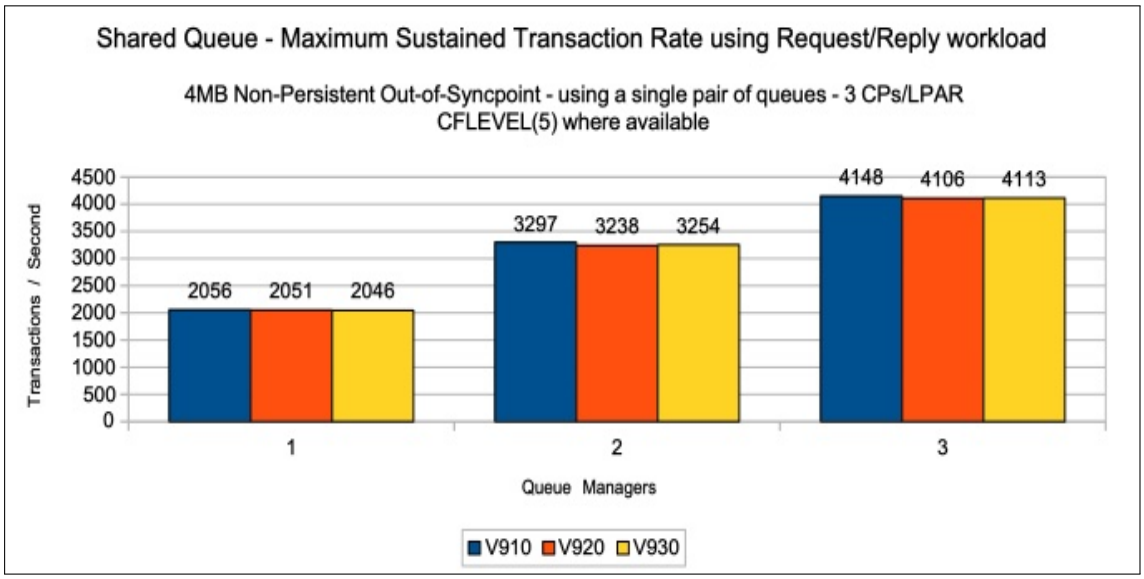
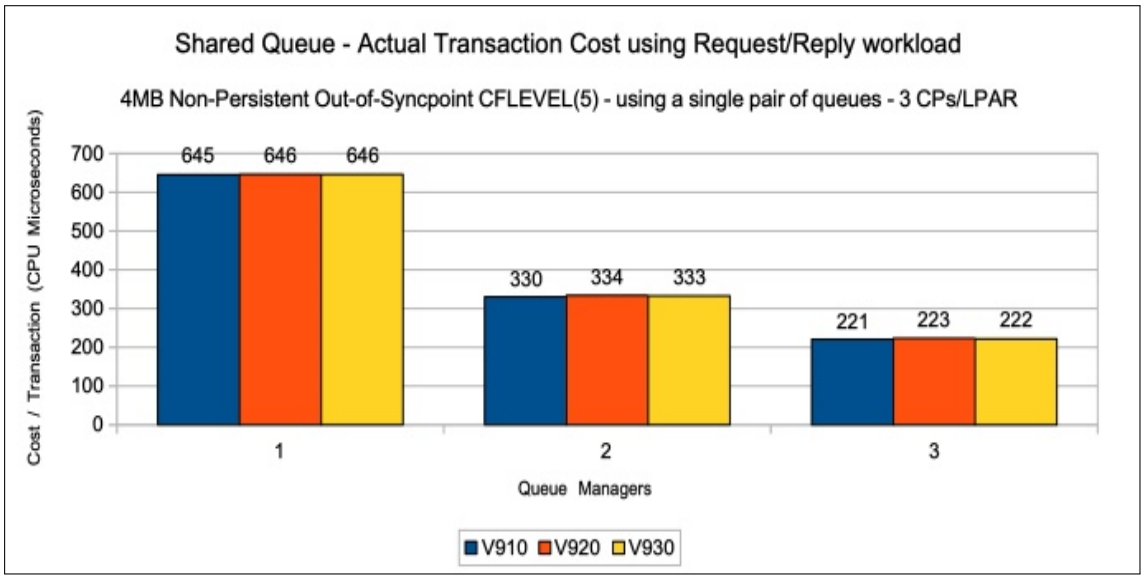


Chart: Transaction cost for non-persistent out-of-syncpoint workload - 4MB messages.



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

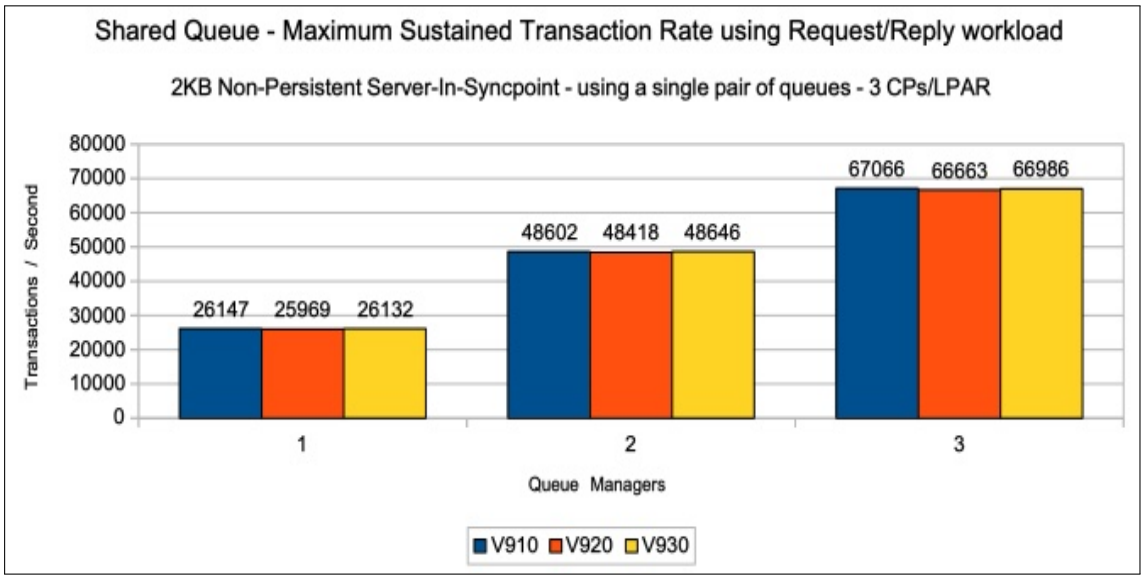


Chart: Transaction cost for non-persistent in syncpoint workload - 2KB

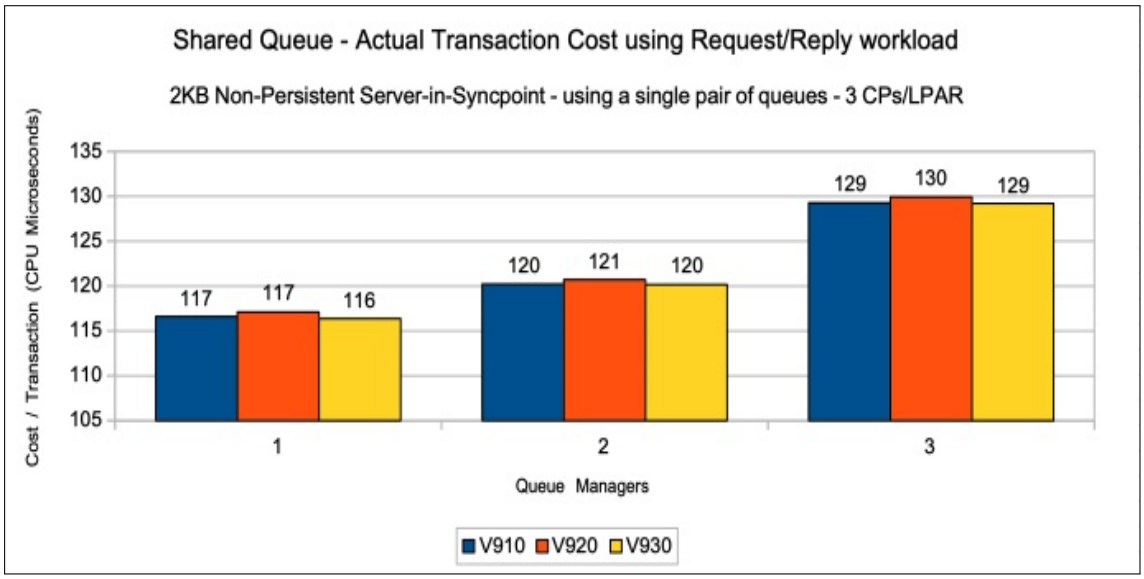


Chart: Transaction rate for non-persistent in syncpoint workload - 64KB

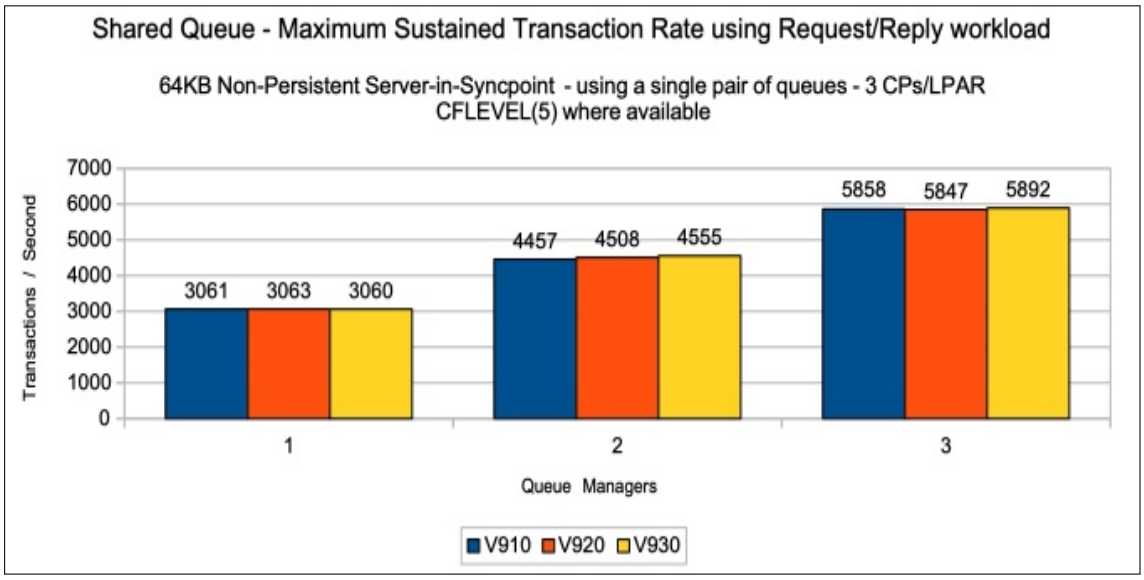


Chart: Transaction cost for non-persistent in syncpoint workload - 64KB

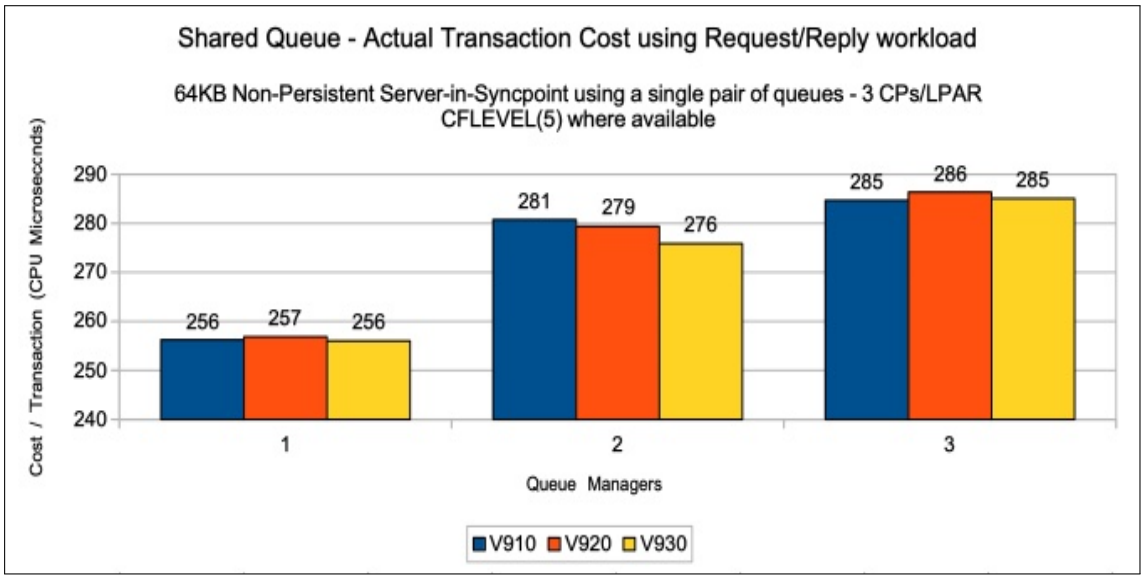


Chart: Transaction rate for non-persistent in syncpoint workload - 4MB

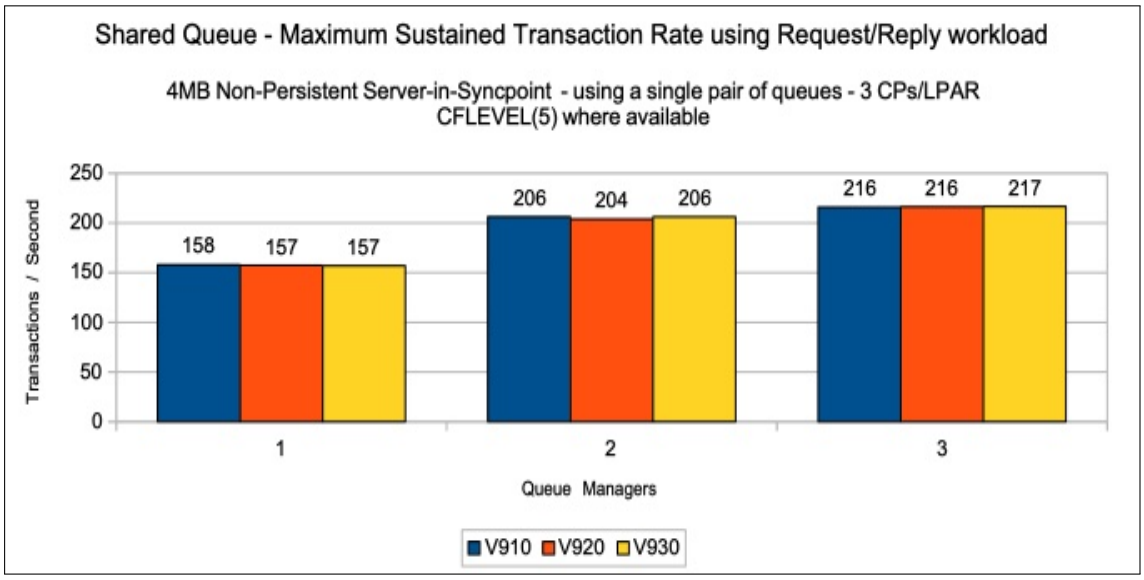
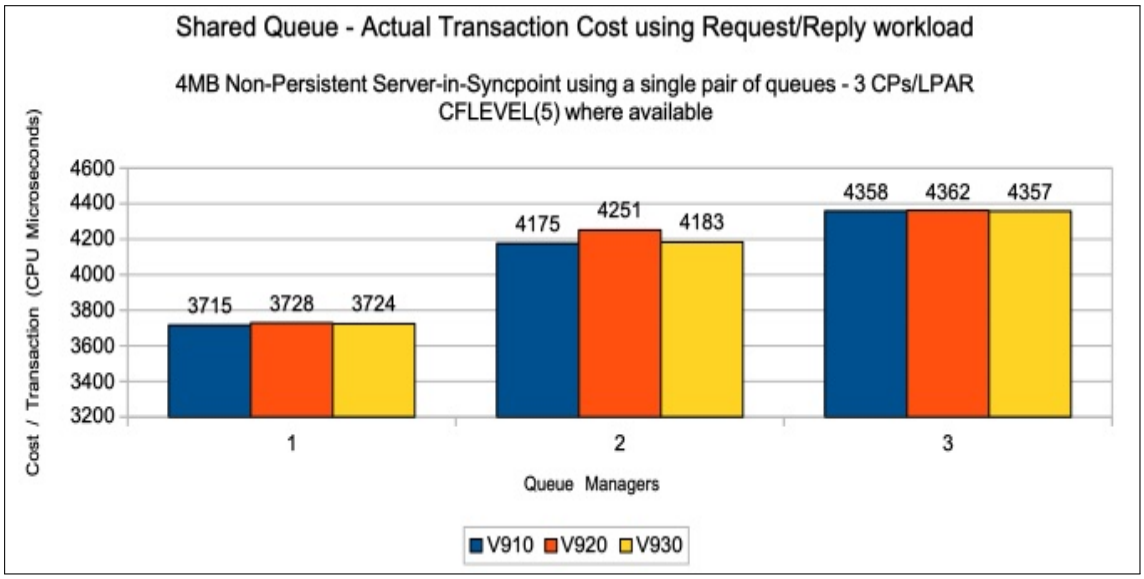


Chart: Transaction cost for non-persistent in syncpoint workload - 4MB



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

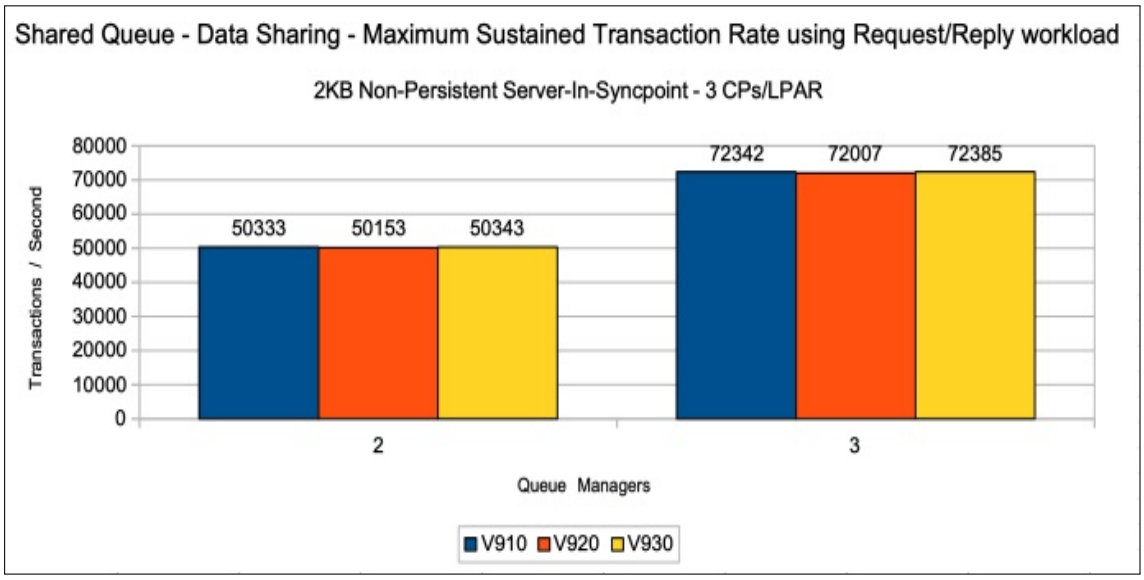


Chart: Transaction cost for data sharing non-persistent in syncpoint workload - 2KB

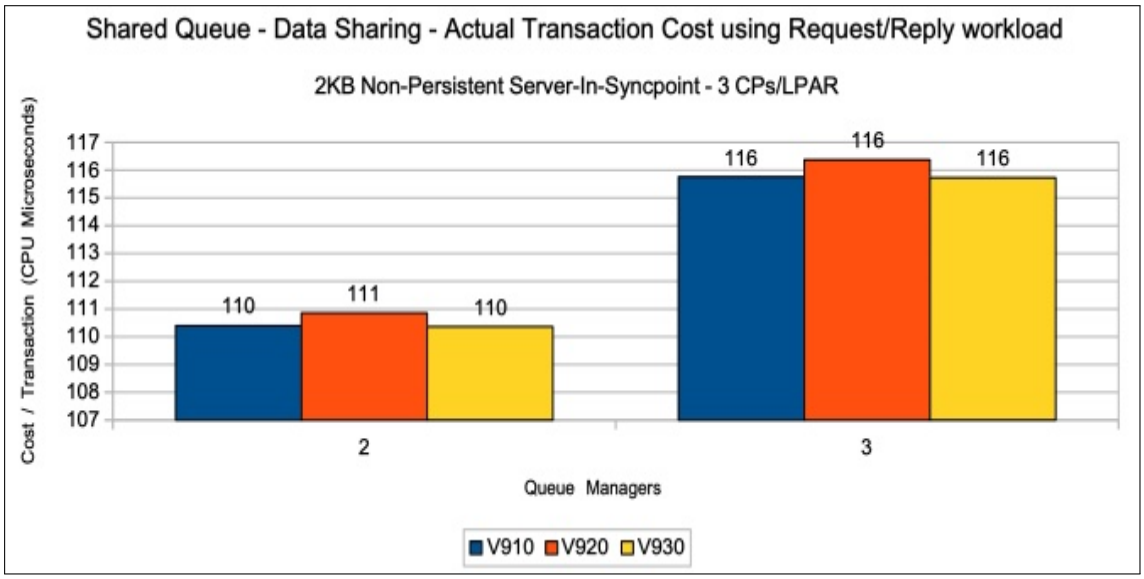


Chart: Transaction rate for data sharing non-persistent in syncpoint workload - 64KB

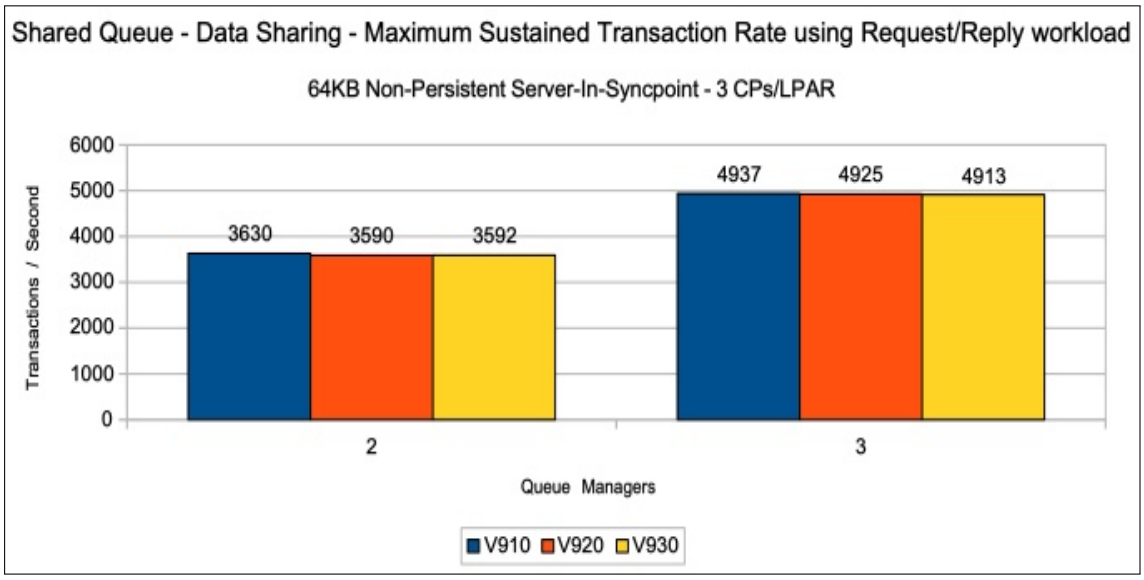
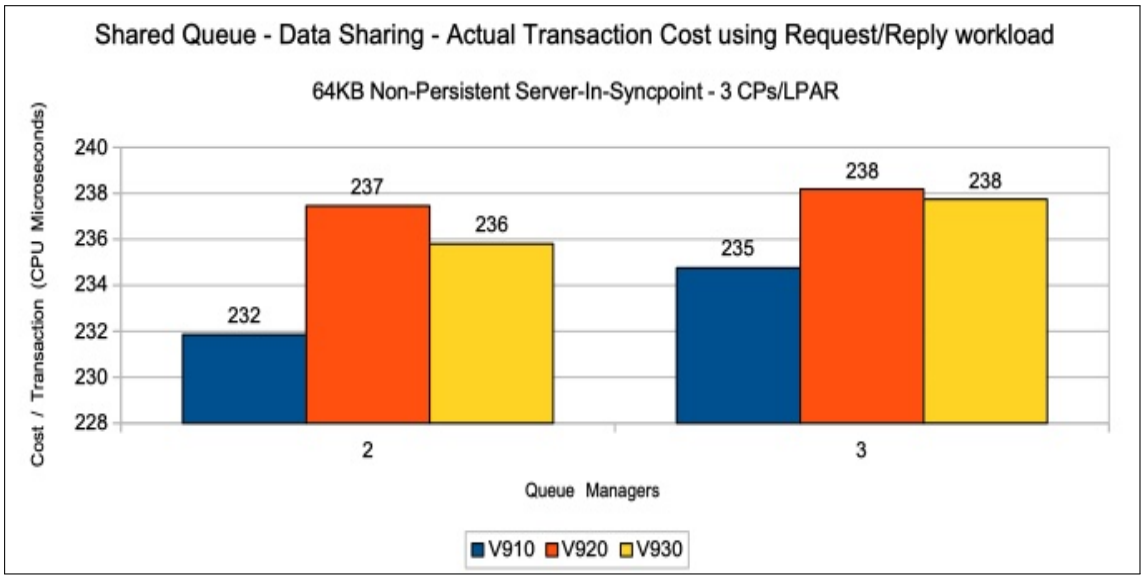


Chart: Transaction cost for data sharing non-persistent in syncpoint workload - 64KB



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 in-syncpoint, so that they are not constrained by logging etc.

Within the channel tests there are measurements with small numbers of channels (1 to 5 inbound plus the same number outbound), which was suitable for driving the LPARs to capacity.

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

Initially measurements are run both with and without the TLS/SSL encryption enabled.

The cipher spec "ECDHE_RSA_AES_256_CBC_SHA384" was used for all TLS/SSL tests.

The SSLRKEYC attribute was set as follows:

- 0 such that the secret key is only negotiated as channel start. This allows us to determine the impact of encryption on the data.
- 1MB of data can flow across the channel before renegotiating the keys.

Channel compression

Since IBM MQ version 8.0.0, the ZLIBFAST compression option is 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".

Additionally, the white paper "MQ for z/OS - Channel Compression" discusses the performance of channel compression on z/OS including where compression may help improve the performance, either by reducing MQ costs or improving channel throughput.

Measurements:

- The measurements using 1 to 5 channels use batch applications to drive the workload at each end of the channel.
- The compression tests use 32KB message of varying compressibility, so there is something to compress, e.g. a message of 32KB that is 80% compressible would reduce to approximately 6.5KB. These tests are run using 1 outbound and 1 inbound channel so include the compression and inflation costs for the request and reply message.

Streaming messages across channels

Many of the performance tests in the regression section use a request/reply model to demonstrate the performance of the MQ channels.

One common use of MQ is to provide the transport mechanism when data is moved between data centres, such as in an IBM InfoSphere Data Replication queue replication (QREP) scenario. This model sends data in a single direction and can show different characteristics to a typical request/reply model. For example the queue depths may naturally vary significantly, and in some cases may drive MQ page set I/O.

Typically a QREP-scenario would use persistent messages and may have different message sizes for online and batch processing. In this section we use 10KB messages for online processing and 1MB messages for batch processing.

In the streaming configuration, the transmit queue is pre-loaded with messages to ensure the channel is able to send full batches (of 200 messages), thus driving the channel to capacity.

Non-persistent in-syncpoint - 1 to 5 sender receiver channels

Chart: Transaction rate with 32KB messages

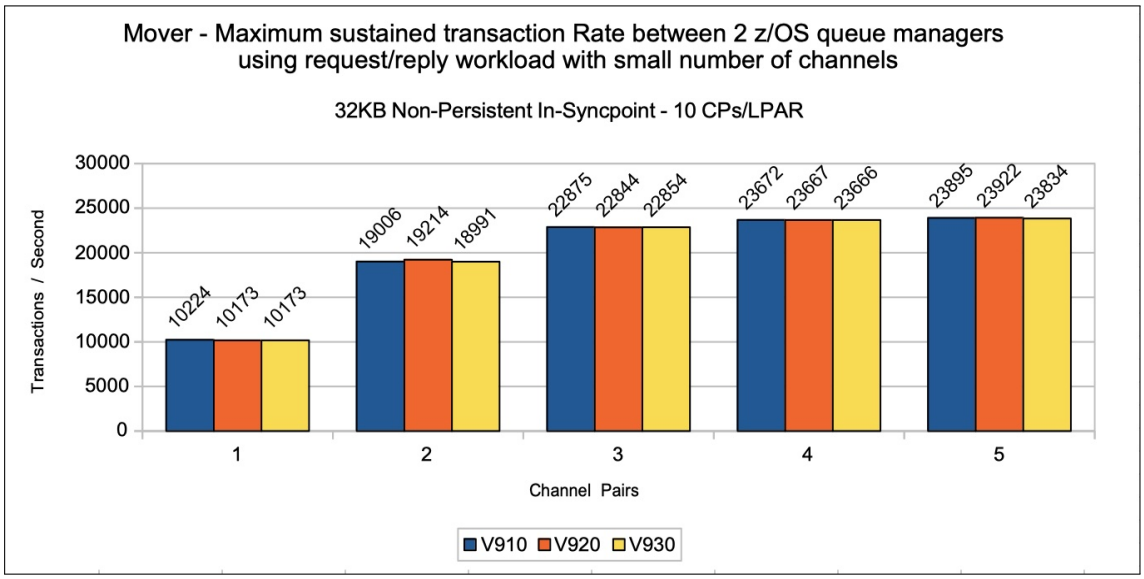


Chart: Transaction cost for 32KB messages

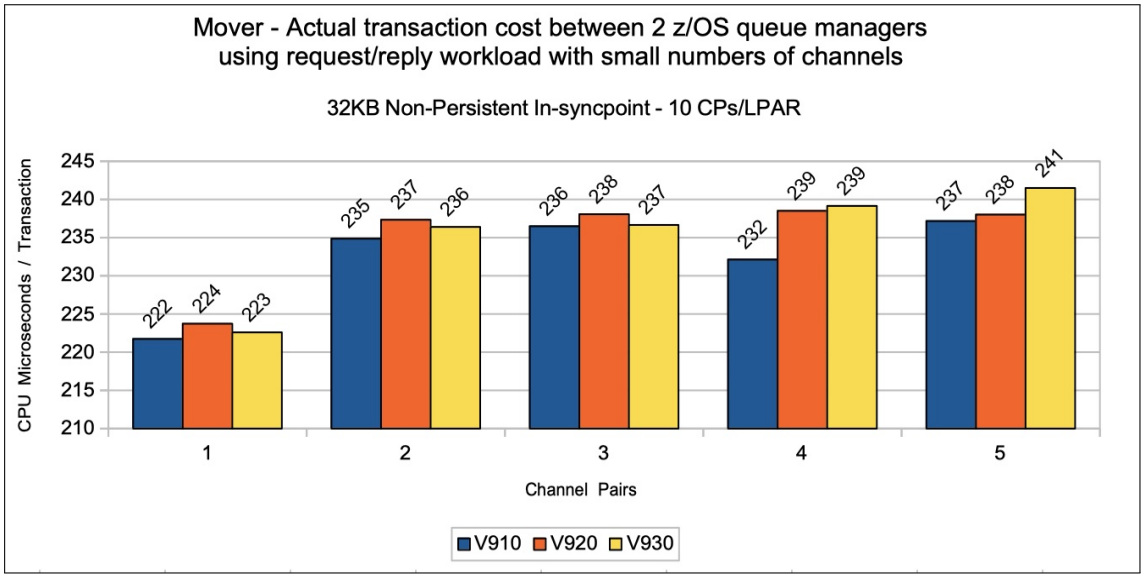


Chart: Transaction rate with 32KB messages over SSL channels with no secret key negotiation

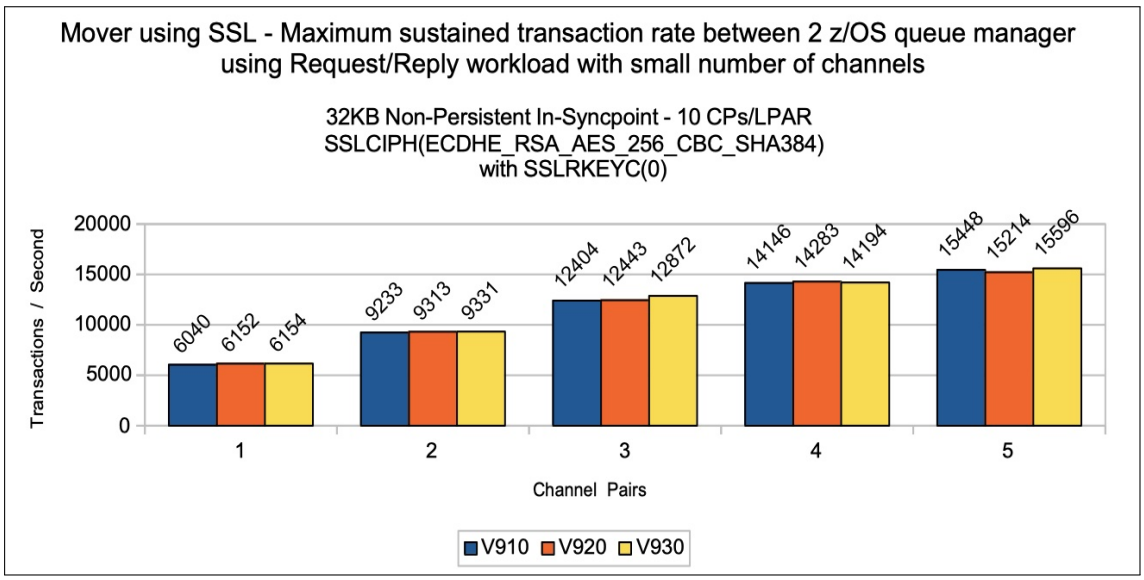


Chart: Transaction cost for 32KB messages over SSL channels with no secret key negotiation

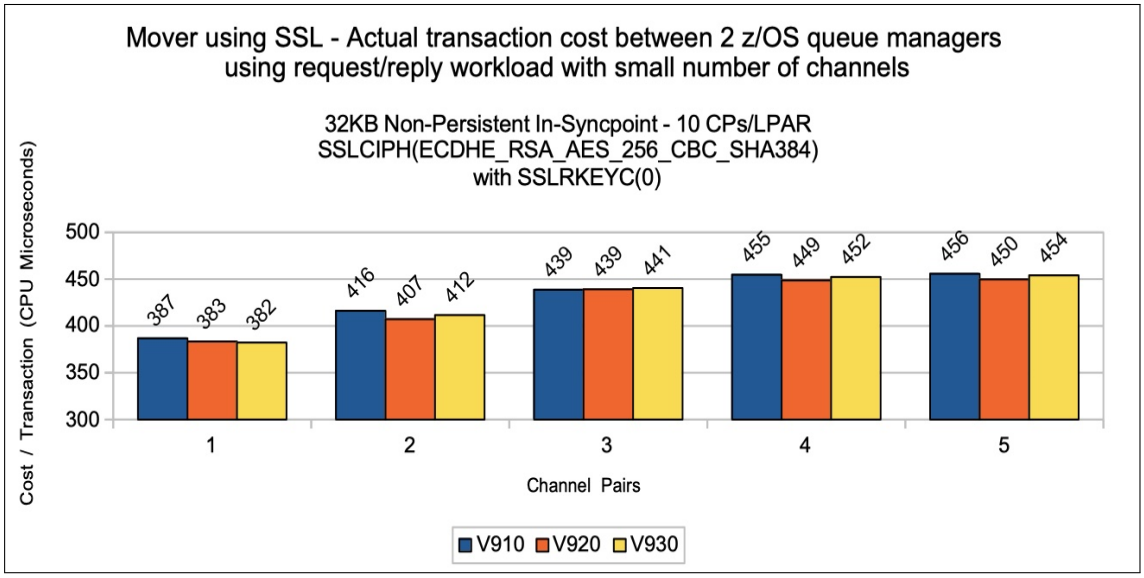


Chart: Transaction rate with 32KB messages over SSL channels with secret key negotiation every 1MB

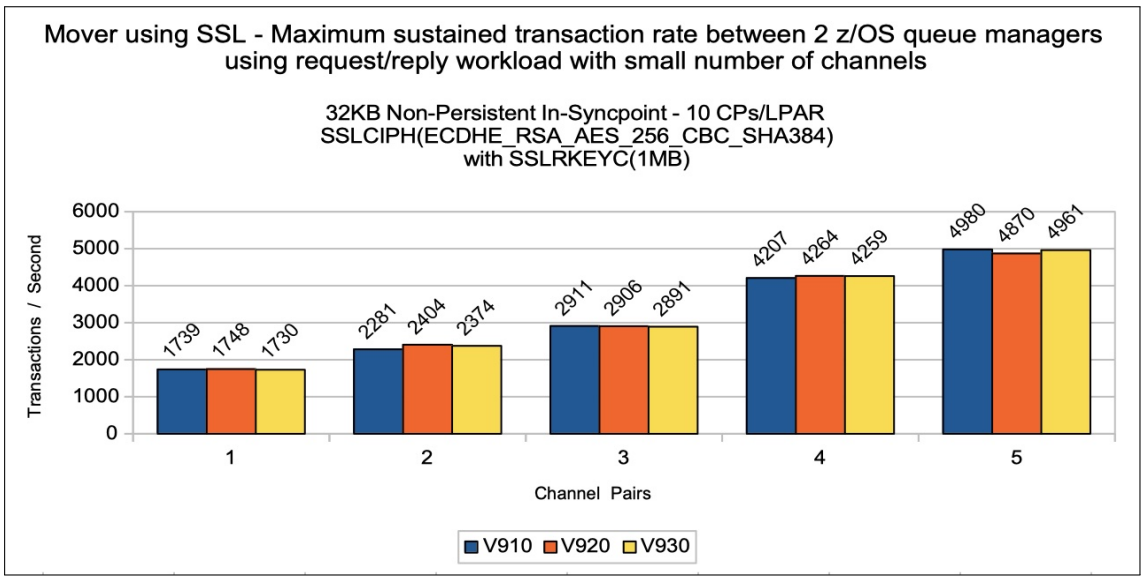
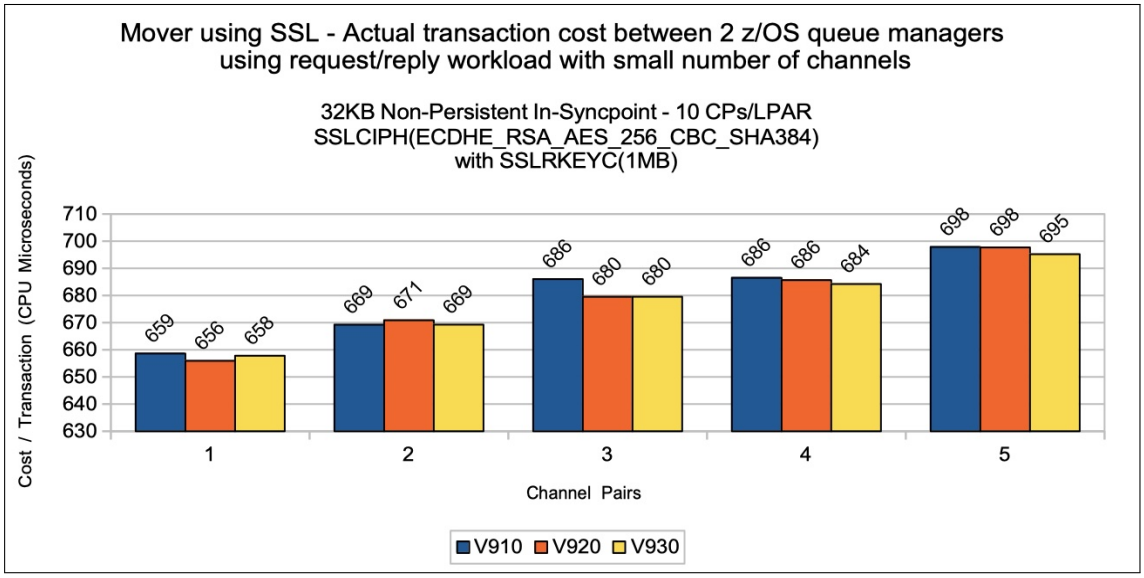


Chart: Transaction cost for 32KB messages over SSL channels with secret key negotiation every 1MB



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 32KB messages

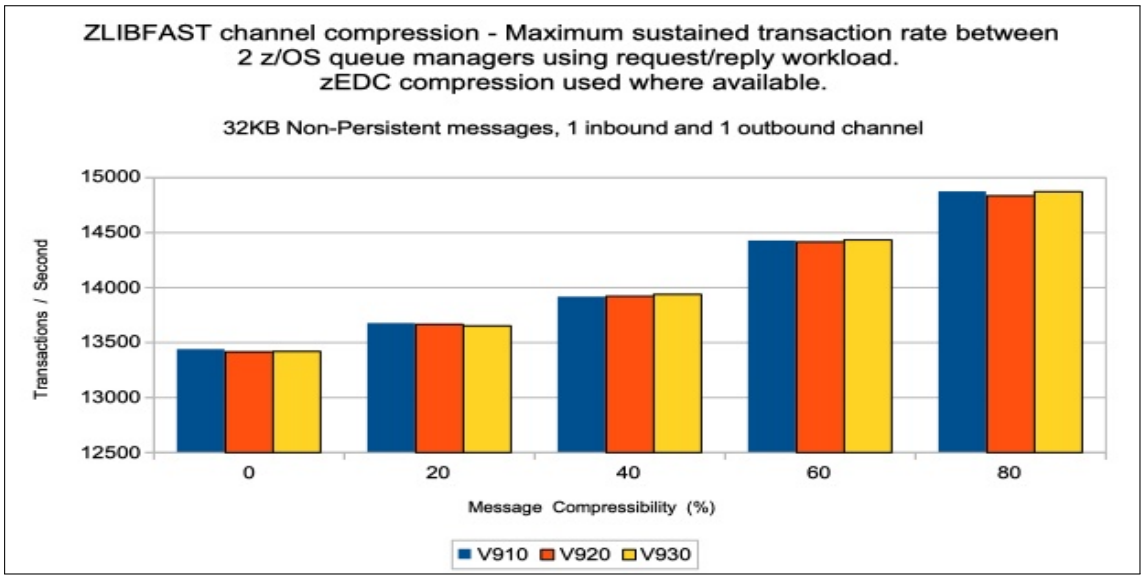
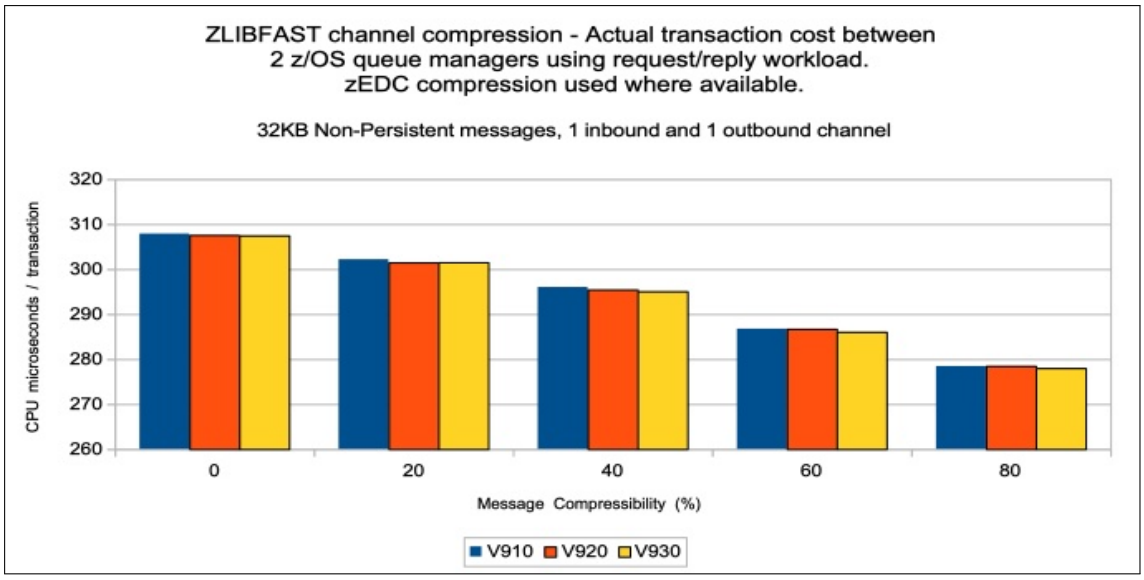


Chart: Transaction cost for ZLIBFAST on 32KB messages



Channel compression using ZLIBHIGH

Chart: Transaction rate with 32KB messages with ZLIBHIGH

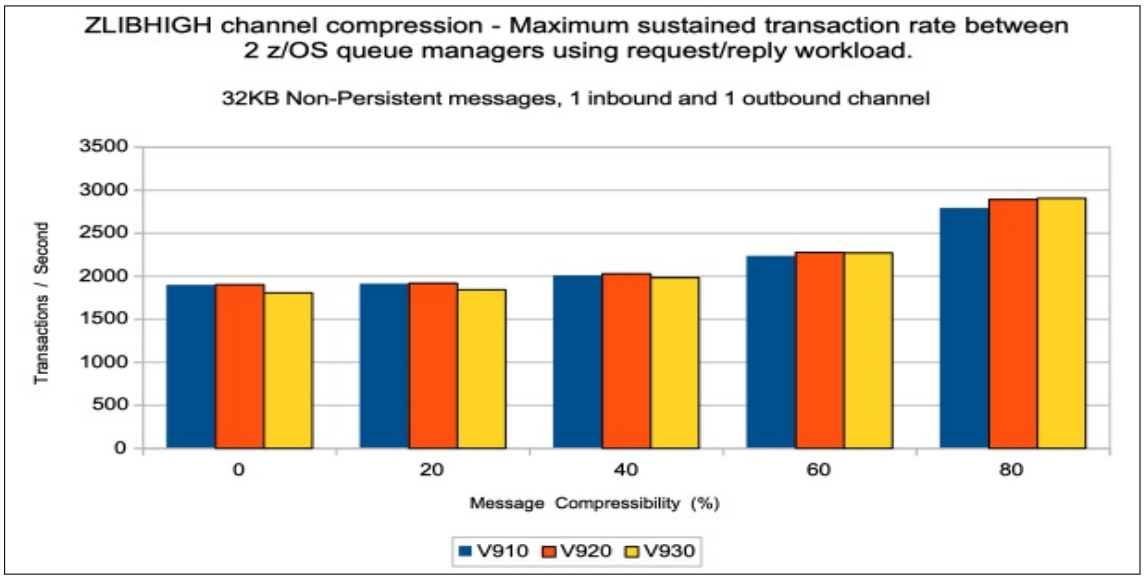
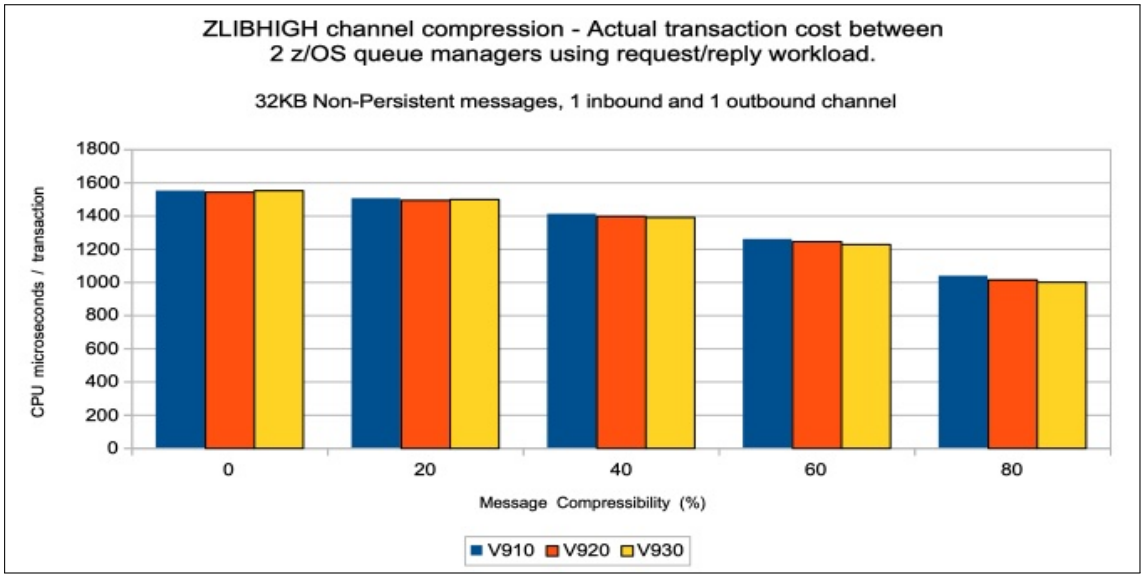


Chart: Transaction cost for 32KB messages with ZLIBHIGH



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 32KB messages with ZLIBFAST on SSL channels

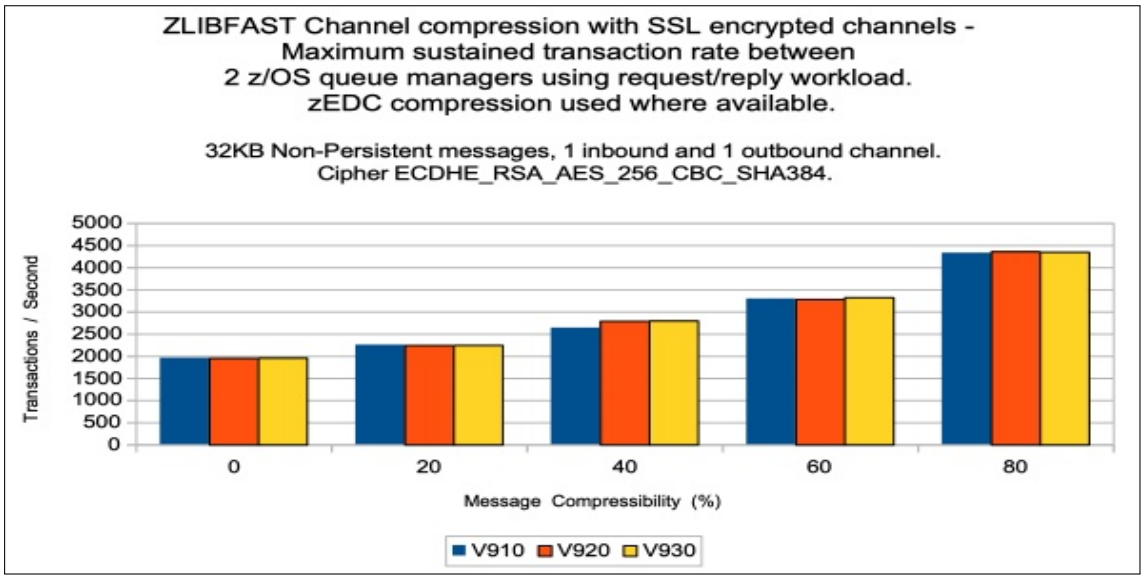
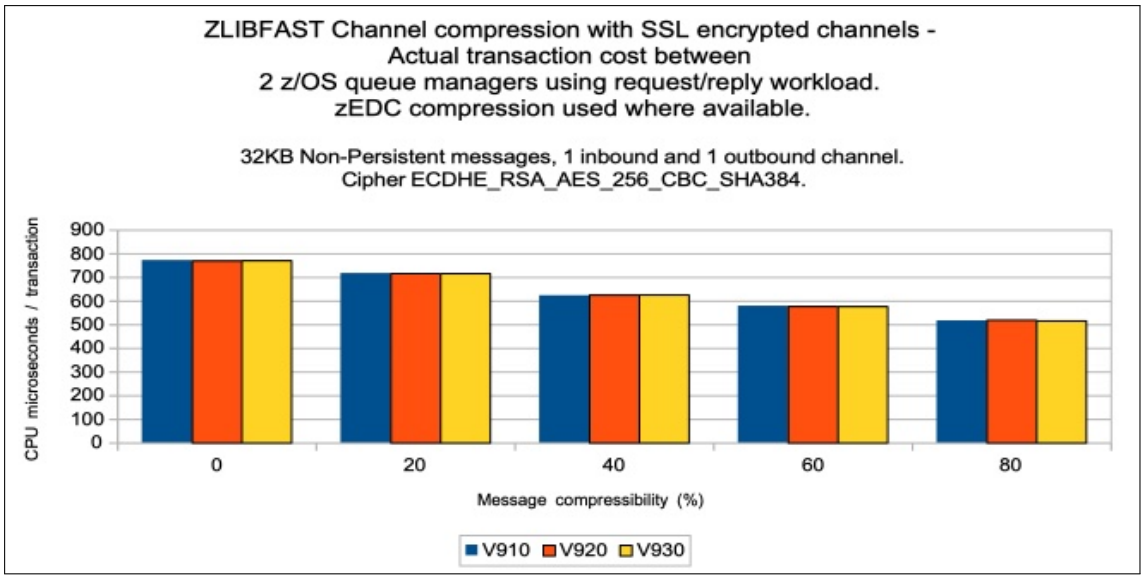


Chart: Transaction cost with 32KB messages with ZLIBFAST on SSL channels



Channel compression using ZLIBHIGH on SSL channels

Chart: Transaction rate with 32KB messages with ZLIBHIGH on SSL channels

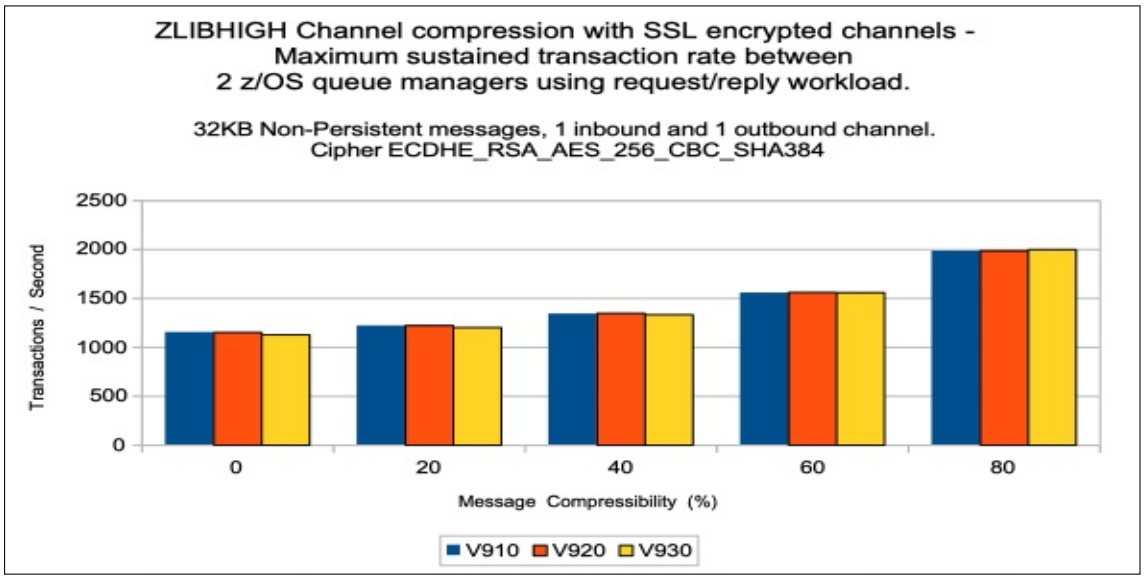
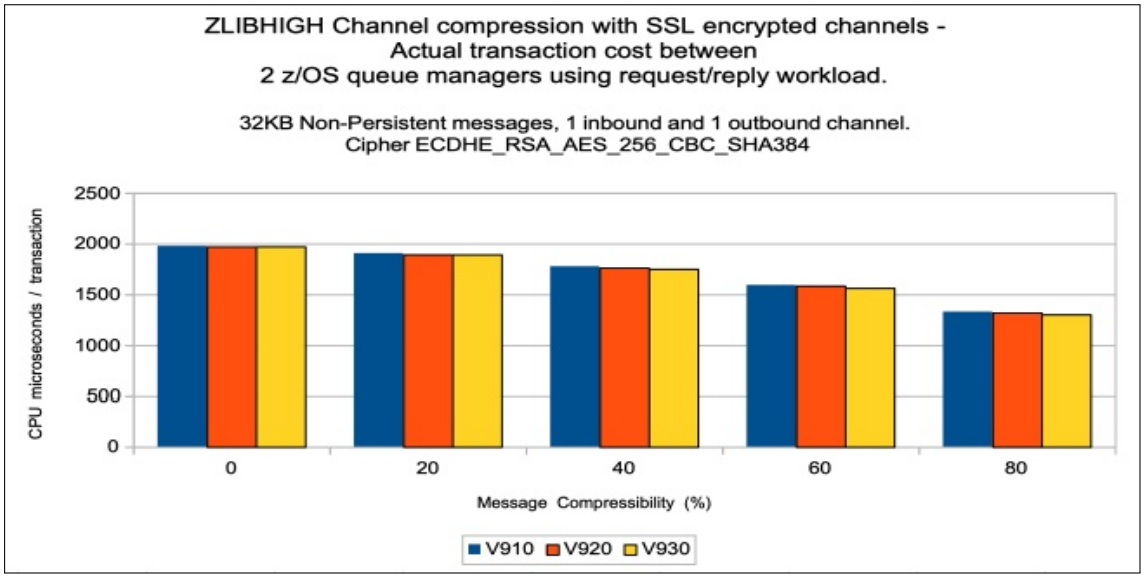


Chart: Transaction cost for 32KB messages with ZLIBHIGH on SSL channels



Streaming workload between 2 z/OS queue managers

Chart: Rate (MB/Second) to stream persistent messages between 2 z/OS queue managers

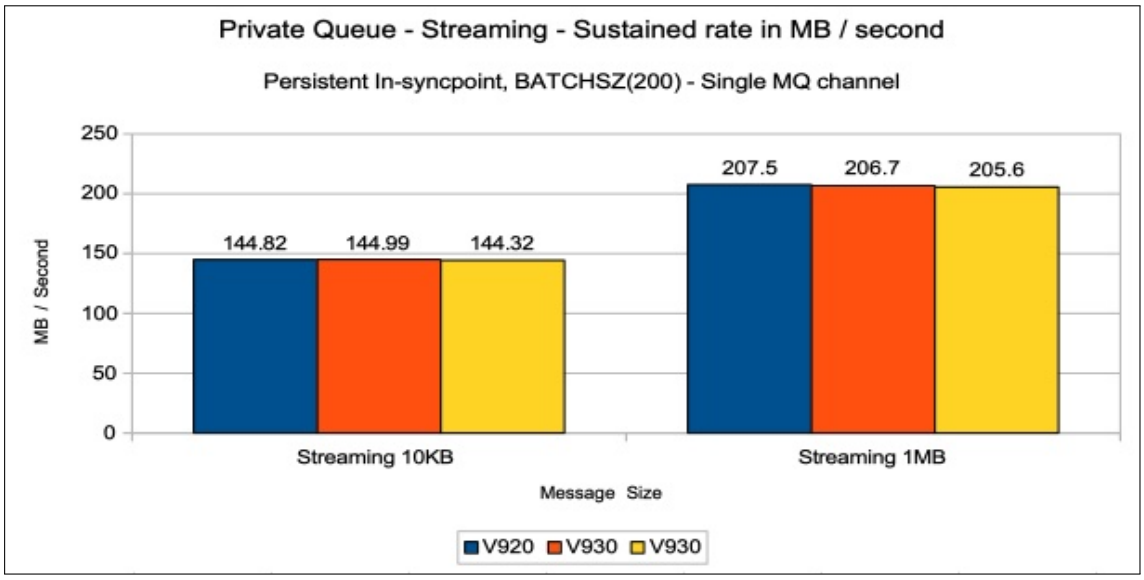
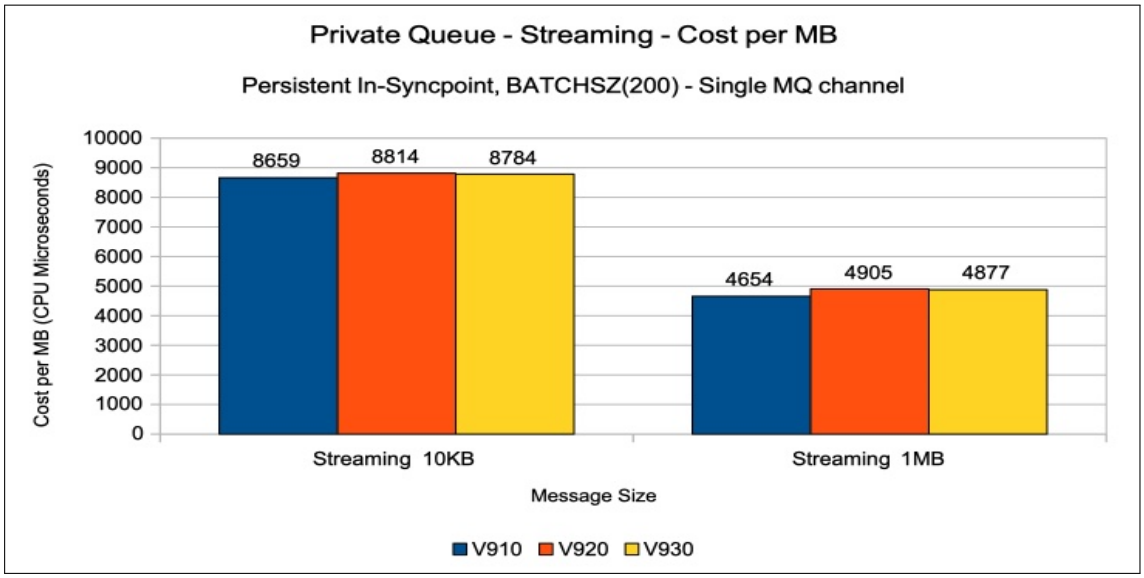


Chart: Cost to stream persistent messages between 2 z/OS queue managers



Moving messages across cluster channels

The regression tests for moving messages across cluster channels are relatively simple, providing an increasing number of destinations for each message put.

The cluster has 8 queue managers - one for the requester workload and the remaining 7 for the server workload. As the test progresses, valid destination queues are defined on an increasing number of the server queue managers. Only the requester queue manager is a full repository - this is not a recommended configuration!

A set of requester tasks is run against one queue manager and the application chooses either bind-on-open or bind-not-fixed. The requester application is written to use an 'MQOPEN, MQPUT, MQCLOSE' model to put messages to the queue. This ensures that the messages are distributed evenly when using the bind-on-open option.

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.

The measurements are run with 2KB messages and again with 32KB messages. In the 2KB workloads, the influence of the selection of the destination and the general cluster "overhead" is more prevalent.

Bind-on-open

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

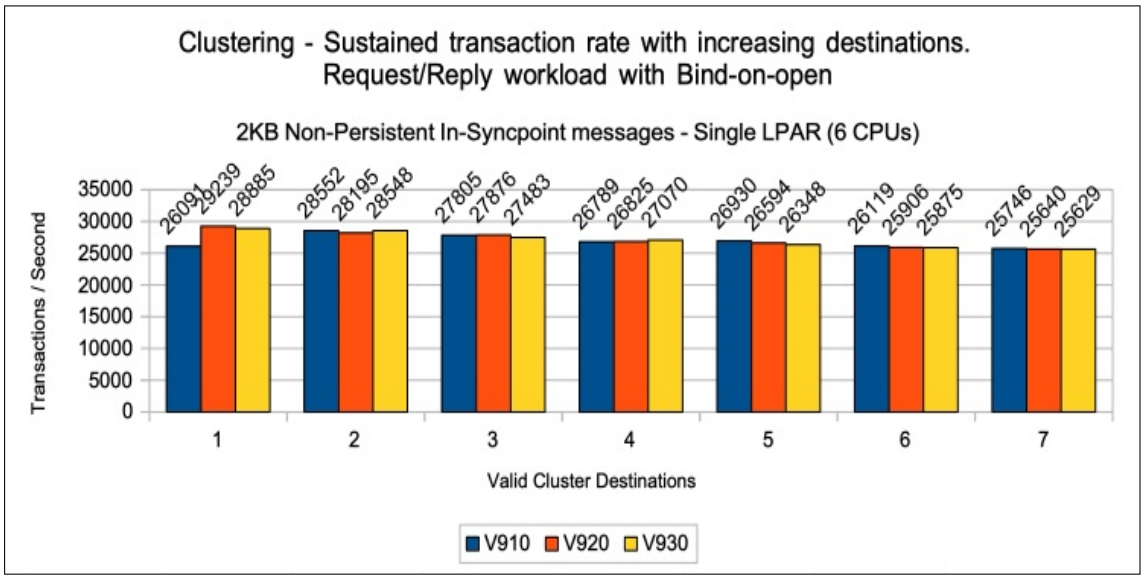


Chart: Bind-on-open transaction cost for 2KB messages

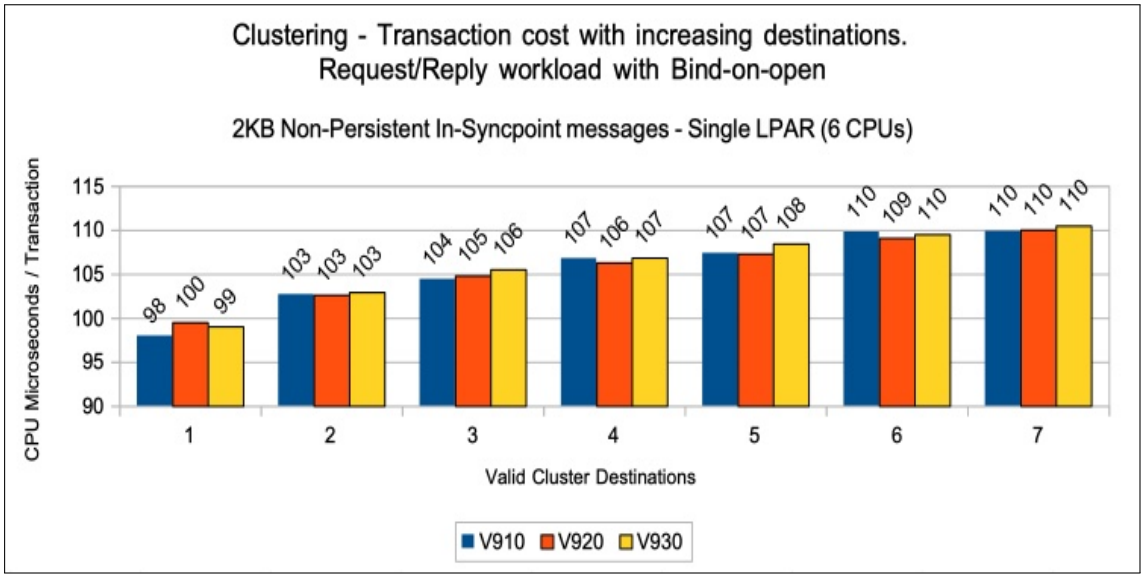


Chart: Bind-on-open transaction rate with 32KB messages

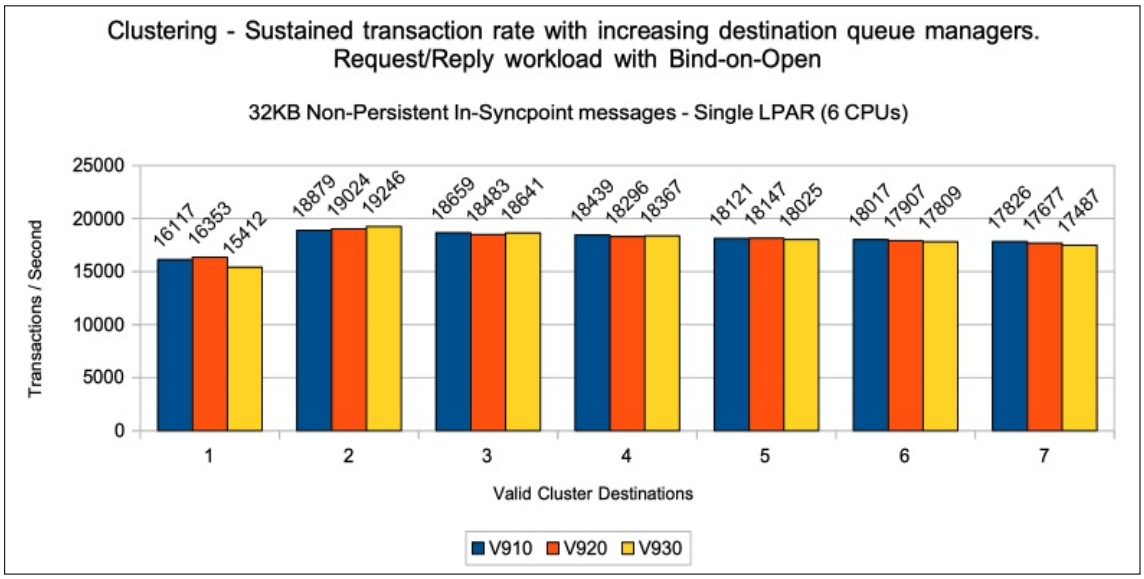
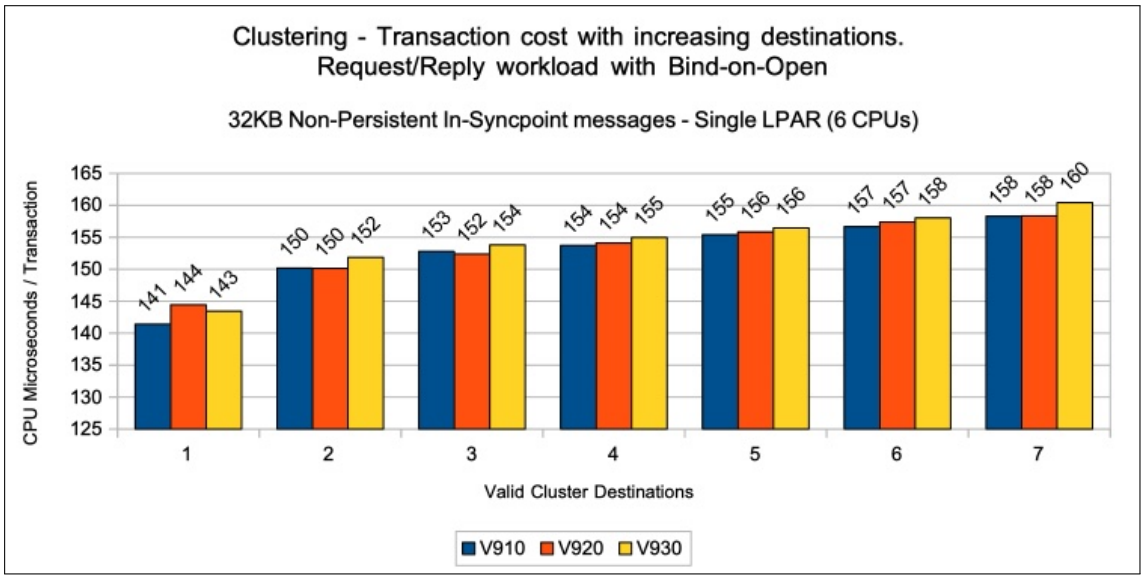


Chart: Bind-on-open transaction cost for 32KB messages



Bind-not-fixed

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

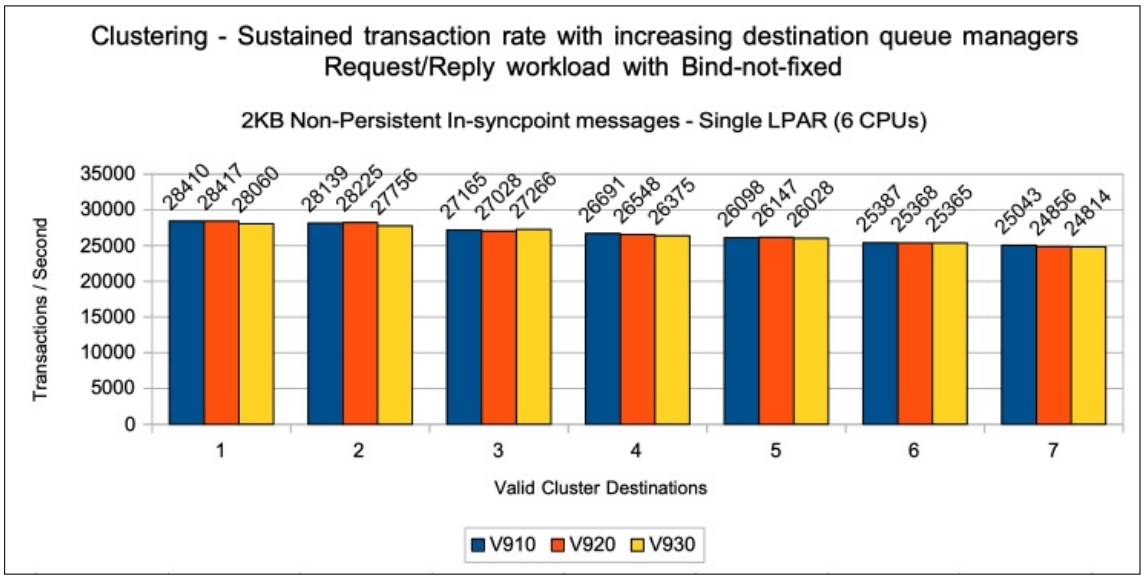


Chart: Bind-not-fixed transaction cost for 2KB messages

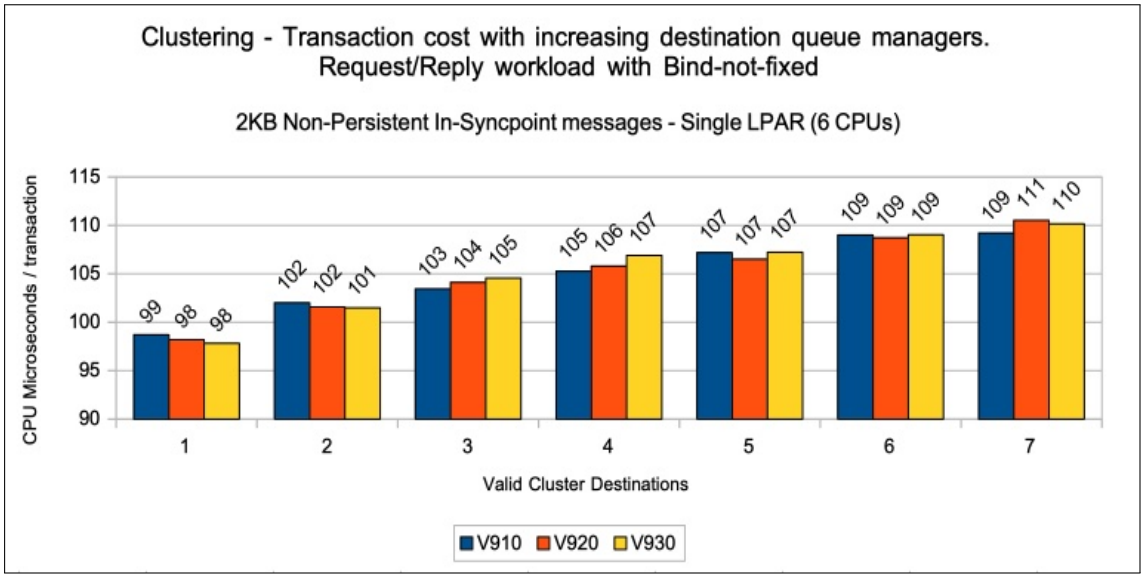


Chart: Bind-not-fixed transaction rate with 32KB messages

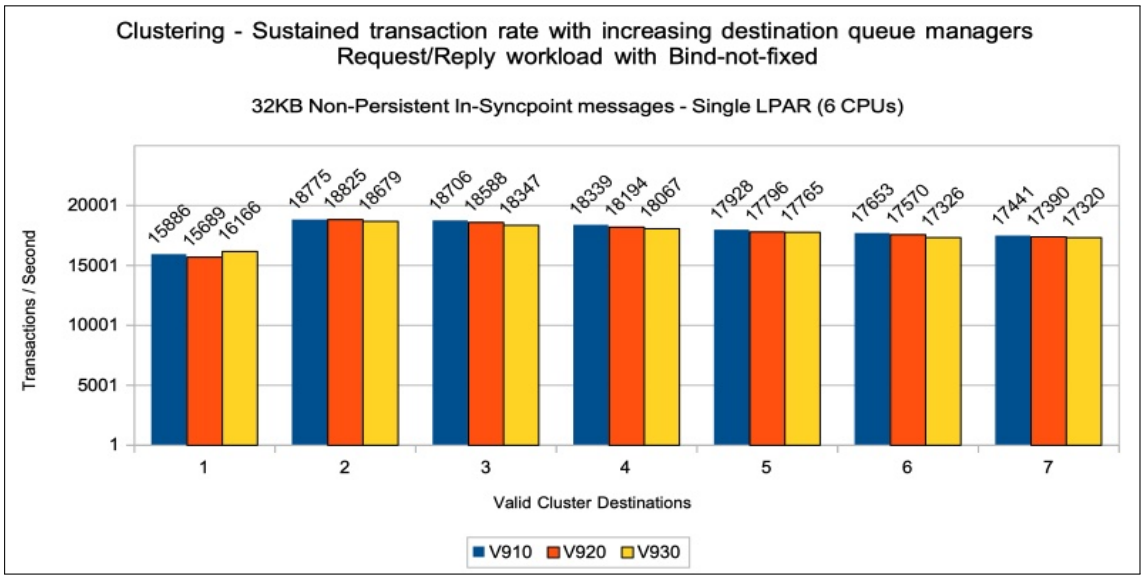
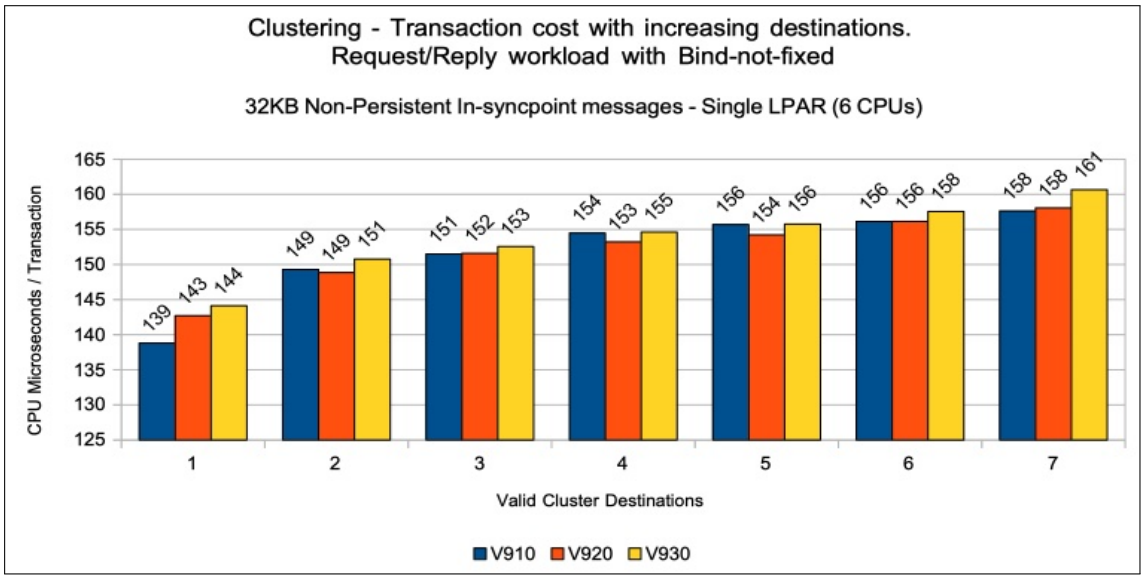


Chart: Bind-not-fixed transaction cost for 32KB messages



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 are 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)

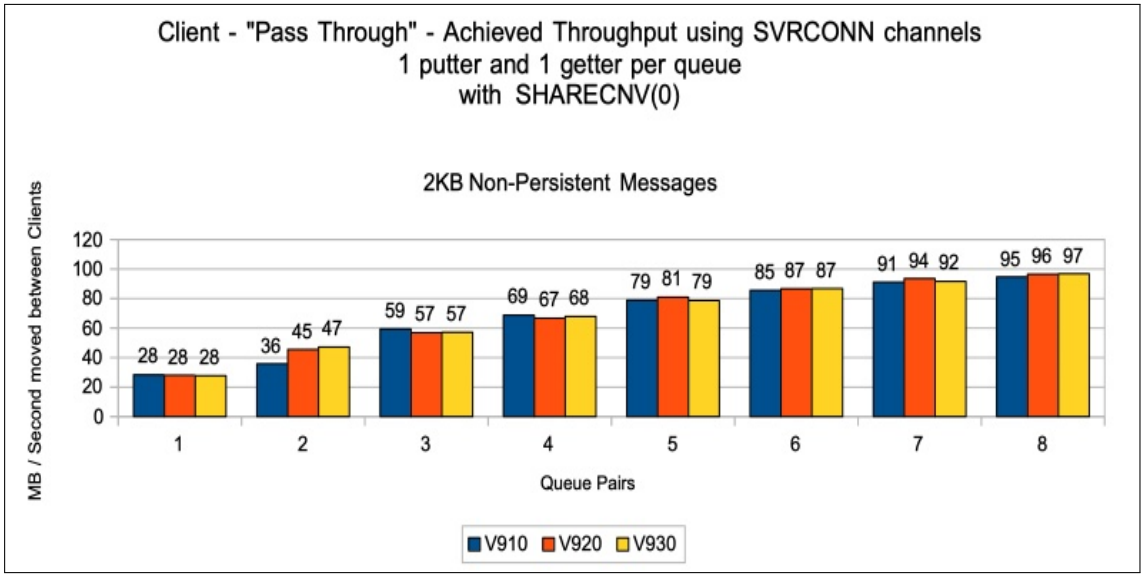
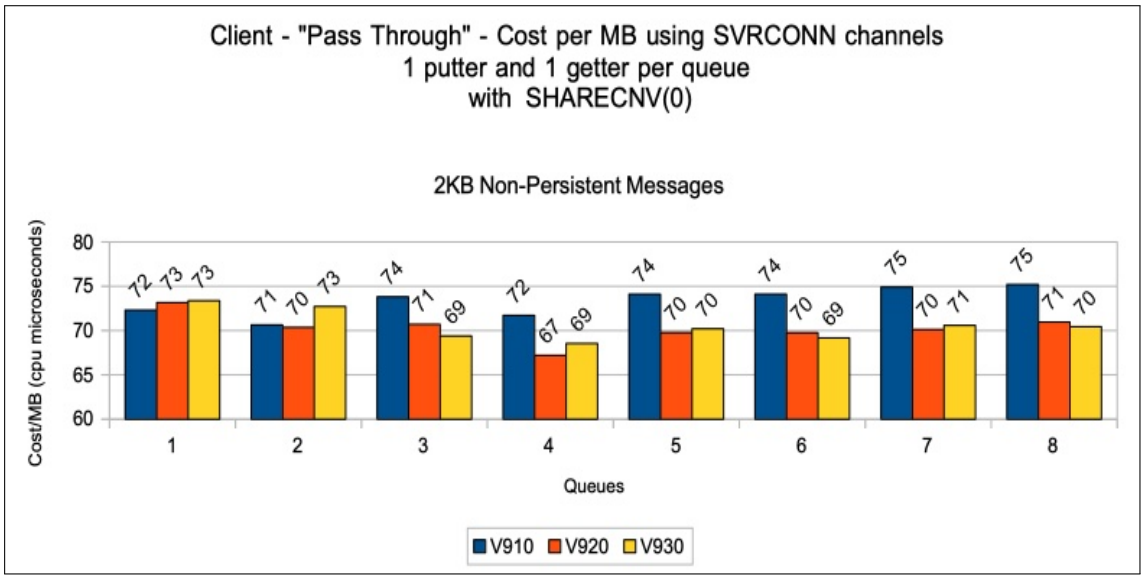


Chart: Cost per MB of client pass through tests with SHARECNV(0)



Client pass through test using SHARECNV(1)

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

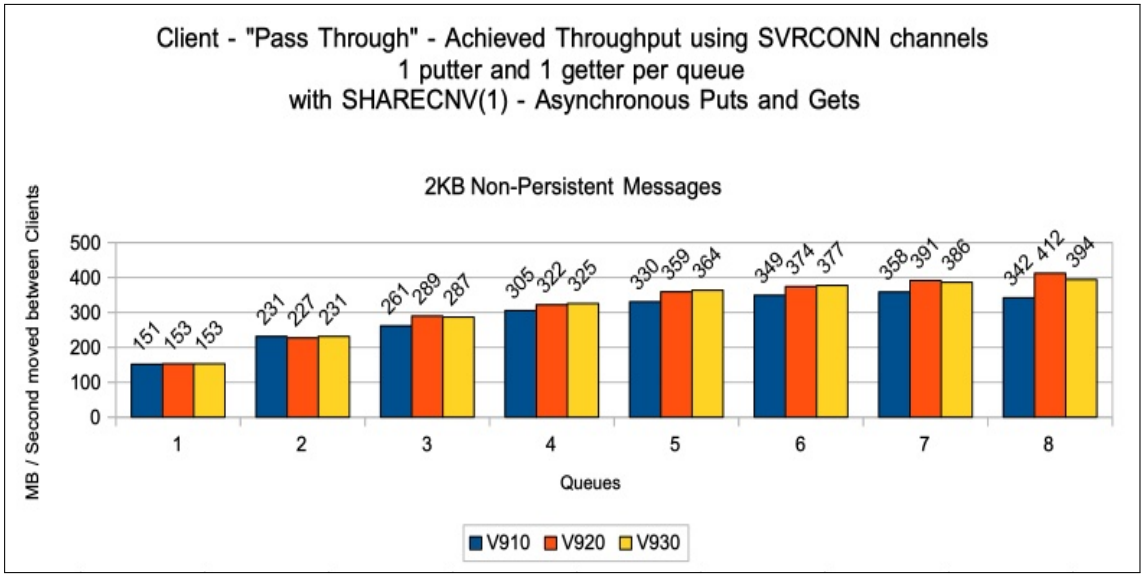
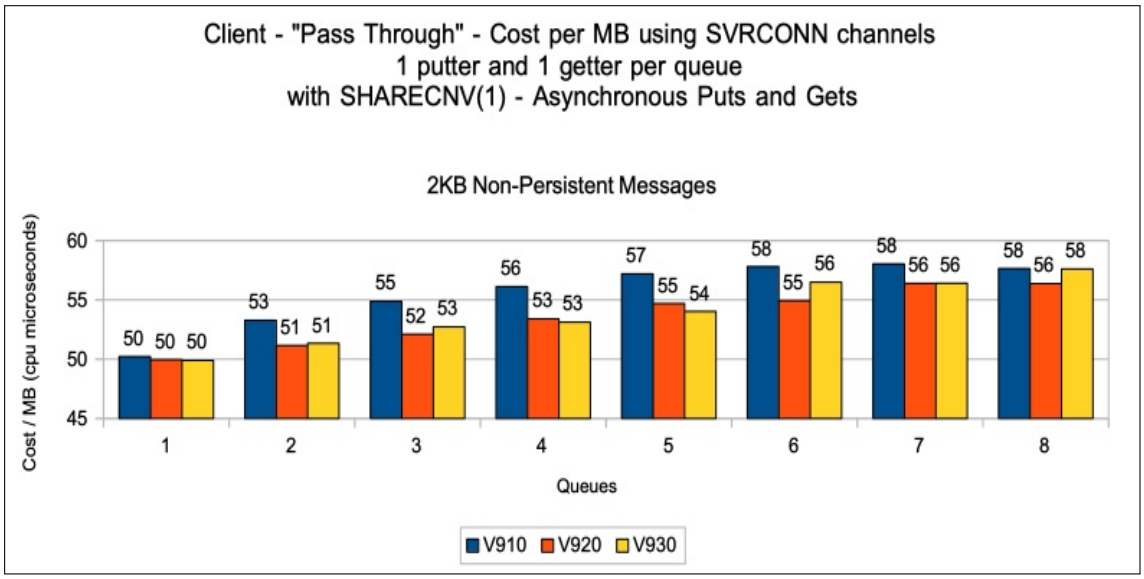


Chart: Cost per MB of client pass through tests with SHARECNV(1)



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

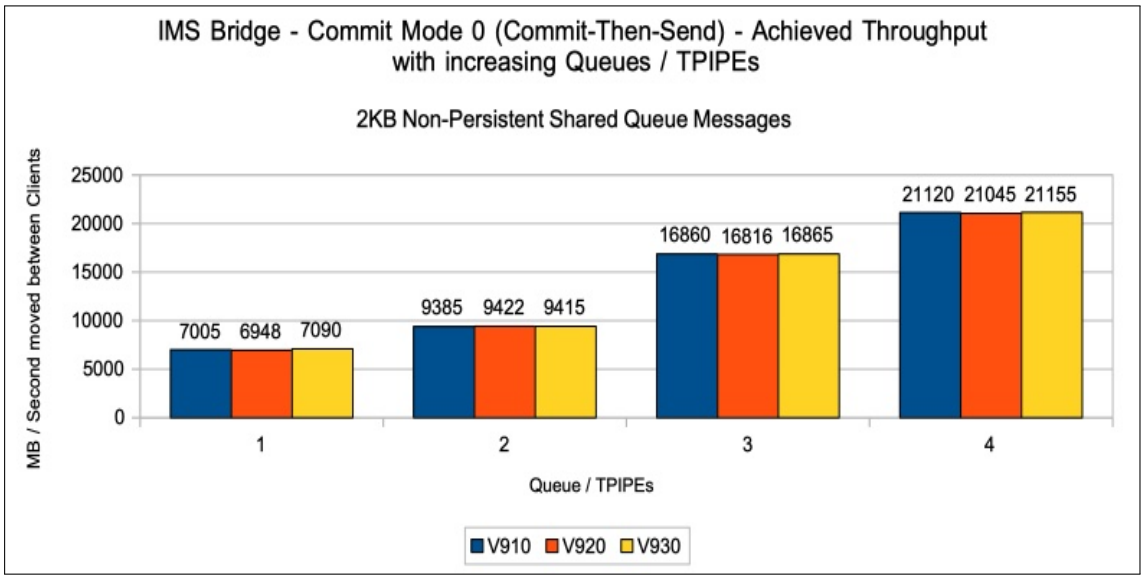
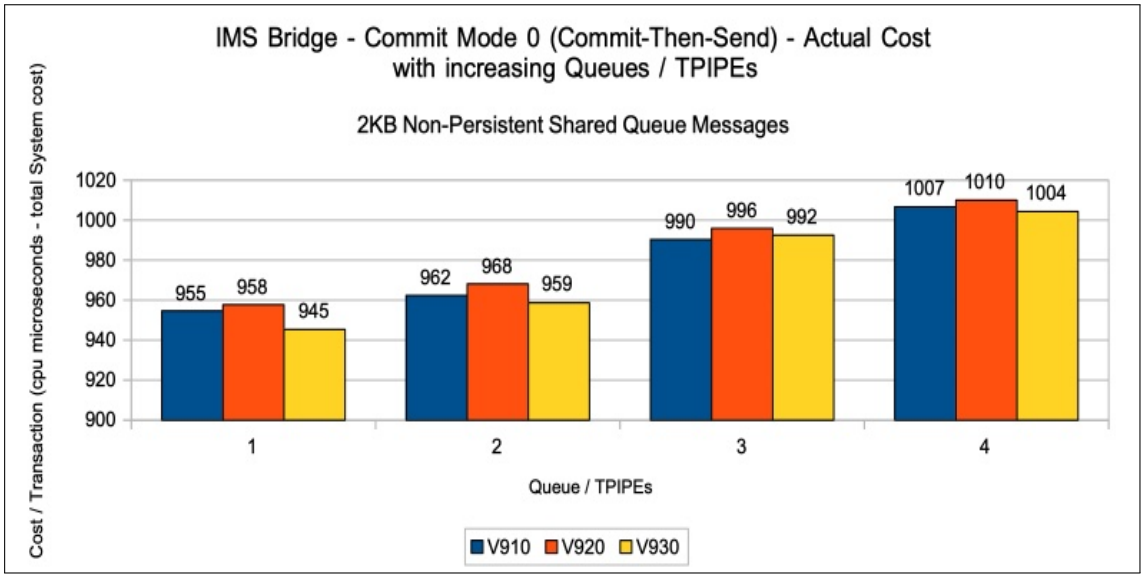


Chart: IMS Bridge commit mode 0 - Transaction cost



Commit mode 1 (send-then-commit)

Chart: IMS Bridge commit mode 1 - Throughput rate

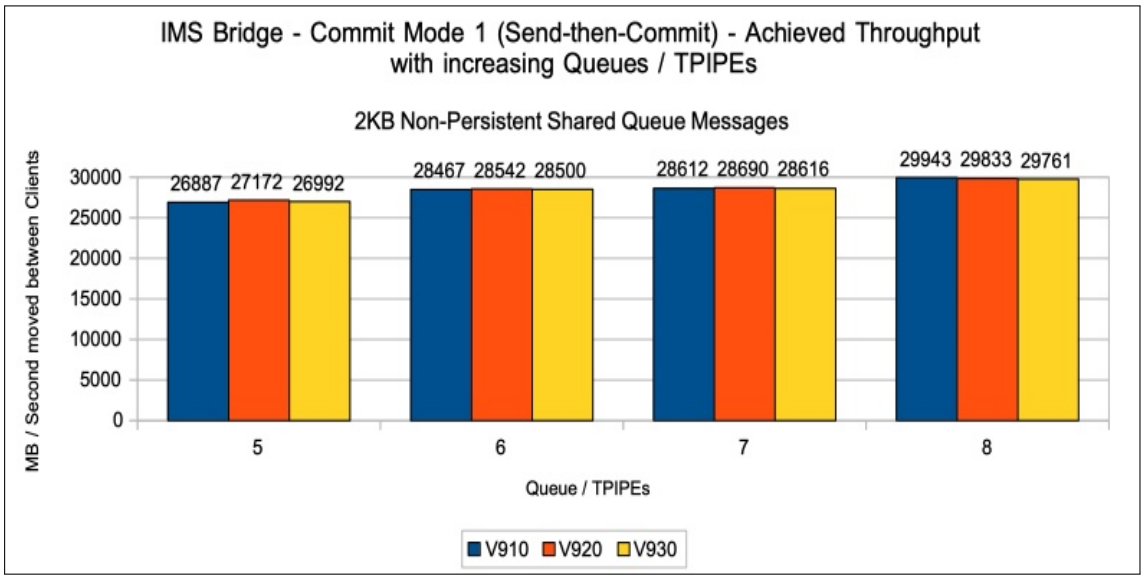
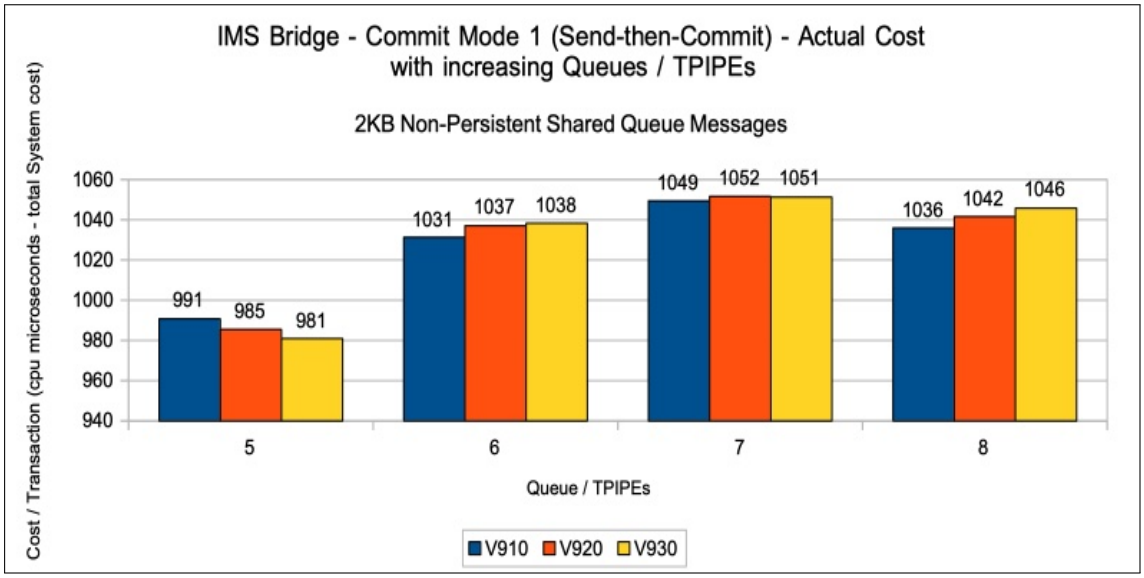


Chart: IMS Bridge commit mode 1 - Transaction cost



Trace

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

Since the performance for 9.1, 9.2 and 9.3 is comparable, the charts in this section show only 9.3 data for the purpose of clarity.

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.

To enable a comparison of impact from enabling MQ's global trace, the results from the equivalent measurements without trace enabled are included.

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

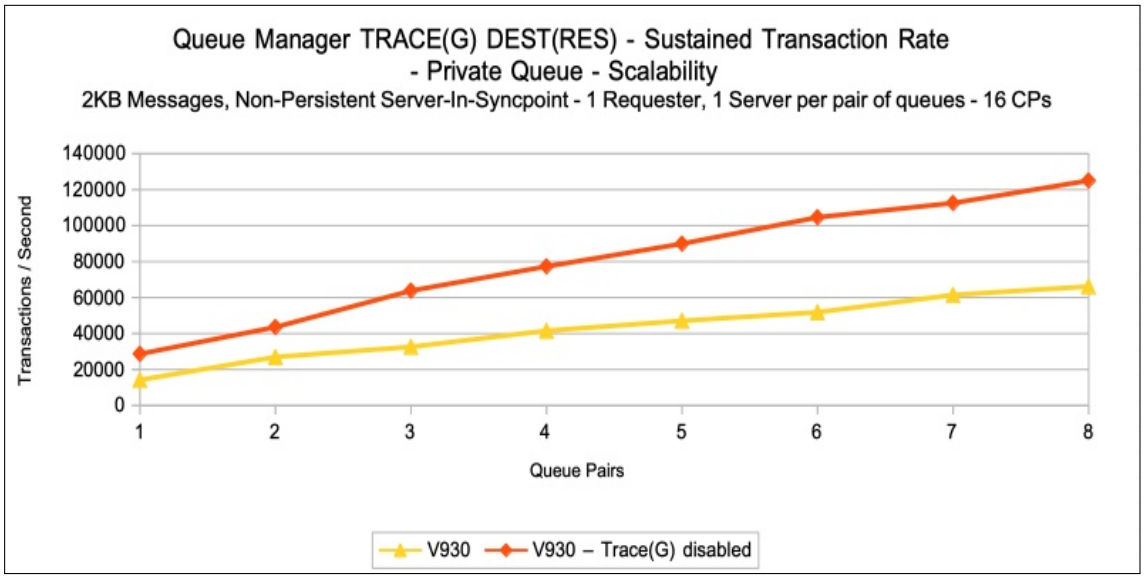
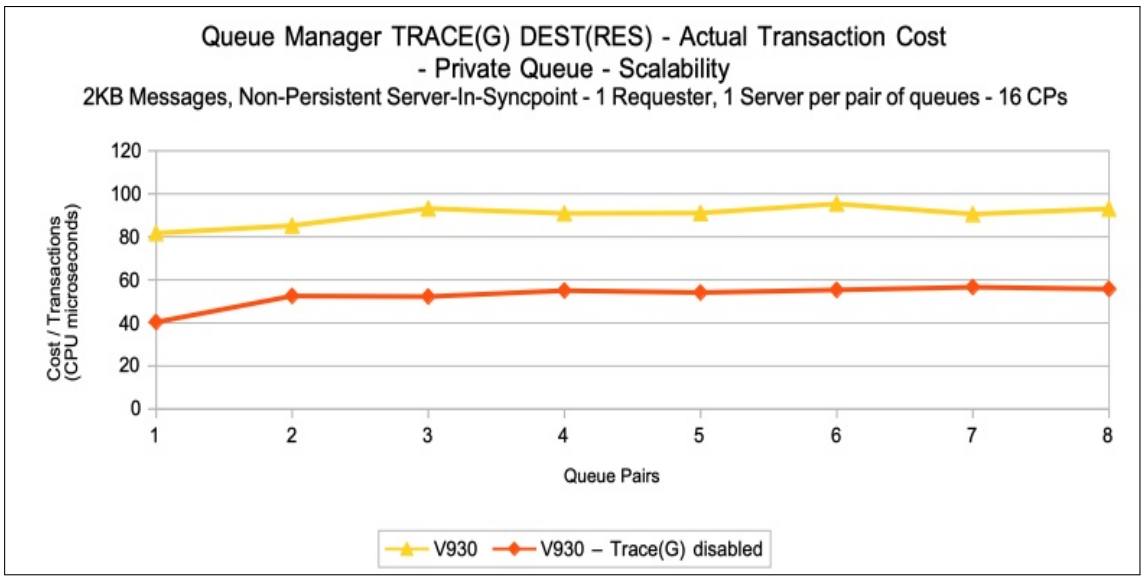


Chart: Queue manager TRACE(G) DEST(RES) - transaction cost



It must be emphasised that whilst the impact from enabling TRACE(G) for these workloads is effectively to double the overall transaction cost, this is primarily due to the high proportion of MQ API work when compared to the overall transaction.

Channel initiator trace

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

To enable a comparison of impact from enabling MQ's channel initiator trace, the results from the equivalent measurements without trace enabled are included.

Chart: Channel initiator TRACE(CHINIT) - throughput rate

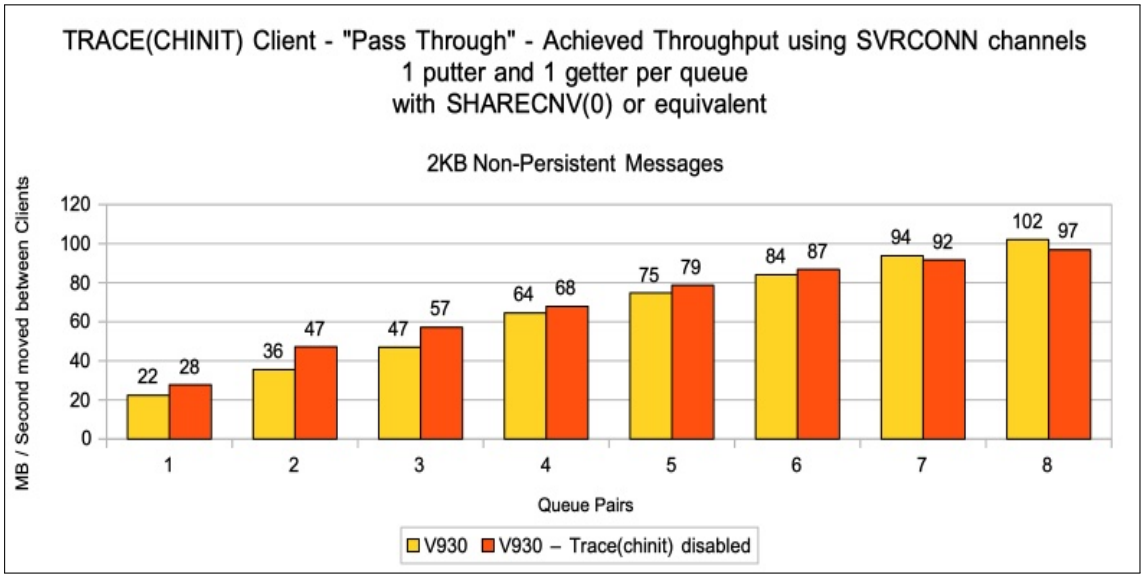
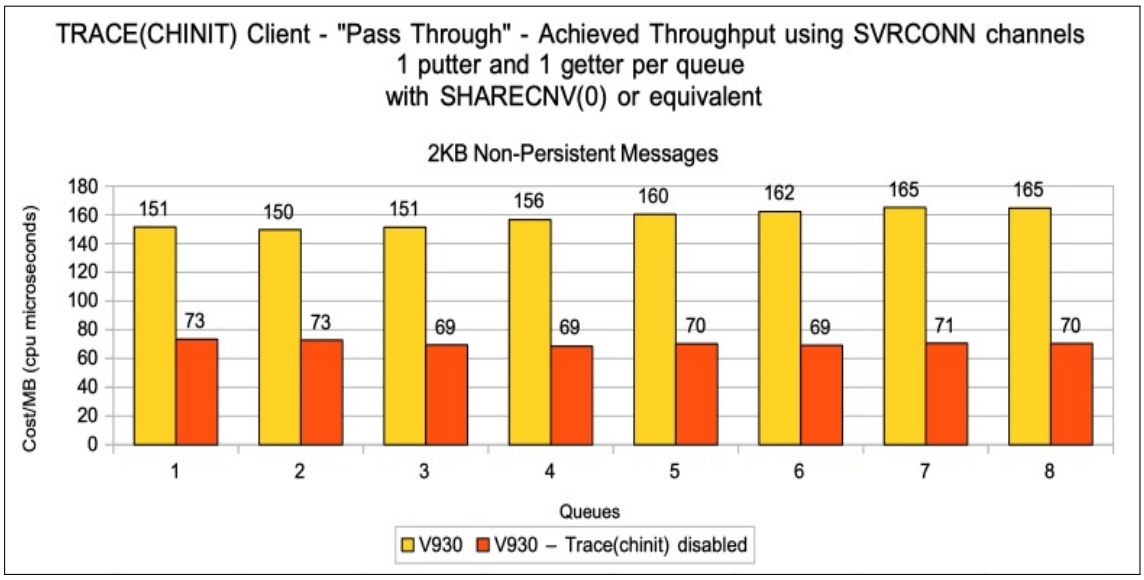


Chart: Channel initiator TRACE(CHINIT) - cost per MB



Note that there was sufficient CPU capacity such that the measurements with TRACE CHINIT enabled were able to achieve similar, or indeed higher, throughput to the measurements where trace was disabled. However, the additional cost would impact the throughput in a CPU constrained system.

Advanced Message Security

IBM MQ Advanced for z/OS VUE 9.3 only

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 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 for z/OS 9.1 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.

AMS regression test configuration

A simple request/reply workload is configured using a single z/OS queue manager with a pair of request and reply queues protected by an AMS security policy. In this instance there is 1 requester and 1 server task that run as long-running batch tasks.

The AMS policies defined are:

- **Integrity** Messages digitally signed with SHA256.
- **Privacy** Messages digitally signed with SHA256 and encrypted with AES256.
- **Confidentiality** Messages encrypted with AES256 and the key can be reused:
  - For 1 message
  - For 32 messages
  - For 64 messages
  - For an unlimited number of messages whilst the application remains connected.

Impact of AMS policy type on 2KB request/reply workload

Chart: Transaction rate for 2KB non-persistent workload with AMS protection applied to request and reply queues

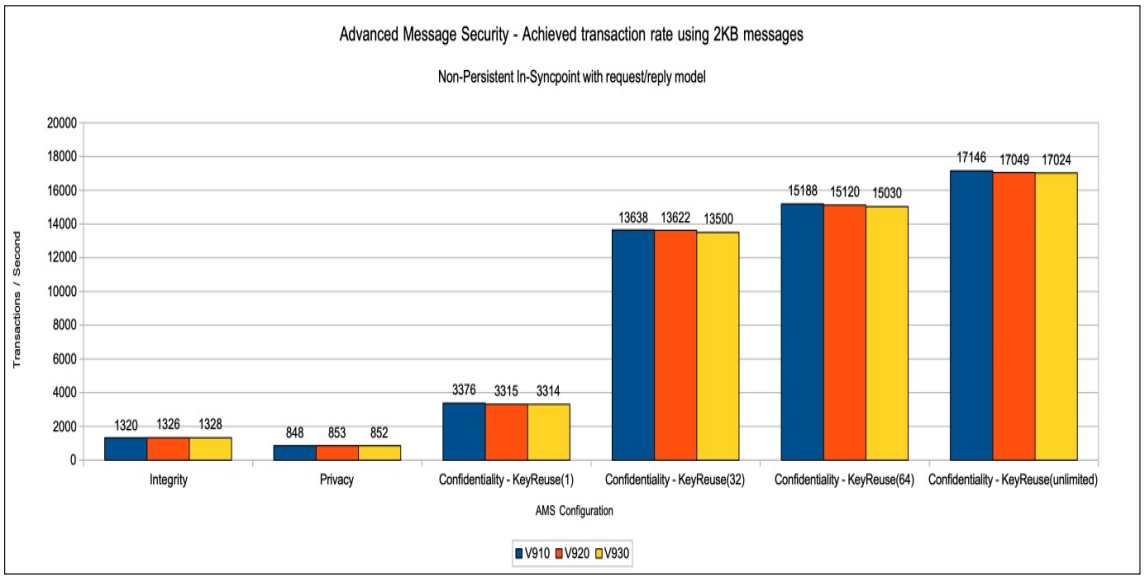
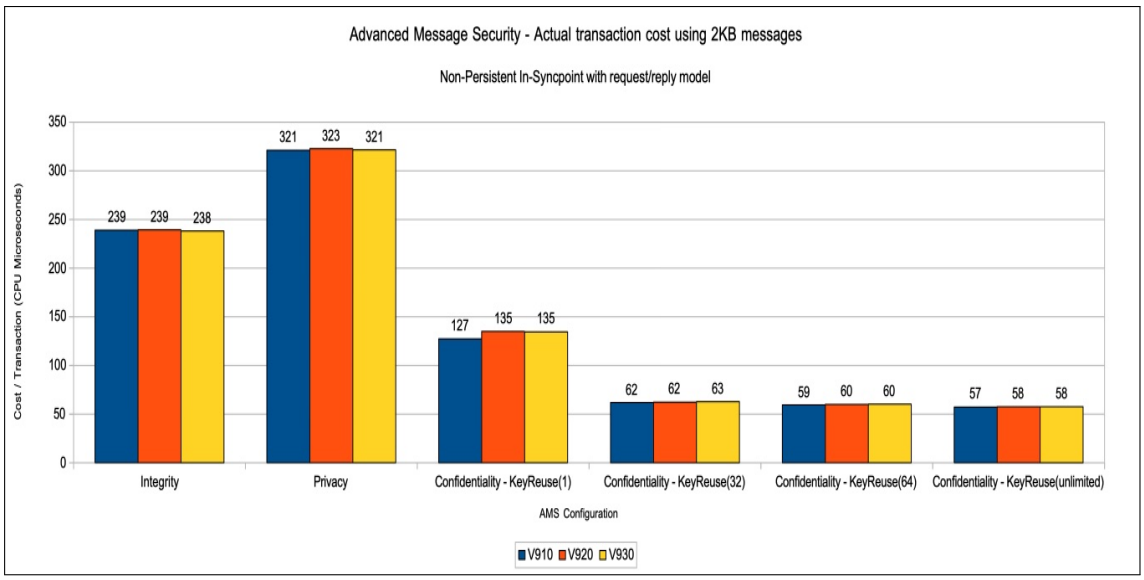


Chart: Transaction cost for 2KB non-persistent workload with AMS protection applied to request and reply queues



Impact of AMS policy type on 64KB request/reply workload

Chart: Transaction rate for 64KB non-persistent workload with AMS protection applied to request and reply queues

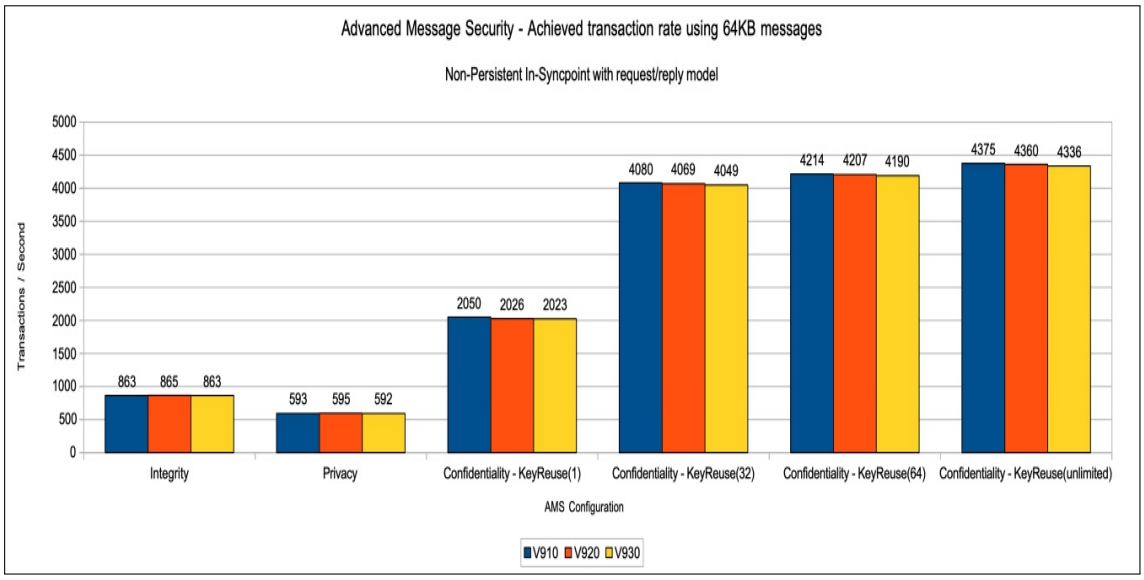
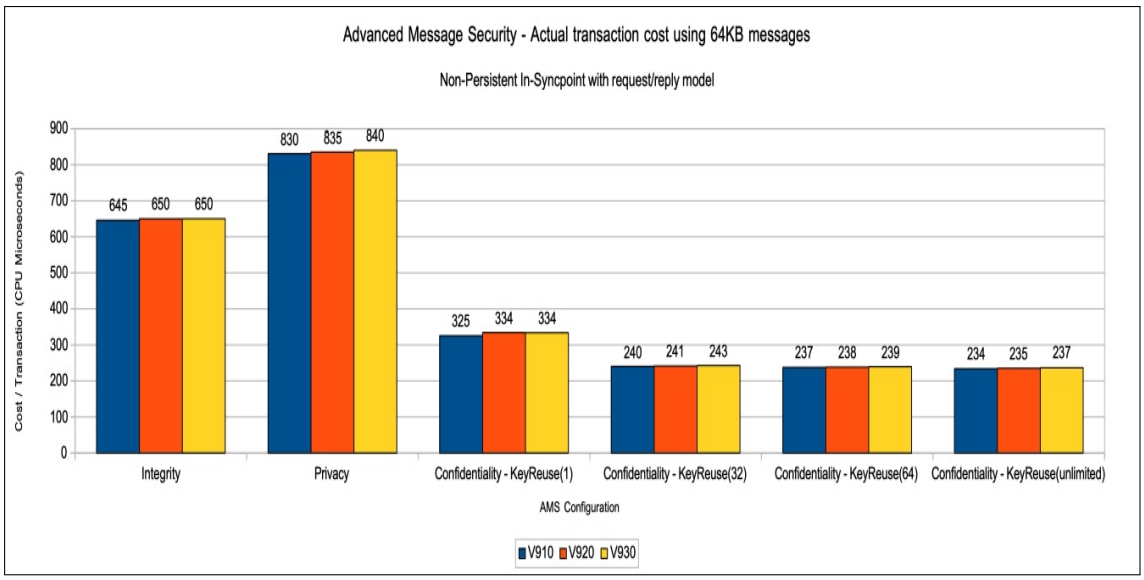


Chart: Transaction cost for 64KB non-persistent workload with AMS protection applied to request and reply queues



Impact of AMS policy type on 4MB request/reply workload

Chart: Transaction rate for 4MB non-persistent workload with AMS protection applied to request and reply queues

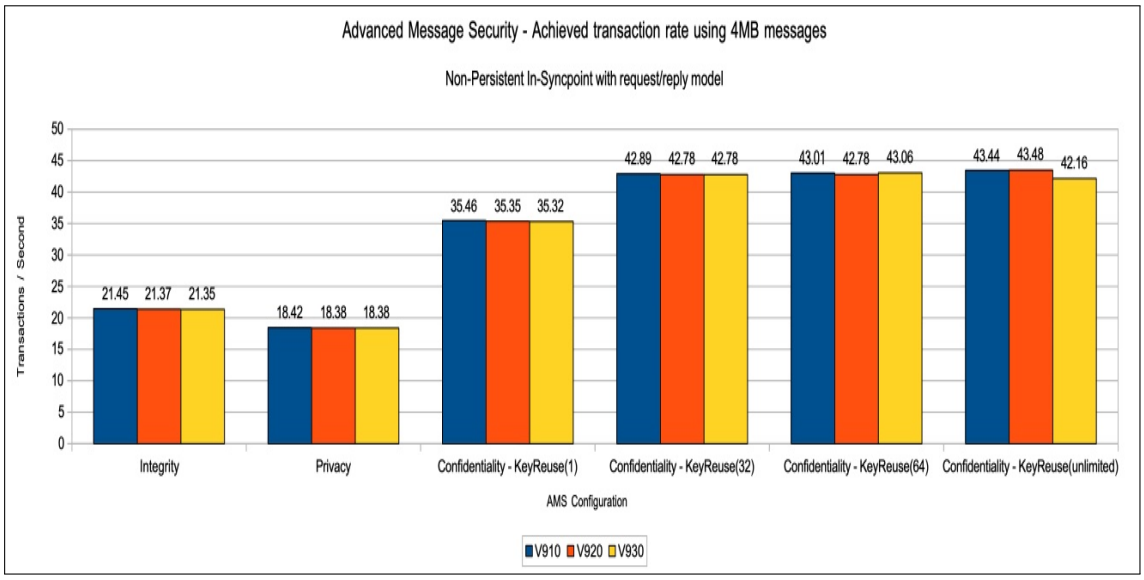
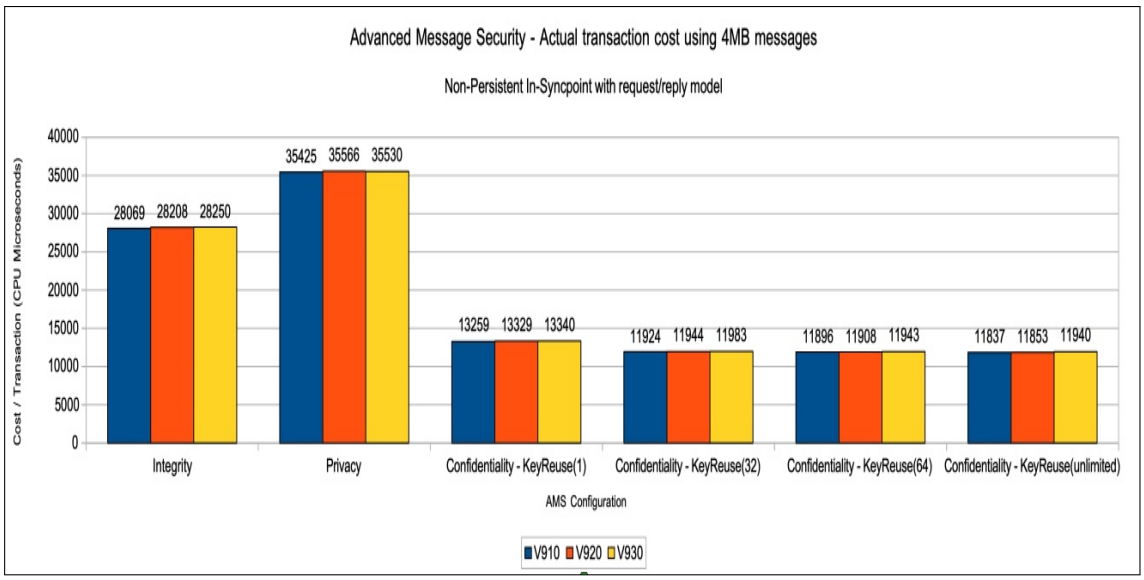


Chart: Transaction cost for 4MB non-persistent workload with AMS protection applied to request and reply queues



Appendix B

System configuration

The majority of the measurements in this report were run on our **IBM MQ Performance Sysplex running on IBM z15 (8561-7J1)** configured thus:

LPAR 1: 1-32 dedicated CP processors, plus 2 zIIP, 144GB of real storage.

LPAR 2: 1-3 dedicated CP processors, 48GB of real storage.

LPAR 3: 1-10 dedicated CP processors, plus 2 zIIP, 48GB of real storage.

Default Configuration:

3 dedicated processors on each LPAR, where each LPAR running z/OS v2r5 FMID HBB77D0.

Coupling Facility:

- Internal coupling facility with 4 dedicated processors.
- Coupling Facility running CFCC level 24 service level 00.30.
- Dynamic CF Dispatching off.
- 3 x ICP links between each z/OS LPAR and CF

DASD:

- FICON Express 16S connected DS8950F.
- 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.
    - SMC-D enabled by default between LPARs 1 and 3.
- zEDC compression available by default - used with MQ channel attribute COMPMSG(ZLIBFAST)..
- Crypto Express7 features configured thus:

- 1 x Accelerator, shared between LPARs 1,2 and 3.
- 2 x Coprocessor on LPAR1.
- 1 x Coprocessor on LPAR2.
- 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 5.5.
  - Db2 for z/OS version 12.
  - IMS 15.1
  - IBM MQ for z/OS 9.2 with latest service applied as of June 2022.
  - IBM MQ for z/OS 9.1 with latest service applied as of June 2022.

Streaming Queue measurements

When testing shared queue support for streaming queues as part of APAR PH49686, the MQ Performance sysplex had moved to IBM z16. As a result all streaming queue data has been remeasured on the system configured thus:

IBM MQ Performance Sysplex running on IBM z16 (3931-7x1) configured thus:

LPAR 1: 1-32 dedicated CP processors, plus 2 zIIP, 144GB of real storage.

LPAR 2: 1-3 dedicated CP processors, 48GB of real storage.

LPAR 3: 1-10 dedicated CP processors, plus 2 zIIP, 48GB of real storage.

Default Configuration:

3 dedicated processors on each LPAR, where each LPAR running z/OS v2r5 FMID HBB77D0.

Coupling Facility:

- Internal coupling facility with 4 dedicated processors.
- Coupling Facility running CFCC level 25 service level 02.44.
- Dynamic CF Dispatching off.
- 3 x ICP links between each z/OS LPAR and CF

DASD:

- FICON Express 32S connected DS8950F.
- 4 dedicated channel paths (shared across sysplex)
- HYPERPAV enabled.