MQ for z/OS Performance Report

IBM MQ Advanced for z/OS VUE version 9.2
and
IBM MQ for z/OS version 9.2

May 2024

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.

Third edition, May 2024. This edition applies to IBM MQ Advanced for z/OS VUE version 9.2 and IBM MQ for z/OS version 9.2 (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 9.2**. 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®
- CICS®
- Db2 for z/OS®
- IMS™
- MVS™
- z15™
- FICON®
- WebSphere®
- IBM MQ®

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

EXPORT REGULATIONS

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

Summary of Amendments

Date	Changes
2024	Version 1.2 - Remove references to end-of-support MQ releases.
2021	Version 1.1 - Update to TLS channel start and stop data.
2020	Version 1.0 - Initial Version.

Table of contents

Chapter 1

Performance highlights

This report focuses on performance changes since previous versions (V9.0 and V9.1) 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 Advanced for z/OS VUE version 9.2, there are a number of features that can benefit the performance of the MQ queue manager:

zHyperWrite support for MQ active logs can reduce the average I/O time by up to 60% and improve the maximum sustainable log rate by up to 2.4 times over active logs mirrored using Metro Mirror.

The **Aspera fasp.io gateway** can improve the performance of streaming-type workloads of over 70 times that achieved with TCP/IP on high latency networks.

In addition, IBM MQ Advanced for z/OS VUE version 9.2 offers improvements to the security of your data, in three ways:

1. Encrypting data on MQ data sets at write time.
2. Providing support for TLS 1.3 ciphers on MQ channels.
3. Allowing Enterprises to protect their data using AMS qualities of protection without mandating that business partners run with the same level of protection.

Chapter 2

Existing function

General statement of regression

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

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

Storage usage

Virtual storage constraint relief has not been a primary focus of this release.

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 v9.2 is similar to both v9.0 and v9.1 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: v9.0, v9.1 and v9.2.

- 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.

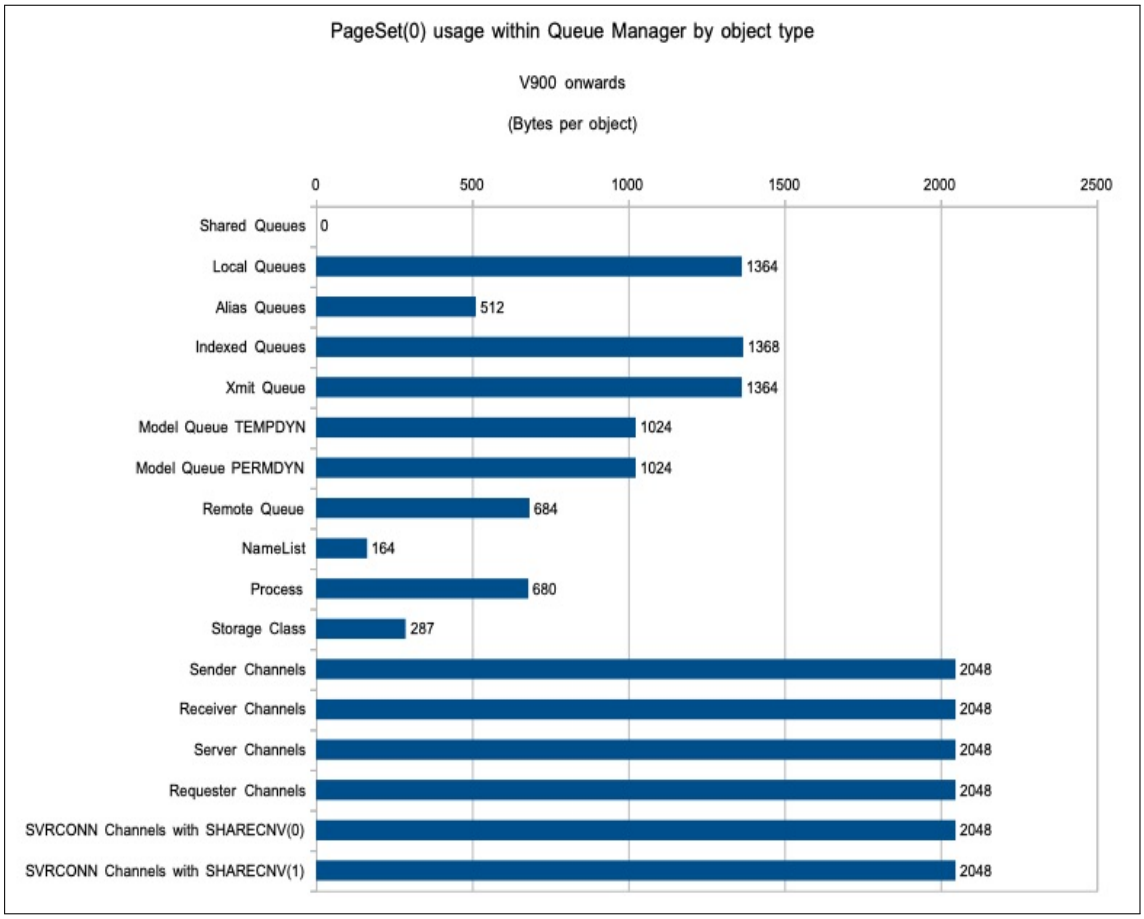
Object Sizes

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.

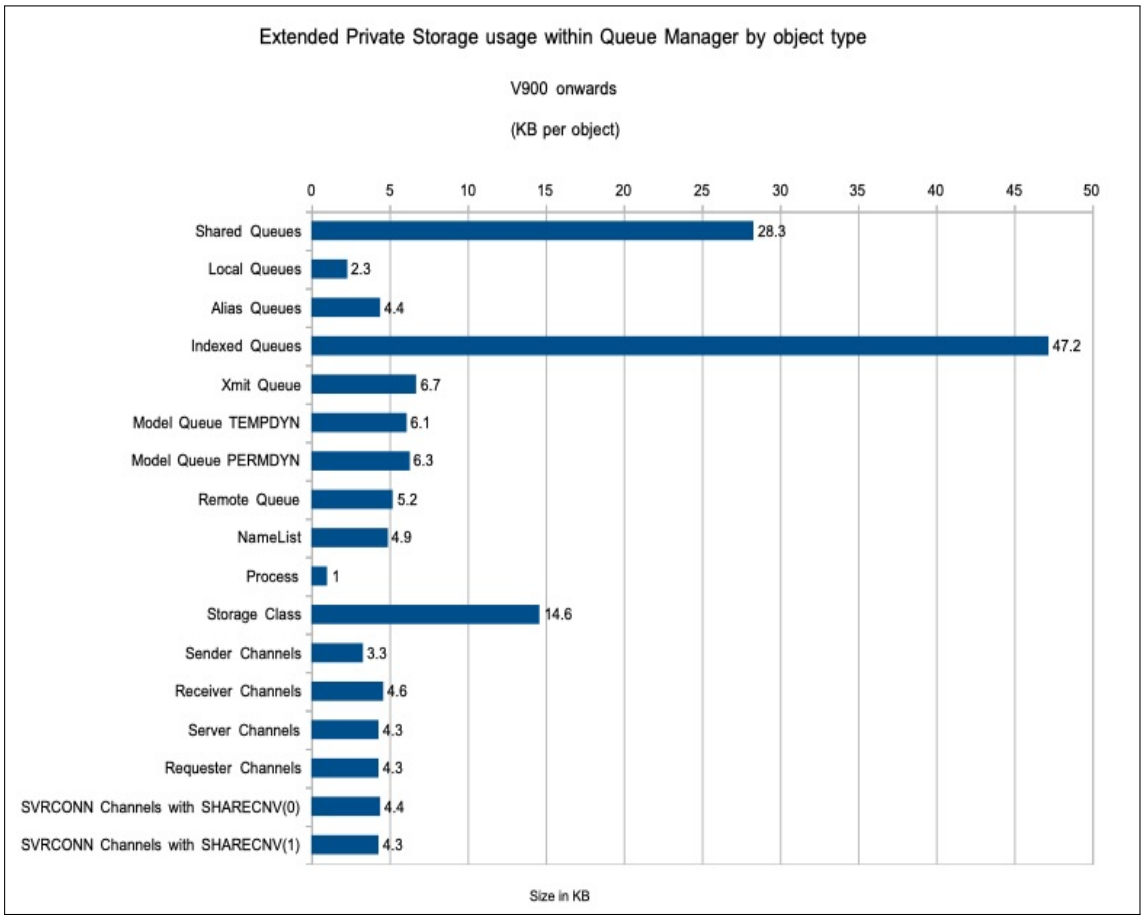
PAGESET(0) Usage

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 15KB per connection, however there is additional usage in the following (non-exhaustive) cases:

- Where connection is over a SHARECNV(1) channel, the usage increases to 24KB.
- Where connection is over a SHARECNV(10) channel and has a CURSHCNV of 10, the usage is 16.3KB per connection (163KB 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 29.8KB, giving an additional shared queue overhead of 5.8KB.

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 SupportPac MP16 "Capacity Planning and Tuning Guide".

		Channel initiator footprint (KB / SVRCONN channel) Message size (KB)			
IBM MQ release	**SHARECNV**	**1**	**10**	**32**	**64**
v9.0	**0**	82	97	158	186
v9.1	**0**	82	98	163	184
v9.2	**0**	82	99	158	187
v9.0	**1**	156	177	263	311
v9.1	**1**	156	178	230	257
v9.2	**1**	157	177	233	259
v9.0	**10**	243	268	656	985
v9.1	**10**	246	268	347	374
v9.2	**10**	241	267	345	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.

If the workload is expected to be clients connecting via SHARECNV(1) SVRCONN channels and using 32KB messages, we could predict that the v9.2 channel initiator could support a maximum of 4,620 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)	91	100	109	1141
Overhead of message size increase on 1KB messages		+9	+19	+1051

Chapter 3

New function for version 9.2

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

- zHyperWrite support for active logs.
- Data set encryption for active logs, page set and SMDS.
- TLS 1.3 support for MQ channels.
- Advanced Message Security (AMS) Interception on Server to Server channels.
- Aspera fasp.io Gateway.

Each of these features will be detailed in subsequent chapters.

Chapter 4

zHyperWrite support for active logs

MQ 9.2 introduces support for zHyperWrite with MQ's active log data sets.

This section discusses improved active log performance, in particular with IBM® Metro Mirror and zHyperWrite.

IBM® Metro Mirror, previously known as Synchronous Peer to Peer Remote Copy (PPRC), is a synchronous replication solution between two storage subsystems, where write operations are completed on both primary and secondary volumes before the write operation is considered to be complete.

The IBM MQ Knowledge Centre section on "Using MetroMirror with IBM MQ" discusses which types of IBM MQ data sets can be replicated using Metro Mirror.

As of MQ 9.2, zHyperWrite is supported with MQ active logs. This offers benefits in a number of areas:

- Reduced I/O times by **up to 60%**.
- Reduced elapsed time for MQ commit by **up to 60%**, which can reduce contention.
- Improved sustained log rate, allowing for each MQ queue manager to process **up to 2.4 times** the volume of workload.

Caveats:

- The benefits of zHyperWrite reduces as the distance of replication increases.
- Whilst improved log performance can be achieved with zHyperWrite, increased SRB time in the MQ MSTR address space might be observed because the extra (Media Manager) I/O requests are processed under MSTR SRB when zHyperWrite is enabled.

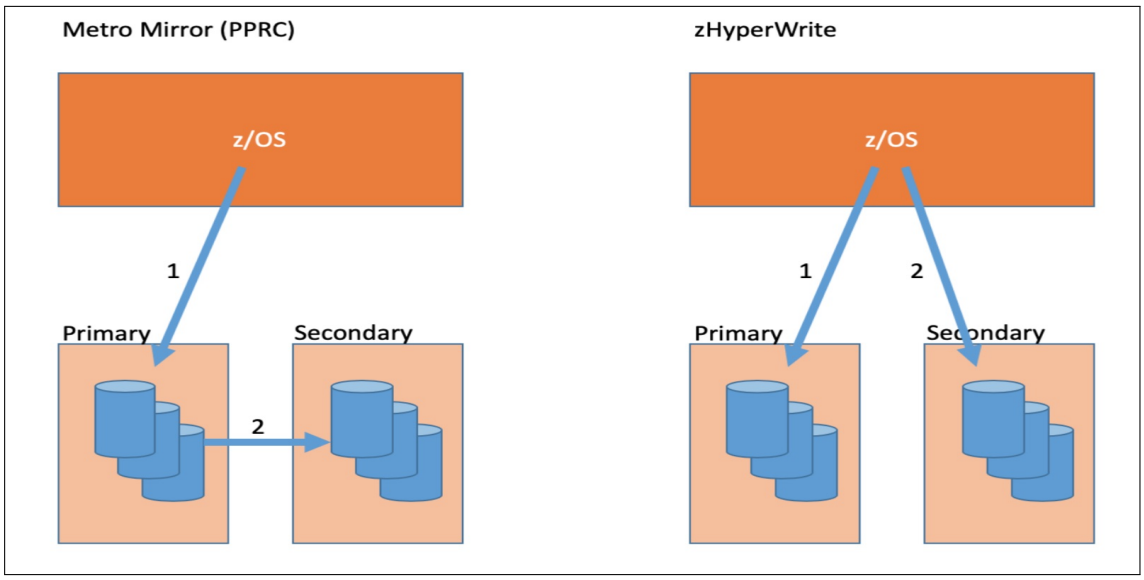
What is zHyperWrite?

The z/OS 2.4 Knowledge Centre describes zHyperWrite as:

IBM zHyperWrite processing can be used by I/O drivers, such as Media Manager, for certain write I/O operations to perform software mirroring to peer-to-peer remote copy (PPRC) devices that are monitored for HyperSwap® processing (with GDPS® or TPC-R). IBM zHyperWrite data replication can be used to reduce latency in these HyperSwap environments. For maximum benefit, IBM zHyperWrite data replication should only be used when all synchronously mirrored relationships are managed by HyperSwap. Devices support IBM zHyperWrite data replication when the following conditions are true:

- The devices support IBM zHyperWrite data replication. Both the primary and secondary devices in a synchronous PPRC relationship must support this function.
- The devices in the synchronous PPRC relationship are managed by HyperSwap (either GDPS HyperSwap or TPC-R HyperSwap).

A simplified view of the difference between PPRC and zHyperWrite is offered in the following diagram:



Notes:

In the PPRC configuration, the mirroring is driven from the primary control unit.

In the zHyperWrite configuration, the mirroring is requested once the I/O to the primary control unit is requested. This can reduce the time to initiate the mirrored request.

Why does my log performance matter?

Improved MQ active log performance is important for a number of reasons:

- Allows more work to get done with the same hardware footprint.
- Better able to handle workload spikes.
- Reduces cost.

Generally, there are a number of ways to make transactions run faster on System Z and z/OS:

1. Faster CPU.
2. Software scaling, reducing contention, faster I/O.
3. Faster I/O technologies such as zHPF, 16Gb channels, zHyperWrite etc.
4. Run at lower utilisations, address Dispatcher Queuing Delays.
5. Faster network, such as SMC-R or SMC-D.

When using high levels of persistent messaging, the rate at which MQ is able to process the workload is largely dependent on the rate at which the MQ logger task is able to complete its I/O. This rate is almost entirely dependent upon the rate at which the I/O subsystem is able to respond to the requests.

The following extract from MQ Statistics data, formatted using program MQSMF (part of support-pac MP1B "Interpreting accounting and statistics data"), demonstrates the single logger task, which runs as an SRB, is 99.82% busy, of which 95.90% is waiting for I/O to complete:

```
Log write rate      272MB/s per copy
Logger I/O busy:    95.90%
Logger task busy:   99.82%
```

In this environment, one way to alleviate this constraint is to reduce the I/O times.

When running synchronous replication technologies such as Metro Mirror (PPRC), the I/O times can be significantly higher than when running without the datasets replicated. The zHyperWrite technology can significantly reduce the I/O times when replicating datasets.

There are a number of benefits that zHyperWrite-enabled active logs can offer to a MQ queue manager:

1. Reduced I/O time.
2. Reduced elapsed time for MQ commit.
3. Improved sustainable log rate.

These benefits do not come at zero cost, and indeed there is an impact to the queue manager costs, which is discussed in the "Impact to MQ queue manager costs" section.

zHyperWrite test configuration

The following measurements were run on a single LPAR of the 3 LPAR MQ Performance sysplex on the IBM z15 (8651-7J1). In these tests, the LPAR was configured with 3 dedicated general purpose processors, and for the purposes of comparison is similar to a 8651-703.

A single MQ queue manager was created in 3 configurations, in each case with single active log and single archive log.

1. Baseline - no mirrored datasets.
2. PPRC - active logs mirrored using Metro Mirror.
3. zHyperWrite - active logs mirrored using Metro Mirror and zHyperWrite enabled at the queue manager.

The queue manager was configured with single active and archive logs as in the dual active and archive log environment, we saw measurements impacted by site hardware limitations. Details of the observed impact can be found in the "Impact of I/O limitations on dual active and dual archive logs" section of the report.

In each configuration, a put-commit/get-commit workload was run using a range of message sizes from 2KB to 4MB.

In the test configuration used, the mirrored DASD was at zero distance. Synchronously replicated DASD over a distance may see different results.

With regards to the disk subsystem, the tests were run on a single frame DS8870 dedicated for performance testing, with dedicated links between the z15 and the DS8870.

The PPRC configuration has 4 dedicated 16Gb Fibre channels for the PPRC-specific traffic. All non-PPRC specific traffic is spread across 4 dedicated 16Gb Fibre channels.

This means that all I/O is written to the non-PPRC specific channels except for the mirrored active logs in the PPRC configuration.

In the zHyperWrite configuration, both primary and secondary copies of each active log are written using the non-PPRC fibre channels.

In all configurations, zHPF (z High Performance FICON) was used.

Note that all I/O is over 16Gb Fibre channels.

The measurements ran for a number of SMF intervals and were designed to run the MQ logger task at the maximum sustainable throughput rate.

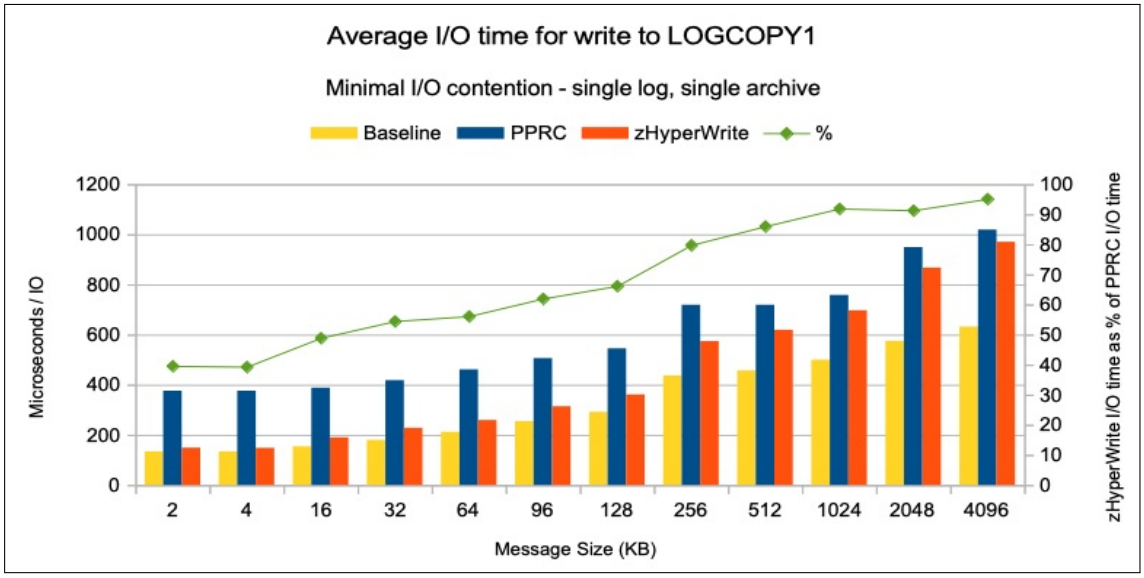
Reduced I/O time

When a persistent workload is throughput constrained, it can be due to the MQ logger task running at capacity. The overall throughput may vary depending on message size(s), and to achieve a higher throughput there are several options:

- Split the workload over multiple queue managers.
- Reduce the I/O response time.

The following chart uses the MQ Statistics data to plot the average I/O times for the logger task for each of the three specified configurations:

Chart: Average I/O time for write to LOGCOPY1



For messages up to 96KB, zHyperWrite is taking less than 60% of the time of the PPRC I/O requests.

Once the messages exceed 96KB, the I/O times get closer, but in this configuration where the I/O subsystem is not constrained, zHyperWrite shows I/O times at least 4% less for all message sizes when compared with the PPRC configuration.

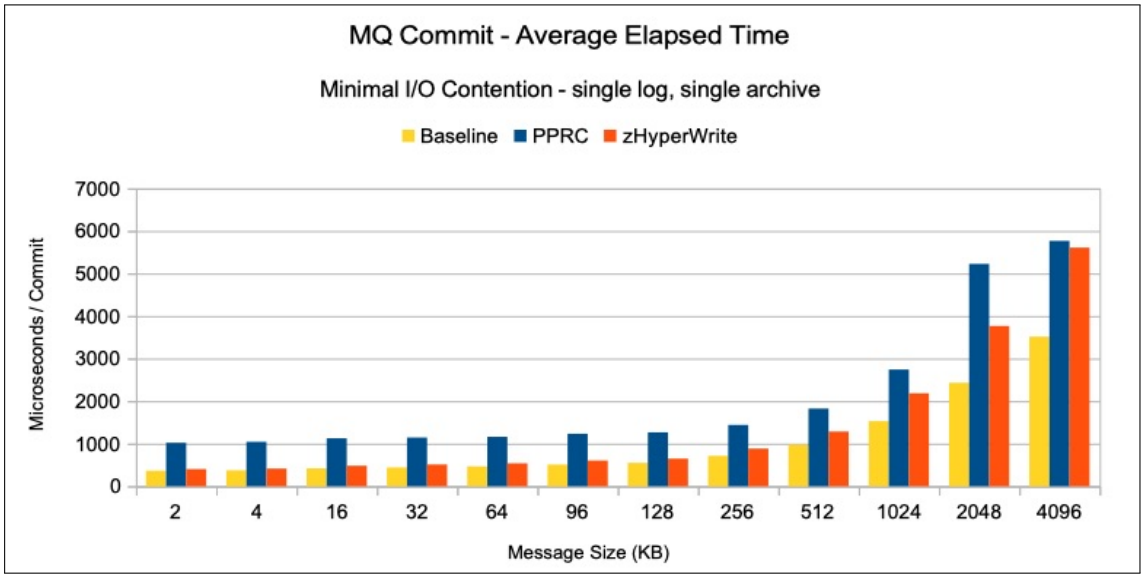
Reduced elapsed time for MQ commit

In transactions, the wait time for the MQ log write I/O, typically at commit time, often makes up a significant proportion of the transaction latency, particularly when disk replication is enabled.

zHyperWrite is designed to reduce the latency of the log writes in a synchronous peer-to-peer remote copy (PPRC) environment. With zHyperWrite enabled, MQ triggers a log write I/O to the secondary disk subsystem in parallel to the log write I/O to the primary disk subsystem. By overlapping the two I/Os, which are run serially without zHyperWrite enabled, a significant reduction in MQ log write I/O write time is possible.

The following chart shows the average elapsed time of the MQ commit for a range of message sizes.

Chart: Average elapsed time of MQ commit



For messages up to 128KB, the average time of the commit in the zHyperWrite configuration is less than 50% of the PPRC Metro Mirror configuration - and largely comparable with the baseline (non-mirrored) configuration.

The reduction in average commit time diminishes once the message size exceeds 128KB until, in the worst case at 4MB messages, the average commit time is still 4% less than the PPRC configuration. Note that with 4MB messages on our system when using single logs, we observed 6% of I/O impacted by Disk Fast Write Bypass (DFWBP), which is discussed further in the "Impact of I/O limitations on dual active and dual archive logs" section of the report. When repeating the tests with archiving disabled, which reduces the load on the I/O further, we observed the average elapsed time for the MQ commit was a minimum of 12% lower when using zHyperWrite in the mirrored environment.

In some cases, another indirect benefit of zHyperWrite can be observed. The reduced log write wait times as a result of having zHyperWrite enabled, can lead to reductions in other types of contention, which in turn can result in more CPU time savings because MQ has fewer contentions, both suspend and resume, to manage.

Note that for most messages, the amount of data logged at put time far exceeds the amount of data logged at get time. See MP16 "Capacity Planning and Tuning Guide", section "How much log space does my message use?" for guidance on how much data is logged.

Improved sustainable log rate

The chart in this section shows the rate in MB per second that the single active log was able to sustain on our systems.

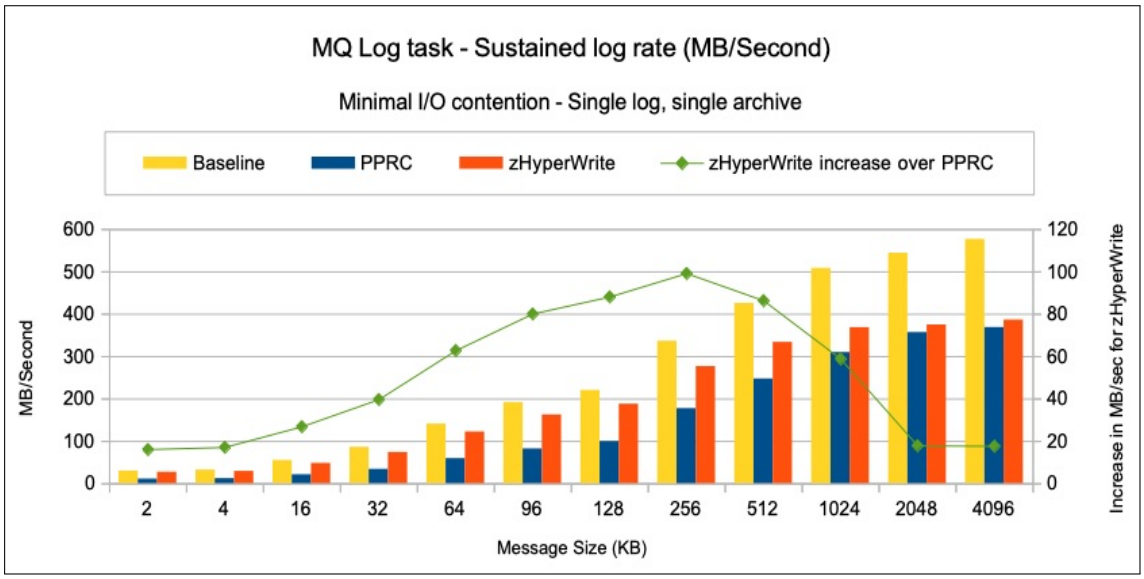
In all cases, the queue manager is configured with single logs and single archives, so for every MB put by the application, there are writes of between 2MB (for the baseline) to 3MB (for PPRC or zHyperWrite) in total volume to the disks i.e.:

- 1 MB to active log (1 MB per log)
- 1 MB to archives - once the test is running, the queue manager is driving the archive process constantly.
- 1 MB to the mirrored logs (1 MB per log).

What this means is that when the charts show a logging rate of 250 MB / second, there is approximately 500 MB / second written to the I/O subsystem in the baseline configuration and 750 MB / second in the PPRC and zHyperWrite configuration.

In the majority of these measurements, the I/O subsystem is not being driven hard enough to fully utilise the available disk cache - although the 2 MB and 4MB workloads do begin to see a small impact from saturating the disk cache. The section "Impact of I/O limitations on dual active and dual archive logs" demonstrates the implications of increased load on the I/O subsystem through the use of dual logs and dual archives, and how that impacts the sustained log rate.

Chart: MQ Log task - sustained log rate



Notes on chart:

The green line indicates the difference in MB/second that zHyperWrite offers over PPRC in our test environment.

For messages up to 128KB, zHyperWrite is achieving double the rate of the PPRC measurement.

The benefits of zHyperWrite are such that with messages of 256KB, we saw an additional 100 MB / second logged with a corresponding increase in transaction rate.

Impact to MQ queue manager costs

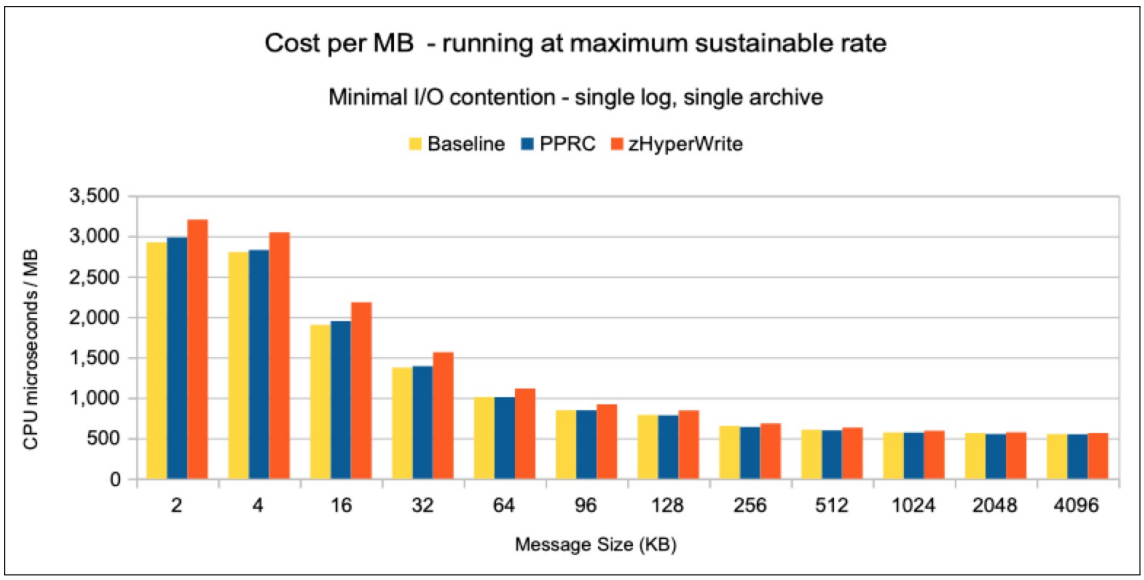
As previously mentioned, the zHyperWrite configuration may see increased SRB costs in the MQ MSTR address space.

There are a number of factors which can affect the actual scale of the impact, and it is difficult to predict exactly how much any particular workload might be affected. Some of the factors include:

- Message size.
- Utilisation of the MQ logger task.
- Amount of data written per I/O.

If we compare the cost per MB in the MQ MSTR address space when running as the maximum sustainable rate, we can plot the following chart.

Chart: Cost to MQ when running at maximum sustained rate



In this example, the maximum increase is using a 32KB workload, zHyperWrite is 12% more expensive than the PPRC configuration.

In these configurations with single active/archive logs, the cost increase is typically the order of 7.5%. In a dual active/archive log configuration, the increase is more typically 6%.

In the majority of these data points associated with smaller messages, the zHyperWrite configuration is logging at a significantly higher rate - which can increase the contention within the queue manager, which in turn can increase costs.

Using a message size where the log rate is similar should minimise any additional costs incurred from higher throughput. In these measurements, the 4MB workload achieves the most similar throughput.

Table: Comparing costs of 4MB workload

Configuration	Rate MB/Second	Average queue manager CPU (including SRB) CPU uSeconds/MB	Average queue manager SRB CPU uSeconds/MB
Baseline	**576**	550	305
PPRC	**368**	548	302
zHyperWrite	**386**	563	317

This example shows that the zHyperWrite configuration adds 15 CPU microseconds per MB to the PPRC configuration costs. This additional CPU is entirely SRB time and equates to an increase of 2.7% of the total MQ MSTR CPU or 4.9% in the total SRB used by the MQ MSTR address space.

These measurements were made from data gathered when the MQ logger task was fully utilised.

In measurements where the transaction rate was artificially constrained to achieve the same logging rate across configurations, essentially driving the logger task at a lower utilisation in the configurations such that the I/O subsystem was more responsive, the I/O times made a difference to the amount of data logged per I/O, which in turn affected the average cost per MB.

For example, with 4MB messages running at a log rate of 75MB per second on dual logs and dual archives:

- PPRC's typical SRB cost per MB was 0.4 CPU milliseconds per MB.
- zHyperWrite SRB cost per MB was 0.44 CPU milliseconds per MB. Part of the reason for the increase was that despite the messaging rate being identical, the amount of data written per I/O request dropped from 105 pages to 84 pages, resulting in extra I/Os and additional cost.

What this variability suggests is that predicting any increase in SRB is dependent on a number of factors, including the timing of the applications committing their messages. A good run, where more data per I/O is written may result in lower costs or minimal increase, but in cases where the faster response time resulted in less data per I/O saw MQ MSTR SRB costs increase up to 22%.

Impact of I/O limitations on dual active and dual archive logs

When running on MQ queue managers configured with dual active and dual archive logs on our single DS8870, the high throughput measurements - typically observed with 256KB messages or larger, saw increased I/O response times due to the saturation of non-volatile storage (NVS) in the I/O subsystem, which manifests itself in the form of increased Disk Fast Write Bypass (DFWBP).

What is Disk Fast Write Bypass?
In the 3990/3390 era, when NVS is full, the write I/O bypasses the NVS and the data is written directly to the disk module (DDM).

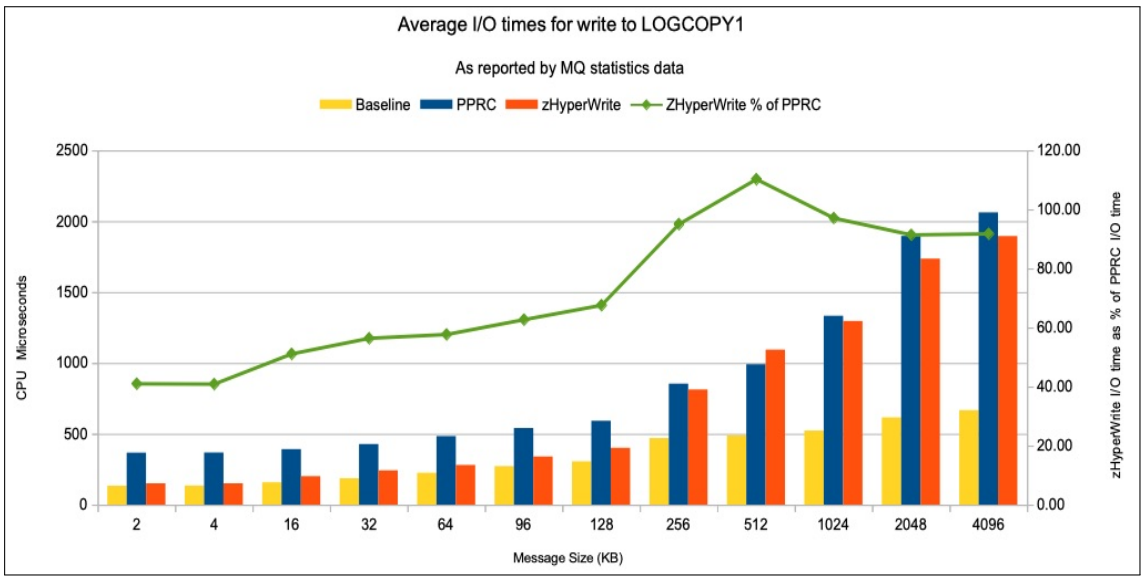
In DS8000, when the NVS is full, the write I/O is retried from the host until the NVS space becomes available. So DFW Bypass must be interpreted as DFW Retry for DS8000. If RMF shows that DFW Bypass divided by the total I/O rate is greater than 1%, that is an indication of NVS saturation.

Note that in our measurements, it was not uncommon to see DFW Bypass exceed 80% of the total I/O rate.

Impact on reduced I/O time
The following chart demonstrates the impact on the I/O time from increase DFWBP.

In our configuration, messages of 256KB and larger resulted in the I/O time for the zHyperWrite measurements being much closer to the PPRC measurements due to NVS saturation in excess of 20% - with the 4MB workload seeing DFWBP for in excess of 100% of requests.

Chart: Average I/O time for write to LOGCOPY1



For workloads that are not impacted by DFWBP, i.e. messages up to 128KB, zHyperWrite is taking between 32% and 60% less time per I/O than the equivalent PPRC request.

Impact on reduced I/O time
The chart in this section shows the rate in MB per second that *each* log copy was able to sustain on our system.

In all of these measurements, the queue manager is configured with dual active and dual archive logs, so for every MB put by the application, there are writes of between 4 MB (for baseline) to 6MB (for PPRC or zHyperWrite) in total volume to the disks, i.e.:

- 2 MB to active log (1 MB per log)

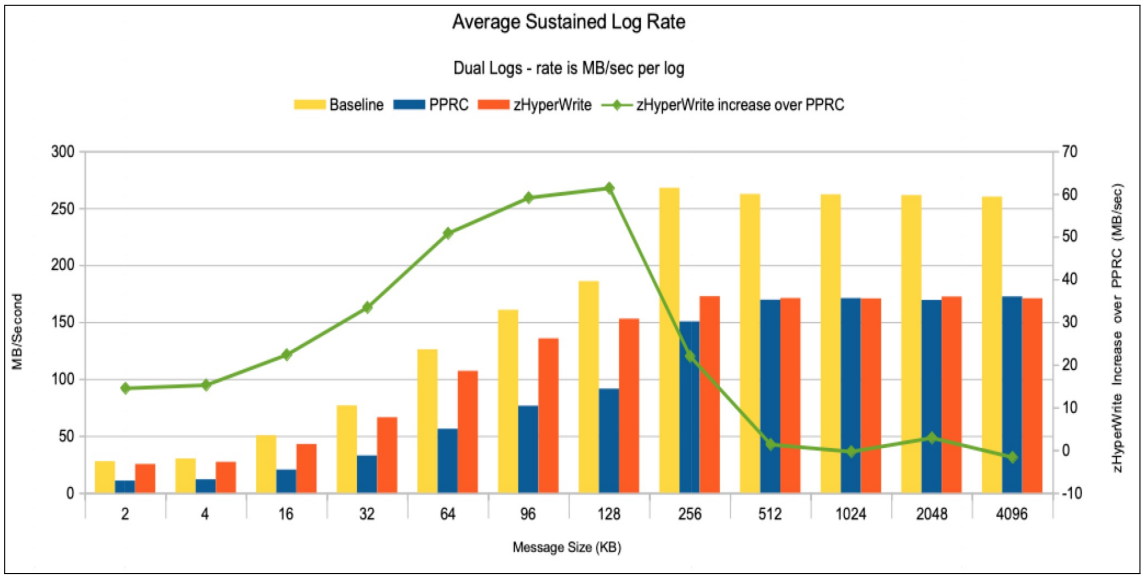
- 2 MB to archives - once the test is running, the queue manager is driving the archive process constantly.
- 2 MB to the mirrored logs (1 MB per log).

What this means is that when the charts are showing 250 MB/second, there is approximately 1GB/second written to the I/O subsystem in the baseline configuration and 1.5GB/second in the PPRC and zHyperWrite configurations.

As discussed previously, at the higher volumes, this causes waits for cache in the I/O subsystem, which is why the performance of PPRC and zHyperWrite becomes similar and the benefits of zHyperWrite are less obvious.

In the case of the PPRC measurement, the system has a separate fibre channel and therefore see less waits for cache - the I/O subsystem is still impacted by DFWBP but not to the same extent as zHyperWrite.

Chart: MQ Log task - sustained log rate with dual logs



Notes on chart:

The green line highlights the difference in MB/second that zHyperWrite offers over PPRC.

For messages of 64KB and less, the zHyperWrite configuration is achieving double the rate of the PPRC measurement.

As noted previously, the benefits of zHyperWrite grow with message size - in this case the actual difference (MB/second) in sustained log rate is peaking with messages sizes between 64KB to 256KB. Workloads with these messages using the zHyperWrite configuration were able to log 50-60 MB/second more than the PPRC equivalent measurement.

After this point, the benefits diminish, largely because the zHyperWrite configuration is being impacted more significantly by I/O cache waits (disk fast write bypass) than the PPRC configuration.

Summary of zHyperWrite benefits

Whilst the zHyperWrite measurements detailed in this report were run on a queue manager configured with single active and single archive logs, this is not the recommended configuration.

Single logs were used to demonstrate the impact of zHyperWrite, particularly when the I/O subsystem can be configured such that any additional I/O load does not cause NVS (Non-Volative Storage) saturation.

At zero distance replication, zHyperWrite can:

- Reduce the elapsed time from an MQ commit by up to 60%.
- Reduce the average I/O time when writing to the MQ active log(s) by up to 60%.
- Improve the maximum sustainable log rate by up to 2.4 times.

Despite the RMF CPU report indicating that the LPAR was less than 25% busy for the entire measurement period, disabling archiving improved the sustained log rate for the 4MB workload by up to 25%. This was due to periods where the number of CPUs available was insufficient for the number of tasks on the LPAR.

As the section "Impact of I/O limitations on dual active and dual archive logs" discusses, the additional load on our I/O subsystem when the queue manager was configured with dual active and dual archive logs, coupled with the mirroring of the active logs was enough to result in NVS saturation. This manifested itself in DFWBP (Disk Fast Write Bypass) and resulted in the performance benefits of zHyperWrite being less distinct.

Chapter 5

Data set encryption

Data set encryption (DSE) was originally implemented in z/OS v2r2, and initial MQ support in version 8.0 was limited to archive logs and BSDS.

In MQ 9.2, support for data set encryption is enabled for active logs, page set and shared message data sets (SMDS).

Table: MQ data set type and encryption support statement

Data set	Pre-9.2	9.2
BSDS	Supported	Supported
Sequential	Supported	Supported
Page set 0	No - Abend 5C6-00C91400	Supported
Page sets 1-99	No MQRC 2193 "Pageset error"	Supported
Active logs	No - Abend 5C6-00E80084	Supported
Archive logs	Yes - V8 onwards	Supported
SMDS	No SMDS marked avail(error)	Supported

Notes on table:

- Data held in buffer pool and SMDS buffers is not encrypted. The use of appropriate AMS policies would ensure encryption of the data held in the buffer pool or SMDS.
- Prior to MQ 9.2, applying data set encryption to SMDS could result in unexpected behaviour depending on the offload rules and potentially how full the structure gets. For example if the SMDS status "avail(error)" is not noticed at the time the message is logged, an application error may not occur until the MQPUT of a message that requires offload is attempted. If the application only uses small messages, that offload may not occur until the CF structure is 80% full, which could be a significant time between the SMDS error being logged and the MQPUTs failing as a result.

Why use data set encryption

z/OS data set encryption provides enhanced data protection for z/OS data sets, giving clients the ability to encrypt all of the data associated with entire applications and databases, without the need to make application changes and without impacting SLAs.

Additional design advantages provided by z/OS data set encryption are:

- Uses CPACF acceleration in collaboration with Crypto Express for protected key cryptography, enabling cryptographic operations to be hardware accelerated while ensuring that key material is not visible in the clear to the OS, Hypervisor or application.
- Encrypt data by policy in a way that is aligned with clients' current access control mechanisms, offering a simplified configuration experience.
- Encrypts at-rest data in bulk, performing efficiently at speed and for low-cost.
- Allows data to remain encrypted with keys managed on IBM Z during replication, backup, and migration.
- Can be configured such that encryption keys are owned and managed by a logical organisational environment, providing cryptographic separation between environments.
- Reduce the risks associated with undiscovered or mis-classified sensitive data.
- Encrypting all of the data associated with an application or database can simplify and reduce the cost of compliance.

The data set encryption feature allows AES encryption of data contained in named data sets using ICSF and RACF, so preventing visibility of sensitive data from systems administrators with a legitimate need to manage those data sets.

Data set encryption with the MQ queue manager

Encryption of data is not free, and as such, consideration should be given as to whether your system has sufficient capacity to support encrypted MQ data sets without impacting the performance of your workloads.

The following sections discuss the additional cost from encrypting MQ data sets based on observations on our dedicated performance systems. To try to simplify the impact, the costs are discussed in three configurations:

1. Active and archive log data sets.
2. Page sets.
3. Shared message data sets.

Typically we might expect that if you are planning on encrypting any of the MQ data sets, you would encrypt **all** MQ data sets, i.e. page set, active and archive logs, SMDS and BSDS.

BSDS might be regarded as less important to encrypt as it contains systems rather than application data, but this may depend on your sites encryption policy.

Despite MQ archive logs supporting data set encryption since IBM MQ for z/OS version 8.0, active and archive logging are closely coupled, and so we offer a section on encrypting all of your MQ log data sets.

The I/O to MQ page set is somewhat different and will depend upon the nature of the workloads using the queue manager. As such, the section on encrypting page set is kept separately from MQ logs and SMDS.

Active and Archive log encryption

Before discussing the impact of encrypting the MQ active and archive logs, consider what happens when logging occurs where the MQ queue manager has dual active and dual archive logs, all of which are encrypted.

- Persistent message workload causes MQ queue manager to **encrypt**+write to active log copy 1 and **encrypt**+write to active log copy 2 data sets.
- At end of current active log:
  - Queue manager reports CSQJ002I "END OF ACTIVE LOG DATA SET".
  - Current active log(s) switches to next active log(s).
  - Checkpoint starts for all buffer pools.
  - Archive task started, processing the recently filled active log:
    - Select either log copy 1 or 2 to provide the data to archive.
    - Read+**Decrypt** chosen active log copy - 1 page at a time.
    - **Encrypt**+write to active log copy 1.
    - **Encrypt**+write to active log copy 2.

The point of this is to show that dual logging and/or dual archiving causes multiple encrypt calls as well as decrypting the data read from the active logs.

With dual logging and dual archiving all protected with data set encryption, the queue manager is encrypting the data 4 times, plus decrypting the data once, for a total of 5 encrypt/decrypt calls for each page logged.

Table: Cost of data set encryption on MQ active and archive logs

Log type	CPU microseconds per 4KB page	CPU microseconds per MB
Archive logs	+0.6	+154
Active logs	+0.7	+180

Notes on table:

- Active logs are accessed via Media Manager, which has additional overheads.
- Costs shown are based on the average across a range of message sizes:
  - Archive logs: range between 0.25 to 1 microseconds per 4KB page.
  - Active logs: range between 0.6 to 0.9 microseconds per 4KB page.

Since we know the impact of data set encryption to both active and archive logging, we can apply this to our queue manager using the following example.

Example: MQ configured with dual active and archive logs, with 1GB log data sets.

Log type	Cost / MB	MB in active log	Factor	Cost (Cost * MB * Factor)
Archive logs	154	1024	2	315 CPU milliseconds
Active logs	180	1024	3	550 CPU milliseconds

Notes on table:

- Factor of 2 used for the separate **encrypt** and write to each copy of the archive log.
- Factor of 3 used for the separate **encrypt** and write to each copy of the active log plus the read and **decrypt** of the log for the archiving task.

In this example, for each 1GB of MQ workload logged, there would be approximately 0.86 CPU seconds *additional* cost in the MQ MSTR address space as a result of enabling data set encryption for both active and archive logs.

Impact of data set encryption to active and archive logs in a workload

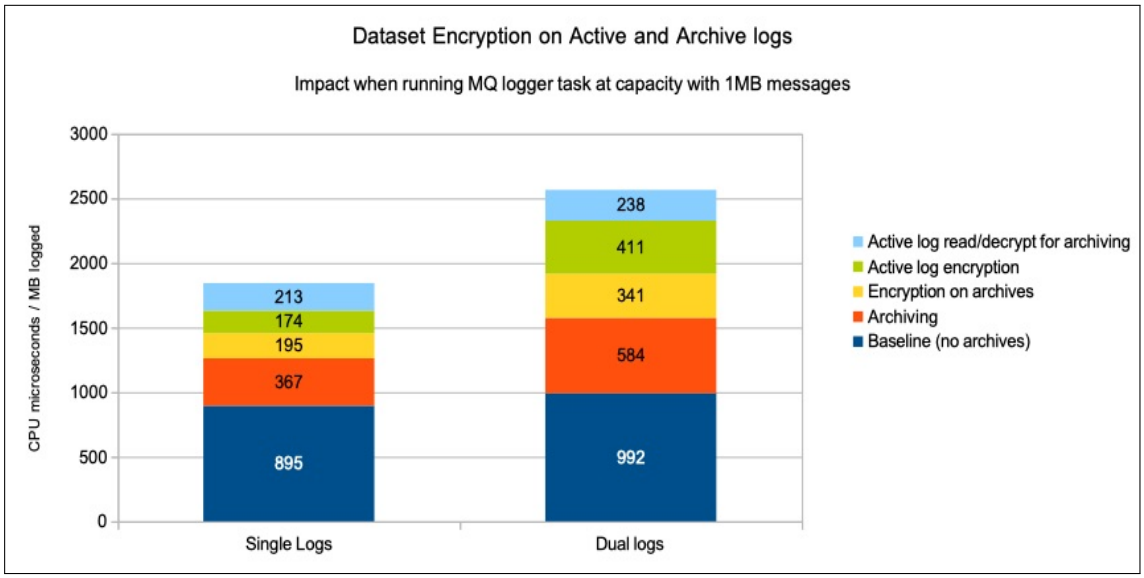
The earlier information identifies the additional cost to the MQ queue manager as a result of encrypting the active and archive logs, but how does that appear if we put the costs into the context of a workload?

In this example, the workload is driving batch applications to achieve a fixed sustainable rate using 1MB persistent messages - the log rate achieved is 270 MB / second per log copy.

The workload is simple - 3 batch applications that put, commit, get, commit 1MB persistent messages for a set time. The rate is a known value and the MQ MSTR costs can be determined from the RMF Workload Activity report.

For this particular workload, the queues are always low depth and there is minimal activity at checkpoint - therefore there would be negligible impact from encrypting the page set, and thus page set encryption costs are not included.

Chart: Cost of processing 1MB of data logged in the MQ MSTR address space



Notes on chart:

1. The cost in the MQ MSTR address space for logging 1MB of data when configured with dual logs was 992 CPU microseconds.
2. Enabling archiving added an additional 584 CPU microseconds per MB, for a total of 1576 microseconds.
3. Applying data set encryption to the archives added a further 341 CPU microseconds, for a total of 1917 microseconds.
    - If data set encryptions was subsequently applied to the active logs, a further 649 CPU microseconds would be added, giving a total of 2566 CPU microseconds per MB. This additional 649 microseconds includes the active log encryption plus the active log read/decrypt for archiving.
4. If data set encryption was applied only to active logging but archiving is still enabled, the cost would be 992 + 584 + 411 + 238 for a total of 2225 CPU microseconds.

Page set encryption

The cost of encrypting MQ page sets is slightly more complicated than encrypting MQ logs due to types of I/O performed.

The program MQSMF, available in supportPac MP1B "Interpreting accounting and statistics data", provides a number of page set related reports, but the one of interest here is PSET. Note that only selected data is shown in the following sample:

```
PS01 BP   1, Pages   1040334, Size 4063MB, ...  Pageset is encrypted
     Pages written in checkpoint       2918219
     Pages written not in checkpoint     10289

PS01 Type :I/O requests.     Pages, Avg I/O time, pages per I/O
PS01 Write:      182403,    2918331,        1968,          16.0
PS01 IMW  :       10177,      10177,         274,           1.0
PS01 GET  :           1,          1,         378,           1.0
```

As highlighted in the sample report, there are three types of I/O performed on MQ page sets, namely WRITE, IMW (immediate write) and GET:

- **GETs** are performed when MQ determines it necessary to read (and decrypt) from page set. Reads are performed one page per I/O request.
- **IMWs** (Immediate writes) occur when the buffer pool reaches 95% full and the data is moved from buffer pool to page set synchronously. Immediate writes are performed 1 page per I/O request. This is an indication that the buffer pool is not sufficiently large for optimal performance.
- **Writes** are performed at 2 points - at checkpoint and when the buffer pool reaches 85% full (also termed "written not in checkpoint"). In each instance, each I/O request will write up to 16 pages.

In terms of the additional cost of data set encryption on page set I/O:

- **GET** +0.7 microseconds / page read.
- **IMW** +0.7 microseconds / page read.
- **WRITE** +2.6 microseconds / page (+41.6 microseconds per I/O).

We can use these numbers in conjunction with the output from the PSET report to estimate the cost impact from applying data set encryption to MQ page sets e.g.

- **GET**: 1 request for an additional 0.7 microseconds, i.e. not a discernible impact from page set encryption.
- **IMW**: 10,177 requests for 10,177 pages - an additional cost of 7124 CPU microseconds.
- **WRITE**: 182,403 requests for 2,918,219 pages - which we would predict at costing an additional 7.59 CPU seconds.

Shared message data set encryption

Typically for messages that can be stored in their entirety in the Coupling Facility, there is no significant impact to the transaction cost from encrypting the SMDS. This is true even when the CF structure is encrypted.

Due to the way that SMDS works, the majority of the encryption costs will be incurred by the application performing the MQPUT, rather than the queue manager address space.

Decrypting the data held on SMDS is charged to the queue manager address space and can partially be offset by reduced cost in the application issuing the MQGET. This is discussed further in "Why are unencrypted gets more expensive than encrypted?"

Furthermore, when accessing messages stored in the local queue manager's SMDS, sufficient DSBUFS can mean the message can be accessed from buffer rather than needing to read and decrypt the data physically stored on the data set.

When data is read from a remote queue managers' SMDS, the decryption costs are charged to the local queue manager performing the read of the SMDS.

Message persistence does not impact the cost of encryption or decryption of SMDS data.

MQPUT to SMDS

The cost of an MQPUT of a message to a shared queue where the message is offloaded to SMDS is largely attributed to the application address space. For non-persistent messages the cost in the queue manager address space is minimal compared to the application cost.

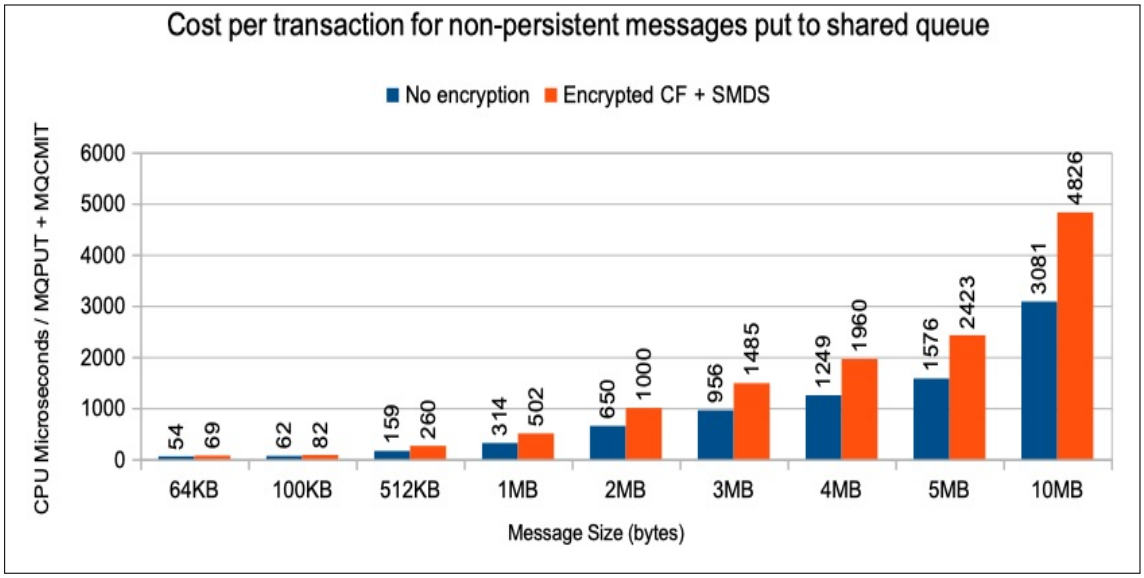
As a result of encrypting the SMDS, putting messages to shared queues where the message is written to SMDS does result in increased cost to the putting application - whether a local application or the channel initiator if the message arrives via an MQ channel.

The following chart compares the cost of non-persistent messages put to a shared queue for 2 configurations:

1. Neither CF nor SMDS are encrypted.
2. Both CF and SMDS are encrypted.

The application performing the MQPUT generates the message once, and performs multiple iterations of MQPUT and MQCMIT before ending. There is no business logic in the program so the costs reported are largely equivalent to the data reported by MQ Accounting class(3).

Chart: Compare cost of MQPUT to Shared Message Data Set



Note: The increase in cost from encrypting the SMDS equates to approximately 0.7 CPU microseconds per 4KB page on IBM z15 regardless of message persistence.

MQGET - When the messages are read from SMDS buffers

As mentioned earlier, when sufficient DSBUFS are available that the messages can be gotten from SMDS buffers, the cost of the MQGET remains the same regardless of whether the data set is encrypted.

Note: *Configuring more DSBUFS to minimize the effect of decrypting data can have a negative effect - due to the time taken to locate the next available buffer (oldest). For example puts and gets to structures defined with DSBUF(3000) were significantly more expensive than messages put to structures defined with DSBUF(10-200).*

MQGET - When the messages are read from local SMDS

The increase in cost from decrypting the message on the SMDS is approximately 0.7 CPU microseconds per 4KB page, regardless of message persistence.

In the queue manager address space there was an increase of approximately 0.8 CPU microseconds per 4KB page in SRB usage. However some of this cost increase was offset by a reduction in the application address space equivalent of 0.1 CPU microseconds per 4KB page. This is described in further detail in Why are unencrypted gets more expensive than encrypted?

MQGET - When the messages are read from remote SMDS

The increase in cost from decrypting the message on the SMDS is approximately 0.8 CPU microseconds regardless of persistence.

In the queue manager address space there was an increase of approximately 0.9 CPU microseconds per 4KB page - as with reads from local encrypted SMDS'. However, as with reads from the local SMDS, some of this cost increase was offset by a reduction in the application address space equivalent of 0.1 CPU microseconds per 4KB page.

When reading from a remote queue managers' SMDS, there are slightly reduced costs charged to the local queue manager, but this is offset by a small increase in the CPU used by the remote queue manager.

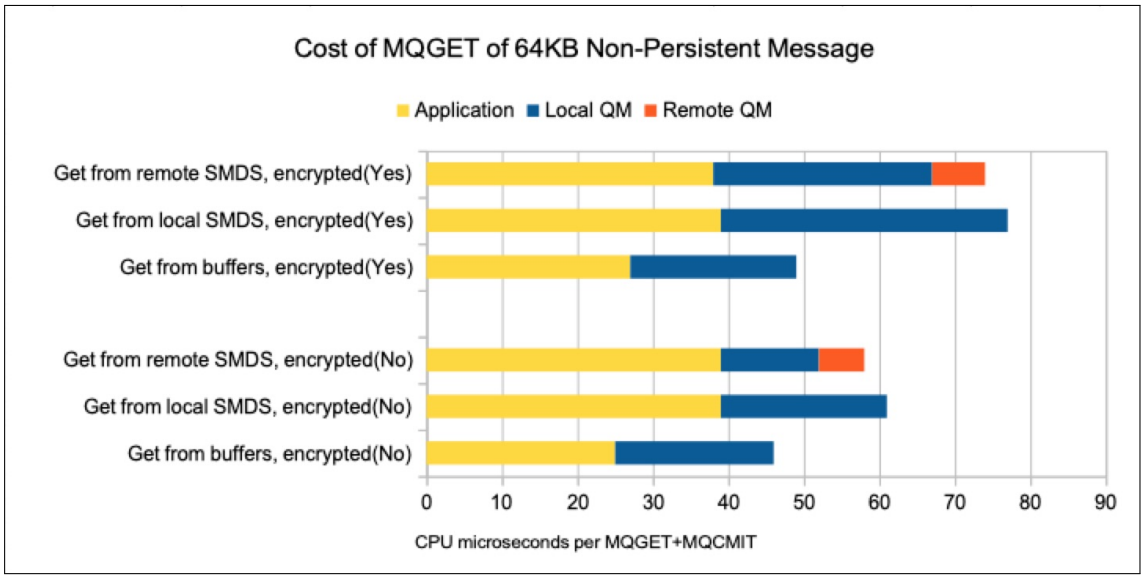
This CPU usage in the remote queue manager usage is due to the remote queue manager deleting the message from its own SMDS and is not affected by data set encryption status.

Our measurements suggest the total cost of MQGET from a remote queue managers' SMDS is within 5% of the cost of an MQGET from a local queue managers' SMDS.

Comparing the cost of MQGETs from shared queue

The following charts offer comparisons of MQGET costs, by address space, for the configurations of MQGETs described previously, for three messages sizes - 64KB, 1MB and 10MB.

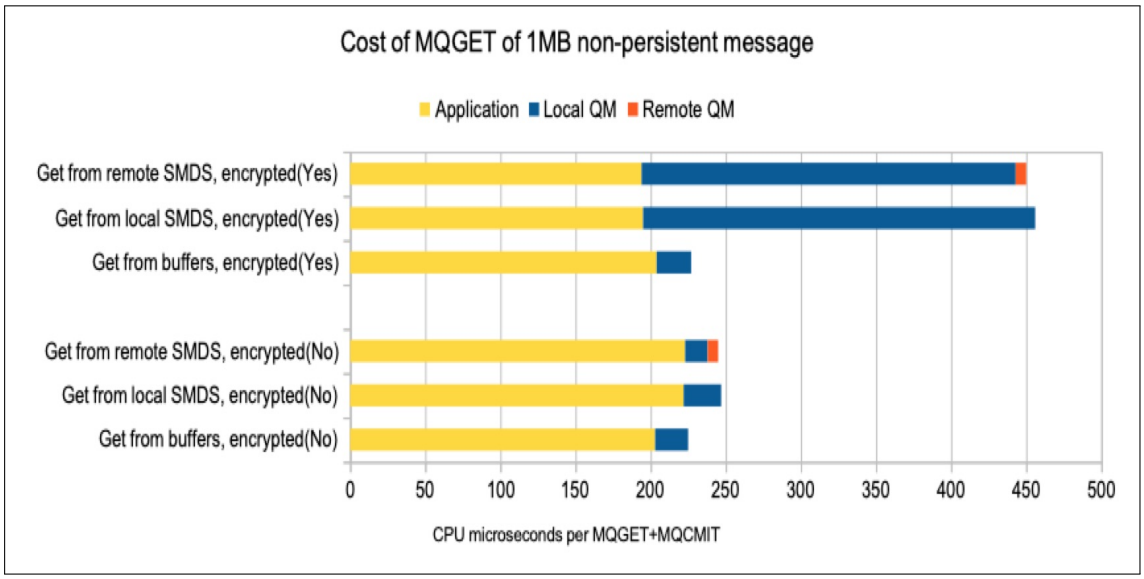
Chart: Compare cost of MQGET of 64KB non-persistent shared queue message



Notes on chart:

- The cost of decrypting the message held on SMDS is charged to the queue manager address space.
- The total cost of MQGET from a remote SMDS is similar to the cost of an MQGET from a local SMDS.
- When a message can be got from SMDS buffers, the encryption status of the SMDS does not affect the cost of the MQGET.
- The cost to the application address space of the MQGET is slightly less when accessing encrypted data than when accessing unencrypted data.

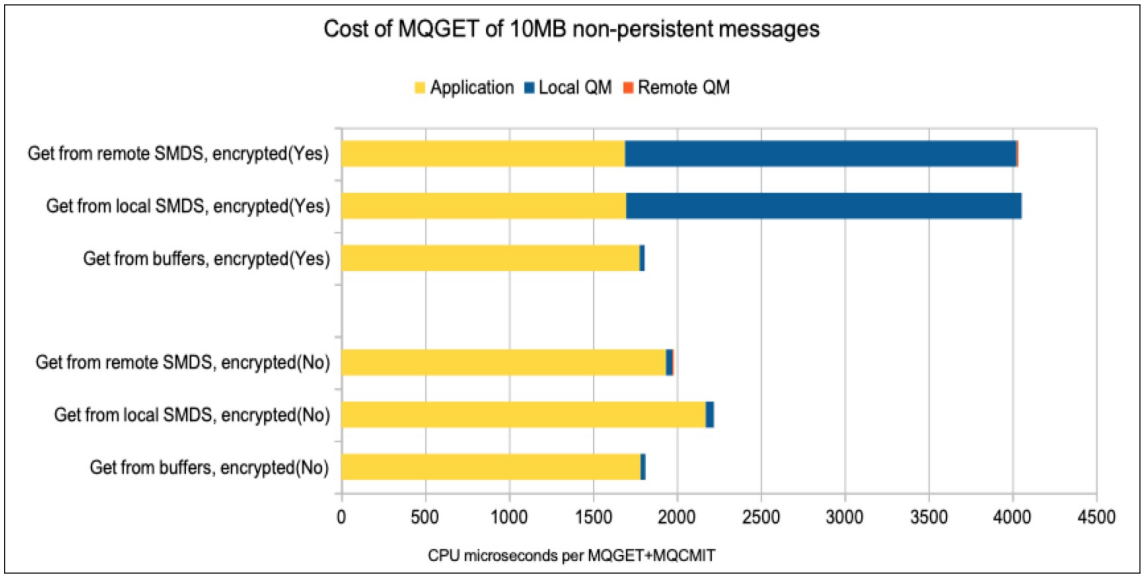
Chart: Compare cost of MQGET of 1MB non-persistent shared queue message



Notes on chart:

- The total cost of the MQGET from a remote SMDS is similar to the cost of the MQGET from a local SMDS - within 5%.
- With these 1MB messages, the application cost, i.e. the reported cost of the MQGET as per MQ accounting class(3), is 13% lower when accessing encrypted data.

Chart: Compare cost of MQGET of 10MB non-persistent shared queue message



Notes on chart:

- With these 10MB messages, the application cost, i.e. the reported cost of the MQGET as per MQ accounting class(3), is 22% lower when accessing encrypted data. This does not take into account the increase in queue manager address space costs from decrypting the data.

Why are unencrypted gets more expensive than encrypted?

According to the MQ Accounting class(3) data and as can be seen in the "MQGET of 1MB" and "MQGET of 10MB" message charts, it appears that the MQGET costs are lower when getting data from encrypted SMDS than when accessing data from unencrypted SMDS.

For example:

Table: Accounting class(3) data for MQGET of 10MB messages

	Unencrypted	Encrypted
Average CPU time	1923 CPU microseconds	1683 CPU microseconds

In this example, the savings from encryption equate to approximately 0.1 CPU microseconds per 4KB page.

This does not take into account the increased cost in the queue manager address space from decrypting the message.

The increase in queue manager cost from MQGETs of message in encrypted SMDS is in SRB and there are no MQ-based statistics to report details of this cost.

Summary of data set encryption costs with the MQ queue manager

As has been discussed in this data set encryption section, the cost of encrypting MQ data sets does not come at zero cost.

The following provides a summary of the CPU overhead from encrypting each of the types of MQ data set, but it is worth considering the impact as a whole, i.e. persistent shared queue messages will see the costs rise due to encrypted active logs, encrypted archive logs as well as encrypted shared message data sets.

Active and archive log data sets costs when operating with data set encryption can be summarised as follows:

- Archive logs cost increases with data set encryption by approximately 0.6 CPU microseconds per 4KB page written.
- Active logs cost increases with data set encryption by approximately 0.7 CPU microseconds per 4KB page read / written.
- Remember to factor in whether running with single or dual logging / archiving.
- Remember that when using encrypted active logs, the data must be decrypted prior to archiving - even if the archives are encrypted too.

Page set costs when operating an encrypted data set can be summarised as follows:

- **GET** +0.7 microseconds / page read.
- **IMW** +0.7 microseconds / page read.
- **WRITE** +2.6 microseconds / page (+41.6 microseconds per I/O).

Shared message data set costs when operating an encrypted data set can be summarised as follows:

- Increased MQPUT costs of approximately 0.7 CPU microseconds per 4KB page.
- Increased MQGET costs of approximately 0.7 CPU microsecond per 4KB page when the data is read from the local queue managers shared message data set.
- Increased MQGET costs of approximately 0.8 CPU microsecond per 4KB page when the data is read from a remote queue managers shared message data set.

The impact of encrypted SMDS can be mitigated for MQGETs when the queues are sufficiently low in depth such that messages can be gotten from SMDS' buffers. Note that it is not recommended to significantly increase the number of SMDS buffers to avoid gets from encrypted SMDS as this can increase the cost of both MQPUTs and MQGETs.

Final observations on data set encryption with MQ:

- The cost per page when writing multiple pages per I/O request is higher than the cost of a single page I/O request.
- Costs may vary depending on how busy your system is - lightly loaded systems or tasks, such as the MQ logger, may see a higher cost per page than reported in this section.
- Data set encryption costs were generally similar on z14 and z15 for our measurements, as they used AES encryption which has been optimised on CPACF. Costs may be higher on earlier generations of IBM z hardware.

- Consider that in CPU constrained systems that the additional cost from encryption may impact existing workload characteristics, resulting in more data written at checkpoint or needing to be read from page set.
- Page set immediate writes and gets can be minimised with sufficiently large buffer pools. *Over-sized buffer pools can minimise immediate writes to page set.*
    - Over-sized buffer pools are buffer pools that are sized at 105% of the corresponding page set.
        - By setting the buffer pool to 105% of the size of the page set, you can avoid additional I/O's for immediate writes.
        - When multiple page sets map to a single buffer pool, over-sizing the buffer pool may not have the desired impact of avoiding immediate writes.

Chapter 6

TLS 1.3 support

MQ 9.2 introduces support for Transport Layer Security (TLS) 1.3 protocols with the additional benefit of aliases.

The ability to specify a TLS alias allows the queue manager and its remote partner, whether another queue manager or a client, to negotiate the ***most secure cipher*** that both partners support.

The determination of the most secure cipher is decided by System SSL on z/OS and GSKit on distributed, and by default the order is the same. The order may be different if the user sets the allowed list on at least one of the sides, in which case this can affect the cipher chosen in the determination process.

The available aliases at MQ 9.2 are as follows (in most to least secure order):

- ANY_TLS13_OR_HIGHER
- ANY_TLS13
- ANY_TLS12_OR_HIGHER
- ANY_TLS12
- ANY - *This potentially allows SSL 3 ciphers to be selected and should be used with caution.*

In order to use an alias, the MQ channel attribute "SSLCIPH" would be set to the value of the preferred alias.

In this chapter, we include information about TLS 1.2 cipher performance as those ciphers may be chosen as a result of using one of the available aliases.

Why use TLS 1.3?

The report Enabling TLS 1.3 in System SSL Applications explains in detail the benefits of the TLS 1.3 protocol and what follows is an extract of that report:

The Transport Layer Security (TLS) 1.3 protocol is a major rewrite of prior TLS protocol standards. After being in the works for many years in the Internet Engineering Task Force (IETF) TLS working group, TLS 1.3 became a formal standard in August 2018 when RFC 8446 was published. In z/OS 2.4, System SSL added support for the TLS 1.3 protocol in order for z/OS applications to take advantage of the security updates.

TLS 1.3 includes the following updates:

- *All handshake messages after the initial client and server handshake messages are now encrypted. In prior protocols, messages were not encrypted until after the final handshake messages were exchanged between the client and server.*
- *Encrypted handshake messages are presented as payload messages and must be decrypted in order to determine whether the message is a handshake, payload or alert message.*
- *The RSA key exchange is no longer supported. It was replaced with Elliptic Curve Diffie-Hellman Ephemeral (ECDHE), which provides forward secrecy.*
- *Prior to TLS 1.3, the negotiated key exchange was part of the cipher suite. In TLS 1.3, the negotiated key exchange is no longer part of the cipher suite and is negotiated separately.*

TLS 1.3 requires key sharing and MQ uses the GSK_CLIENT_TLS_KEY_SHARES and GSK_SERVER_TLS_KEY_SHARES options, which will generate public/private key pairs for each key share group, which can be computationally expensive and might impact performance.

MQ's current implementation of TLS 1.3 protocol support means that all of the key share groups are supported, so whether a TLS 1.3 protocol is explicitly specified, or an alias is specified that allows a TLS 1.3 cipher to be negotiated, the channel start costs *may* be significantly higher than if TLS 1.2 protocols are specified. On IBM z15, ICSF FMID HCR77D1 reduces the cost of the key share handshake to the extent that TLS 1.3 ciphers are no longer significantly more expensive than TLS 1.2 ciphers.

Using an alias in the channel definition will only affect the channel start rate and cost, i.e. the rate and cost of the channel stop, secret key negotiation and encryption/decryption of data will depend on the performance of the negotiated cipher.

The impact of the alias on channel start performance can be viewed in section "Starting TLS channels using Aliases".

How do I know what ciphers are in use?

To determine what cipher has been **requested** on a particular MQ channel, use:

```
DISPLAY CHANNEL(VTS1_VTS2_0001) SSLCIPH
```

In our configuration this returns:

```
CHANNEL(VTS1_VTS2_0001)
CHLTYPE(SDR)
QSGDISP(QMGR)
SSLCIPH(ANY_TLS13)
```

In this example, we can see that the ANY_TLS13 alias has been specified. To determine what cipher has been **negotiated** on the started MQ channel use:

```
DISPLAY CHSTATUS(VTS1_VTS2_0001) SSLCIPH
```

In our configuration this returns:

```
CHSTATUS(VTS1_VTS2_0001)
CHLDISP(PRIVATE)
XMITQ(VTS2_XMIT_0001)
CONNAME(10.20.36.103)
CURRENT
CHLTYPE(SDR)
STATUS(RUNNING)
SUBSTATE(MQGET)
STOPREQ(NO)
RQMNAME(VTS2)
SSLCIPH(TLS_AES_256_GCM_SHA384)
```

Which cipher should I choose?

From a security perspective, you may want the most secure cipher, but be aware that not all ciphers perform the same.

There are 3 areas where the cipher used can affect performance:

- Starting and stopping of the channel. If this occurs frequently the cost may be a factor in choosing the cipher.
- Key (re-)negotiation time - the impact of this is dependent upon the value of SSLRKEYC and the volume of data flowing over the MQ channel. If small volumes of data flow over the MQ channel, or the SSLRKEYC is configured such that there are few key re-negotiations, the negotiation cost may not be a factor.
    - Note: Since TLS 1.3 has key re-negotiation as part of the protocol, reaching the SSLRKEYC-threshold does not have the same effect when using a TLS 1.3 cipher compared to a TLS 1.2 cipher. There is cost associated with SSLRKEYC(non-zero) even with TLS 1.3 ciphers, and those are reported in the key (re-)negotiation section. For best performance with TLS 1.3 ciphers, specify SSLRKEYC(0).
- Cost of encryption and decryption of data - there are ciphers that do not have hardware support for encryption and decryption of data, which can impact the cost and the time to encrypt the message data.

Note: In all charts and tables in this section, the ciphers are listed in most to least secure order as returned by System SSL.

Starting TLS channels

The rate at which MQ is able to start channels is impacted by the cipher.

The following table gives an indication of the cost and rate at which MQ is able to start channels with the named ciphers.

The ciphers are shown in the order that System SSL returns as the most to least secure.

TLS	hex cipher	CipherName	Cost CPU ms	Channels started / second
TLS 1.3	1303	TLS_CHACHA20_POLY1305_SHA256	3.87	100.64
	1302	TLS_AES_256_GCM_SHA384	3.34	105.08
	1301	TLS_AES_128_GCM_SHA256	3.30	105.06
TLS 1.2	009D	TLS_RSA_WITH_AES_256_GCM_SHA384	2.31	135.75
	C030	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	3.94	91.73
	003D	TLS_RSA_WITH_AES_256_CBC_SHA256	2.30	134.59
	C024	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	3.81	87.81
	C028	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	3.91	92.86
	009C	TLS_RSA_WITH_AES_128_GCM_SHA256	2.34	135.39
	C02F	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	3.93	91.2
	003C	TLS_RSA_WITH_AES_128_CBC_SHA256	2.29	141.12
	C023	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	3.80	88.82
	C027	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	3.89	93.6
	003B	TLS_RSA_WITH_NULL_SHA256	2.31	135.75

As mentioned previously, TLS 1.3 ciphers may be more expensive, and as a result take longer to start, than TLS 1.2 ciphers and this is due to the TLS 1.3 requirement to support key shares. In the table above, and the following 2 charts, ICSF FMID HCR77D1 has been applied to the performance system, and this results in comparable performance between TLS 1.2 and TLS 1.3 ciphers at channel start.

In all cases, part of the channel start cost has been offloaded to CryptoExpress hardware (co-processor / accelerator).

The data in the table is shown in the following 2 charts.

Chart: TLS Channel start rate

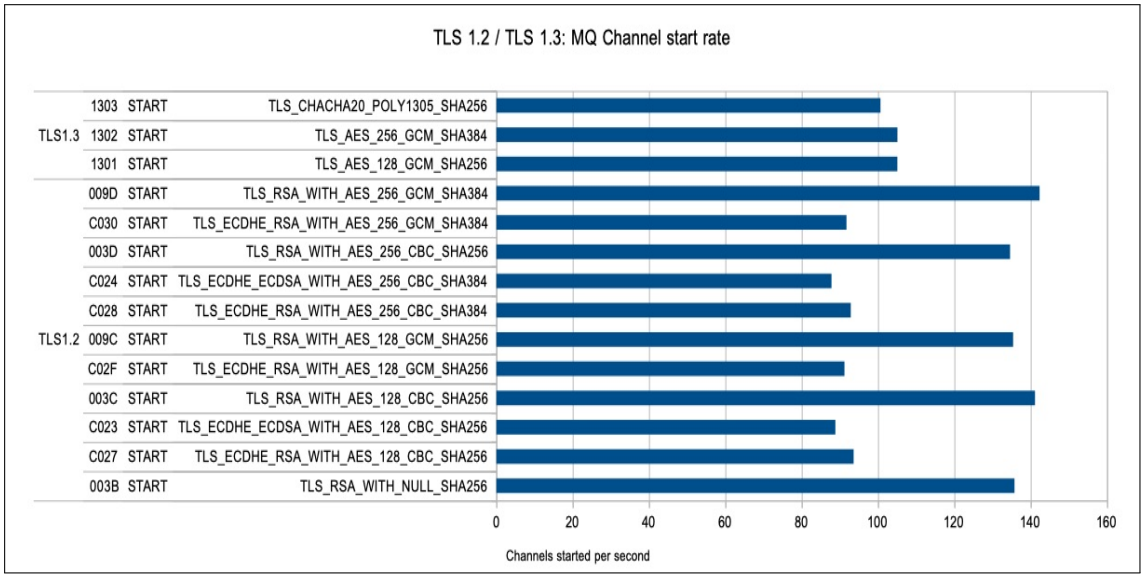
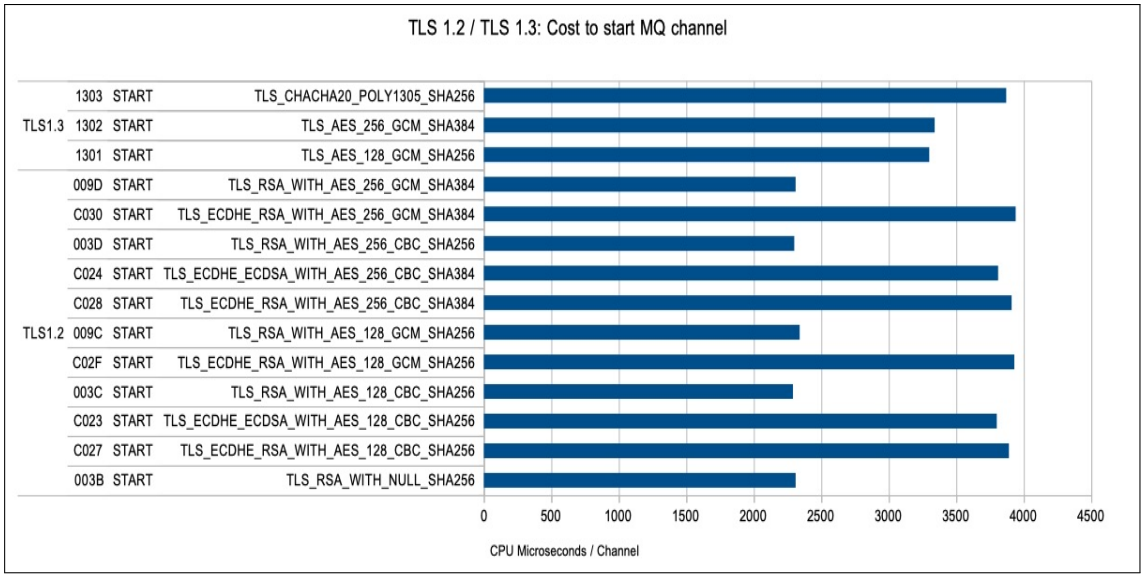


Chart: TLS Cost of starting MQ channel



Starting TLS channels using aliases

In the earlier section "Starting TLS channels", it was demonstrated that the cipher used can impact the rate the channels could start as well as affect the cost of starting the channel.

By specifying an alias, such as ANY_TLS13, MQ is able to negotiate the most secure common cipher supported by both ends of the channel.

Using an alias that allows TLS 1.3 protocols to be included in the negotiation phase of channel start means that MQ must generate public/private key pairs for *each potential* key share group.

The following table demonstrates how the specified alias can impact the channel start rate and cost.

Table: Impact of alias on channel start

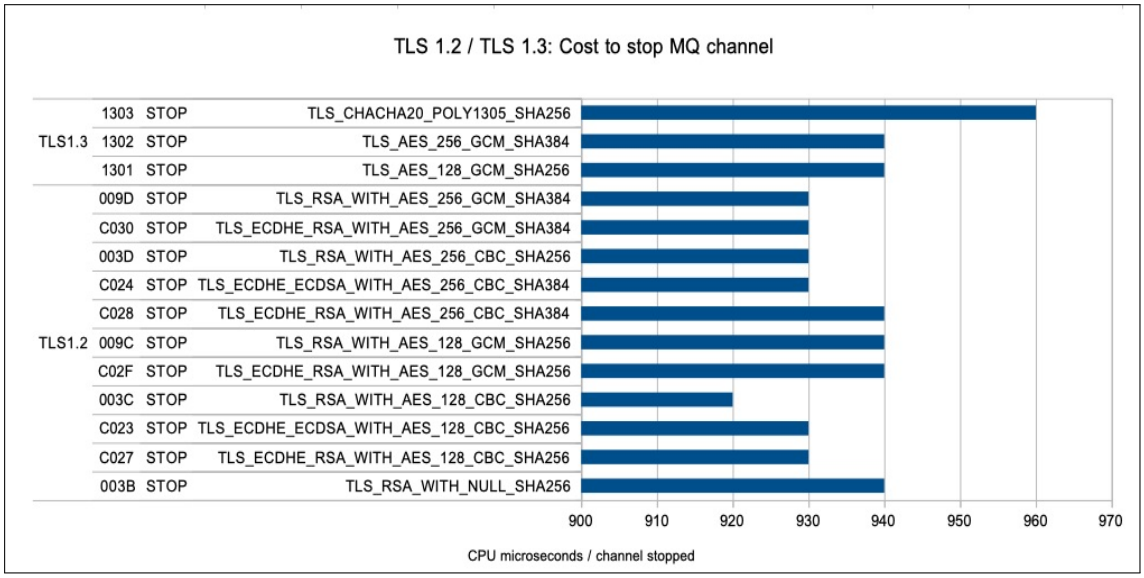
Alias	Negotiated Cipher	Cost CPU ms	Channels started / second
ANY_TLS13_OR_HIGHER	TLS_CHACHA20_POLY1305_SHA256	3.89	101
ANY_TLS13	TLS_CHACHA20_POLY1305_SHA256	3.88	102
ANY_TLS12_OR_HIGHER	TLS_CHACHA20_POLY1305_SHA256	3.85	104
ANY_TLS12	TLS_RSA_WITH_AES_256_GCM_SHA384	2.33	139
ANY	TLS_CHACHA20_POLY1305_SHA256	3.86	100

Note: Where a TLS 1.3 cipher *may* be negotiated, there is is additional cost due to the need to generate key pairs for all key share groups. This has an impact on the rate at which the MQ channels may start. The impact of this key pair generation is significantly reduced with ICSF FMID HCR77D1 on IBM z15.

Stopping TLS channels

In terms of stopping MQ channels, we saw TLS 1.2 ciphers typically cost 920 to 940 CPU microseconds per channel and TLS 1.3 ciphers cost between 940 and 960 CPU microseconds on IBM z15.

Chart: TLS Cost of stopping MQ channel



Note: Stopping a channel where the SSLCIPH used an alias does not affect the cost or rate at which the channel is stopped.

Secret key (re-)negotiation costs

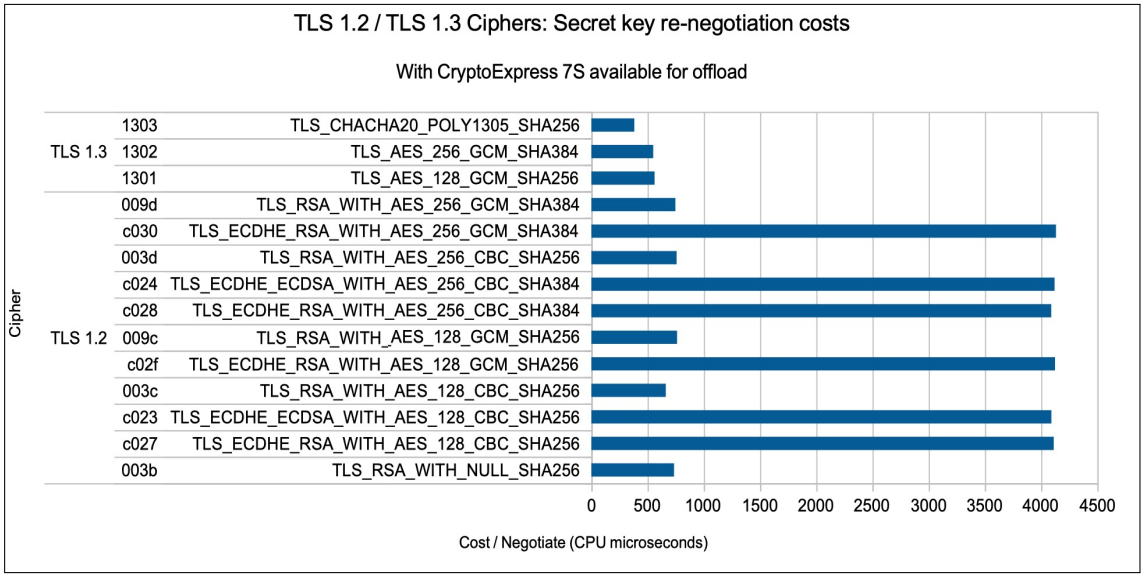
The cost of re-negotiating the secret key can vary significantly depending on the cipher used.

TLS 1.3 ciphers have key re-negotiation as part of the protocol, so the processing triggered by reaching the threshold as defined by the SSLRKEYC attribute does not perform key re-negotiation. However, even though MQ does not attempt to re-negotiate the key, there will be cost associated with a non-zero SSLRKEYC for TLS 1.3 ciphers.

Ideally, for queue managers using TLS 1.3 ciphers, the SSLRKEYC attribute would be set to 0, but as the attribute is defined at the queue manager level, this may affect any TLS channels that run using TLS 1.2 ciphers.

The following chart shows the cost of each key negotiation on IBM z15 for all of the TLS 1.3 and TLS 1.2 ciphers currently supported.

Chart: TLS Cost of secret key re-negotiation



How did we calculate the key re-negotiation costs?

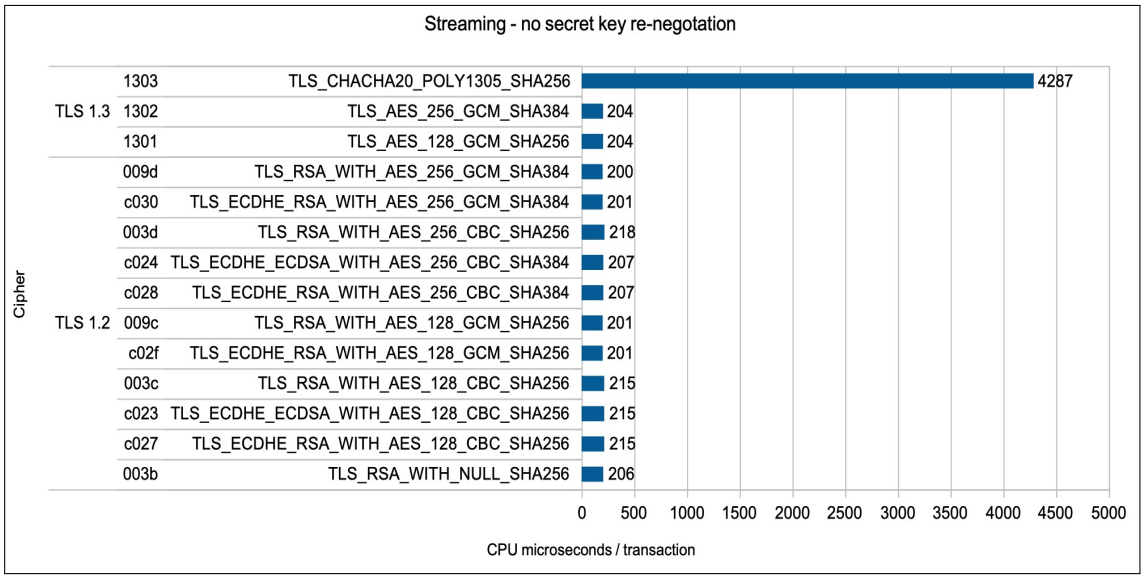
The key re-negotiation costs were calculated as follows:

- Workload run with SSLRKEYC(0) configured and the average transaction cost was determined.
- Workload run with SSLRKEYC(1MB) configured and the average transaction cost was determined.
- As we knew the message size was constant, we could determine how many messages would flow between key negotiations, taking into account that the workload is a request / reply workload.
- The difference between the 2 transaction costs multiplied by the number of messages per negotiation is the cost of re-negotiating the secret key.

Cost of Encryption / Decryption of data using TLS ciphers

The cost of encrypting and decrypting data flowing over MQ channels is typically reduced with the use of CPACF (Central Processor Assist for Cryptographic Functions) and is relatively consistent, however cipher `TLS_CHACHA20_POLY1305_SHA256` is not supported by hardware encryption and must be processed using software.

Chart: TLS Transaction cost when streaming 32KB non-persistent messages



This workload represents the transaction cost including the putting application, sending MQ subsystem, receiving MQ subsystem and getting application and includes the cost of encrypting and decrypting the messages once per transaction.

Can I influence which ciphers are chosen?

Yes. When using an alias to allow MQ to negotiate from a set of supported TLS ciphers, it is possible to restrict the choice of cipher to a subset of specific ciphers.

For example, whilst `TLS_CHACHA20_POLY1305_SHA256` is regarded as the most secure cipher currently available and supported on z/OS, the cost of the encryption may mean that particular cipher is less palatable from a performance perspective.

To configure the queue manager to use only a subset of the available ciphers, code the `CSQMINI DD` card in the MSTR JCL, i.e. the following sample will limit the selection of TLS 1.3 ciphers to `TLS_AES_256_GCM_SHA384` and `TLS_AES_128_GCM_SHA256`.

```
\\CSQMINI DD *
TransportSecurity:
AllowTLSV13=TRUE
AllowedCipherSpecs=TLS_AES_256_GCM_SHA384,TLS_AES_128_GCM_SHA256
\*
```

The "AllowedCipherSpecs" keyword is not limited to TLS 1.3 ciphers, for example to limit the ciphers to `TLS_RSA_WITH_AES_256_CBC_SHA256` specify the contents of the `CSQMINI DD` card as:

```
\\CSQMINI DD *
TransportSecurity:
AllowedCipherSpecs=TLS_RSA_WITH_AES_256_CBC_SHA256
\*
```

Chapter 7

AMS Interception on server to server message channels

IBM MQ Advanced for z/OS VUE version 9.2 only

AMS interception on server to server message channels is an enhancement to the Advanced Message Security feature.

AMS Interception on server to server message channels is not the same, and should not be confused with AMS MCA Interception, which is a distributed only feature and can be used to selectively enable policies to be applied for server connection-type channels.

For the purposes of this section, we refer to AMS interception on server to server message channels as "AMS Interception on z/OS".

The IBM MQ Knowledge Centre should be regarded as the primary repository for information on configuring MQ for z/OS with AMS.

This section will discuss what AMS Interception on z/OS is, how to configure your MQ for z/OS systems to use this new function, and what impact you may see to the performance of your AMS-enabled queue manager.

What is AMS Interception on z/OS and when would I use it?

AMS Interception on z/OS provides a means to control whether messages should have any applicable AMS policies applied to them, when sender-type message channel agents get messages from transmission queues, and receiver-type message channel agents put messages to target queues.

This allows AMS protection to be enabled on a queue manager when communicating using server to server channels of type sender, server, receiver and requester, with a queue manager that does not have AMS enabled.

In turn, this means that AMS protected messages in AMS enabled queue managers can be unprotected prior to being sent to non-AMS enabled queue managers, and unprotected messages received from non-AMS enabled queue managers can be protected, by applicable AMS policies, on AMS enabled queue managers.

AMS Interception on z/OS allows the enterprise to protect their data using AMS qualities of protection without mandating all of their business partners apply AMS protection or even the same AMS quality of protection.

AMS interception on z/OS can also be used to ensure data hosted by discrete "data islands" within the same business can protect their local data, without needing to co-ordinate the certificate management. The use of TLS cipher protection over MQ channels may be sufficient to protect the data in-flight between those islands, based on the business' security policies, whilst relying on AMS to provide protection of data at rest in the island. Note that dumps of the channel initiator may contain data that is not protected.

Also note that AMS is not a replacement for using TLS on channels, however for simplicities sake, the measurements in this document do not use TLS cipher protection on the MQ channels. Given the above use cases for AMS Interception on z/OS, it is anticipated that the typical configuration will use the AMS Confidentiality quality of protection. As such, the measurements in this document will focus primarily on relying on Confidentiality to encrypt the data, although examples with Integrity and Privacy are also provided.

Terminology used with AMS Interception on z/OS

Given the scope of the AMS Interception on z/OS feature, all workloads will be queue manager to queue manager using MQ server to server channels. Each workload has a domain where the data is protected at a similar level of protection. We are using the term "data island" to represent these areas of data that are protected in the same manner.

This section will consider scenarios of the following types:

- **Single data island** across 2 queue managers, either with:
  - unprotected workload.
  - or end-to-end AMS protection.
- **Multiple data islands**, one protected with AMS Interception on z/OS and one unprotected.
- **Multiple data islands**, protected independently with AMS Interception for z/OS. For example one of the data islands could be protected with AMS Confidentiality policies and the other with AMS Privacy policies. Please note that we do not expect this to be a common scenario.

Configuring AMS Interception on z/OS

AMS Interception on z/OS is configured using the **SPLPROT** attribute on channel types (CHLTYPE) of SDR, SVR, RCVR or RQSTR. The available options to configure the behaviour as dependent on the channel type specified.

PASSTHRU
Pass through, unchanged, any messages sent or received by the Message Channel Agent (MCA) for this channel. This value is valid for channels with a channel type of SDR, SVR, RCVR or RQSTR and is the default value.

REMOVE
Remove any AMS protection from the message received from the transmission queue by the Message Channel Agent (MCA), and send the messages to the remote partner.

When the MCA gets a message from the transmission queue, if an AMS policy is defined for the transmission queue, any AMS protection is removed from the message prior to sending the message across the channel.

If the AMS policy is not defined for the transmission queue, the message is sent unchanged.

This value is valid for channels of type SDR or SVR.

ASPOLICY
Based on the policy defined for the target queue, apply AMS protection to inbound messages prior to putting them on the target queue.

When the MCA receives an inbound message, if an AMS policy is defined for the target queue, AMS protection is applied to the message prior to the message being put to the target queue. If an AMS policy is not defined for the target queue, the message is put to the target queue unchanged.

This value is valid for channels of type RCVR or RQSTR.

User ID for AMS Interception on z/OS

The requirement for user IDs used with AMS Interception on z/OS are the same as those for existing AMS enabled applications. For a running channel, the sending MCA gets messages from a transmission queue and the receiving MCA puts messages to target queues. The MCA user ID (MCAUSER) field, set on server-to-server channels, defines the user ID under which the MCAs perform put and get requests.

With AMS Interception, AMS functions are performed during put and get requests, as with other AMS enabled applications. Therefore MCA user IDs have the same requirements as those for AMS application user IDs.

The MCAUSER used to perform the put and get is configurable, and dependent on whether it is an inbound or outbound channel. See the MQ Knowledge Centre on MCAUSER for details on how the chosen user ID performs actions on the MCA. As such, the user ID that the channel initiator is running under is the user ID that is to be used for AMS functions performed during server-to-server message channel interception. Therefore, these user IDs have the same requirements as those for AMS application user IDs.

Message size and MAXMSGL

Due to AMS protection, the message size of protected messages will be larger than the original size.

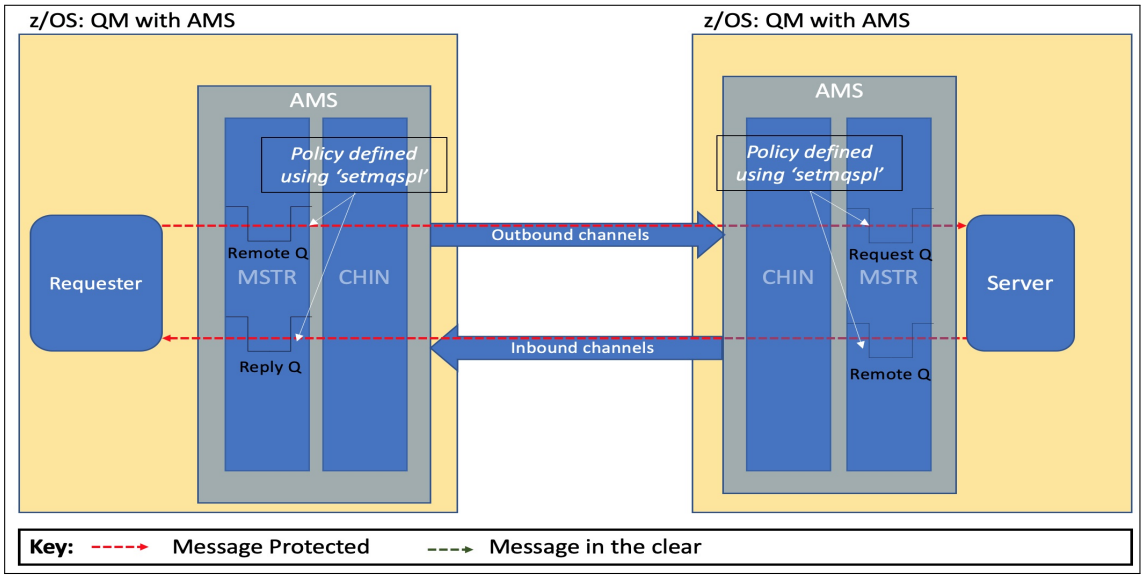
Protected messages are larger than unprotected messages. Therefore, the value of the MAXMSGL attribute on both queue and channels, might need to be altered to take into account the size of protected messages.

As a guide:

- Signing the message adds between 1150 and 1200 bytes.
- Encrypting the message adds a further 280 bytes.
- Authenticating the message does not make a difference to the size of the message.
- Adding more recipients to the policy increased the size of the message by approximately 220 bytes per recipient.

Configuration - both queue managers are AMS-enabled

This initial example shows the configuration where both queue managers are AMS-enabled and a request/reply workload is used.



This configuration can be regarded as a single data island with end-to-end AMS protection - where the data is protected across the Enterprise.

Note that in this example there are **two** queues defined on each queue manager with AMS policies - the remote queue for outbound messages and the inbound queue for messages arriving from the partner.

This example has the request message protected from the time it is put by the requester application until it is successfully gotten by the server application and similarly when the reply message is sent back from server to requester.

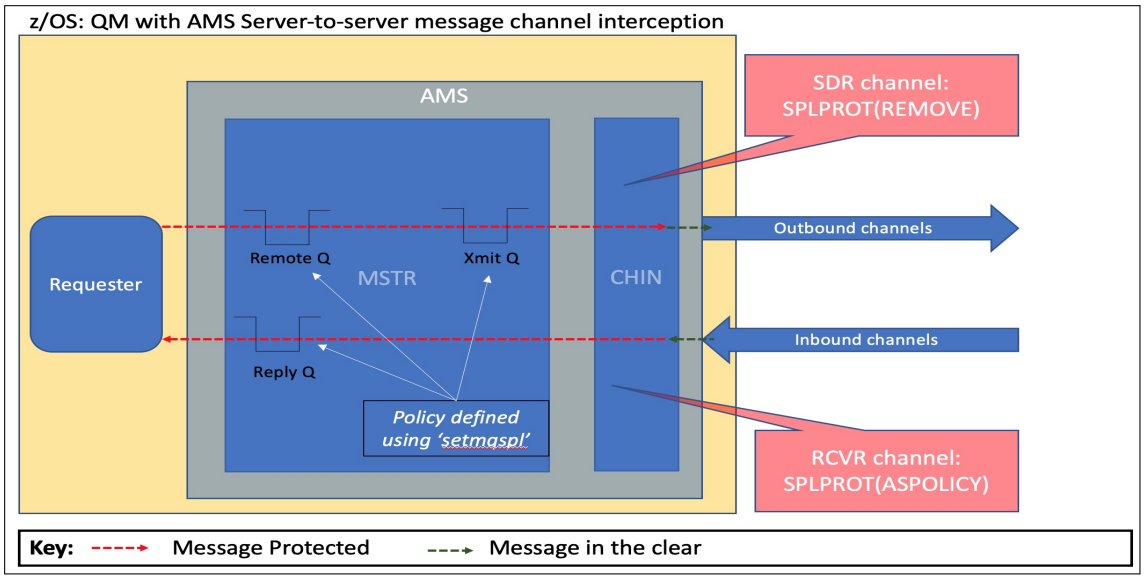
Configuration - only one queue manager is AMS-enabled

Contrast the previous full AMS protection configuration with one using AMS Interception on z/OS, where only one queue manager is AMS-enabled.

The following diagram provides an example for AMS Interception for z/OS where the remote queue manager (not shown) is non-AMS enabled.

In this example, note that there are **three** queues that have AMS policies defined - on the outbound messages the remote queue and the transmission queue both have policies defined, as does the reply queue for inbound messages.

Since the remote queue managr is non-AMS enabled, outbound messages need to have their AMS protection removed before sending, so the sender channel has SPLPROT "REMOVE". For inbound messages, AMS message protection is required, so the receiver channel uses SPLPROT "ASPOLICY" to ensure the queue manager applies the AMS policy, if one exists.



Note that in addition to both the SDR and RCVR channels requiring the SPLPROT attribute being defined, the transmission queue also has a policy defined, in order to support AMS server to server message channel interception.

This configuration is an example of multiple data islands, where one island is protected and the other island is not protected by AMS qualities of protection.

Configuration - queue managers are independently configured with AMS Interception on z/OS

This is an example of the "multiple data islands configured AMS Interception on z/OS" configuration.

This configuration is similar to the previous configuration in that the local queue manager is configured with AMS Interception on z/OS.

In addition, the remote z/OS queue manager is also configured with AMS Interception on z/OS i.e.:

1. 3 queue with AMS policies defined:
    - Inbound queue
    - Remote queue
    - Transmission queue
2. The receiver channel will have SPLPROT(ASPOLICY).
3. The sender channel will have SPLPROT(REMOVE).

AMS Interception on z/OS and the MQ channel initiator

From an MQ performance perspective, the largest change introduced with AMS Interception on z/OS is that the channel initiator is potentially performing the removal of AMS protection from outbound messages and applying AMS protection to inbound messages.

This AMS protection, whether being removed or applied, is performed by one of the adapter tasks in the channel initiator, and is selected from the next available task from one of set of adapter TCBs as specified by the CHIADAPS parameter.

Whilst applying or removing AMS protection from the current message, the adapter task will be unavailable for other work, either using CPU or waiting for cryptographic work to be completed by the available cryptographic hardware.

MQ Statistics trace class(4) data can be used to see the impact of AMS Interception on z/OS on the channel initiators' adapter tasks, for example:

Table: Adapter usage when applying AMS protection to inbound messages.

Configuration	Average CPU time	Average Elapsed time	Comments
Unprotected	8	8	
Confidentiality with key reuse 32	15	18	1. CPU time is higher than unprotected, but minimal wait for offload to cryptographic hardware. Given these numbers are an average, the 3 microsecond wait could be extrapolated to determine the key generation elapsed time e.g. 3 * 32 = 96 microseconds.
Privacy	18	338	1. Increased time difference between CPU and elapsed, due to waiting for cryptographic work or *local lock*. 2. Increased cost from signing the message and encrypting the key.

The impact of AMS Interception on z/OS on the adapter tasks is discussed further in "channel initiator tasks and local lock".

How the AMS policy type and the increased load on the adapters can affect workloads is discussed further in the "Business partner streaming to AMS Interception-protected Enterprise" section.

As a result of the increased load on the adapter tasks when using AMS Interception on z/OS, it may be necessary to ensure additional channel initiator adapter tasks are made available to avoid impacting other non-AMS specific workload.

Typically the simplest way to determine if there are sufficient adapter tasks available is when reviewing the adapter report to ensure that at least 1 adapter task is unused for the SMF interval.

Note: More adapter takes may be specified using the `ALTER QMGR CHIADAPS(integer)` command, however these tasks will not be available until the channel initiator address space is restarted.

What is the performance impact of AMS Interception on z/OS?

When implementing AMS Interception on z/OS, you will have experience of at least one of the following two configurations:

- No AMS protection.
- End to end AMS protection.

Enabling AMS Interception for z/OS from either configuration will result in some impact to your system, whether additional CPU cost, increased latency on the transaction or re-distribution of cryptographic workload.

Moving from no AMS protection to AMS Interception on z/OS

When moving from an environment with non-AMS enabled queue managers to AMS enabled queue managers, there will be a performance impact.

Moving from end to end AMS protection to AMS Interception on z/OS

When moving from an environment where AMS protection spans multiple queue managers to one where AMS Interception on z/OS allows messages to flow to non-AMS enabled queue managers, there is an impact on both the queue manager implementing AMS message channel interception and the hosting LPAR.

Consider an environment where there is data flowing between 2 AMS enabled queue managers, where protection is applied on LPAR A and removed on LPAR B.

- On each LPAR, there will be AMS-related cost on the applications using the AMS protected queue, as well as both the AMSM and MSTR address spaces.

Compare this to the AMS Interception on z/OS configuration where the protection is applied on LPAR A and then the channel initiator removes the AMS protection prior to sending to LPAR B.

- The original cost of protection will still be incurred by the putting application as well as the MSTR and AMSM address spaces.
- In addition, the cost of unprotecting the message will be incurred by the channel initiator, as well as some cost in the MSTR and AMSM address spaces.

The net cost of protecting and unprotecting the message will be similar whether performed in the end-to-end configuration or in the AMS Interception on z/OS configuration.

Despite the net cost of AMS protection remaining similar, consider the impact to both the CPU and Cryptographic processors on the LPAR with AMS message channel interception.

- Since the unprotection is occurring on the same system as the protection, there will be additional CPU utilisation.
- Similarly, load on the cryptographic processors may double. This additional load may add latency due to the requests being queued waiting for cryptographic processors.

Channel throughput

As mentioned previously the work performed in the channel initiator relating to AMS Interception on z/OS is completed by adapter tasks.

Messages flowing over a single channel are processed in series, i.e. the current message must be processed in its entirety, whether getting from a queue and unprotecting, or protecting and putting to a queue, before the subsequent message is begun to be processed.

The increased time spent protecting or unprotecting the message by the adapter task means that the queued messages are processed more slowly.

If those queued messages are on a remote system, possibly belonging to a business partner, they may see their MQ queue deeper than before AMS Interception on z/OS was implemented.

There is potential for the queue on the remote system to fill and it is suggested that the owner of the remote system is made aware that AMS Interception on z/OS is being implemented on the target system, and that they should monitor for deep queues.

It may be necessary to increase the maximum depth of the queue on the remote system or to modify the application to wait-then-retry in the event of queue full.

If message expiry is used on the remote system. it may be necessary to review the value used.

Channel initiator tasks and local lock

Typical guidance for tuning adapter tasks is to verify that at peak volumes there is a least one unused adapter task. This can be verified using the MQ statistics class(4) data.

Many z/OS subsystems including the channel initiator uses local lock to serialise access to certain resources, such as storage allocation.

This local lock may be taken by the adapter tasks, and with the additional work performed by AMS Interception on z/OS, increased local lock time can result.

Therefore, the configuration of the channel initiator tasks such as the adapters (CHIADAPS) and dispatchers (CHIDISPS) when using significant numbers of channels using the SPLPROT attribute can affect the amount of local lock time, particularly if additional channel initiator tasks are added.

The tables and charts that follow in this section aim to demonstrate the impact of local lock times on the adapter tasks when running a simple request/reply workload across 50 channel pairs, protected by AMS Interception on z/OS, when using AMS Confidentiality with key reuse (KR) of 32. This configuration should mean a relatively light usage of any Crypto Express hardware.

A symptom of increased local lock time can be see in the MQ statistics class(4) data with a significant difference between the adapter CPU time and the adapter elapsed time.

Why might the adapter elapsed time be significantly larger than the CPU time?

When the is sufficient CPU, the MQ Statistics trace class(4) adapter report will generally show the average elapsed time to be similar to the average CPU time, but there are a number of times where there will be a more significant difference between the two values. Some of these reasons include:

1. Persistent messages, waiting for log writes.
2. Message selection - this is discussed in report MP16 "Capacity Planning and Tuning Guide".
3. Buffer pools used by the queue are at capacity and the queue manager is enforcing immediate writes to page set.
4. AMS Interception for z/OS - waiting for local lock and/or cryptographic work offloaded to Crypto Express hardware.

How do we know this is the impact from local lock?

Local lock can be relatively difficult to quantify in terms of the impact to the messaging rate, but with regards to these measurements we know that increased transaction round-trip time occurs despite:

- Cryptographic hardware is not constrained,
- CPU is lightly loaded and for less than 15% of the interval there are more tasks active than CPUs available,
- Workload is not performing disk I/O,
- Dedicated performance network is capable of significantly more throughput.

We can confirm the presence of local lock by formatting a standalone channel initiator dump using the IPCS "SYSTRACE PERFDATA" function and checking the time spent in local lock.

Varying the number of dispatcher tasks

In this section we demonstrate how varying the number of dispatchers can affect the usage of the adapter tasks as well as the overall throughout.

The workload used is a request/reply type workload using 32KB non-persistent messages using 50 sets of queues. In this example the two queue managers represent separate data islands - where the requester queue manager protects its data using AMS Confidentiality qualities of protection and the server queue manager has no AMS protection applied.

The adapter data shown in the following table and chart is taken from the requester queue managers' adapter report, generated from MQ Statistics class(4) data.

In all cases, the queue managers are defined with 10 adapter tasks.

Table: Vary dispatcher tasks for AMS Interception for z/OS Request/Reply workload.

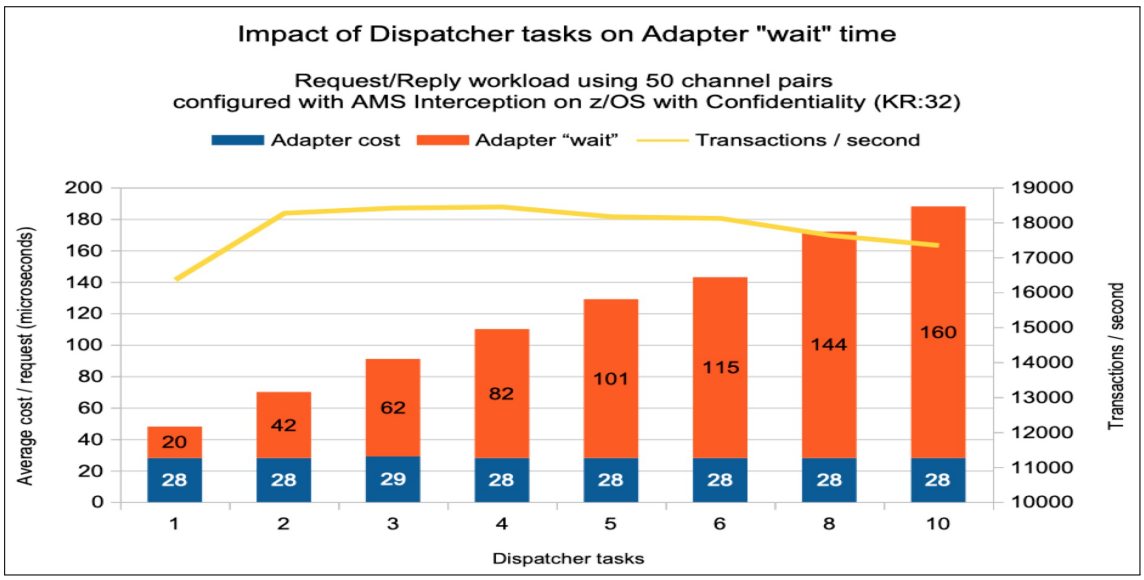
Dispatcher tasks	Transactions per second	Cost per transaction (microseconds)	Adapter average elapsed (microseconds)	Adapter average % busy
1	16369	379	48	24
2	18277	395	70	39
3	18433	398	91	51
4	18456	398	110	62
5	18176	400	129	71
6	18128	400	143	78
8	17645	401	172	92
10	17354	401	188	98

In the above table, the average CPU time per adapter request remains constant at 28 CPU microseconds, but as the number of dispatcher tasks increases, the average elapsed time per adapter request increases significantly, from 48 to 188 microseconds.

The average use of the 10 available adapters also increases significantly - note that the peak throughput was achieved with 2-4 dispatcher tasks, but the 10 adapter tasks are 40-60% busy. When the number of dispatchers is increased to 10, the overall throughput has dropped but the 10 adapters are all 96-99% busy (for an average of 98%) - and spending approximately 160 microseconds per adapter call waiting for local lock.

In none of these measurements are the configured dispatcher tasks more than 60% utilised over the SMF intervals.

The following chart is a representation of the key items from the table when varying the number of dispatchers.



Notes on chart:

Wait time is calculated by subtracting the average CPU time from the average elapsed time in the statistics class(4) adapter report.

The chart shows that the peak rate was achieved with 2-4 dispatcher tasks and this was despite the adapter tasks being less than 62% busy. These configurations saw relatively small adapter wait (local lock) times. As more dispatchers were made available to drive the adapter tasks, the time spent waiting for the lock increased significantly and the adapter usage increased - solely in wait time.

With an adapter task spending more time waiting for the local lock, the adapter is unable to process work from other channels, which may themselves not need to take the local lock, thus potentially affecting other channels that are not using the AMS Interception for z/OS function.

Varying the number of adapter tasks

In this section we have used the optimal configuration of 3 dispatchers from the previous section, and vary the number of adapters to illustrate the usage of those available adapter tasks and how that can affect the overall transaction rate.

Table: Vary adapter tasks for AMS Interception for z/OS Request/Reply workload.

Adapter tasks	Transactions per second	Cost per transaction (microseconds)	Adapter average elapsed (microseconds)	Adapter average % busy
1	11414	373	29	100
2	15904	390	42	100
3	17134	392	59	100
4	17633	394	76	100
5	17913	394	86	92
6	18149	398	88	81
8	18306	397	92	64
10	18423	398	91	51

In the above table, the dispatcher tasks were never more than 25% busy for the measurements. The average CPU time per adapter request was typically 25-29 CPU microseconds, but by limiting the number of available dispatcher tasks, we have reduced the effect of local lock on the adapter tasks as a side effect of constraining the number of concurrent channels flowing data through the channel initiator.

Guidance for adapters and dispatchers with AMS Interception on z/OS

It is worth noting that the local lock does not necessarily cost in terms of CPU - it is primarily time waiting for resource, but it does block the adapter task from performing other work.

Guidance for configuring the number of adapter and dispatcher tasks is available in report MP16 "Capacity Planning and Tuning Guide", but a reminder is provided below.

Dispatchers:

As few as possible, as long as the available dispatchers are not used at capacity. As the earlier section "varying the number of dispatcher tasks" shows, too few dispatchers can inhibit throughput.

It may be that limiting the number of dispatchers can be used to reduce local lock time, or at least the "wait" time in the adapter tasks that we have discussed but this can affect non-AMS Interception on z/OS channel throughput.

Adapters:

General guidance for adapters depends on the type of workload.

For example, workload using persistent messages can take much longer than those for non-persistent messages, due to log I/O, thus a channel initiator processing a large number of persistent messages across many channels may need more than the default 8 adapter tasks for optimum performance.

Similarly, workloads using AMS Interception for z/OS to apply or remove AMS protection to messages may benefit from additional adapter tasks.

Where serialisation for resource occurs in the channel initiator, such as with AMS Interception for z/OS or persistent messages and as per MP16 guidelines, we suggest CHIADAPS(30) for systems with up to 4 processors and then increase CHIADAPS by 8 for each additional processor up to a maximum of CHIADAPS(120). We have seen no significant disadvantage in having CHIADAPS(120) where this is more adapter tasks than necessary.

At a minimum, we suggest that there is **at least 1** adapter task unused during peak workloads.

By ensuring there are unused adapter tasks even at peak workloads, this should prevent adapters serialised and waiting for resource (local lock) whilst processing AMS Interception for z/OS workload from impacting workload on channels that are not AMS Interception on z/OS-enabled.

Finally, and to re-iterate, we would advise monitoring the utilisation of the channel initiator tasks using MQ statistics class(4) data to determine whether there are sufficient tasks available. If the tasks are constrained and there is sufficient CPU available, some benefit may be achieved with small incremental changes to the number of available channel initiator tasks.

Indexing of SYSTEM.PROTECTION.POLICY.QUEUE

When starting your queue manager you may see message:

```
CSQI004I @VTS1 CSQIMGE3 Consider indexing SYSTEM.PROTECTION.POLICY.QUEUE by
               CORRELID for BATCH connection VTS1AMSM, xxx messages skipped
```

The data held on this queue is also cached within the AMSM region, so the majority of access uses cached data.

The queue is only read by the AMSM address space on start-up and when the REFRESH command is issued.

As a result, unless there are hundreds of policies defined, it is unlikely that indexing the SYSTEM.PROTECTION.POLICY.QUEUE will offer a benefit.

Example scenarios using AMS Interception on z/OS

Single AMS protected data island - Request / Reply workloads

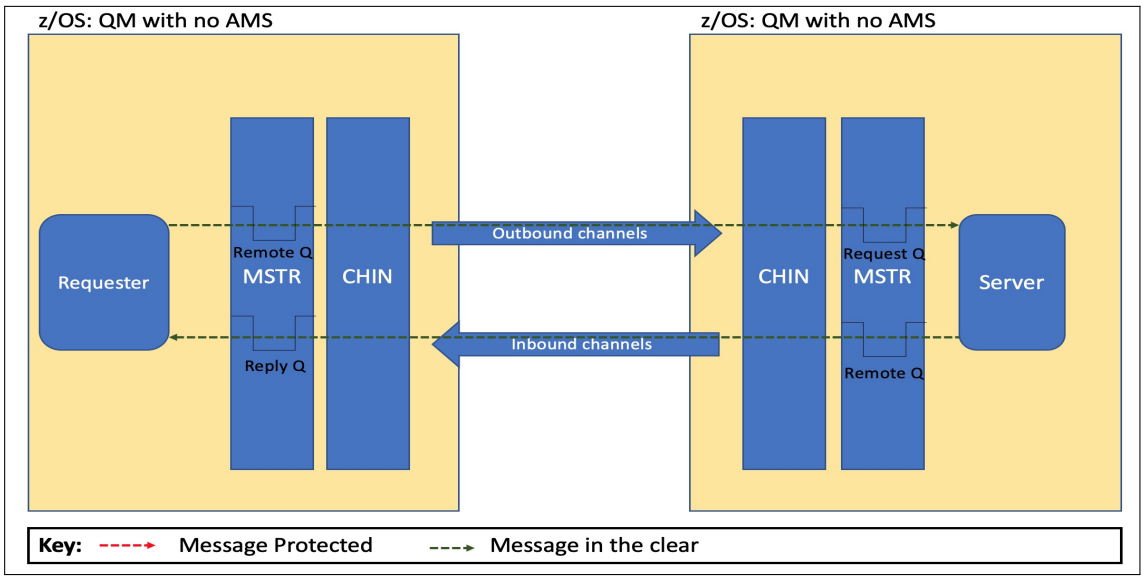
This first scenario uses a simple request/reply workload to demonstrate the impact of AMS Interception on server to server message channels when compared with both a non-AMS workload and a full end-to-end AMS configuration.

To that end, there are three configurations being used, and they are:

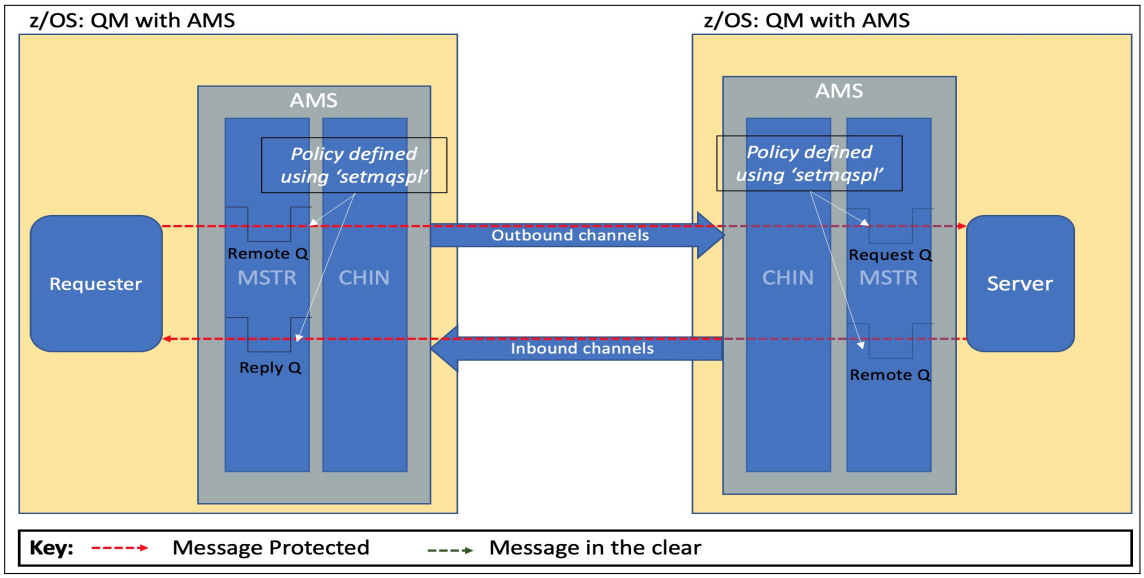
1. Single data island, with unprotected workload.
2. Single data island, with end-to-end AMS protection.
3. Multiple data islands, one protected with AMS interception on z/OS.

The three configurations can be represented graphically thus:

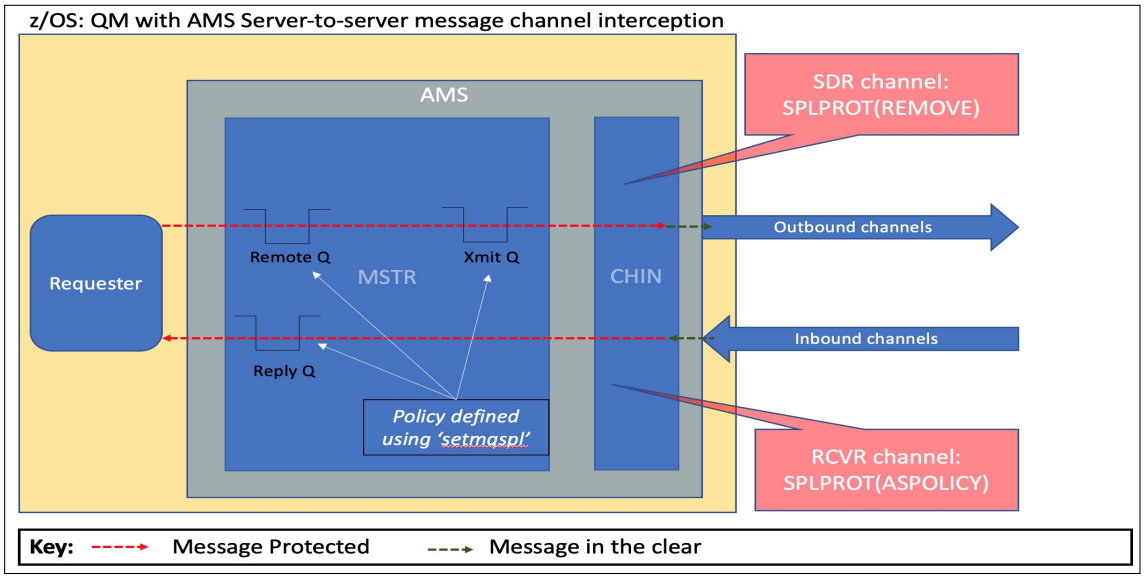
Workload *(single data island, with unprotected workload)*, configured without AMS protection:



Workload *(single data island, with end-to-end AMS workload)*:



When using AMS Interception on z/OS on the requester-side of the configuration, the configuration is represented, with the server-side having no AMS protection configured *(multiple data islands, one protected with AMS Interception on z/OS)*:



The measurements that follow use a single pair of request / reply tasks with a 32KB non-persistent message per channel pair.

For the request / reply workloads we focus primarily on the AMS Confidentiality quality of protection as these are the most likely use cases.

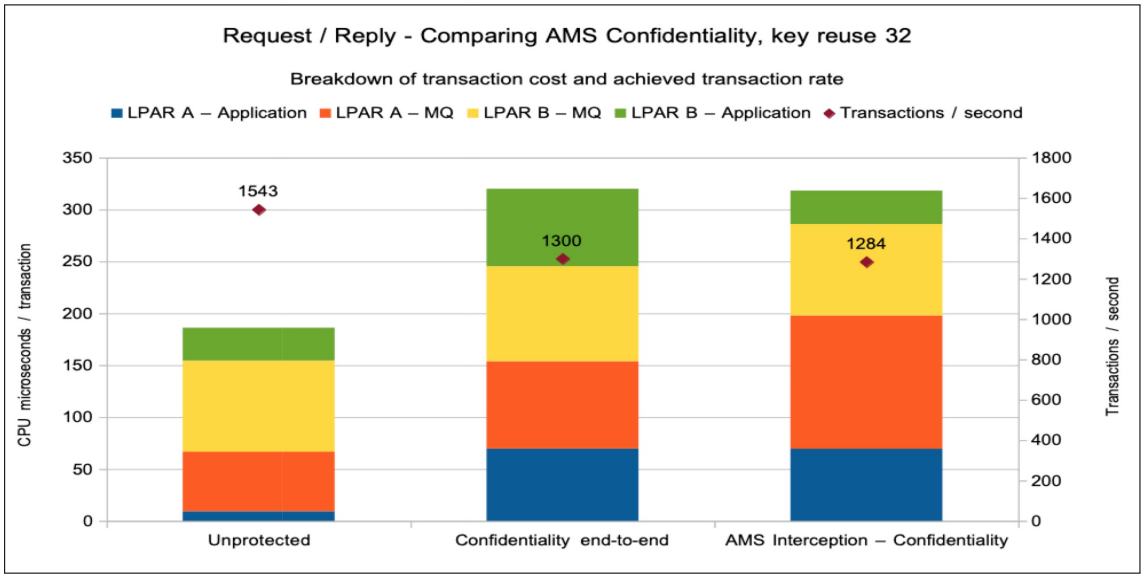
AMS Confidentiality with key reuse 32

For the purposes of the following measurements, a key reuse of 32 has been used, which benefits from reduced use of asymmetric cryptography whilst still re-generating the key on a regular basis.

Reducing the use of asymmetric cryptography means that the Crypto Express hardware is used less frequently.

As will be demonstrated later, the performance of AMS Confidentiality with key reuse of 32 is significantly better than AMS Privacy, regardless of whether the AMS protection is end-to-end or on a single end via the AMS Interception on z/OS.

As such, the impact from AMS Confidentiality over the unprotected configuration is much smaller and can be demonstrated in the following chart:



Notes on chart:

MQ costs relate to those in the MSTR, CHIN and AMSM address spaces.

Applying AMS Confidentiality with key reuse of 32 to the workload saw the transaction cost increase by 135 microseconds, which for this workload equates to a 70% increase.

It should be emphasised that the increase of 135 microseconds would be seen in the 32KB non-persistent workload whether the application cost was 10 microseconds or 1000 microseconds.

It should also be noted that this additional 135 microseconds is for the request/reply workload which involves protecting and unprotecting the message twice - once on the outbound message and again on the inbound message. It may be assumed that a uni-directional flow would cost of the order of 68 CPU microseconds.

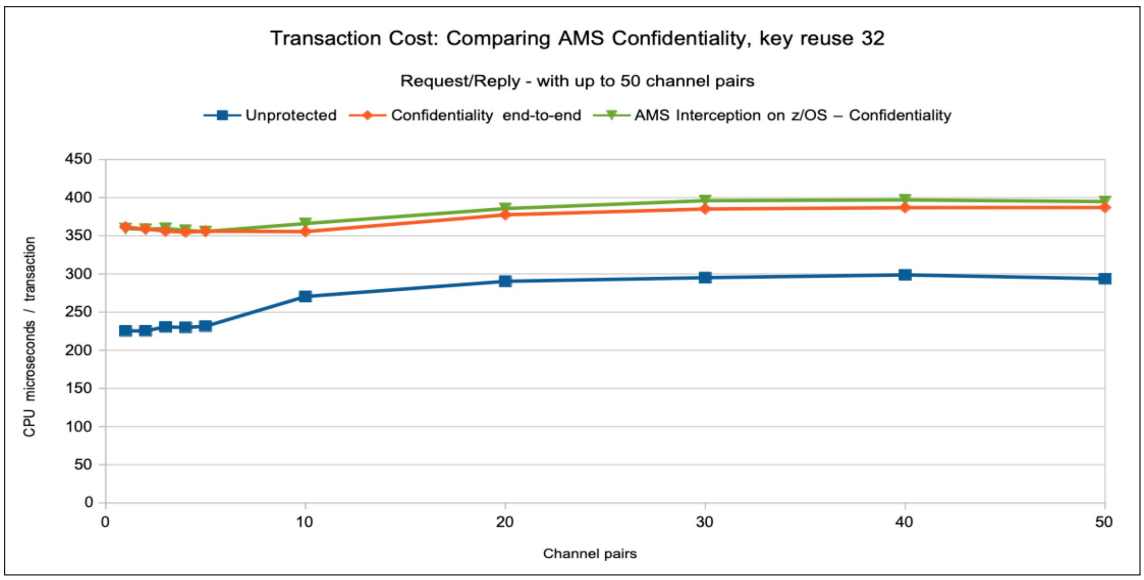
This increased cost resulted in the transaction rate dropping by 16% when compared to the unprotected measurement.

Does AMS Confidentiality scale similarly to the "no protection" model?
With the increased load in the channel initiator for AMS Interception on z/OS, the throughput rate when running under significant load can be affected for a number of reasons including:

1. Increased load on the channel initiator adapter(s) from applying / removing the encryption from the message.
2. Increasing the number of adapter tasks can result in increased local lock time.

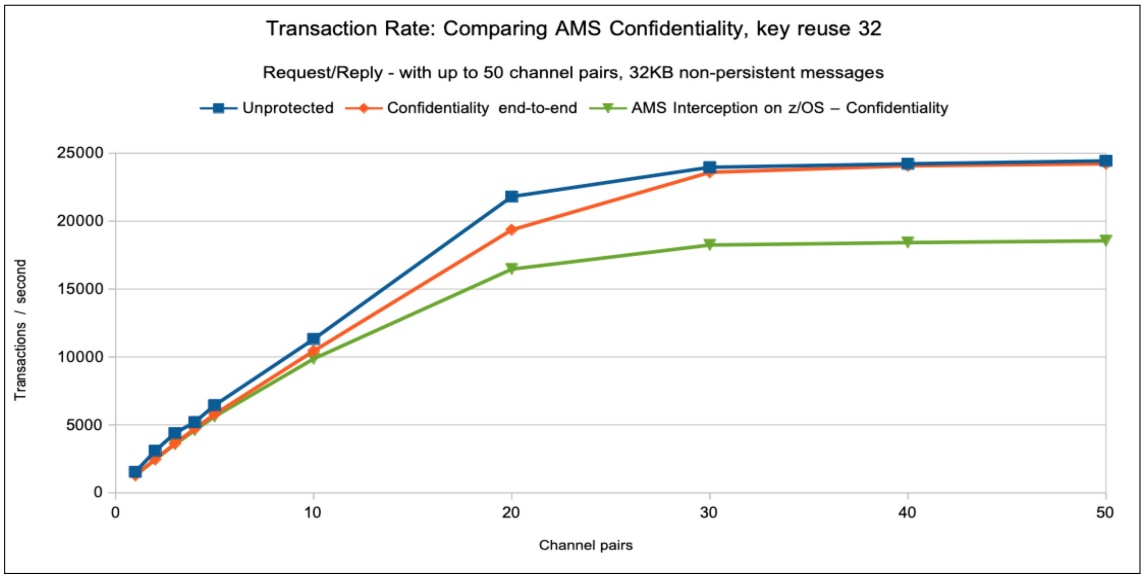
The first chart shows that the transaction cost for the AMS Interception on z/OS when configured with AMS Confidentiality qualities of protection is similar to the AMS Confidentiality end-to-end configuration.

AMS Confidentiality transaction cost:



However the second chart, showing the achieved transaction rate, shows markedly worse performance when running with significant numbers of channels where AMS Interception on z/OS is applied.

AMS Confidentiality transaction rate:



Notes on transaction rate chart:

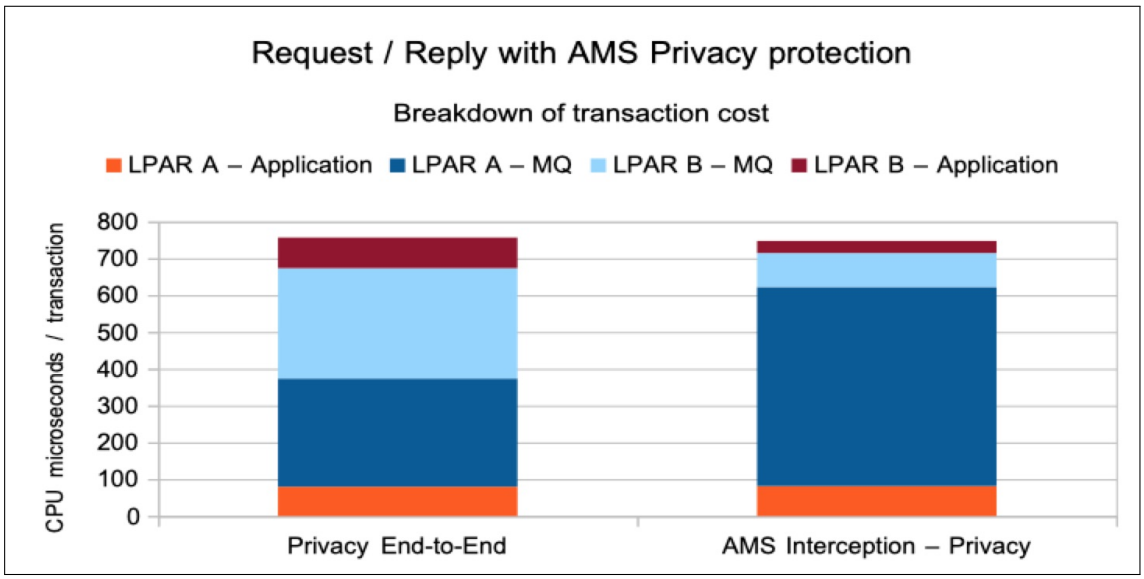
- In this instance the reason for the AMS Interception on z/OS measurement flattening out at 18,500 transactions per second was due to increased local lock in the channel initiator.
- We found that reducing the number of adapter tasks, from 20 to 5, did reduce the local lock but did not improve the throughput.

AMS Privacy

As mentioned previously, moving from an AMS end-to-end protection model to an AMS Interception on z/OS model where the AMS protection is only on one queue manager, means that the net cost impact from AMS remains similar.

The following chart demonstrates the balancing of the cost between the AMS end-to-end and AMS Interception on z/OS models.

AMS Privacy - Breakdown of transaction cost:



Notes on transaction rate chart:

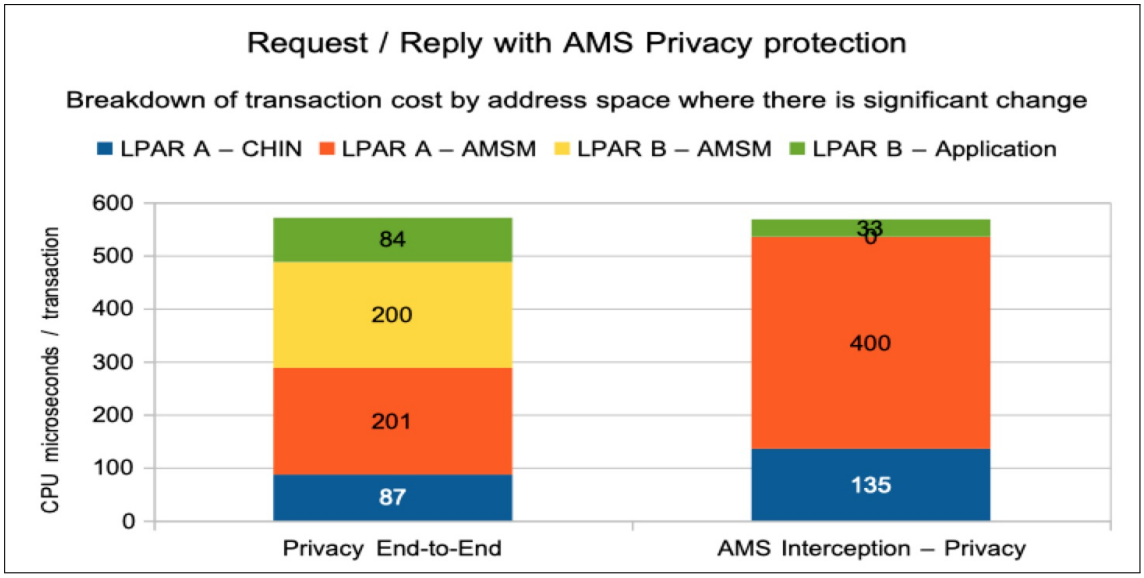
The MQ cost includes the cost of the MQ MSTR, CHIN and AMSM address spaces.

The total transaction cost remains similar whether the AMS protection is applied across both LPARs or only on LPAR A.

Moving from the end-to-end protection model to the AMS Interception on z/OS model, we see the cost move from LPAR B's MQ and application into LPAR A's MQ.

Using the data from the previous chart and expanding to only show address spaces where there was *significant change* to the cost, results in the following chart.

AMS Privacy - Breakdown of transaction cost where there is significant change:

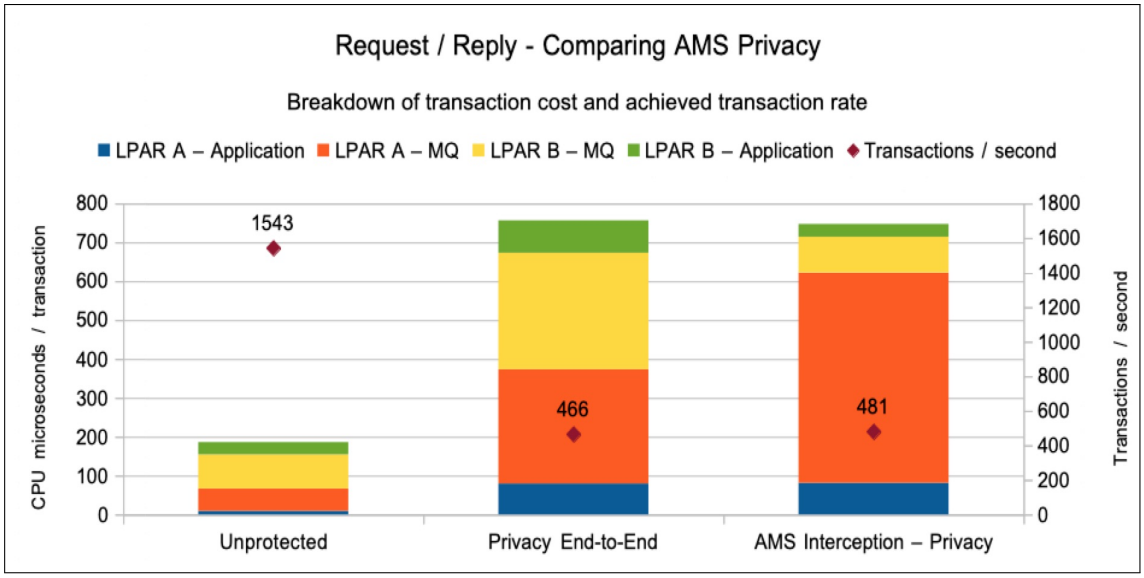


Notes on chart:

This chart shows that there is increased cost in LPAR A's channel initiator address space when moving to the AMS Interception on z/OS model. This increased cost is offset by the reduced cost in LPAR B's application.

The net cost attributed to the AMS address spaces remains constant - again the cost shifts from LPAR B to LPAR A for the AMS Interception on z/OS model.

At this point we have seen how moving from the AMS Privacy end-to-end protection model to the AMS Interception on z/OS using AMS Privacy model shifts the balance of the AMS cost from LPAR B to the protected LPAR A, but how does this compare to the costs and transaction rate when running with no AMS protection?

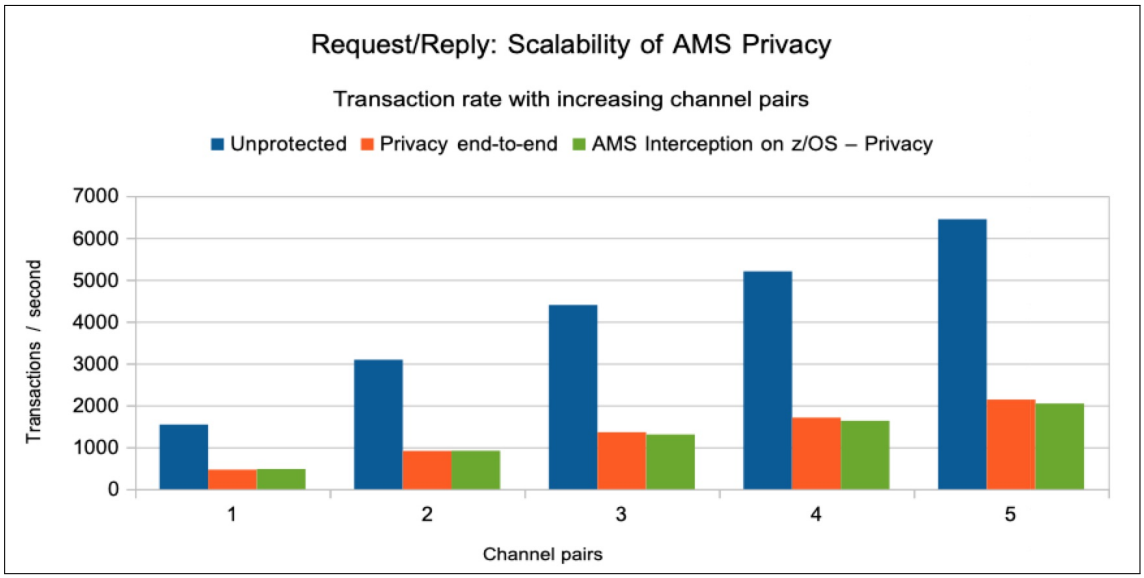


In these measurements, the impact of applying AMS Privacy to a simple request/reply workload is to increase the transaction cost 4 times and reduce the transaction rate to 30% of the unprotected rate.

It should be noted that in the measurements, the application cost appears to grow significantly when using AMS protection. However the *application* in use, has no processing outside of the message processing, and the increased cost of AMS protection would be similar (+70 microseconds for protect and unprotect) regardless of whether the application cost 10 microseconds or 1000 microseconds per "transaction".

Does AMS Message Channel Interception scale similarly to the "no protection" model?

Provided there is sufficient resource, then the AMS Interception on z/OS configuration should show a similar pattern as workload is increased, subject to local lock waiting for serialised resources, for example:



Notes on chart:

The chart shows the achieved transaction rate as more request / reply workloads are added, each using their own set of MQ channels.

Whilst the unprotected configuration appears to scale at a faster rate, the achieved throughput with 5 channel pairs is 4.1 times that of 1 channel pair.

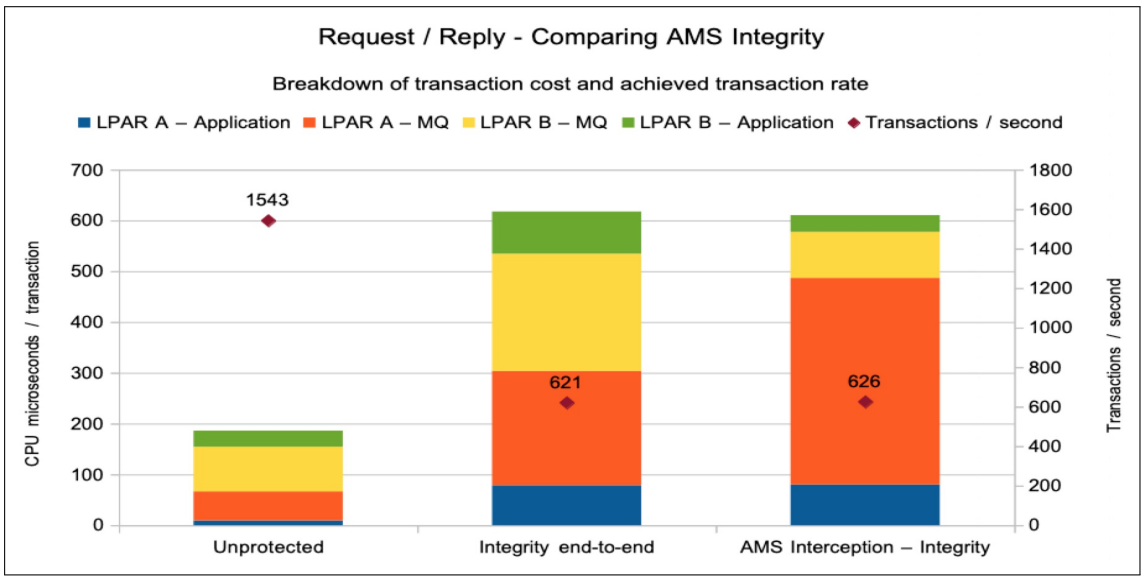
For the AMS Privacy end-to-end configuration, the achieved throughput with 5 channel pairs is 4.6 times that of 1 channel pair.

The AMS Interception for z/OS using privacy configuration achieved 4.2 times the throughput when moving from 1 to 5 channel pairs.

Given the transaction cost for the 5 channel pair measurements is the same for the AMS Privacy end-to-end configuration and the AMS Interception on z/OS with Privacy configuration, the reason for the difference in transaction rate, approximately 5%, is due to all of the cryptographic work being performed on a single LPAR - placing more load on the available Crypto Express coprocessors.

AMS Integrity

Moving from AMS Integrity end-to-end to AMS Interception on z/OS with Integrity protection shows the same net cost effect that we have seen with both Privacy and Confidentiality, as demonstrated in the following chart.



Notes on chart:

This chart shows that there is increased cost in LPAR A's channel initiator address space when moving to the AMS Interception on z/OS configuration. This increased cost is offset by the reduced cost in LPAR B's application.

The net cost attributed to the AMS address remains consistent - again the cost shifting from LPAR B to LPAR A for the message channel interception configuration.

Transaction rate between the AMS Integrity end-to-end and the AMS Interception on z/OS with Integrity configurations is similar.

Two data islands, independently protected using AMS Interception on z/OS

This configuration may appear to be preferable when protecting data in an Enterprise as it can mean that certificate management does not have to be coordinated across data islands, nor do the data islands need to run the same quality of AMS protection. Should the channels use TLS cipher protection, then certificate management would still need to be co-ordinated across the data islands.

There are a number of reasons why using independently protected data islands may not perform as well as an Enterprise-wide data island, including:

- Increased load on cryptographic hardware.
- Increased MQ cost in MQ channel initiator(s) and AMSM regions.
- Increased latency on end-to-end message flows.
- Security - data would be unprotected in a channel initiator dump.

For example, consider the additional processing involved when comparing a two independently protected data islands over a single Enterprise-wide data island when sending a message between 2 queue managers:

- Additional unprotect in sending-side channel initiator prior to send over network.
- Increased MQ cost in MQ channel initiator(s) and AMSM regions.

In each case, these additional pieces of work will add cost to the flow as well as increasing the time taken to flow the messages between applications.

Depending on the AMS quality of protection, and in the case of AMS Confidentiality the key reuse value, these additional unprotect and re-protects may result in the doubling of the use of the associated cryptographic processors when compared to the single Enterprise-wide data island use case.

The measurements in this section compare the following three scenarios:

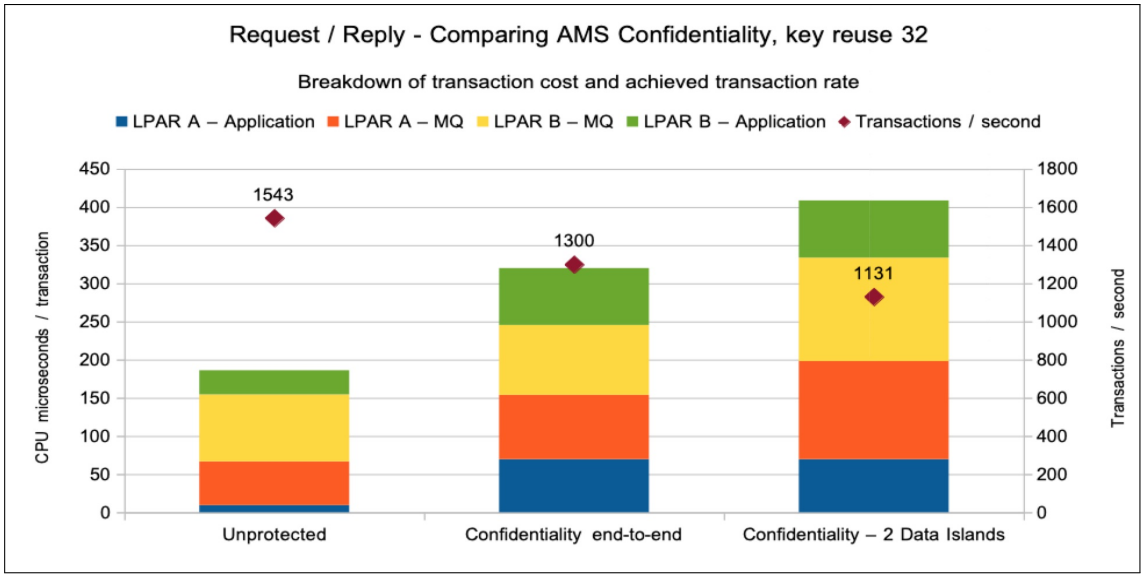
1. Unprotected single data island, request / reply workload.
2. Enterprise-wide single data island, request / reply workload.
3. Two data islands, independently protected using AMS Interception on z/OS.

As with the earlier request / reply workloads, the initial measurements that follow use a single pair of request / reply tasks with a 32KB non-persistent message per channel pair, using the AMS Confidentiality quality of protection and key reuse is set to 32 as this provides a good balance of protection at minimal cost.

Comparison of single channel pair performance

The use of two independently protected data islands, when compared with an end-to-end AMS protection model, adds both cost and latency to the transaction.

The following chart demonstrates the impact on the cost and the achieved transaction rate when implementing two independently protected data islands.



Notes on chart:

MQ costs are based on the average transaction cost in the MSTR, CHIN and AMSM address spaces.

By using multiple data islands, the transaction rate drops by 13% from the achieved rate when using AMS end-to-end protection.

With regards to transaction cost, there is additional cost in both LPARs for primarily the channel initiators, but also the AMSM regions. In this example the additional cost amounts to 90 CPU microseconds, or a 28% increase in transaction cost.

The transaction cost on LPAR A is, unsurprisingly similar to the observed in the earlier measurements when using 2 data islands where AMS Message Channel Interception was applied on LPAR A only.

The costs observed in LPAR B are similar to those costs in LPAR A.

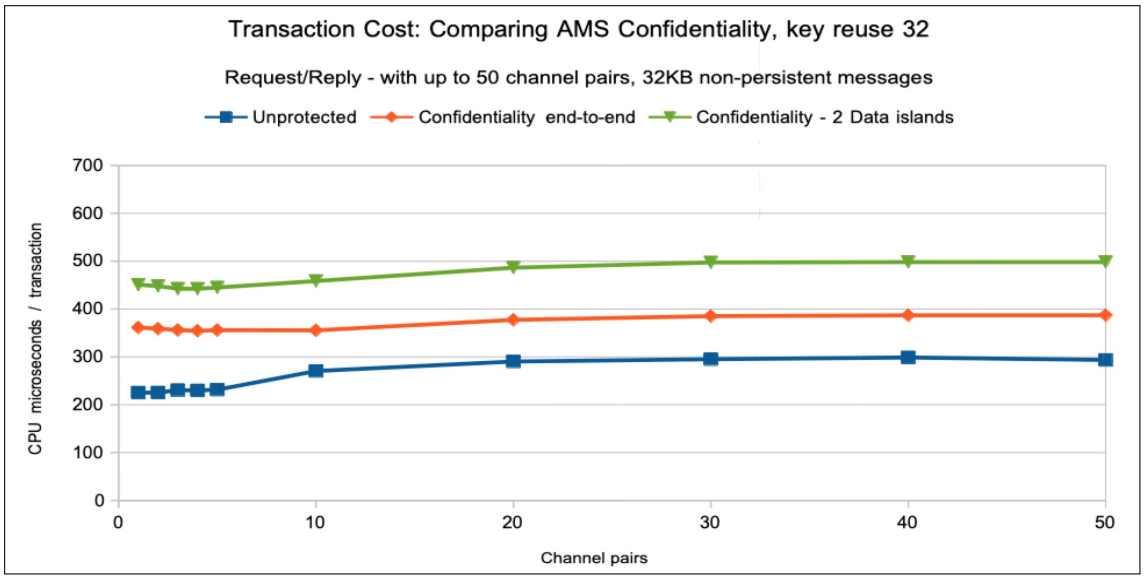
Scalability when using two independently protected data islands

Earlier measurements comparing the performance of multiple data islands, where only one was protected using AMS Interception on z/OS showed a significant difference in the performance when compared with end-to-end AMS protection using the same AMS quality of protection. This was in part due to the increased difference between the adapter CPU time and elapsed time, which was largely due to increased local lock.

With the increased load on **each** channel initiator in the independently AMS Interception on z/OS-protected data island, we see a larger impact on both the transaction cost and the transaction rate when compared with AMS protection end-to-end.

The first chart shows the transaction cost for the AMS Confidentiality 2 data islands is markedly higher than the AMS Confidentiality end-to-end measurement.

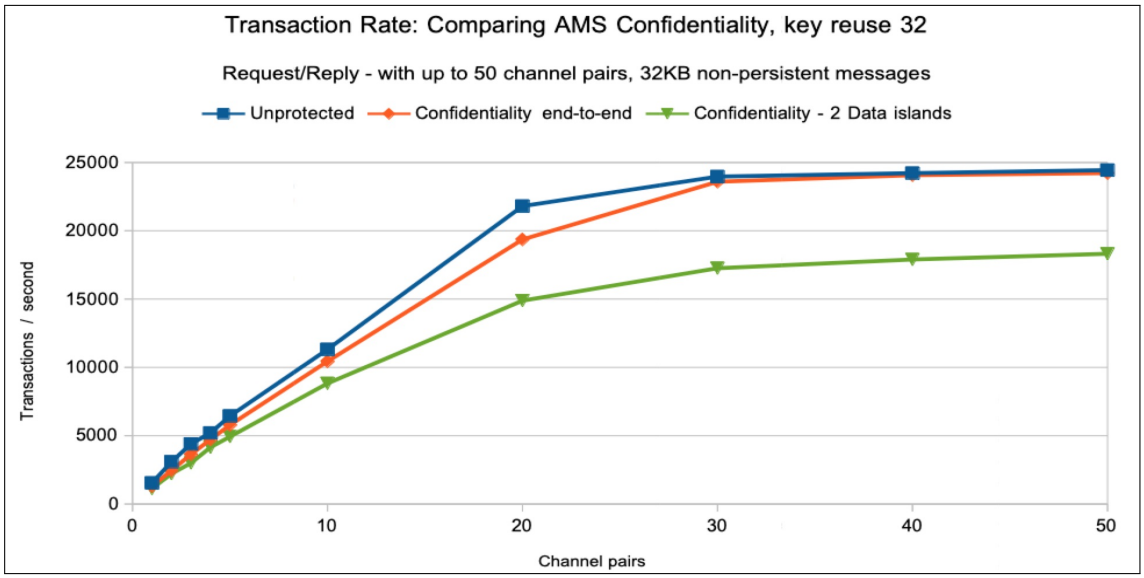
AMS Confidentiality - transaction cost up to 50 channel pairs:



Notes on chart: For the two independently protected data islands, the increase in transaction cost is of the order of 90 to 110 microseconds per transaction over the equivalent configuration where the data is protected end-to-end in a single data island.

In the second chart, showing transaction rate, we see a similar pattern to that in the *"Multiple data islands, one protected with AMS Interception on z/OS"* configuration, but in terms of transaction rate we have seen drop in overall throughput of up to 12% compared to the single AMS Interception on z/OS' throughput.

AMS Confidentiality - transaction rate up to 50 channel pairs:



Notes on chart:

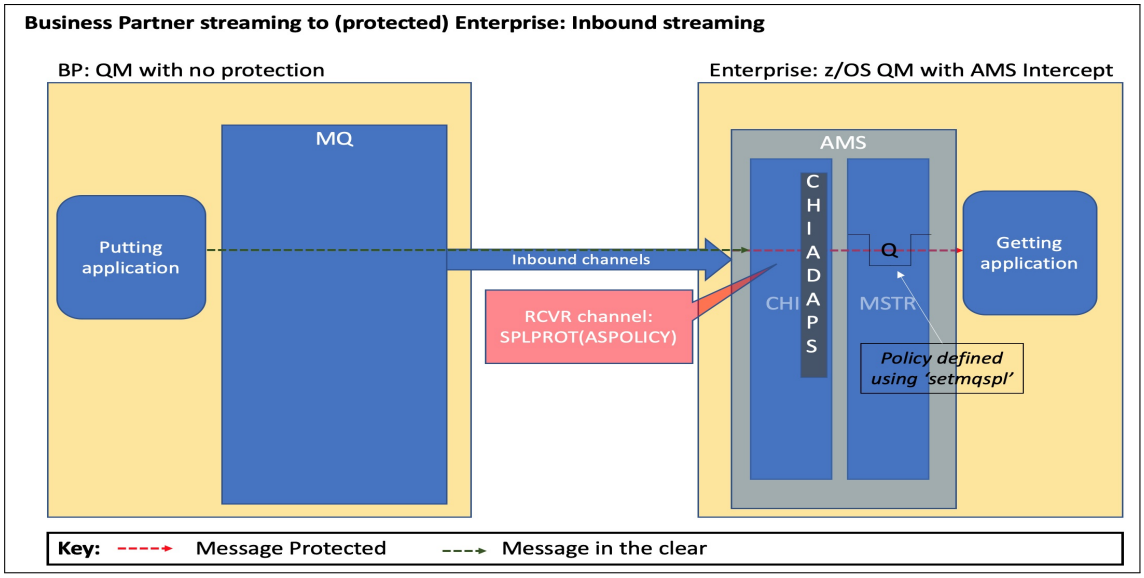
Once more we see the transaction rate for the AMS Interception on z/OS measurements tail off far earlier than in the end-to-end measurement.

In the two independently protected data islands measurements, both channel initiators see increased disparity between adapter CPU time and elapsed time, indicating impact from local lock.

Business Partner streaming to AMS Interception-protected Enterprise

This scenario aims to demonstrate the impact of a business partner streaming messages to an Enterprise-hosted queue manager configured with AMS Interception on z/OS, and is expected to be the most common use of this new function.

The configuration can be represented thus:



In these measurements, the business partner is running on a distributed non-AMS enabled queue manager.

The putting application on the distributed partner is attempting to put messages at a rate of 10,000 2KB non-persistent messages per second.

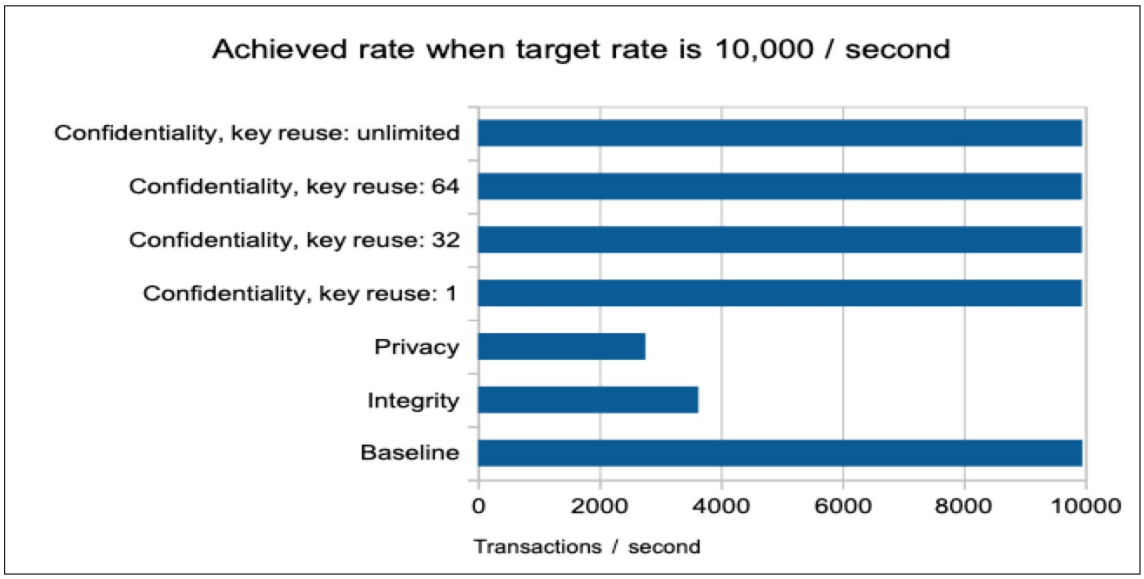
These messages flow over a SDR-RCVR type channel to the AMS-enabled queue manager, where the channel initiator applies AMS protection to the message being put on the target queue.

The authorised application gets and unprotects the message, discarding it after use.

The types of AMS protection used in these measurements are:

- None (baseline).
- Integrity, message is signed using SHA256.
- Privacy, message is signed using SHA256 and encrypted with AES256.
- Confidentiality, message is encrypted with AES256 and key reused:
  - 1 time
  - 32 times
  - 64 times
  - Unlimited.

The first chart in this section shows the achieved throughput rate on the AMS-enabled system.



This chart shows that the target rate of 10,000 messages per second was not sustainable in 2 of the 7 configurations, namely Integrity and Privacy.

When the target rate of 10,000 was not sustainable, the distributed queue manager saw a backlog of messages building up on the queue. In addition, the time spent on the transmission queue (XQTIME) is significantly higher when sending messages to the AMS message channel intercept-enabled queue manager.

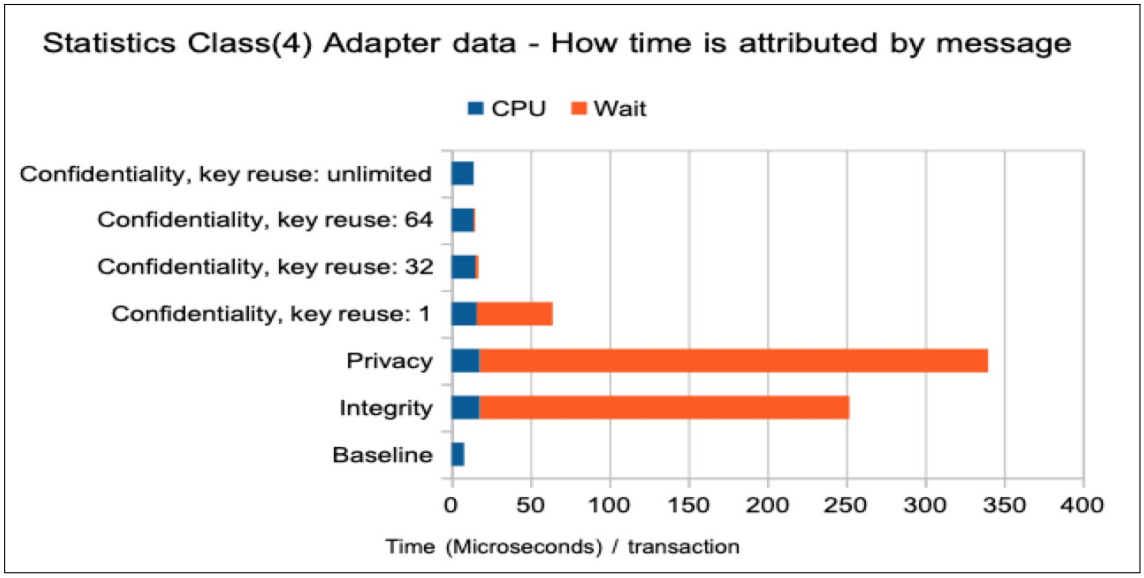
To prevent the putting application failing, 2 changes were made:

1. The MAXDEPTH attribute on the queue on the distributed queue manager was altered to have a higher value than the default of 5000.
2. The application was altered to wait-then-retry when the MQRC_Q_FULL reason code was returned from the MQPUT call.

The backlog of messages in the distributed queue manager was due to the increased time in the adapter task on the AMS-enabled queue manager, whilst protecting the inbound message.

A secondary affect occurred on the z/OS queue manager for the Confidentiality configuration where key reuse of 1 was specified. In this instance, the rate at which the messages were encrypted and put to the queue out-paced the rate at which the getting task was able to remove the messages from the queue. This resulted in page set I/O, and eventually the arriving message rate would be slowed to allow for immediate writes to page set. This was alleviated with larger buffer pools in the z/OS queue manager.

The second chart represents the MQ statistics class(4) data for the adapter task for each configuration.



In the above chart, the Integrity, Privacy and Confidentiality with key reuse 1 configurations show significant time in wait-state. This time spent waiting means the adapter task is blocked from processing other work.

Two of these three configurations are the same as the two configurations that were unable to sustain 10,000 messages per second.

The wait time is calculated from the average elapsed time minus the average CPU time per adapter request.

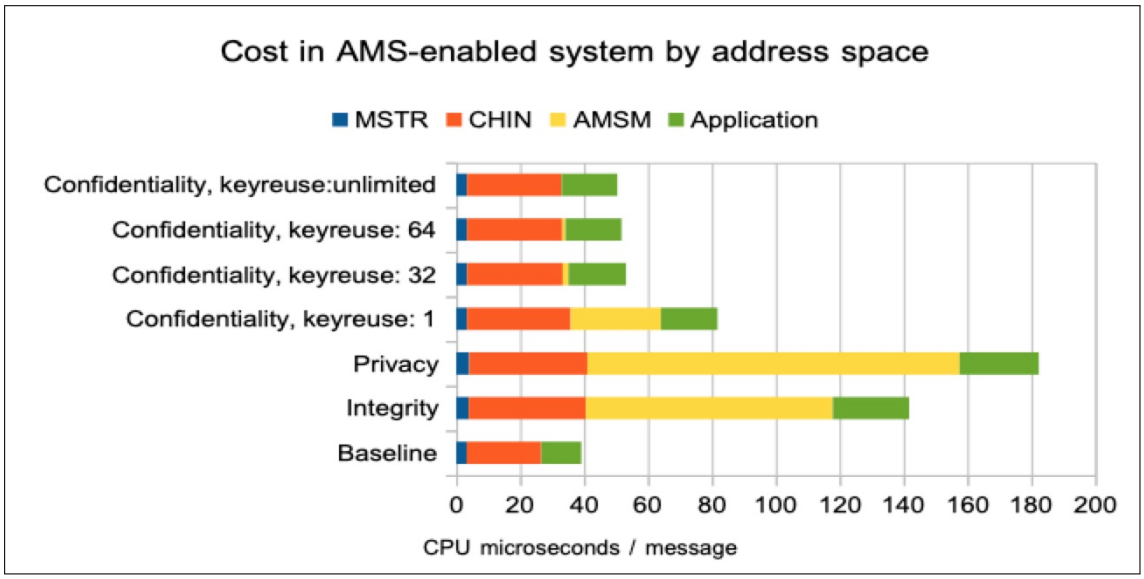
In this scenario, for *Privacy*, *Integrity* and *Confidentiality key reuse 1* configurations, this wait time is largely spent in the cryptographic certificate processing, some of which is charged to the AMSM address space.

Note that the encryption is performed on CPACF (CP Assist for Cryptographic Functions) and is recorded as CPU time, which is why the wait time on the confidentiality configurations reduces as key reuse increases.

The final chart in this section shows the cost per message in the MQ address spaces - MSTR, CHIN and AMSM as well as the cost in the getting application.

Note that the getting application contains minimal processing apart from the MQGET, so the impact of the AMS protection appears more significant than in an application where MQ forms 10% of the total cost of the transaction.

Chart: Breakdown of cost on z/OS when streaming 2KB messages at 10,000 per second:



When the channel initiator is processing AMS Confidentiality messages at the same rate as unprotected messages, there is an increase in cost per message of 7 CPU microseconds, i.e the cost of AMS Confidentiality protection being applied in the *channel initiator* to a 2KB message is 7 CPU microseconds.

The application address space costs were impacted by the type of protection.

- Integrity added 11.5 CPU microseconds per MQGET.
- Privacy added 12.5 CPU microseconds per MQGET.
- Confidentiality added 4.5 CPU microseconds per MQGET.

Summary

AMS Interception on server to server message channels provides a solution to a specific problem, namely:

- Ensuring an Enterprise or line of business can protect their MQ data at rest without mandating their partner(s), whether internal or external, implement the same quality of protection, or at the same point in time.

The impact of AMS Interception on z/OS will vary depending on whether you are currently AMS-enabled or not.

It is worth considering whether the increased flexibility from AMS Interception on z/OS outweighs the lower AMS end-to-end protection costs.

Implementing AMS Interception on z/OS may require reviewing your CHIADAPS and CHIDISPS settings on the proposed queue managers in order to minimise the impact on any non-AMS Interception for z/OS workloads.

If the channel initiator is **only** processing AMS Interception on z/OS-type work, increasing the number of adapter tasks may result in increased local lock time rather than improving throughput. However if the channel initiator is processing a mixture of AMS Interception on z/OS-type work, AMS end-to-end work and non-AMS work, increasing the number of adapter tasks may benefit the overall throughput.

Using AMS Interception on z/OS may increase the load on your cryptographic hardware - review the RMF Cryptographic report to determine there is sufficient capacity available for the expected increase in cryptographic work.

Applying AMS protection over server-to-server channels, whether in an end-to-end or a message channel interception configuration, will add both cost and latency to the end-to-end transaction. In a system that is constrained for CPU or cryptographic resource, the use of AMS Interception on z/OS could be a significant factor in a change in the behaviour of the workload.

Given the impact that AMS protection over server-to-server message channels can have on throughput and latency, it is always worth reviewing your settings for:

- Maximum queue depth.
- Maximum message length - AMS protected messages will be larger than their unprotected counterparts.
- Expiry - with increased latency on AMS Interception on z/OS, are messages going to be expired before they can reach their target destinations?

Finally, ensure the user ID for the channel initiator applying or removing AMS protection is authorised to perform this processing.

Chapter 8

Aspera fasp.io gateway

IBM MQ Advanced for z/OS VUE version 9.2 only

The IBM® Aspera fasp.io gateway provides a fast TCP/IP tunnel that can significantly increase network throughput for IBM MQ.

The Aspera gateway can be used to improve the performance of queue manager channels. It is especially effective if the network has high latency or tends to lose packets, and it is typically used to speed up the connection between queue managers in different data centres.

However, for a fast network that does not lose packets there is a decrease in performance when using the Aspera Gateway, so it is important to check network performance before and after defining an Aspera Gateway connection.

A queue manager running on any entitled platform can connect through an Aspera gateway. The gateway itself is deployed on Red Hat®, Ubuntu Linux® or Windows. This means that the gateway can be deployed on Linux on Z or in a z/OS Container Extension (zCX).

When deploying the Aspera fasp.io gateway in a z/OS Container Extension, the CPU used is largely eligible for offload to zIIP. In our measurements, it was typical to see in excess of 98% of the zCX costs to be eligible for zIIP offload.

When deploying the fasp.io gateway other than in zCX, consider that there may be additional latency when communicating between the MQ channel initiator and the gateway.

Aspera fasp.io gateway performance highlights

The following section details some of the performance environments where use of the Aspera fasp.io gateway can provide significant improvements to throughput, in particular:

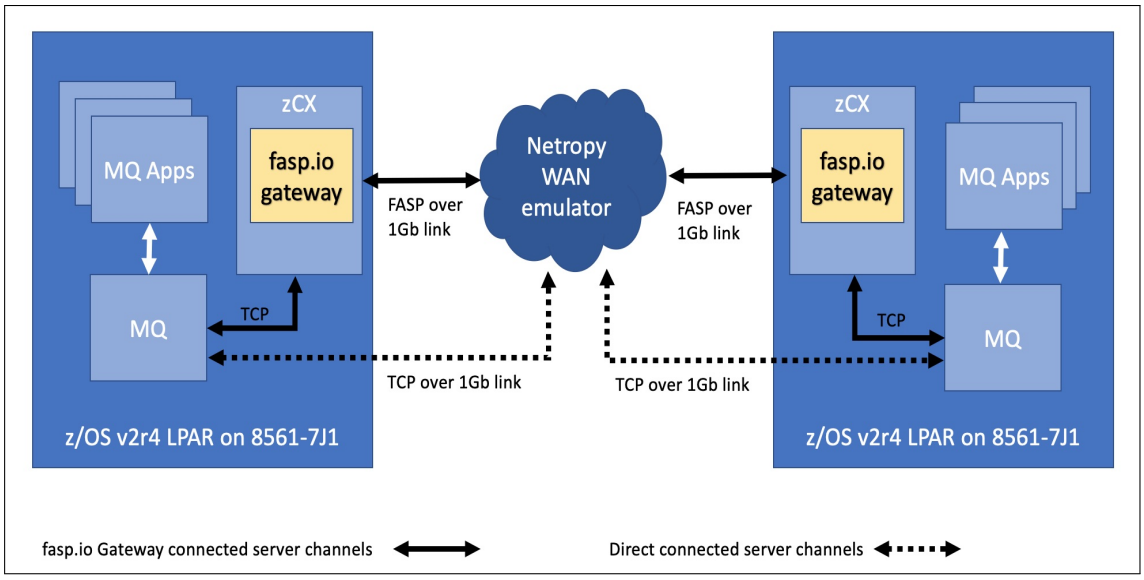
- Streaming-type workloads with large messages - up to 72 times throughput improvement.
- Streaming-type workloads with small messages - up to 11 times throughput improvement.
- Request/Responder-type workloads with medium messages - up to 60% throughput improvement.
- Reduction in overall transaction cost with the use of zIIPs.
- Reduced impact from unreliable networks.

Aspera fasp.io gateway test configuration

The Aspera fasp.io gateway benefit can be seen in networks with high latency and/or a tendency to lose packets. To simulate these environments, we use a Netropy 10G4 WAN emulator, which we configure dynamically to simulate a range of network configurations.

In all tests shown in this section, the following test topology is used.

Chart: Test topology when using the Aspera fasp.io Gateway



Note:

- All communication, whether using the Aspera fasp.io gateway or not, is routed through the Netropy WAN emulator.
- All communication is limited to 1Gb / sec.
- The fasp.io gateway on each LPAR is running in a zCX.

Workload configuration

There are two models of workload that we have considered for use with the Aspera fasp.io Gateway.

1. Uni-directional (streaming) - send/receive workload, simulating Queue Replication with 1 capture task and 1 apply task. These tests are modelled on the streaming tests in the Regression section of this document.
2. Bi-directional (Request/Responder) workload utilising 1 pair of server channels with multiple applications. These tests are modelled on the request/reply workloads using 1 channel pair in the Regression section of this document.

Each of these workloads is measured in 5 configurations, with different latency and packet loss, as below:

- **N0:** 0ms network latency (no packet loss)
- **N1:** 5ms network latency (no packet loss)
- **N2:** 25ms network latency (no packet loss)
- **N3:** 40ms network latency (0.1% packet loss)
- **N4:** 50ms network latency (0.5% packet loss)

An additional configuration, N5, is used on occasion to demonstrate the impact from packet loss on high latency networks.

- **N5:** 50ms network latency (no packet loss)

Note: Latency is applied to both directions, so request/responder workloads will be affected by 2 times the latency.

Aspera fasp.io gateway streaming workload

The streaming workloads use 2 message sizes, 10KB and 1MB, to simulate online and batch processing models respectively.

In each case, the MQ channels are configured to have a batch size of 200 and sufficient buffers are available to minimise impact from I/O to MQ page set on both the sending and receiving queue managers.

10KB streaming workload

In a streaming workload, smaller messages, such as 10KB, tend to result in worse performance when using the Aspera fasp.io gateway on relatively low-latency networks.

As the latency increases, such as in the N2 configuration, parity is achieved with the TCP/IP configuration, but as the latency and packet loss begins to occur, the fasp.io gateway shows significant improvement in throughput.

- **N3**: 4.8x TCP performance
- **N4**: 11.75x TCP performance

The following charts shows the sustained channel throughput in MB/second across a single MQ channel and the cost per transaction.

The first chart demonstrates the improved performance as the network latency and packet loss increases. The second chart compares the cost of running the workloads and demonstrates the benefits that may be achieved provided sufficient zIIP capacity is available.

Chart: Achieved channel throughput when streaming 10KB persistent messages.

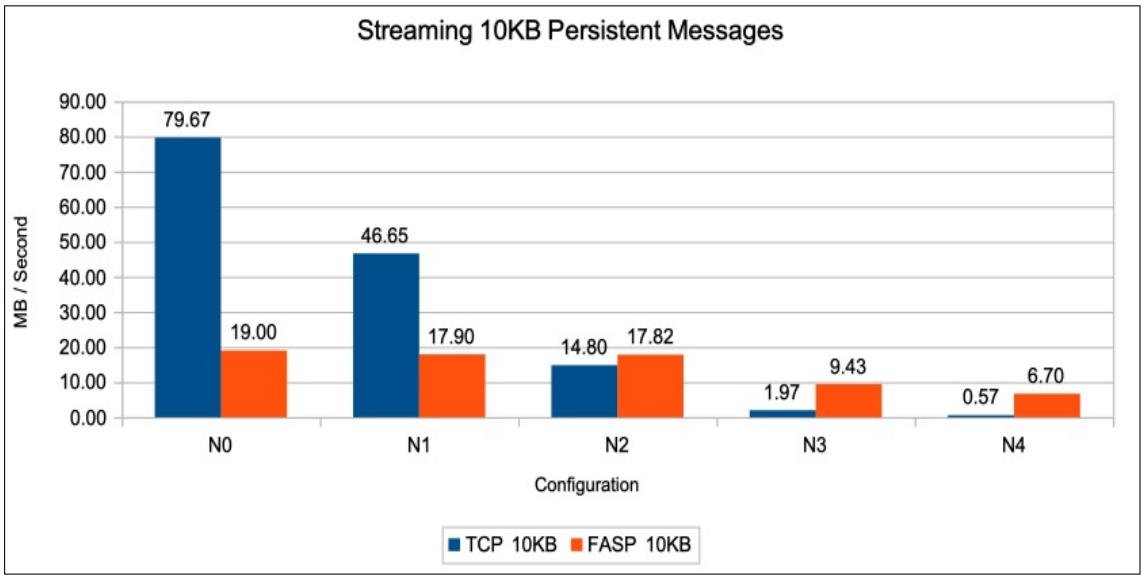
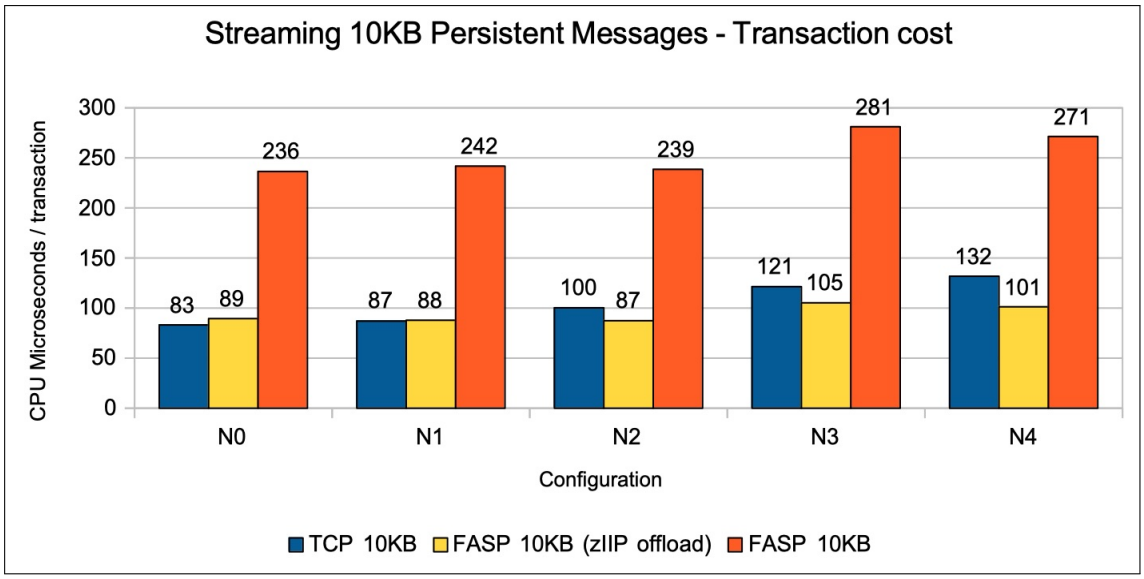


Chart: Actual transaction cost when streaming 10KB persistent messages.



Notes on transaction cost chart:

The transaction cost is calculated using the data from the RMF Workload report and includes the costs for the MQ MSTR, channel initiator, zCX, TCP/IP and application address spaces.

The costs reported for the "FASP 10KB (zIIP offload)" columns are based on the reported zIIP-eligibility for those address spaces and are the optimum costs based on those calculations.

1MB streaming workload

In a streaming workload, larger messages, such as 1MB with larger achieved batch sizes (XBATCHSZ) tend to result in performance that is comparable with TCP/IP even in low latency networks, and far exceed the performance of TCP/IP on high-latency networks or those suffering from packet loss.

- **N1**: +17% TCP performance
- **N2**: 3.6x TCP performance
- **N3**: 22.4x TCP performance
- **N4**: 72.25x TCP performance

The following charts shows the sustained channel throughput in MB/second across a single MQ channel and the cost per transaction.

The first chart demonstrates the improved performance as the network latency and packet loss increases. The second chart compares the cost of running the workloads and demonstrates the benefits that may be achieved provided sufficient zIIP capacity is available.

Chart: Achieved channel throughput when streaming 1MB persistent messages.

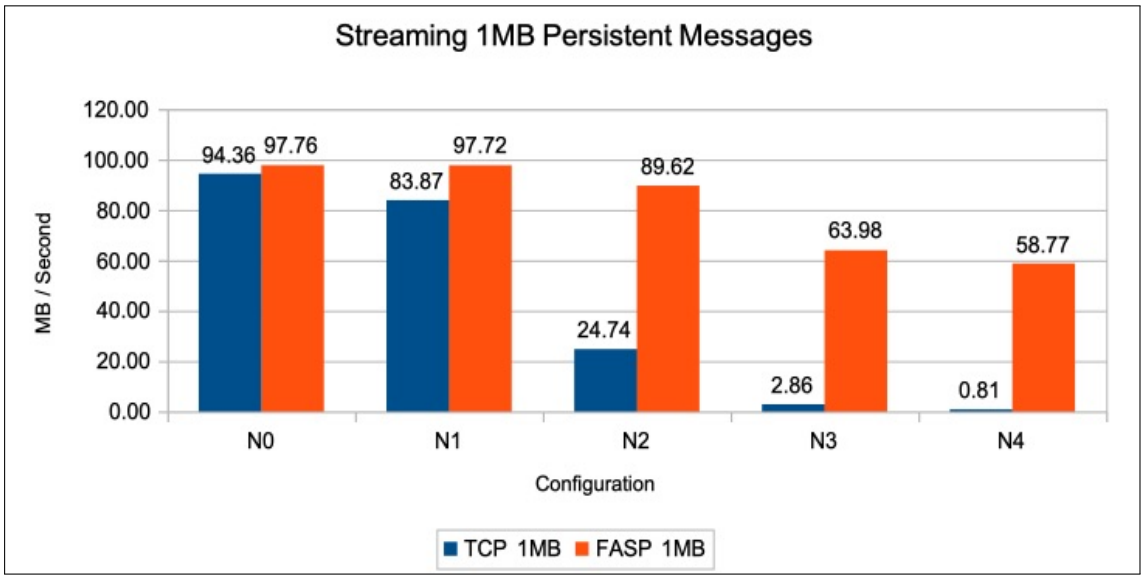
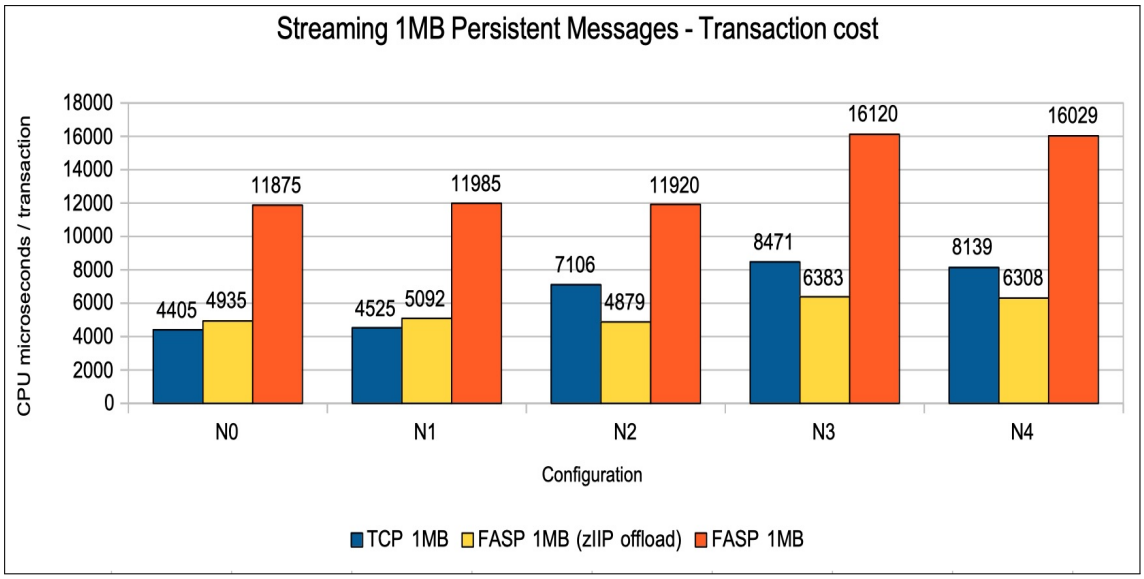


Chart: Actual transaction cost when streaming 1MB persistent messages.



Notes on transaction cost chart:

The transaction cost is calculated using the data from the RMF Workload report and includes the costs for the MQ MSTR, channel initiator, zCX, TCP/IP and application address spaces.

The costs reported for the "FASP 1MB (zIIP offload)" are based on the reported zIIP-eligibility for those address spaces and are the optimum costs based on those calculations.

Aspera fasp.io gateway request/responder workload

32KB request/responder

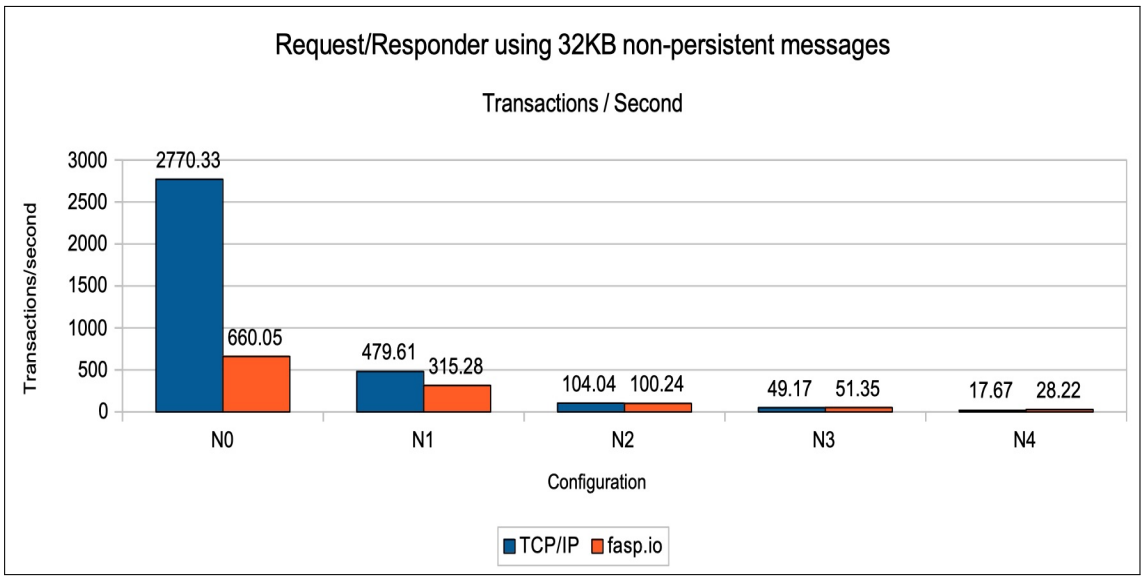
Request/Responder workloads do not benefit from using the fasp.io gateway to the same extent as streaming workloads.

There are many factors which impact the performance of a request/responder workload when routing the network traffic through a pair of fasp.io gateways, and these include:

- Batch size - this is the number of messages in the batch, and the amount of data flowing over the channel between end-of-batch flows.
- Message size.

In the following example, the measurement uses 32KB messages where a small number of requester applications each put a message and wait for a reply of equivalent size.

Chart: Achieved transaction rate with 32KB messages in request/responder workload.

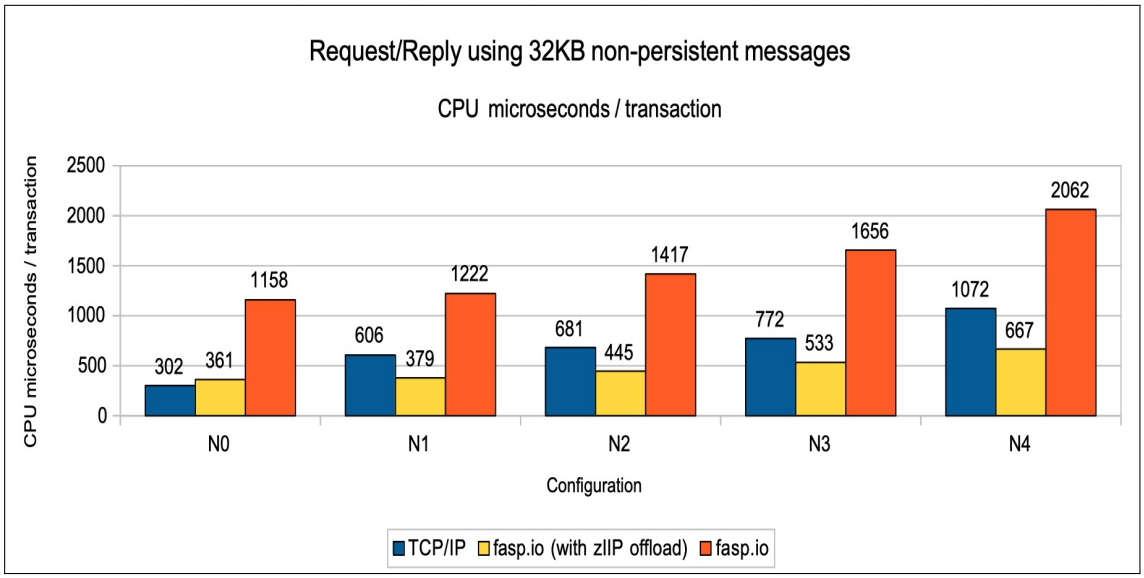


Notes on chart:

At configurations N2 and N3, the fasp.io configuration is able to achieve parity with the TCP/IP configuration. As both latency and packet loss increases, the benefits of the Aspera fasp.io gateway become more obvious, such that with configuration N4, 50ms latency (in each direction) and 0.5% packet loss, the throughput improves by 60%.

When zIIP offload is not available, the fasp.io gateway hosted on zCX configuration shows costs that are between 2-3 times that of the TCP/IP configuration. Providing sufficient zIIP resource can bring CPU savings in excess of 20% over the comparable TCP/IP configuration.

Chart: Actual transaction cost with 32KB messages in Request/Responder workload.



Notes on chart:

The chart shows the fasp.io cost in two modes; when running the gateway on zCX using general purpose processors and again when the zIIP-eligible work is fully offloaded. Once network latency is introduced, provided there is sufficient capacity to offload the zCX work to zIIP, the fasp.io workload can offer a reduction in overall CPU cost.

The increase in cost in the TCP/IP configuration as latency increases is primarily incurred in the network layer, whether in the TCP/IP address space or MQ's dispatcher tasks in the channel initiator.

The increase in cost in the fasp.io configuration as latency increases is again primarily in the network layer, but in this configuration the cost is incurred by TCP/IP and the zCX address spaces - and as a result of zCX being largely zIIP eligible, the overall transaction cost is lower than the equivalent TCP/IP measurement when zIIP-offload is fully utilised.

How much impact is there from packet loss on high latency networks?

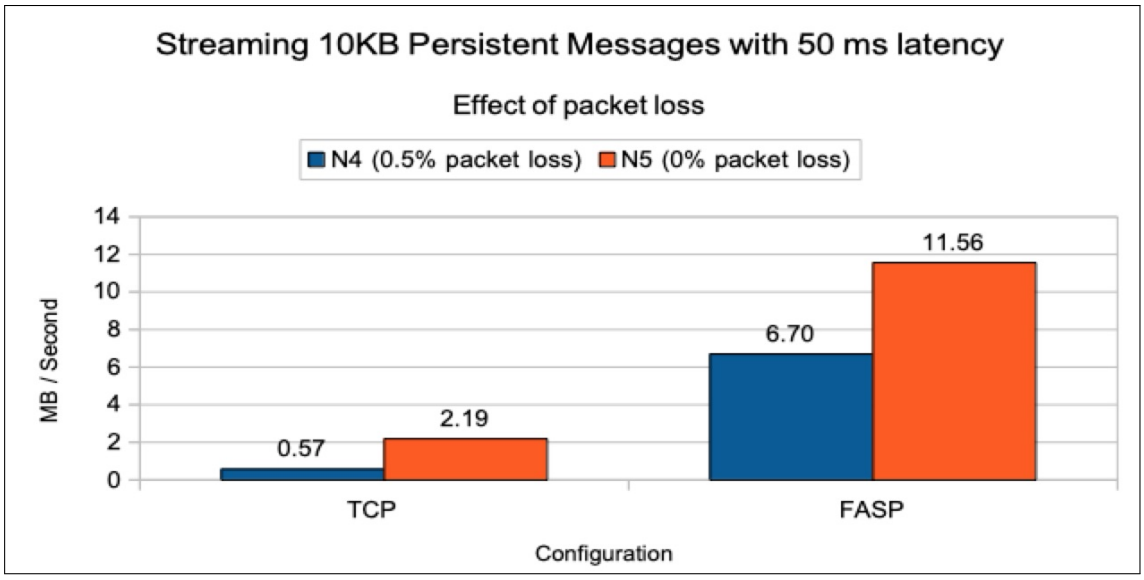
TCP/IP is a conversational-type protocol such that packet loss is noticed relatively quickly and results in data being re-sent. This can result in TCP/IP being unable to optimise its send and receive buffers to be able to benefit from dynamic right sizing.

By contrast, the fasp.io gateway uses a proprietary user defined protocol (UDP) which sends the data and at certain points performs scrutiny on the received data and in the event of lost data, requests that data is re-sent.

As a result, networks that experience packet loss see a larger relative impact to the TCP/IP performance than the fasp.io gateway performance.

If the streaming tests are repeated such we can compare the N4 (0.5% packet loss) and N5 (0% packet loss) configurations, we observe the performance changes as per the following 2 charts:

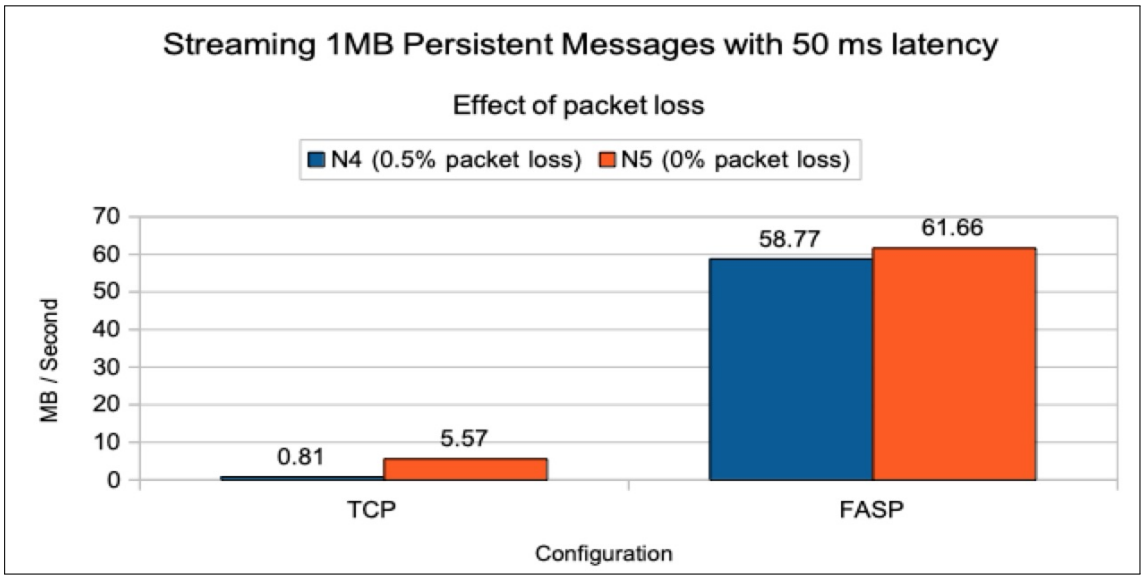
Chart: Zero loss channel throughput when streaming 10KB persistent messages.



For the N5 configuration with 0% packet loss:

- TCP/IP throughput increased from 0.57 MB/second to 2.19 MB/second - an increase of 3.8x or 1.6 MB/second.
- fasp.io throughput increased from 6.7 MB/second to 11.56 MB/second - an increase of 1.72x or 4.86 MB/second.

Chart: Zero loss channel throughput when streaming 1MB persistent messages.



For the N5 configuration with 0% packet loss:

- TCP/IP throughput increased from 0.81 MB/second to 5.56 MB/second - an increase of 6.8x or 4.75 MB/second.
- fasp.io throughput increased from 58.77 MB/second to 61.66 MB/second - an increase of 4% or 2.89 MB/second.

Does achieved batch size affect Aspera fasp.io gateway performance?

In our measurements, the batch confirm flow was the major inhibitor to the fasp.io performance, particularly on low-latency networks. You can see the impact in the NETTIME value, which shows the network cost of the flow.

The following table shows the expected NETTIME values based on the latency and the reported NETTIMEs for all five configurations.

Table: NETTIME - Expected vs Achieved

Configuration	Expected	TCP/IP actual	fasp.io actual
N0	< 1	1	5-6
N1	10	10	16-18
N2	50	50	54-58
N3	80	80-100	90-97
N4	100	120-180	110-130

Note on table: The expected, actual TCP/IP and actual fasp.io times are reported in milliseconds and are as reported by the DISPLAY CHSTATUS MQSC command.

The expected latency is double the latency configured, as the confirm flow requires data to flow in both directions.

In the fasp.io configurations, the NETTIME was typically 4+ milliseconds longer than the equivalent TCP/IP configuration - until configuration N3 where the impact of packet loss affected the TCP/IP configuration far more significantly.

Note that it is not clear whether the additional time reported by NETTIME is an artifact of hosting the gateway on zCX or is directly related to the fasp.io gateway.

Mitigating impact of BATCHSZ on fasp.io gateway

In the higher latency configurations, the achieved batch size (XBATCHSZ) indicated that the batches were not full. Given the significant impact of the end of batch flow, ensuring each batch contained more messages resulted in the transaction rate more than doubling, e.g. for the request/responder workload using 32KB messages, the N4 configuration originally achieved 28 transactions per second with XBATCHSZ(34), but when achieving XBATCHSZ(50) the transaction rate increased to 63 (2.25x) with the side effect of increased latency per round-trip.

This can be achieved in a number of ways:

- Increase BATCHSZ - only of benefit if the XBATCHSZ already matches the BATCHSZ. If the channel sends a wide range of message sizes, it may be beneficial to use the BATCHLIM attribute to ensure that the amount of data (MB) is not excessive such that the target buffer pool is sufficiently large to hold at least one full batch without writing to page set.
- Increase BATCHINT - to keep the channel batch open for longer. Be aware that increasing this can result in increased response times because the batches last longer and messages will remain uncommitted for longer.

Does message size affect Aspera fasp.io gateway performance?

Yes, but to a far less extent than the achieved batch size.

In a request/responder-type workload with a 5 millisecond latency, as per configuration N1, we found that messages up to 32KB achieved a transaction rate approximately 33% lower when using the fasp.io gateway rather than TCP/IP.

Messages between 64KB and 100KB achieved a transaction rate 12-20% higher in the TCP/IP configuration when measured with 5 milliseconds of latency (as per N1).

How does channel compression affect the Aspera fasp.io gateway?

IBM z15 introduced on-chip compression, which replaced the optional zEnterprise Data Compression feature on IBM z14 and earlier platforms.

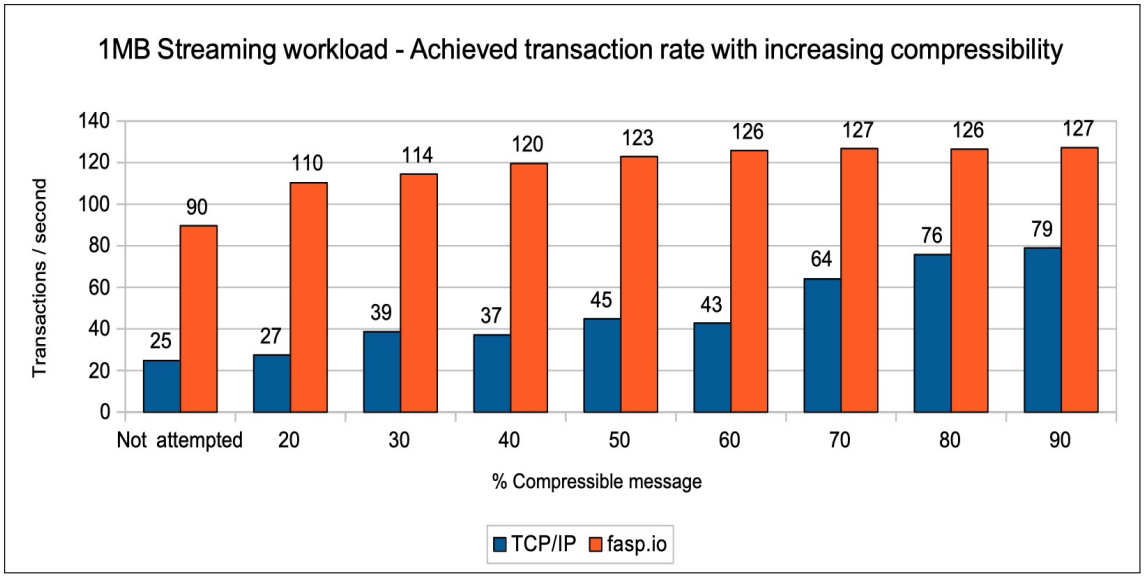
Since IBM MQ version 8.0, MQ has supported the use of hardware compression on channels, and this can be enabled by setting `COMPMSG(ZLIBFAST)`. It should be noted that if hardware compression or the data to be compressed falls below the minimum threshold for hardware compression, then compression and decompression will be performed by standard general purpose processors by software functions. Further detail relating to the performance of channel compression on z15 can be found in the IBM MQ for z/OS on z15 performance report.

One of the more common use-cases for channel compression is on high latency networks such as those where the fasp.io gateway also provides significant benefit, so can channel compression be used in conjunction with the fasp.io gateway? The answer is yes - and indeed in our measurements the threshold for seeing benefits from channel compression is lower than when compressing and sending data over networks without the gateway.

In the following examples, the 1MB streaming workload is run, where the message payload is incrementally more compressible - from 20 to 90% compressible, i.e. at its most compressible, the 1MB message results in 100KB of data being sent over the network.

For simplicity's sake, the measurements are run only on configuration N2 (25 milliseconds of latency).

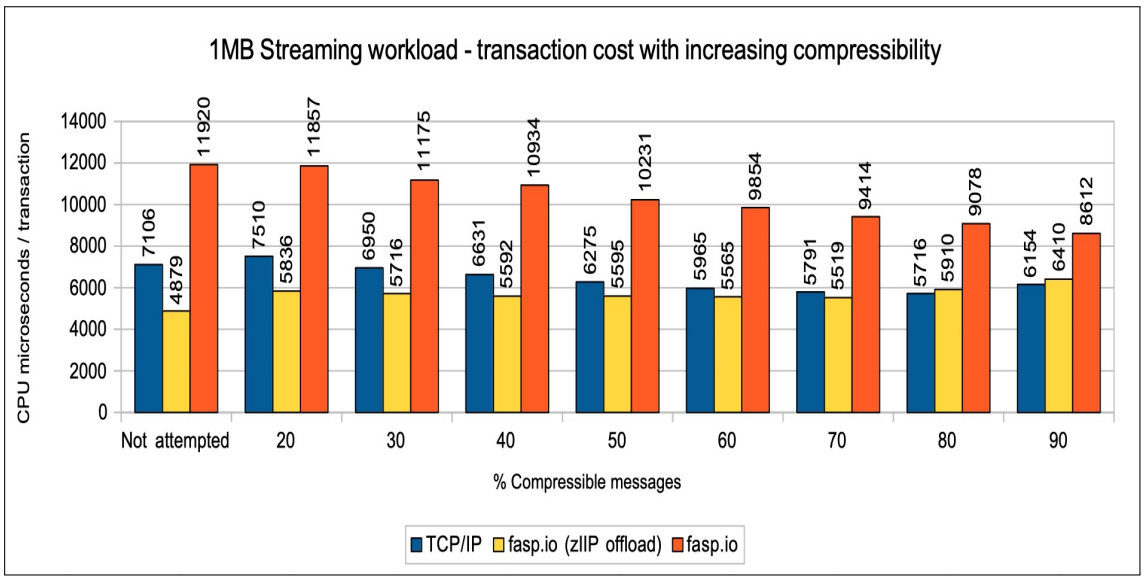
Chart: Achieved transaction rate - Streaming 1MB workload with 25 millisecond latency and increasing compressibility.



Note that the "not attempted" column is the baseline measurement where `COMPMSG(NONE)` is defined.

With regards to transaction rate, the TCP/IP measurements see a 3 times improvement in throughput with messages that are highly compressible, but even those highly compressible messages were unable to achieve more than 62% of the equivalent fasp.io measurement.

Chart: Transaction cost - Streaming 1MB workload with 25 millisecond latency and increasing compressibility.



Notes on transaction cost chart:

The transaction cost is calculated using the data from the RMF Workload report and includes the costs for the MQ MSTR, channel initiator, zCX, TCP/IP and application address spaces.

The costs reported for the "fasp.io (zIIP offload)" columns are based on the reported zIIP-eligibility for those address spaces and are the optimum costs based on those calculations.

Appendix A

Regression

When performance monitoring both IBM MQ Advanced for z/OS VUE version 9.2 and IBM MQ for z/OS version 9.2, 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 V9.0 (V900), V9.1 (V910) and V9.2 (V920) 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

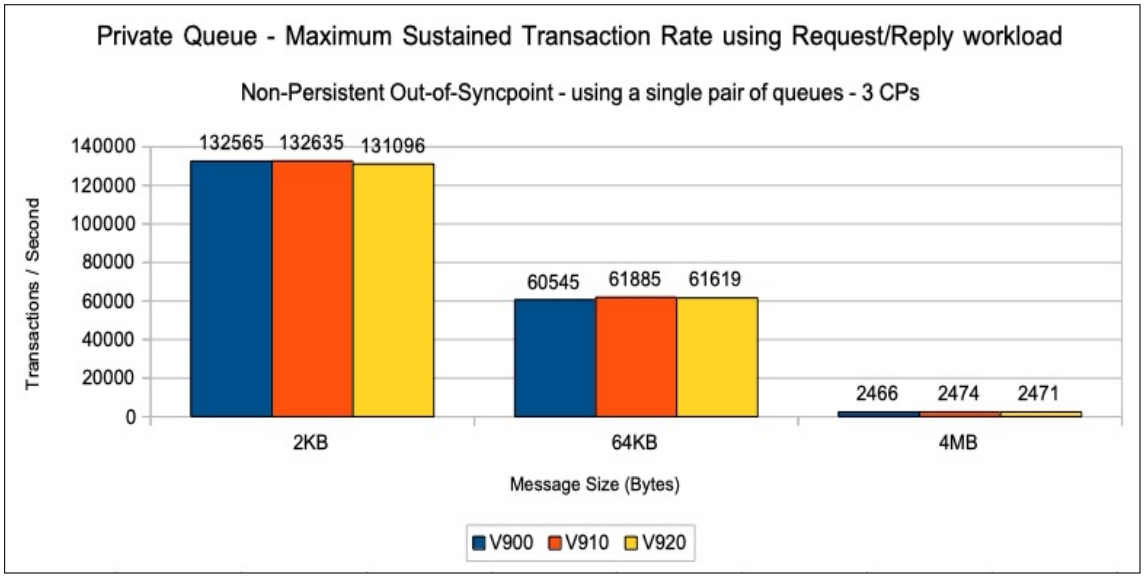
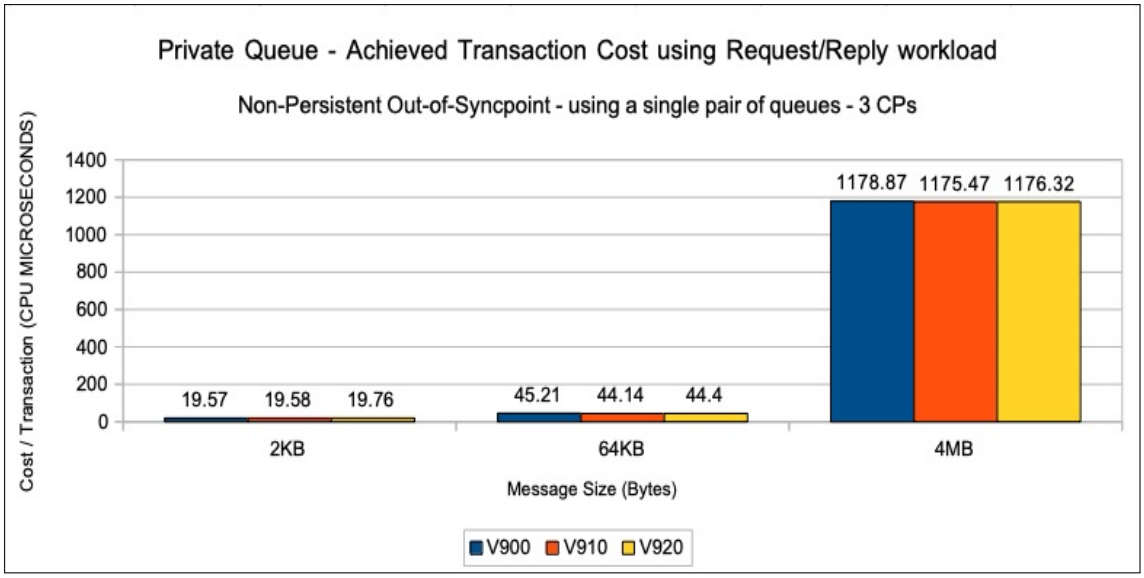


Chart: Transaction cost for non-persistent out-of-syncpoint workload



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

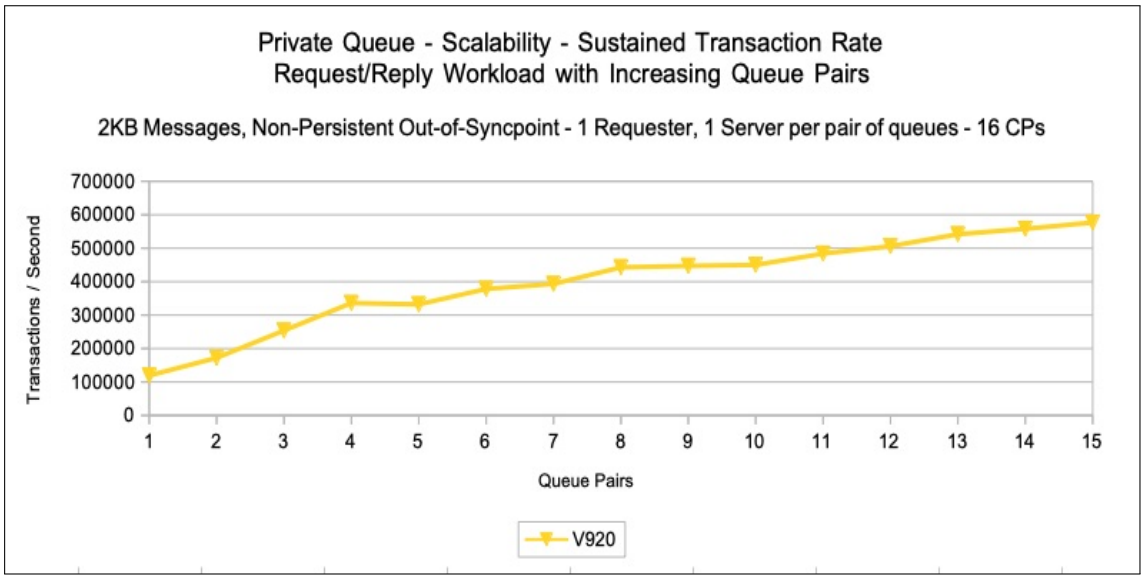
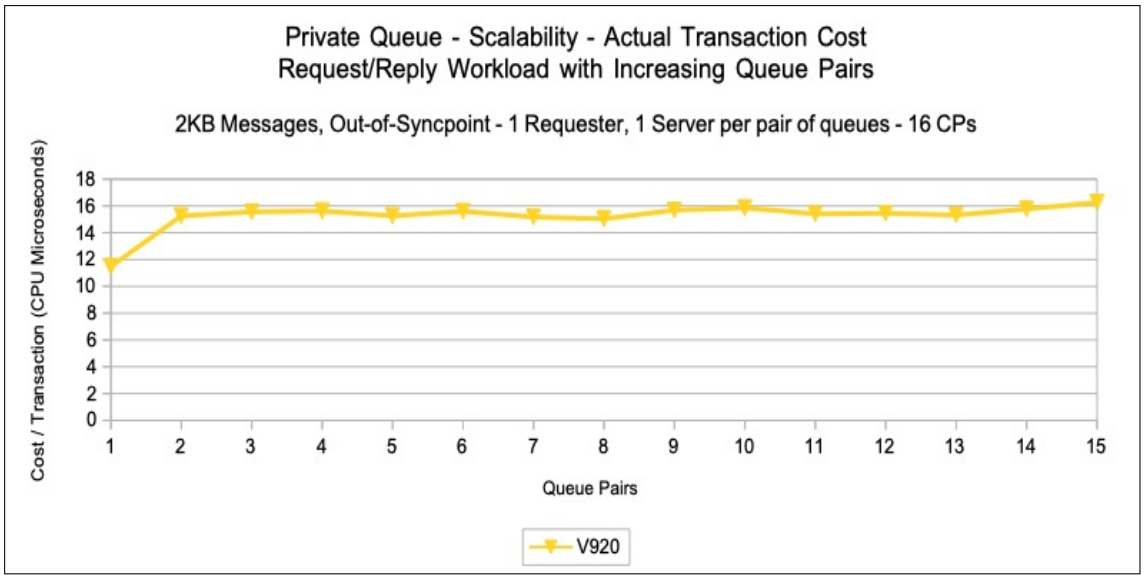


Chart: Transaction cost for non-persistent out-of-syncpoint scalability workload



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 V900, V910 and V920 is comparable, the charts show only V920 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

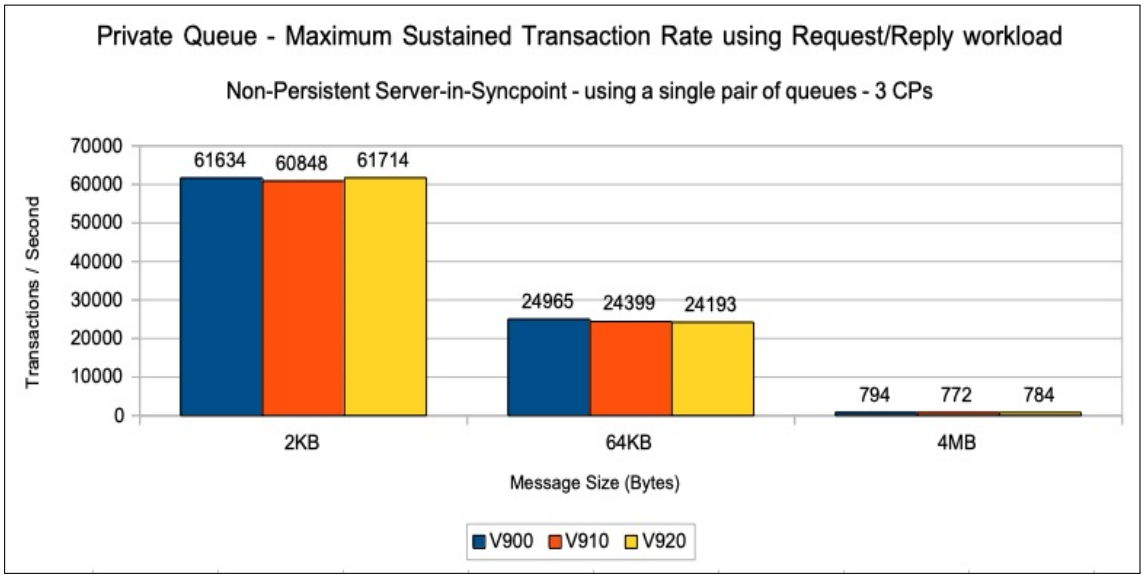
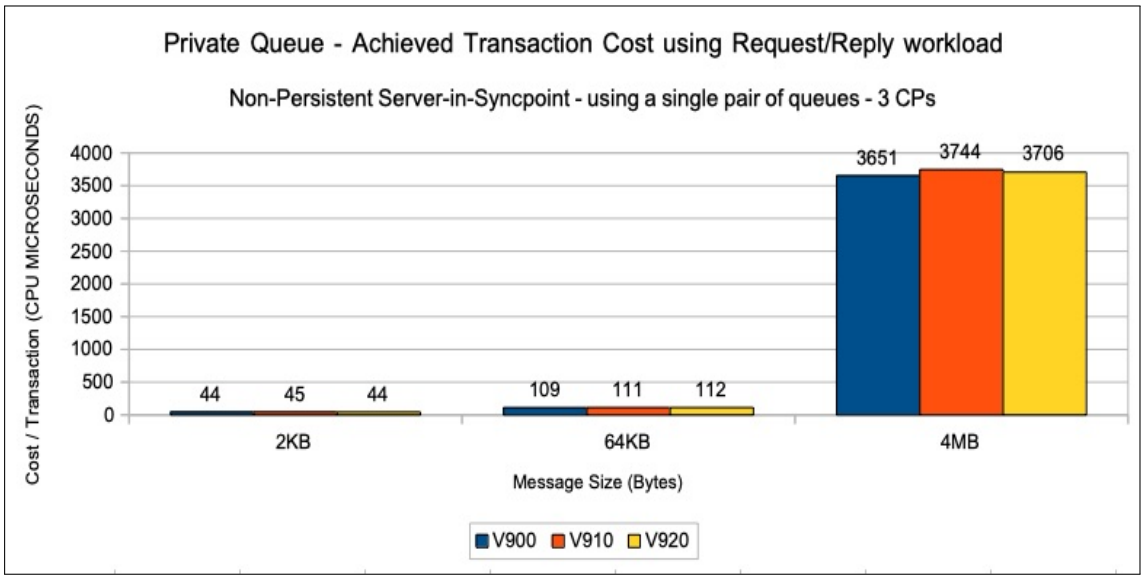


Chart: Transaction cost for non-persistent in syncpoint workload



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 V900, V910 and V920 is comparable, the charts show only V920 data for the purpose of clarity.

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

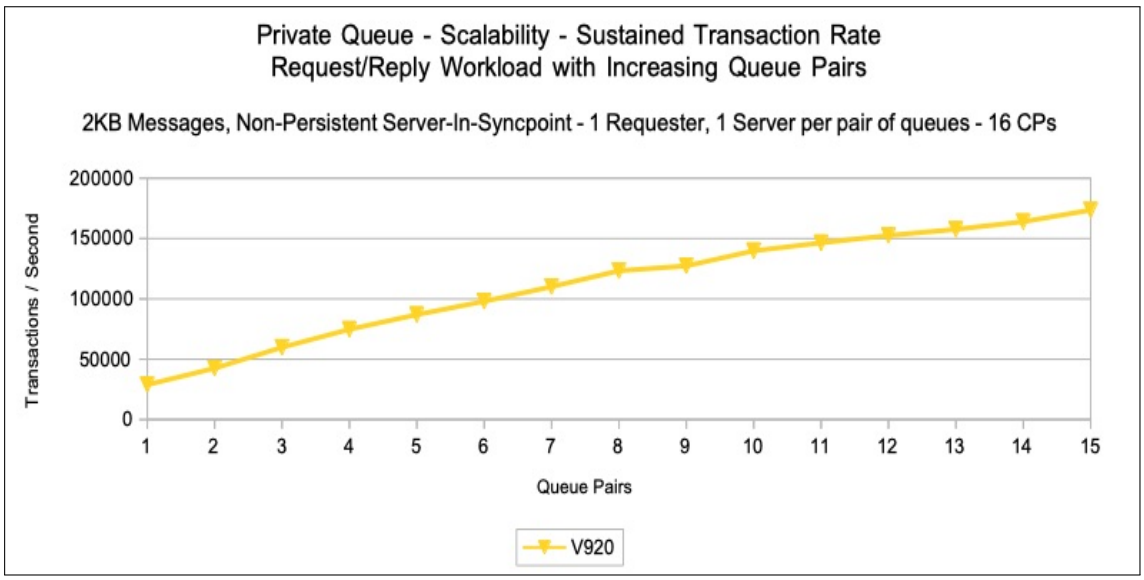


Chart: Transaction cost for non-persistent in syncpoint scalability workload with 2KB messages

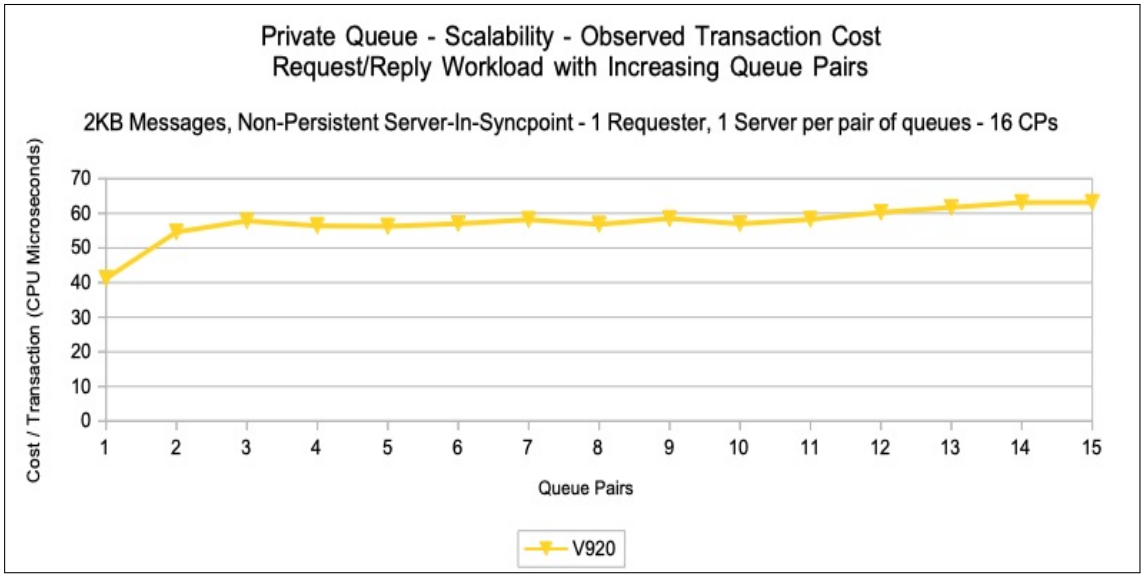


Chart: Transaction rate for non-persistent in syncpoint scalability workload with 64KB messages

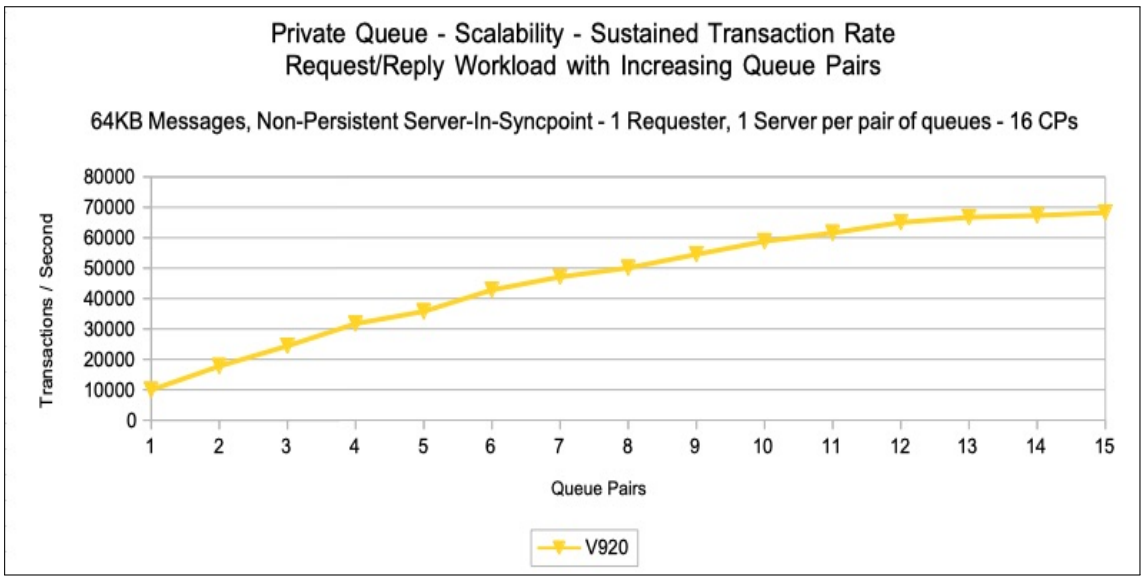


Chart: Transaction cost for non-persistent in syncpoint scalability workload with 64KB messages

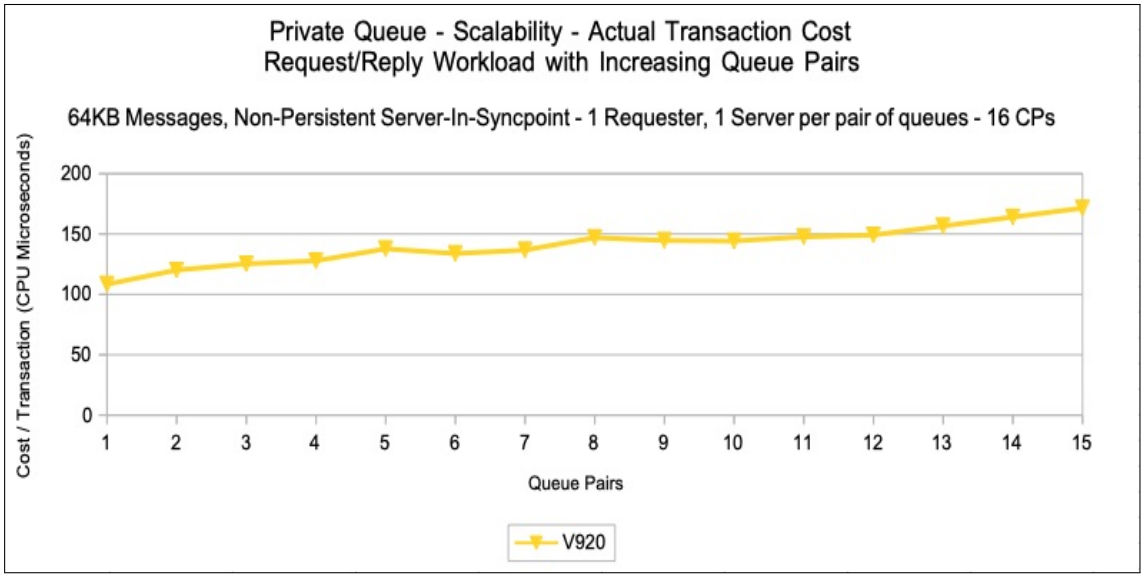


Chart: Transaction rate for non-persistent in syncpoint scalability workload with 4MB messages

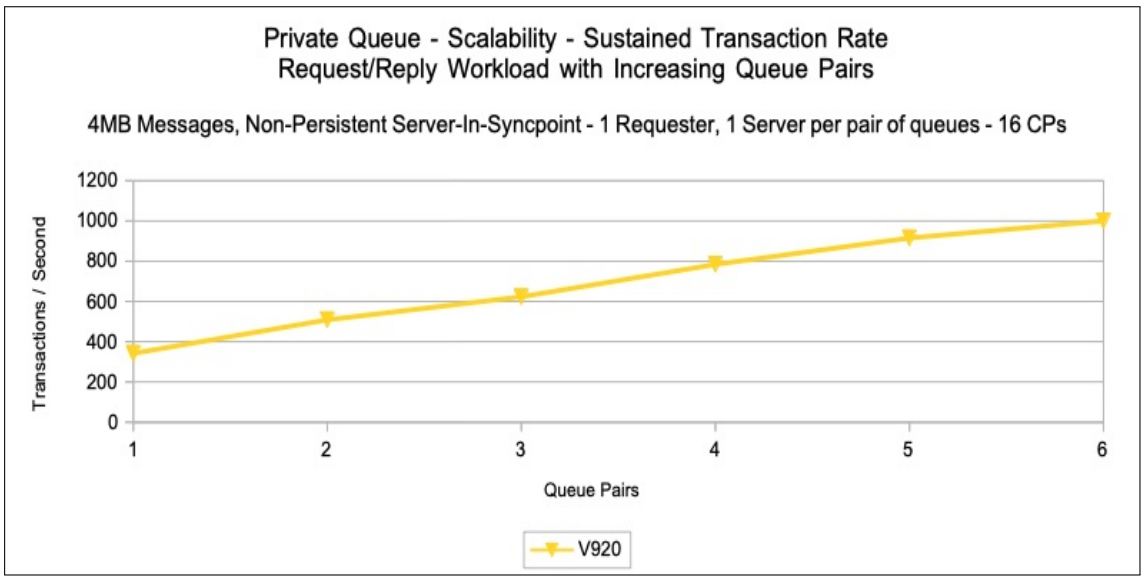
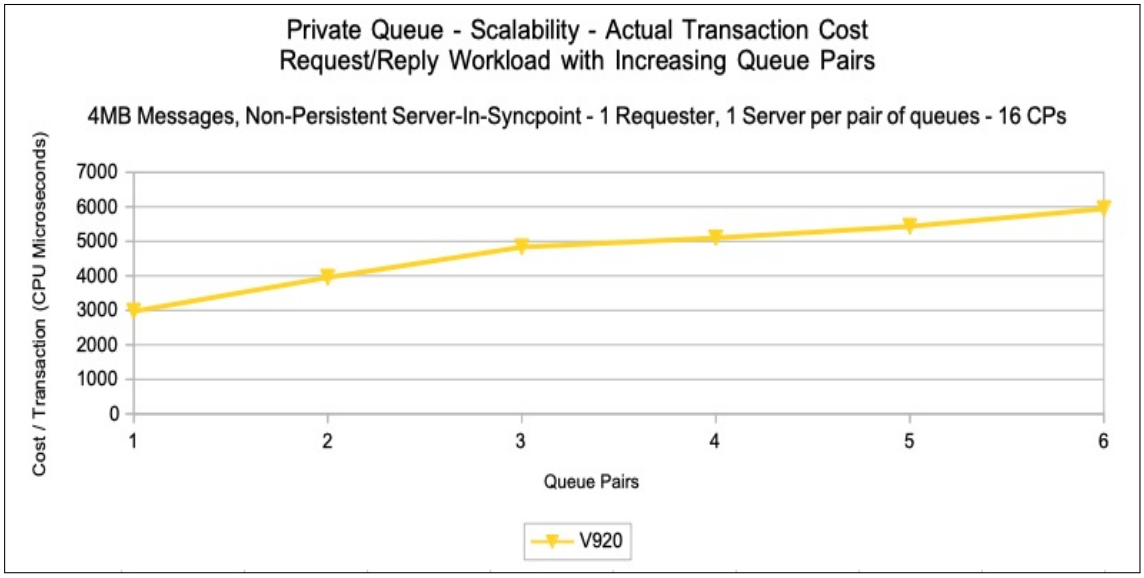


Chart: Transaction cost for non-persistent in syncpoint scalability workload with 4MB messages



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

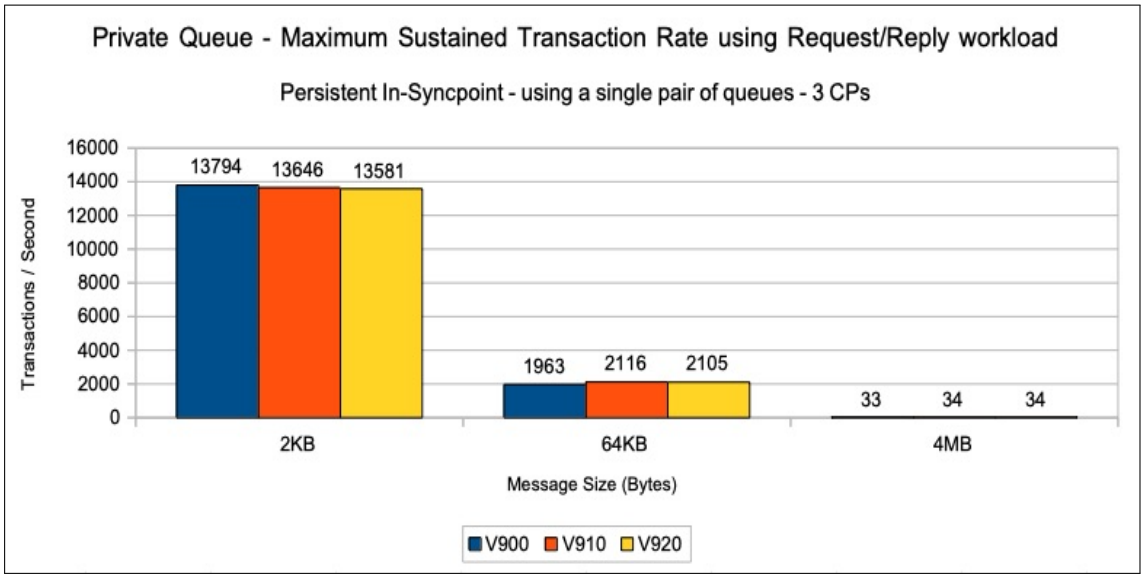
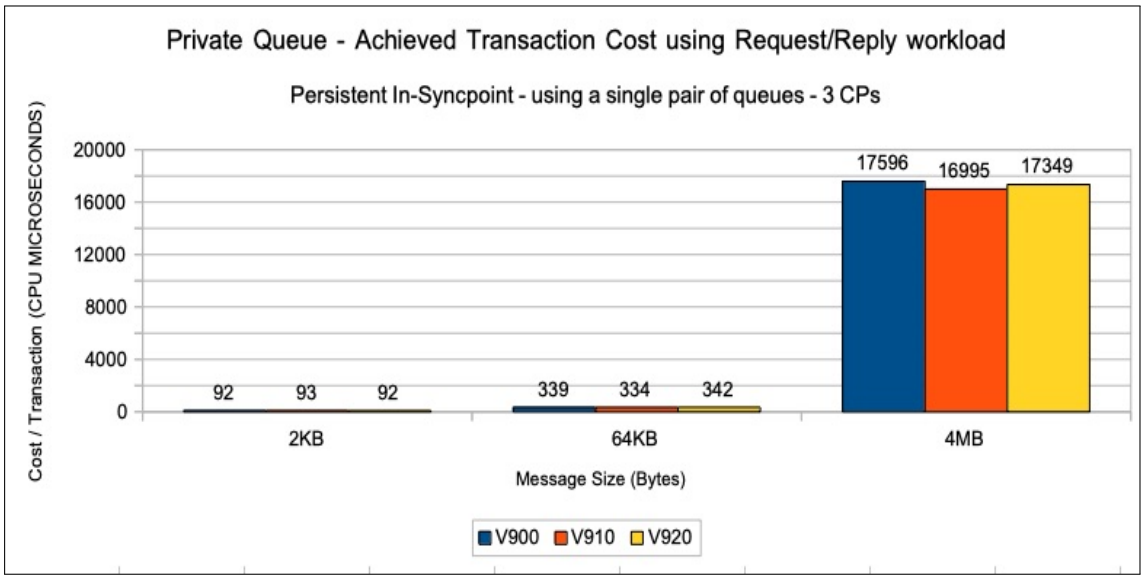


Chart: Transaction cost for persistent in syncpoint workload

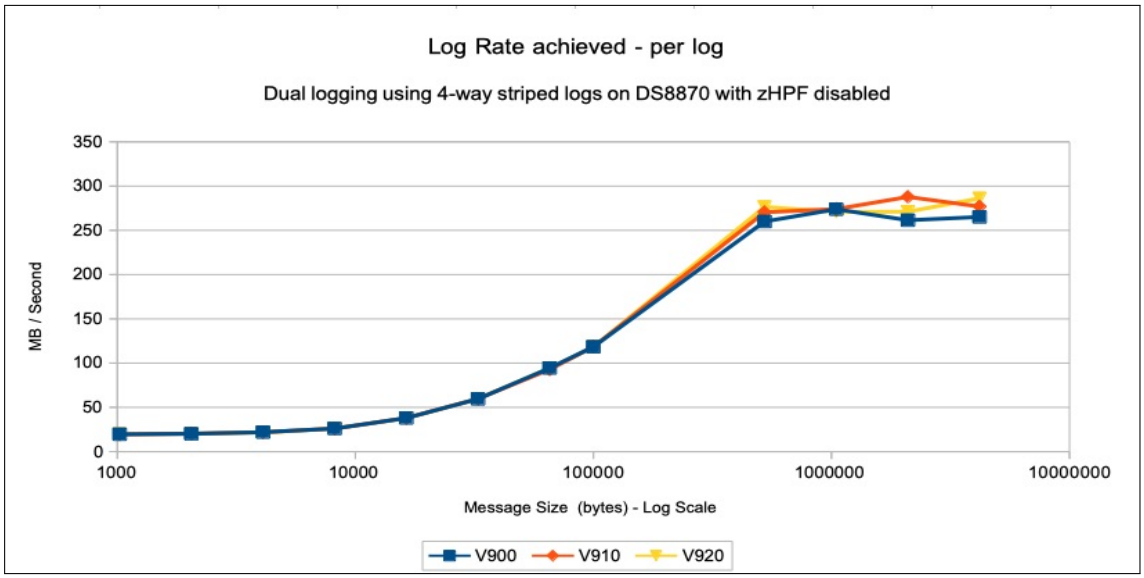


Upper bounds of persistent logging rate

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 250MB per second is for each copy of the dual logs, i.e. there is more than 500 MB of data being written to IBM MQ log data sets per second.

These logs are configured with 4 stripes.

From the RMF™ CACHE Subsystem Summary report we have determined that the peak rates are being impacted by NVS (Non-volatile storage) storage constraints which causes DASD FAST WRITE (DFW) delays.

DASD FAST WRITE provides support for the caching of write requests. NON-VOLATILE STORAGE (NVS) provides backup storage for this function. When you write data out to a volume, a write request is made to the storage control unit. The storage control unit accepts the data and puts it in the storage control unit cache. Later, the storage control unit schedules the data for writing.

When the DFW delays are not present, the peak log rate increases to a rate in excess of 300MB per second for each copy of the dual logs.

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.

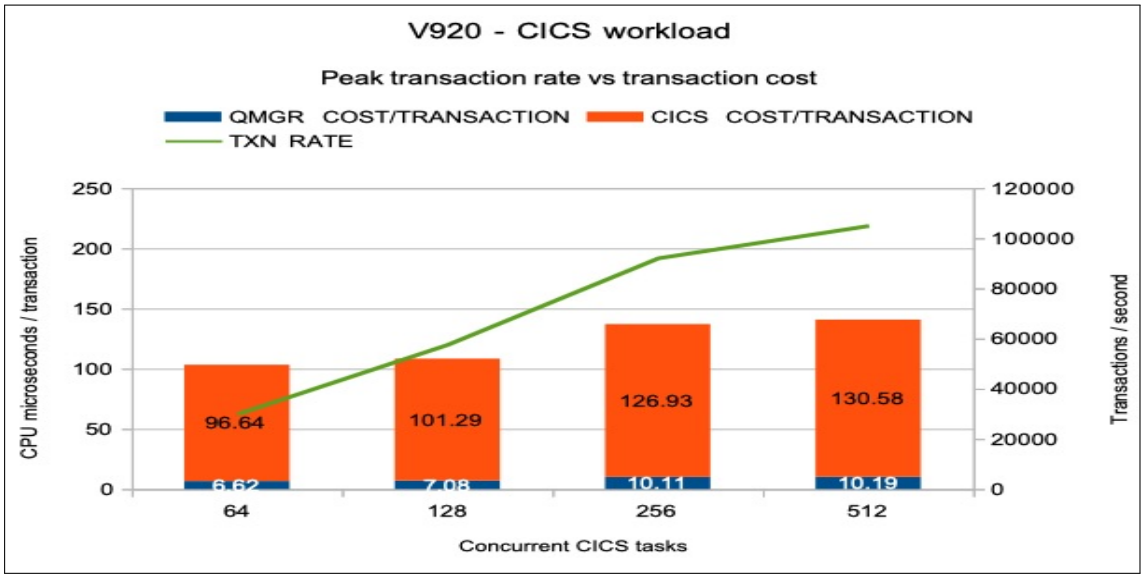
The following chart uses a set of simple CICS transactions that open queues, put a 2KB non-persistent message, get a 2KB non-persistent message, close queues and initiate a new CICS transaction. Running multiple sets of these transactions simulates a number of concurrent transactions across multiple 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 V900, V910 and V920 data is similar, only V920 data is shown.

Chart: V920 CICS workload



Shared Queue

Version 7.1.0 introduced CFLEVEL(5) with Shared Message Data Sets (SMDS) as an alternative to Db2 as a way to store large shared messages. This resulted in significant performance benefits with both larger messages (greater than 63KB) as well as smaller messages when the Coupling Facility approached its capacity by using tiered thresholds for offloading data. For more details on the performance of CFLEVEL(5), please refer to performance report MP16 "Capacity Planning and Tuning Guide".

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

Non-persistent out-of-syncpoint workload

Maximum throughput on a single pair of request/reply queues

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

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

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

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

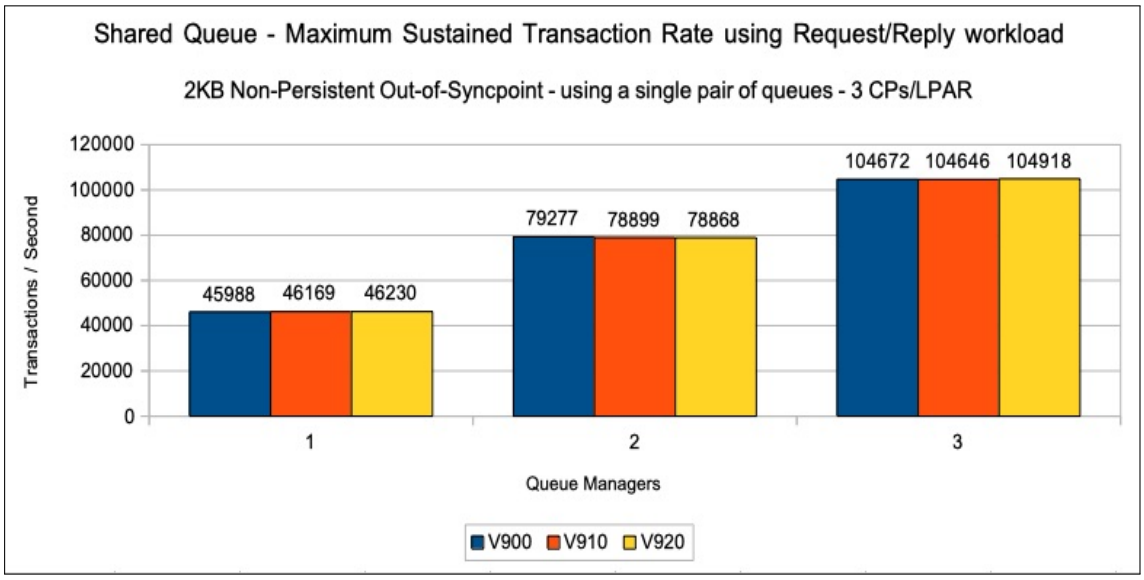


Chart: Transaction cost for non-persistent out-of-syncpoint workload - 2KB messages.

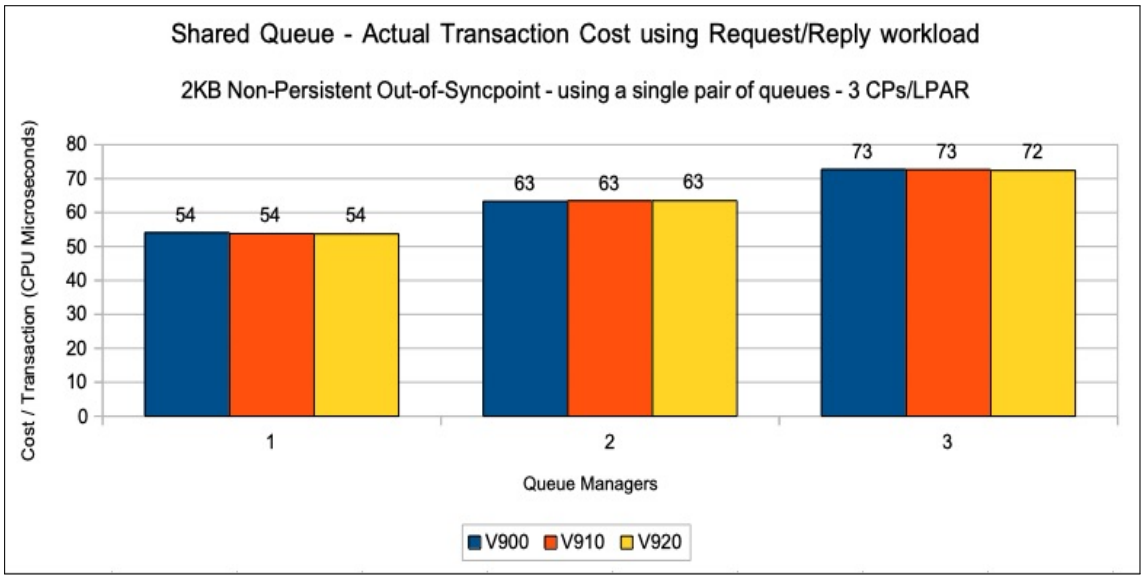


Chart: Transaction rate for non-persistent out-of-syncpoint workload - 64KB messages.

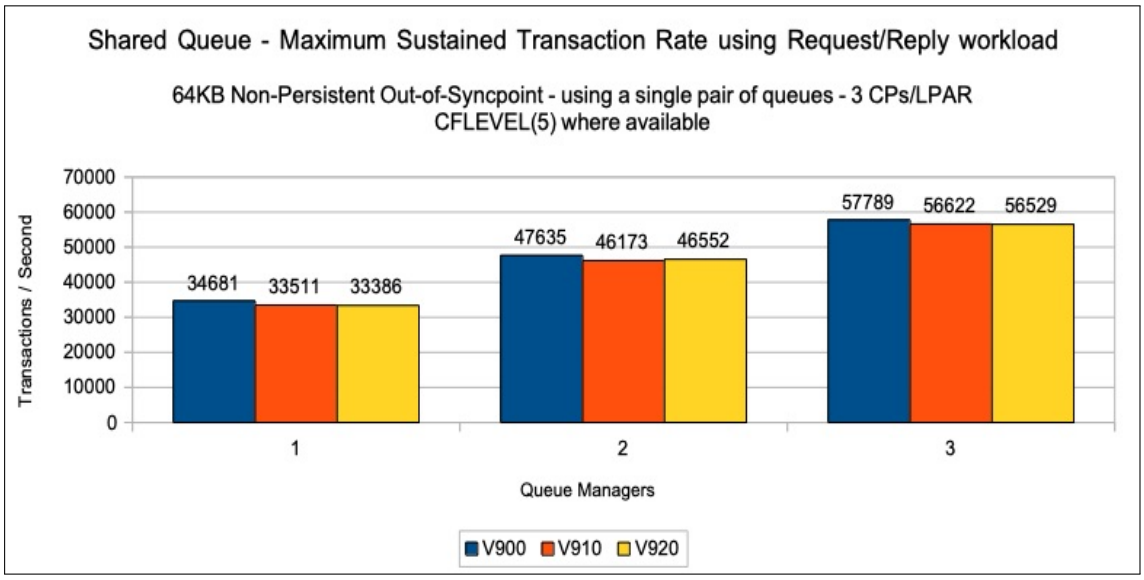


Chart: Transaction cost for non-persistent out-of-syncpoint workload - 64KB messages.

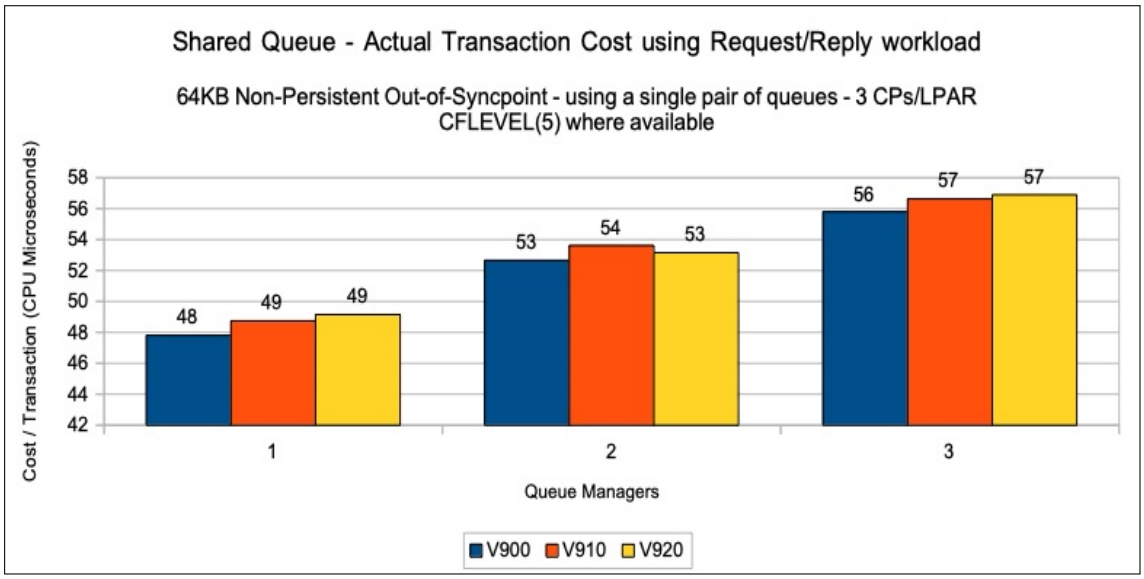


Chart: Transaction rate for non-persistent out-of-syncpoint workload - 4MB messages.

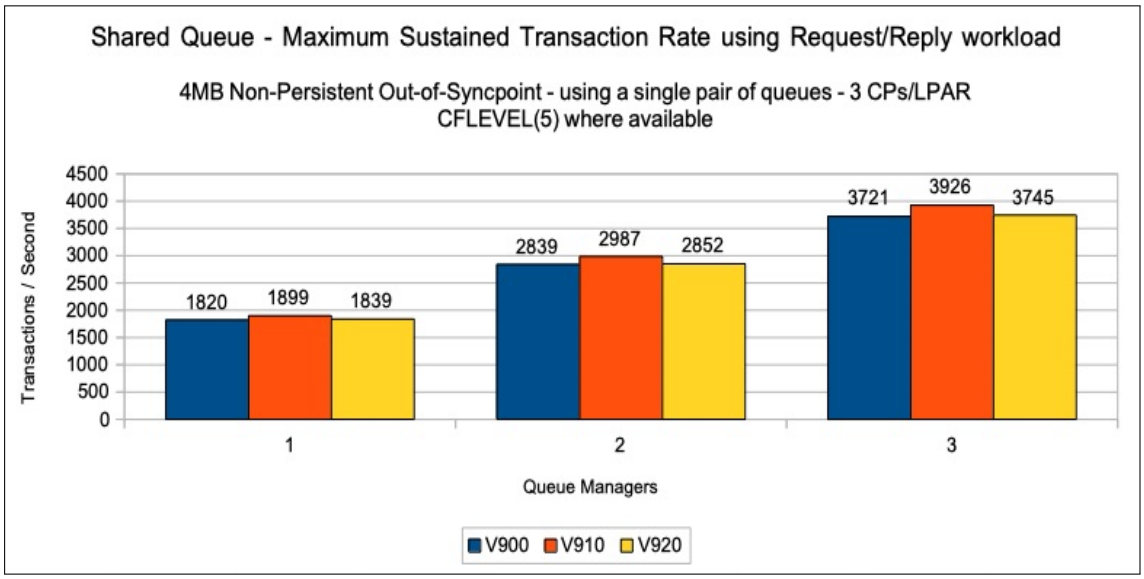
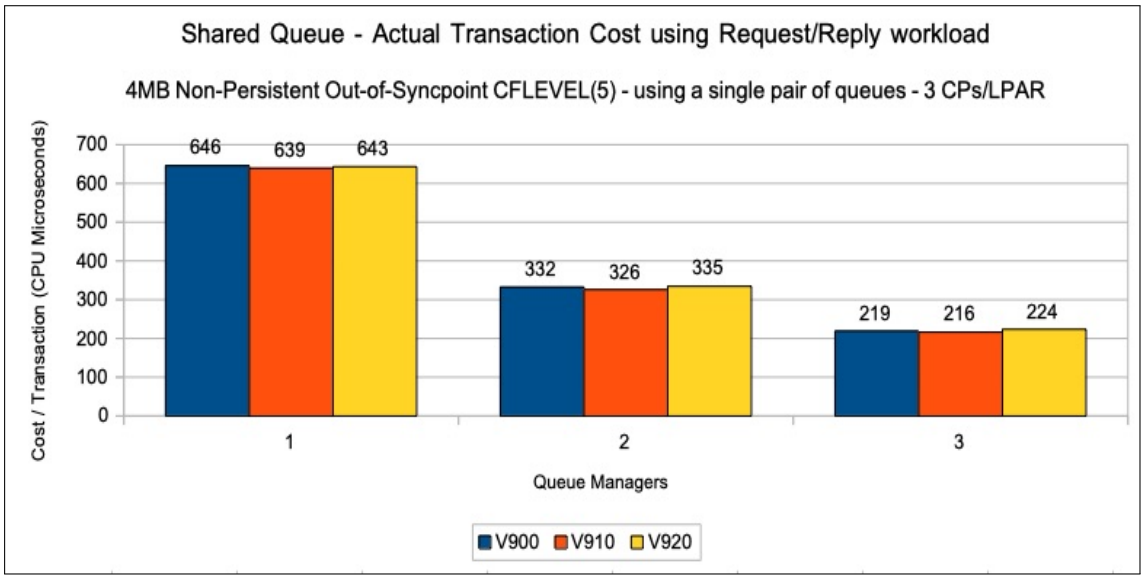


Chart: Transaction cost for non-persistent out-of-syncpoint workload - 4MB messages.



Non-persistent server in-syncpoint workload

Maximum throughput on a single pair of request/reply queues

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

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

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

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

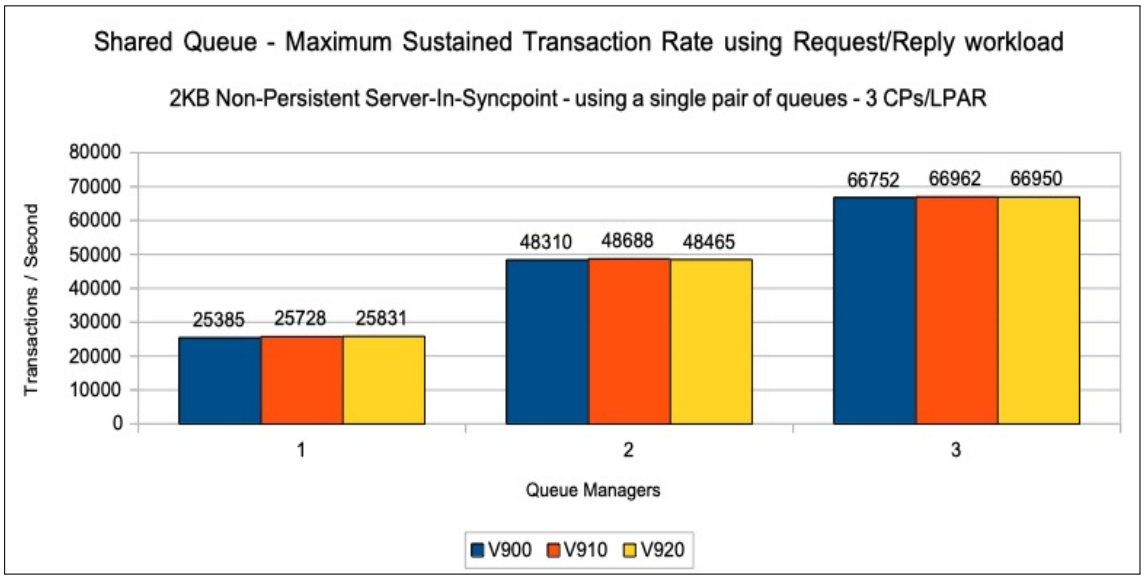


Chart: Transaction cost for non-persistent in syncpoint workload - 2KB

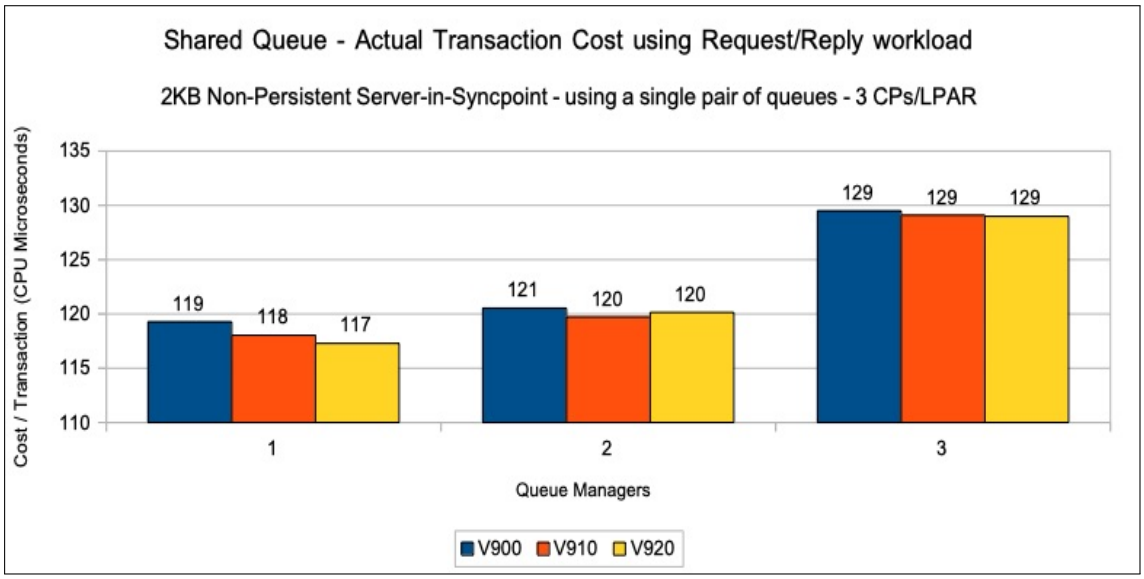


Chart: Transaction rate for non-persistent in syncpoint workload - 64KB

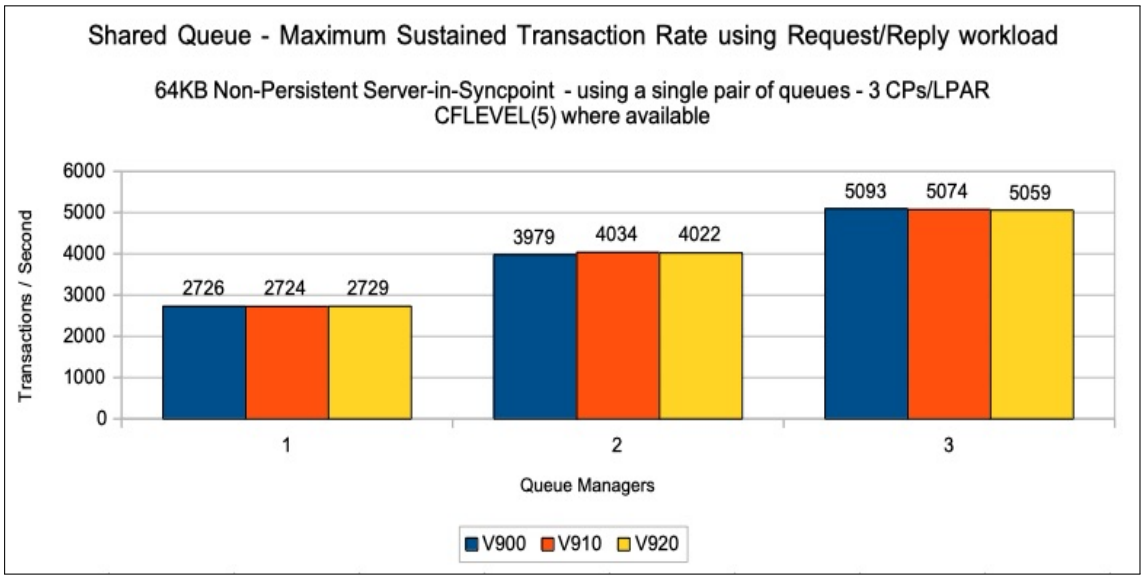


Chart: Transaction cost for non-persistent in syncpoint workload - 64KB

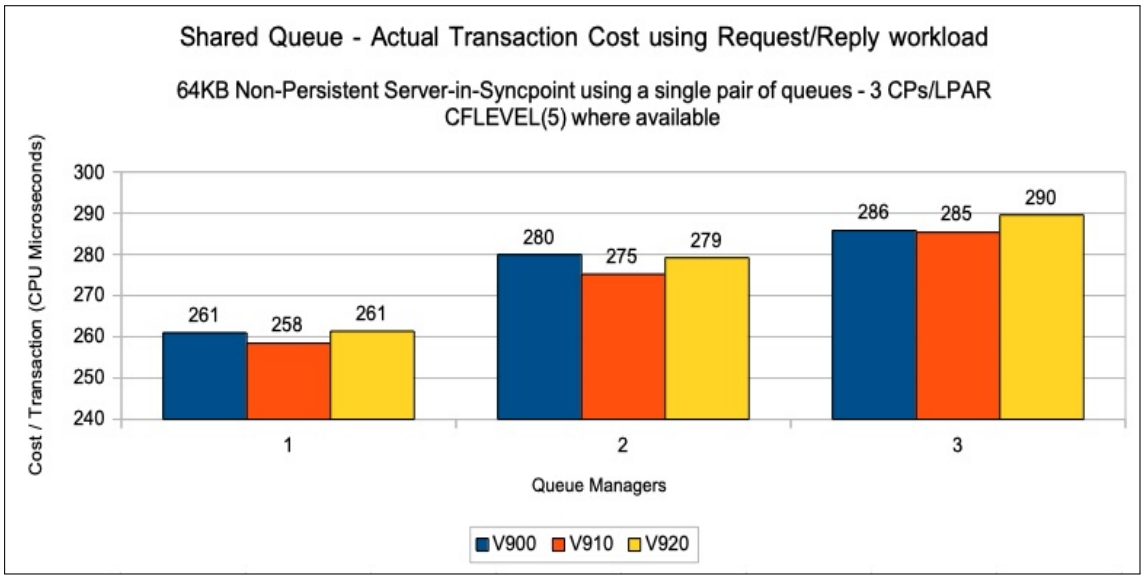


Chart: Transaction rate for non-persistent in syncpoint workload - 4MB

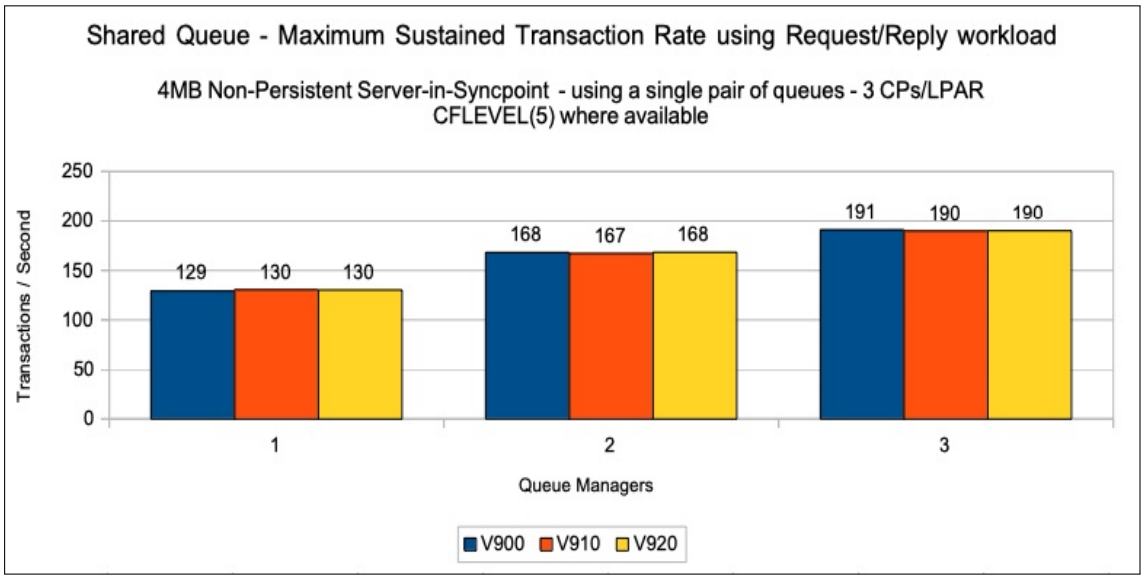
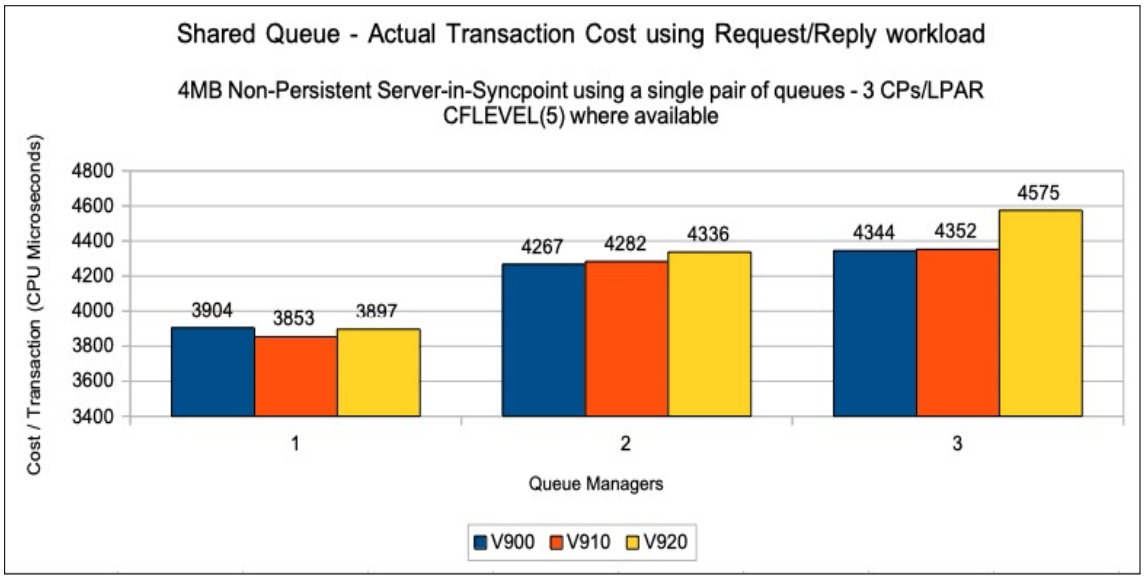


Chart: Transaction cost for non-persistent in syncpoint workload - 4MB



Data sharing non-persistent server in-syncpoint workload

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

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

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

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

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

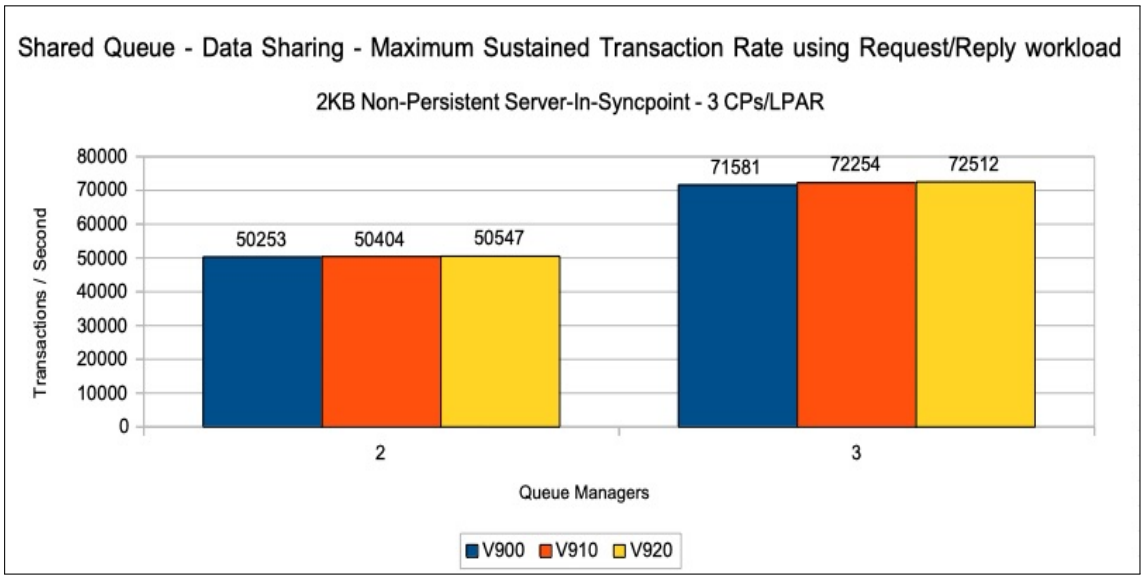


Chart: Transaction cost for data sharing non-persistent in syncpoint workload - 2KB

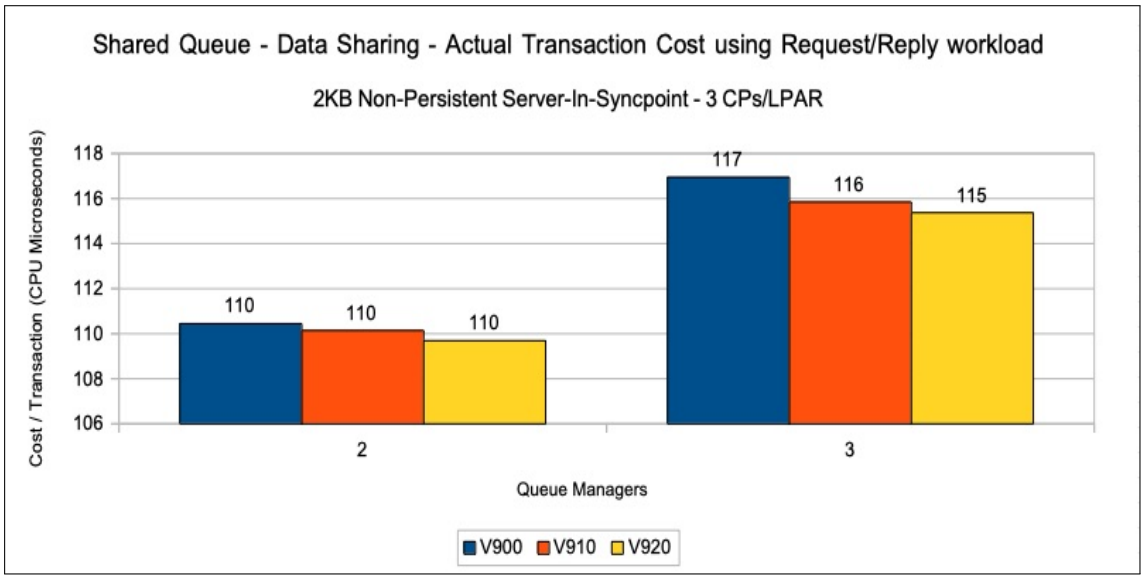


Chart: Transaction rate for data sharing non-persistent in syncpoint workload - 64KB

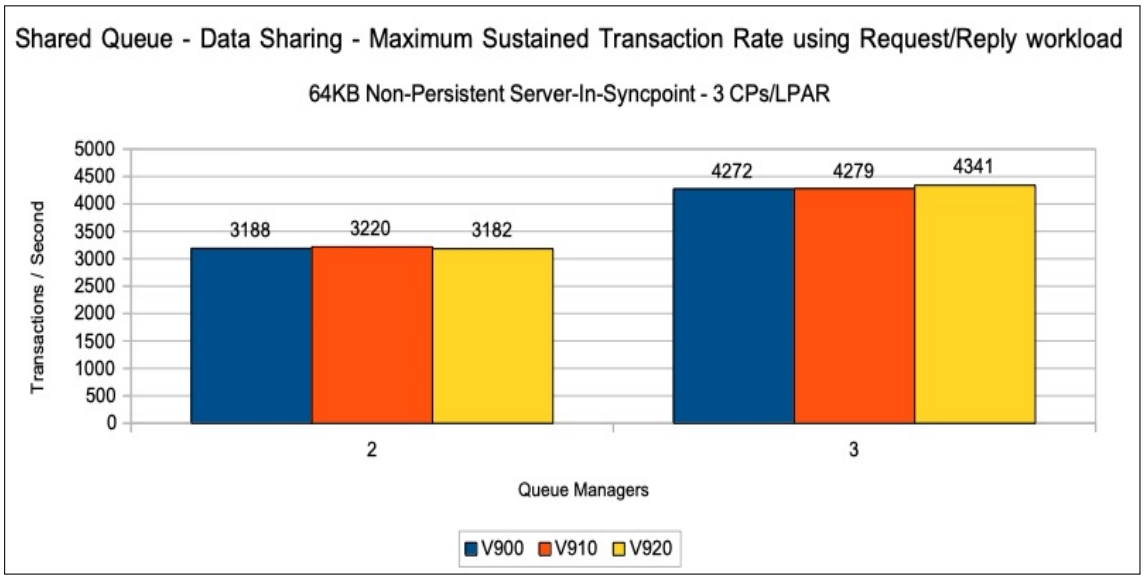
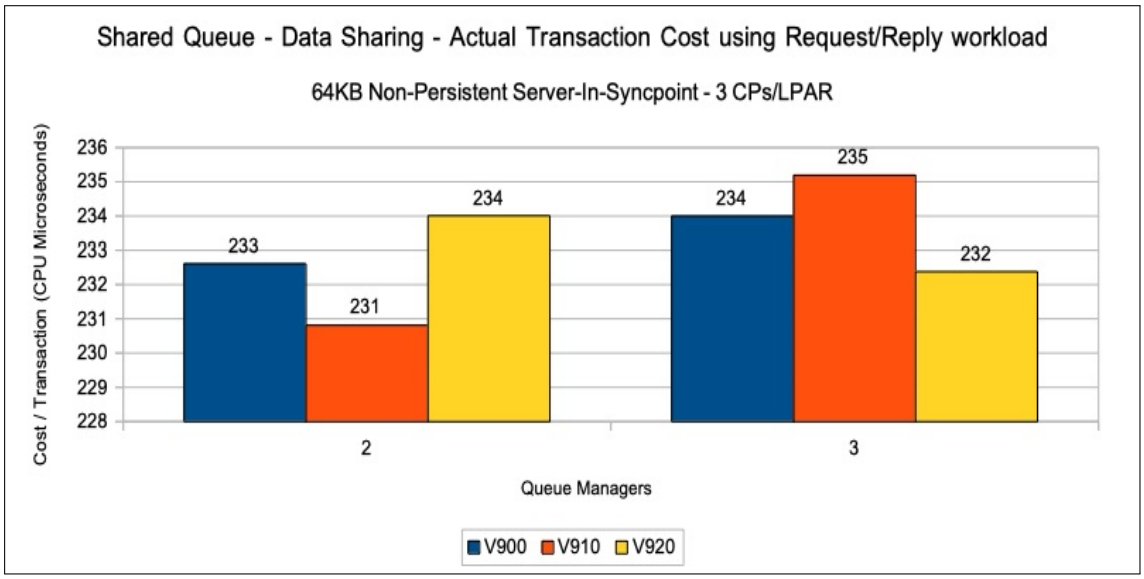


Chart: Transaction cost for data sharing non-persistent in syncpoint workload - 64KB



Moving messages across channels

The regression tests for moving message across channels e.g. sender-receiver channels, are designed to use drive the channel initiator such that system limits rather than MQ are the reason for any constraints, and typically it is CPU that is the constraining factor. Therefore the tests use non-persistent messages 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 Channel compression on MQ for z/OS. The compression measurements shown in this document use zEDC hardware compression where possible.

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

Measurements:

- The measurements using 1 to 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

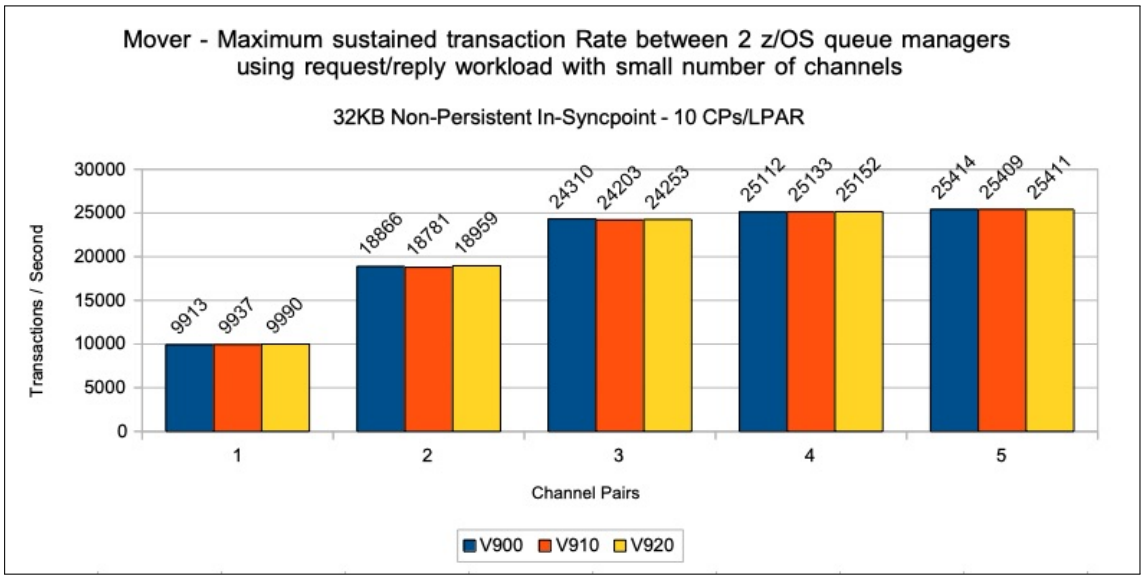


Chart: Transaction cost for 32KB messages

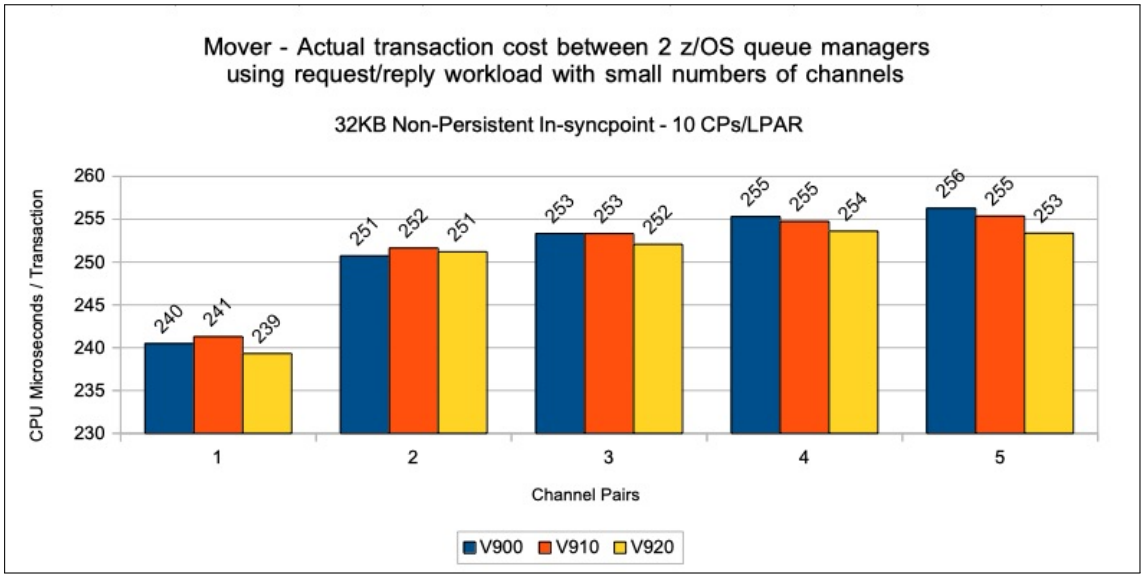


Chart: Transaction rate with 32KB messages over SSL channels with no secret key negotiation

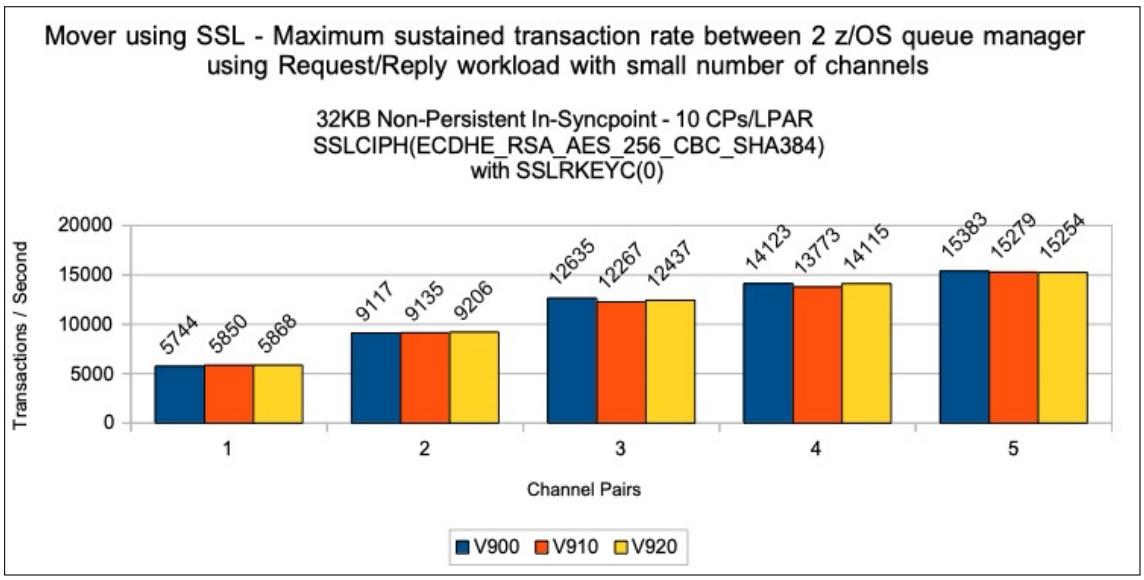


Chart: Transaction cost for 32KB messages over SSL channels with no secret key negotiation

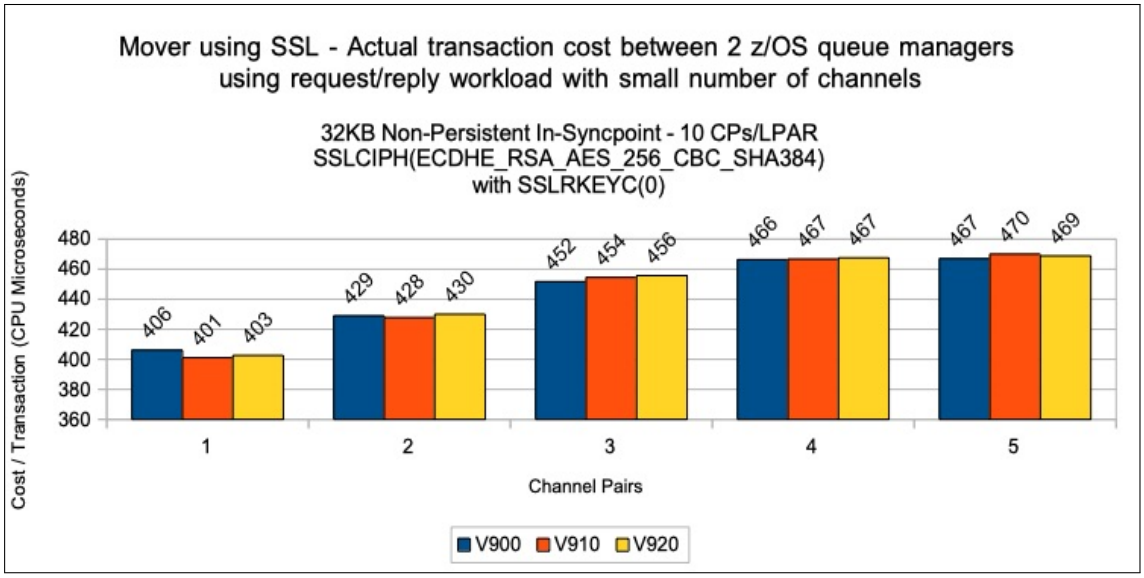


Chart: Transaction rate with 32KB messages over SSL channels with secret key negotiation every 1MB

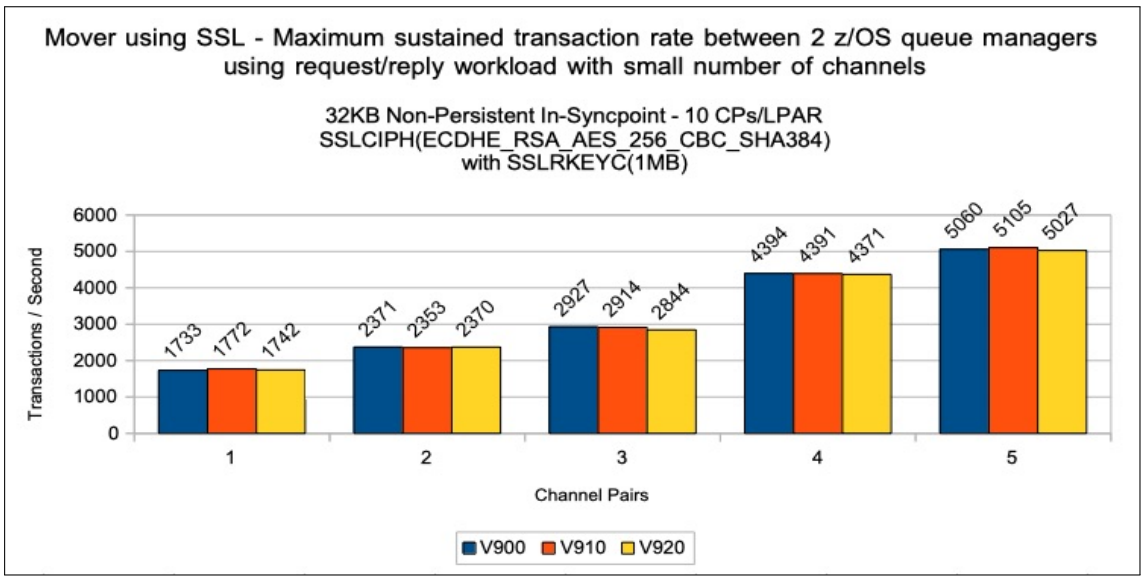
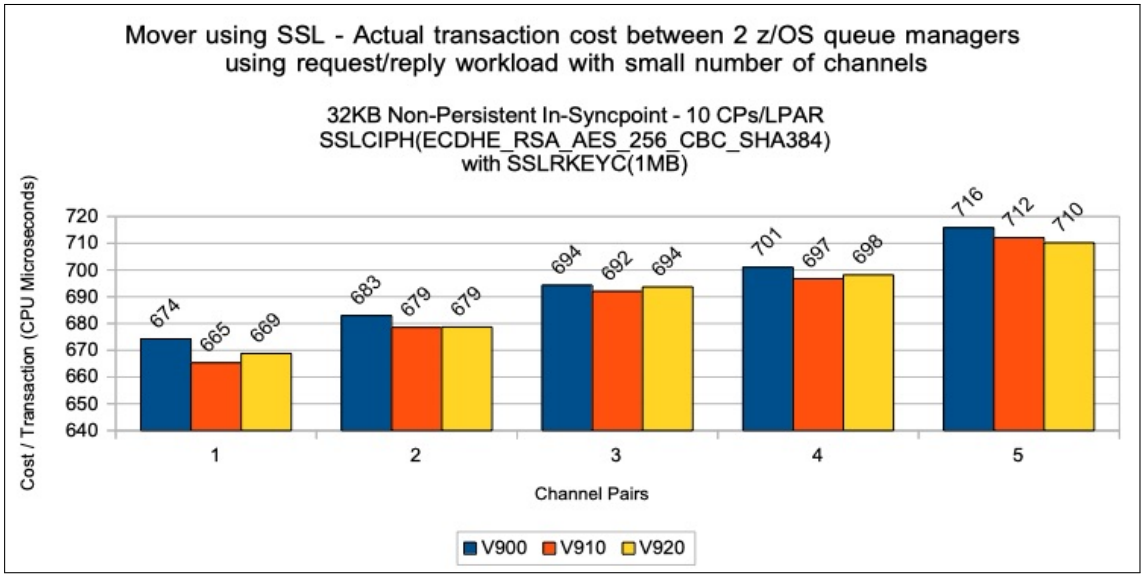


Chart: Transaction cost for 32KB messages over SSL channels with secret key negotiation every 1MB



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

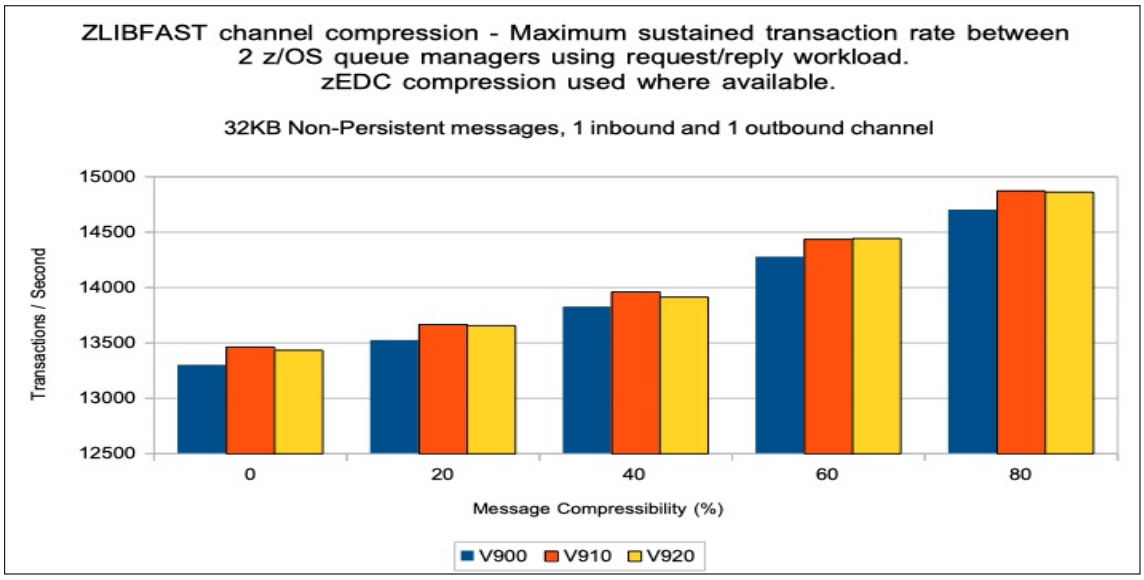
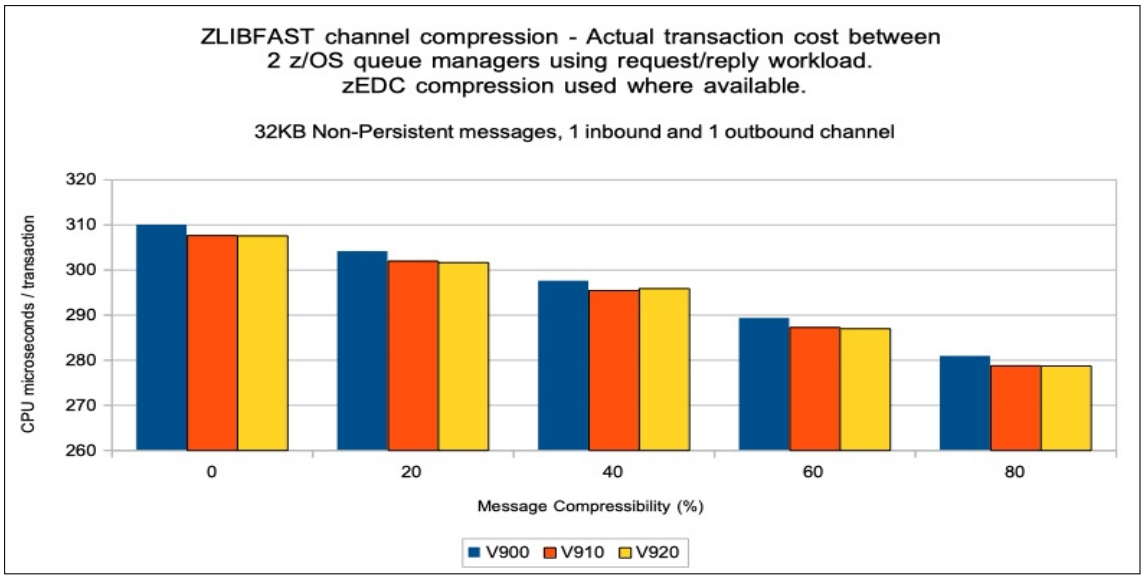


Chart: Transaction cost for ZLIBFAST on 32KB messages



Channel compression using ZLIBHIGH

Chart: Transaction rate with 32KB messages with ZLIBHIGH

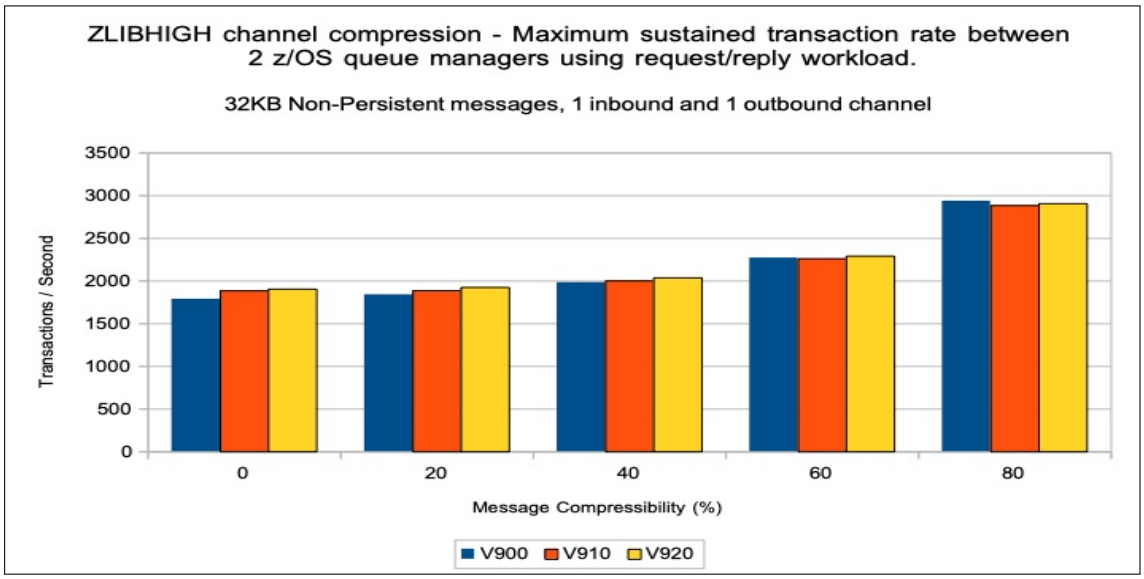
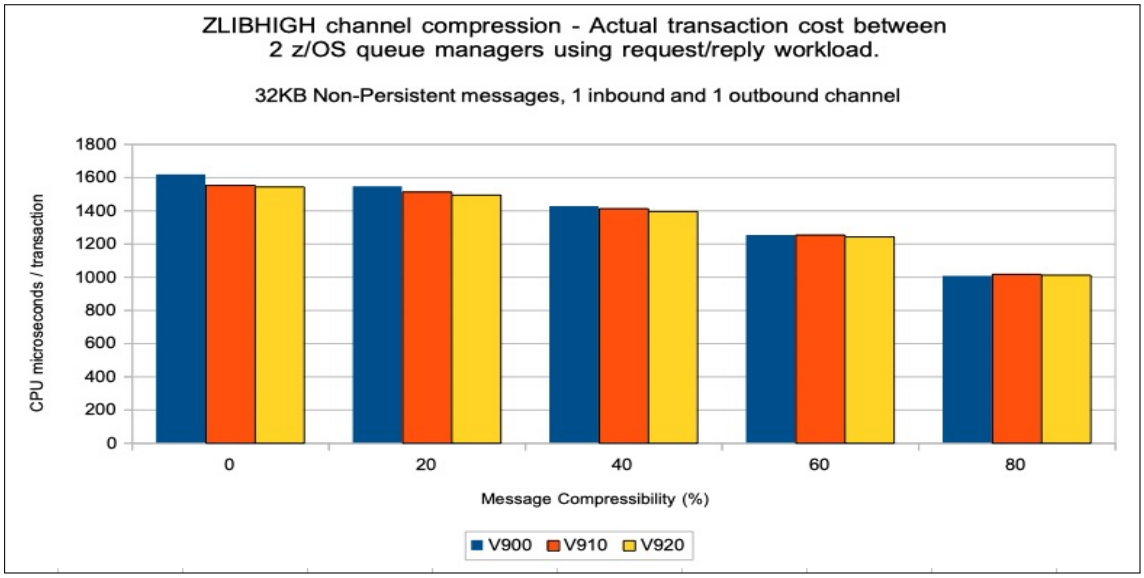


Chart: Transaction cost for 32KB messages with ZLIBHIGH



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

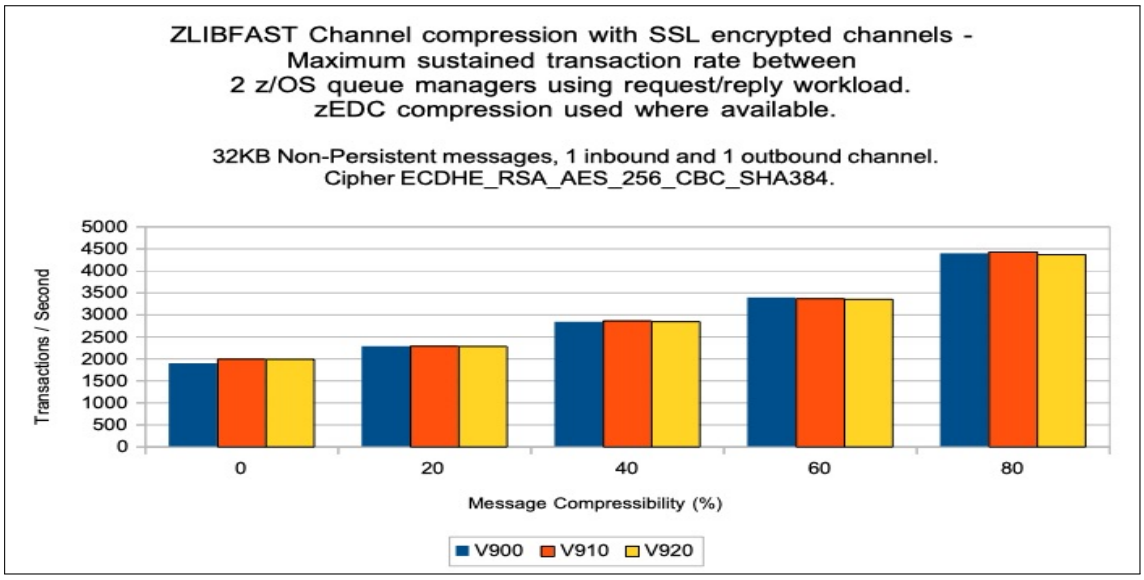
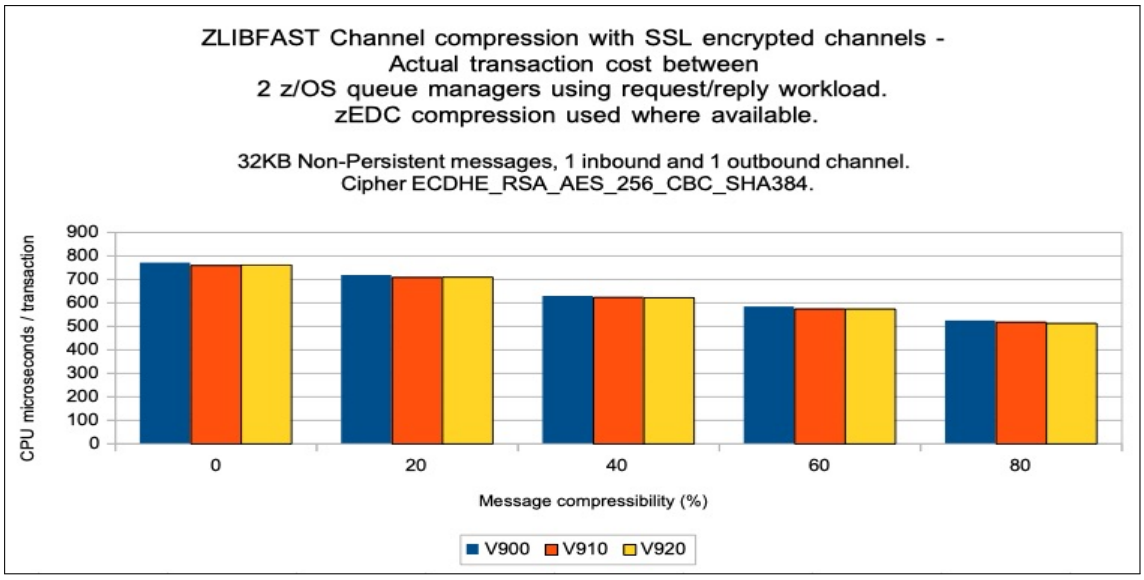


Chart: Transaction cost with 32KB messages with ZLIBFAST on SSL channels



Channel compression using ZLIBHIGH on SSL channels

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

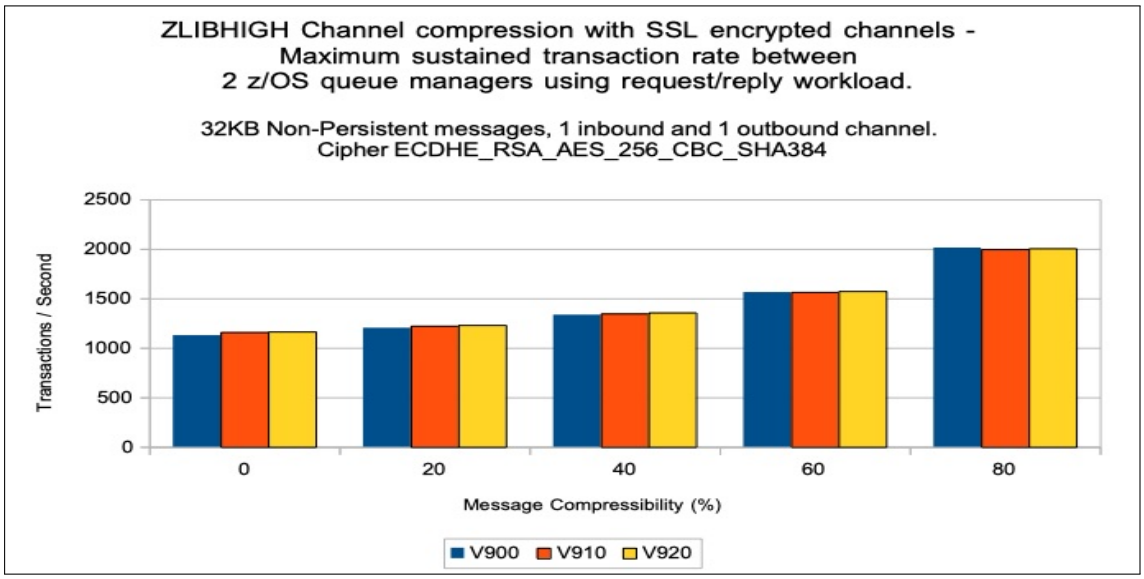
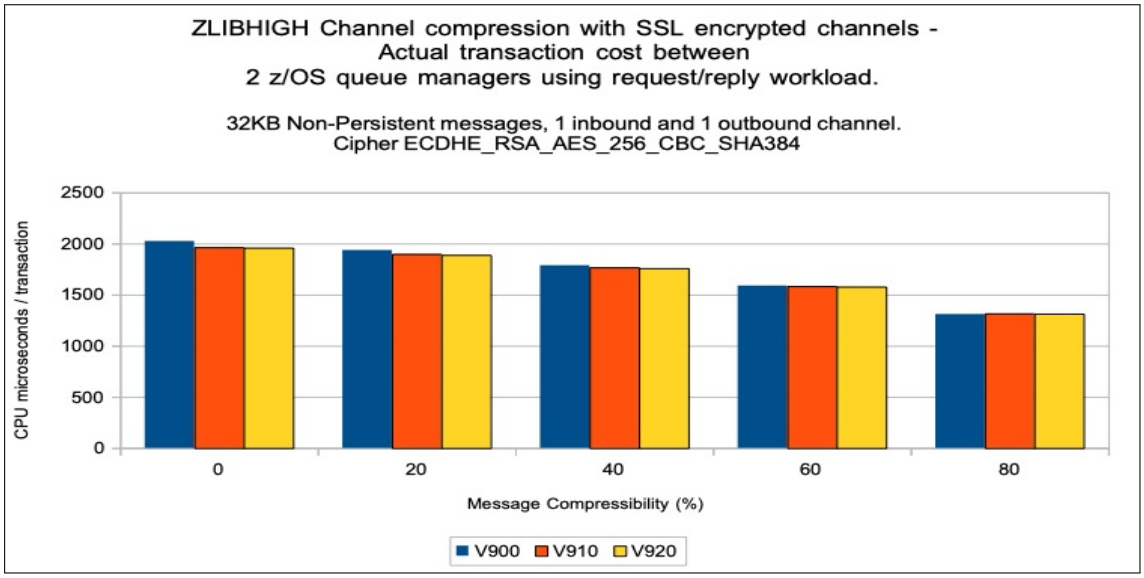


Chart: Transaction cost for 32KB messages with ZLIBHIGH on SSL channels



Streaming workload between 2 z/OS queue managers

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

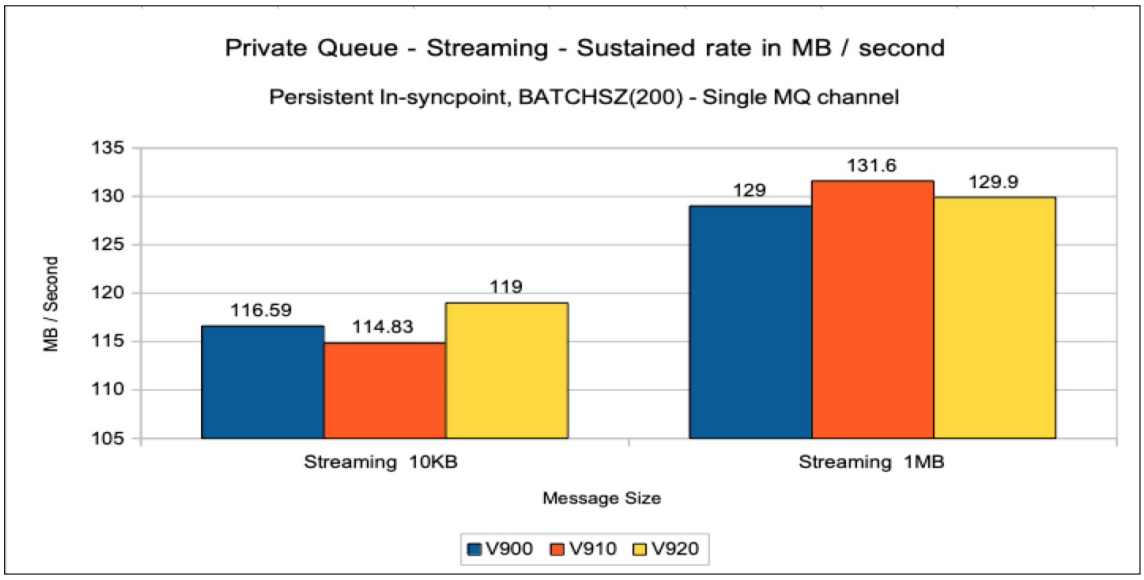
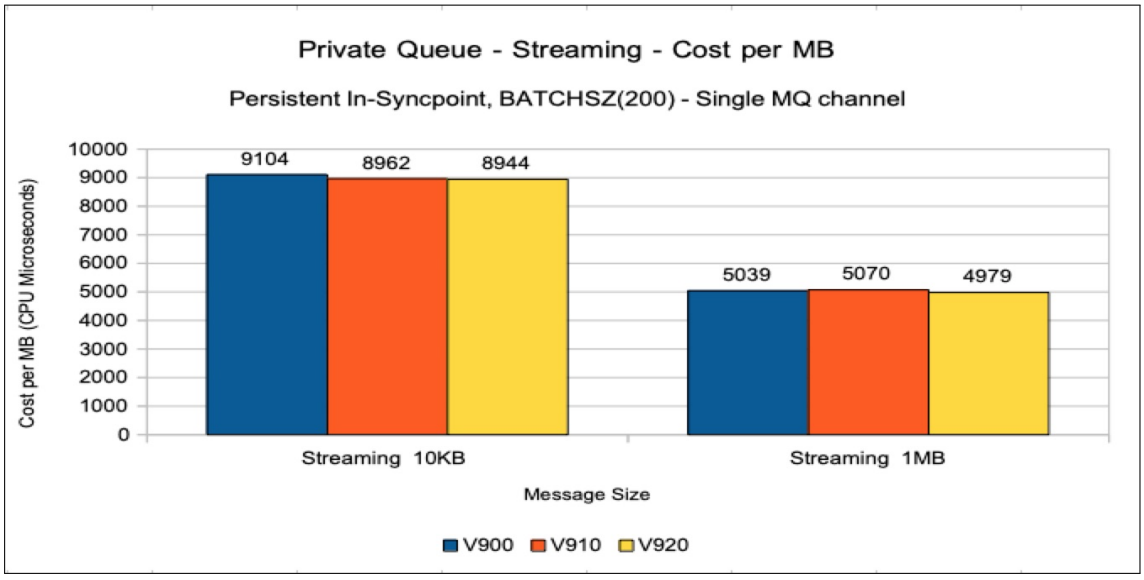


Chart: Cost to stream persistent messages between 2 z/OS queue managers



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.

Note: Since V900, there is a small degradation in performance, which is particularly noticeable with smaller messages in our benchmark tests. This is the result of changes for the cluster workload exit being moved in to channel initiator code and requiring switching to/from authorised state.

Bind-on-open

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

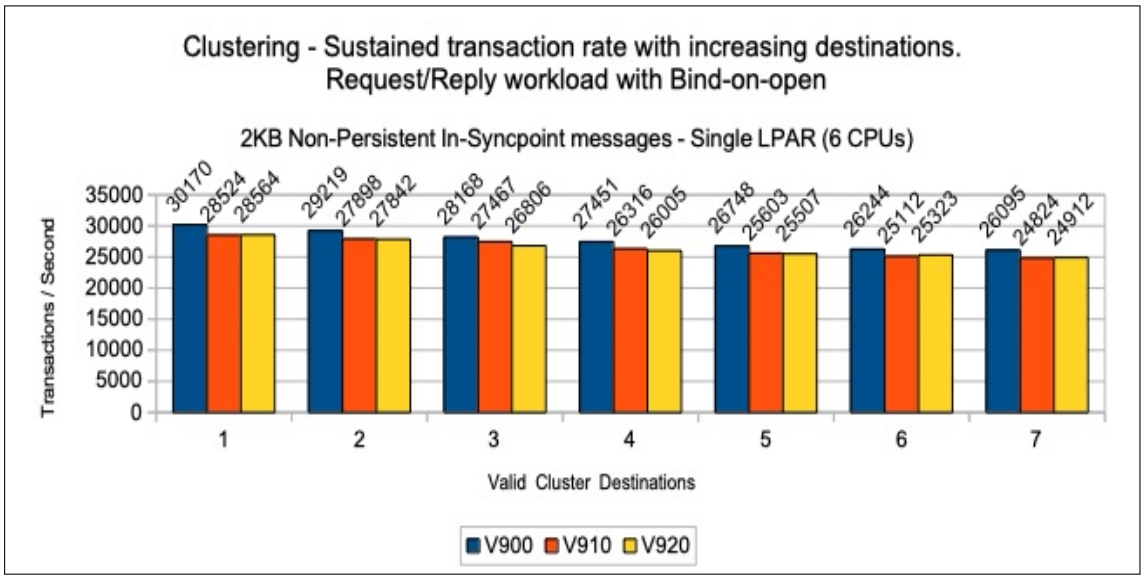


Chart: Bind-on-open transaction cost for 2KB messages

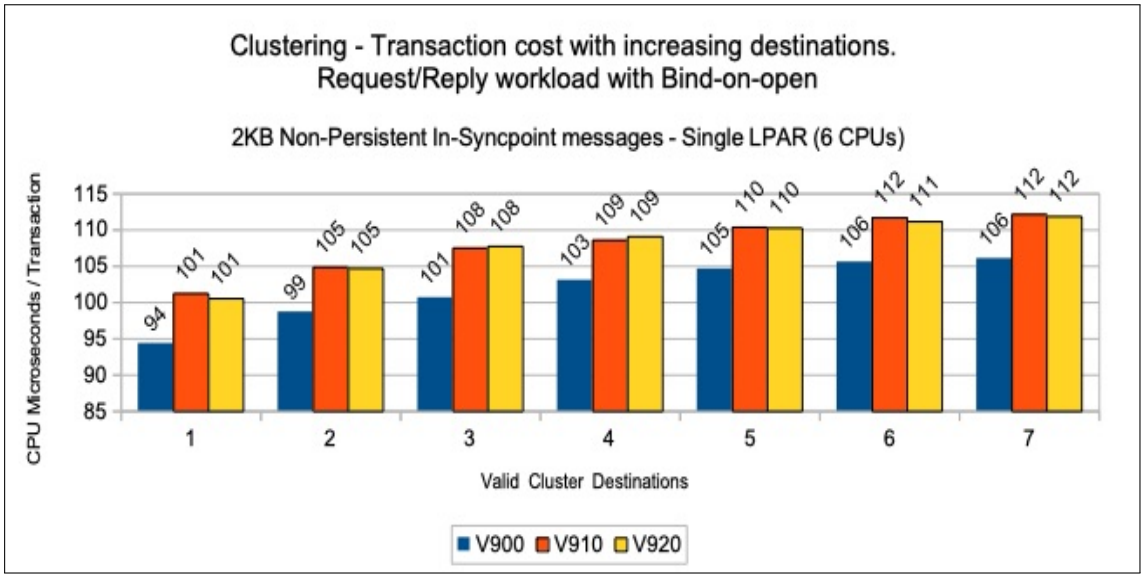


Chart: Bind-on-open transaction rate with 32KB messages

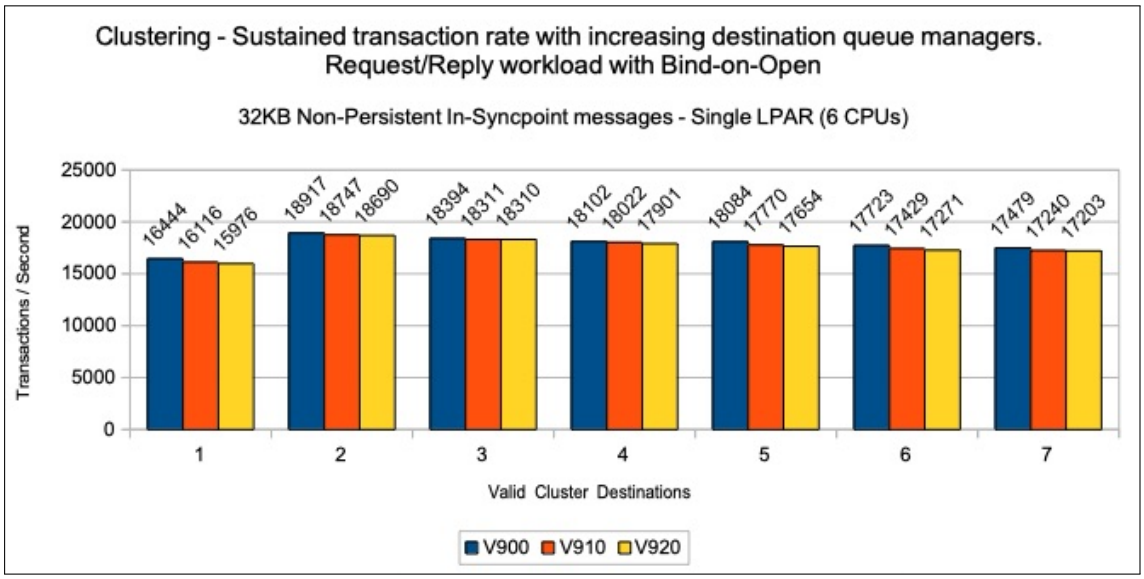
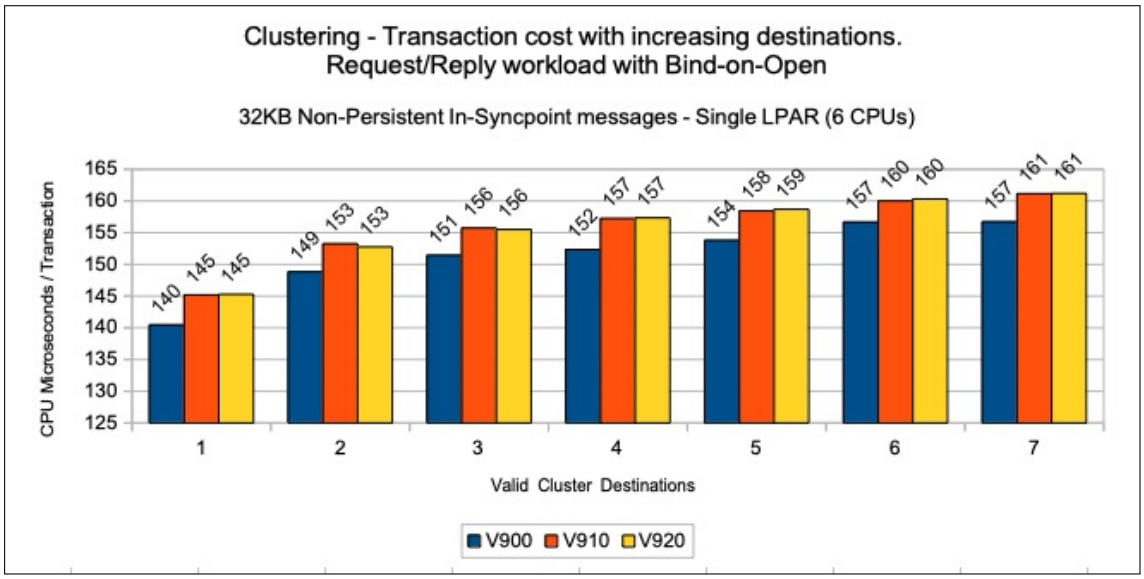


Chart: Bind-on-open transaction cost for 32KB messages



Bind-not-fixed

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

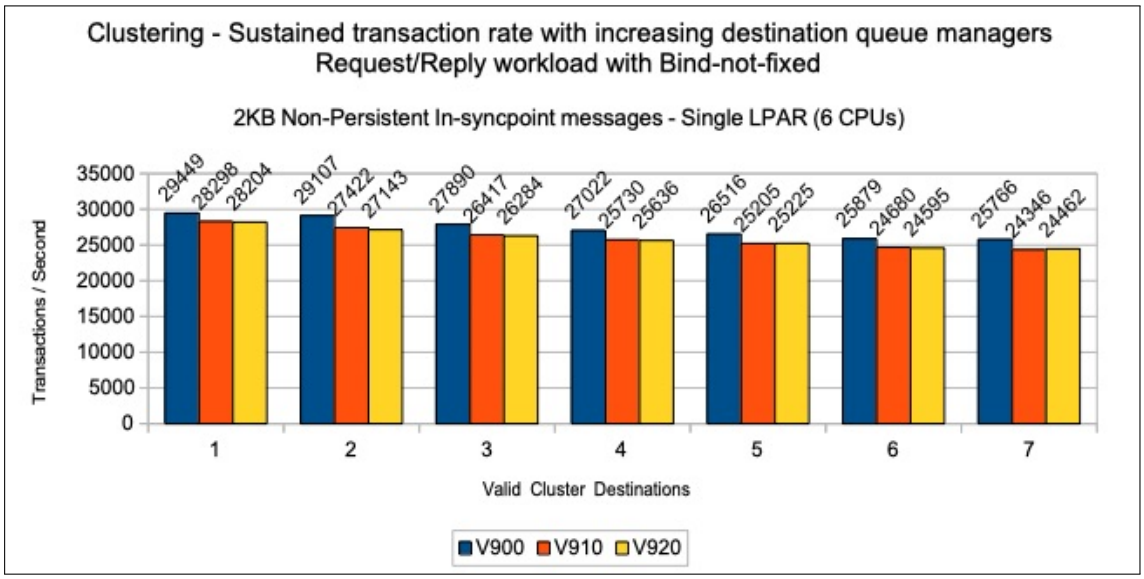


Chart: Bind-not-fixed transaction cost for 2KB messages

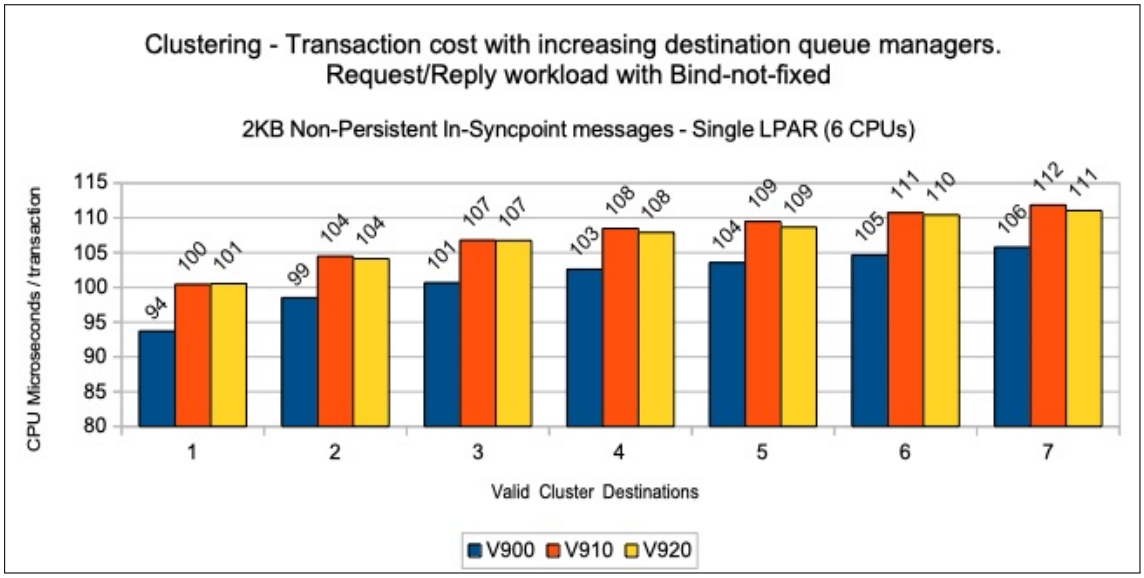


Chart: Bind-not-fixed transaction rate with 32KB messages

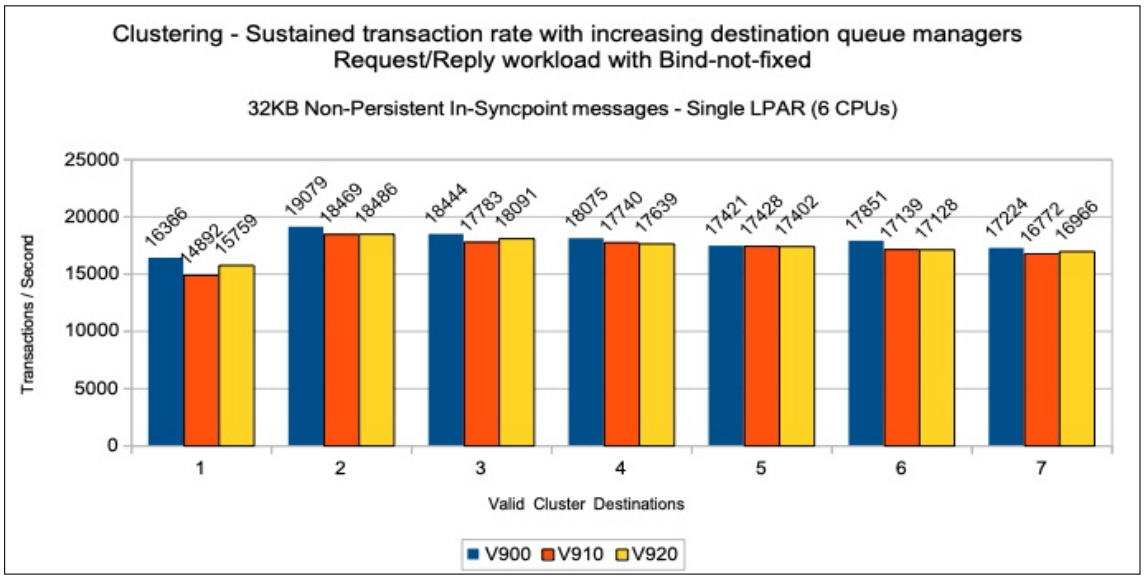
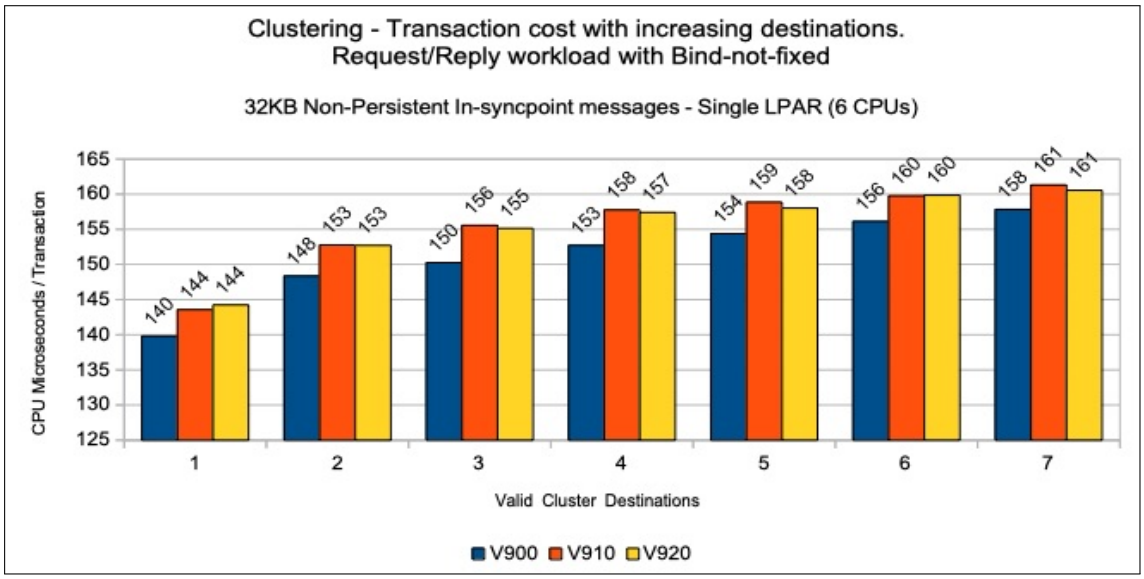


Chart: Bind-not-fixed transaction cost for 32KB messages



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)

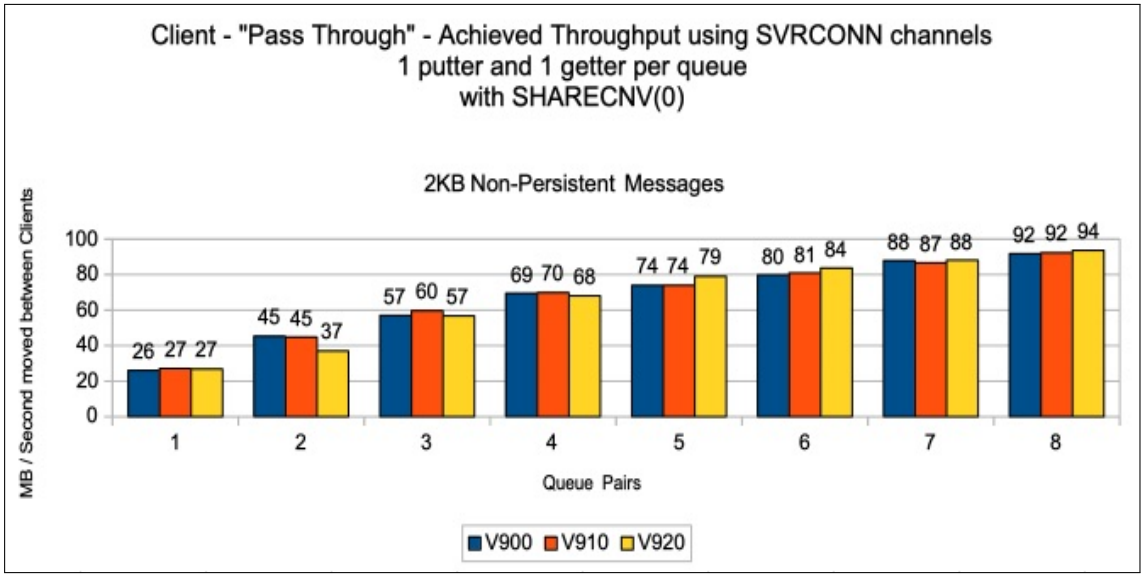
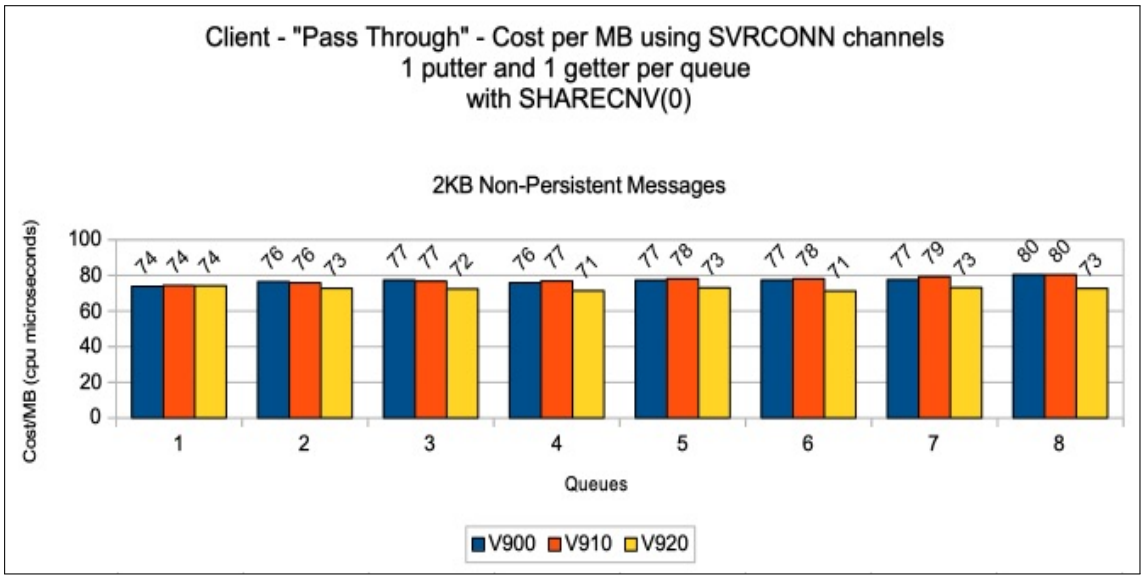


Chart: Cost per MB of client pass through tests with SHARECNV(0)



Client pass through test using SHARECNV(1)

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

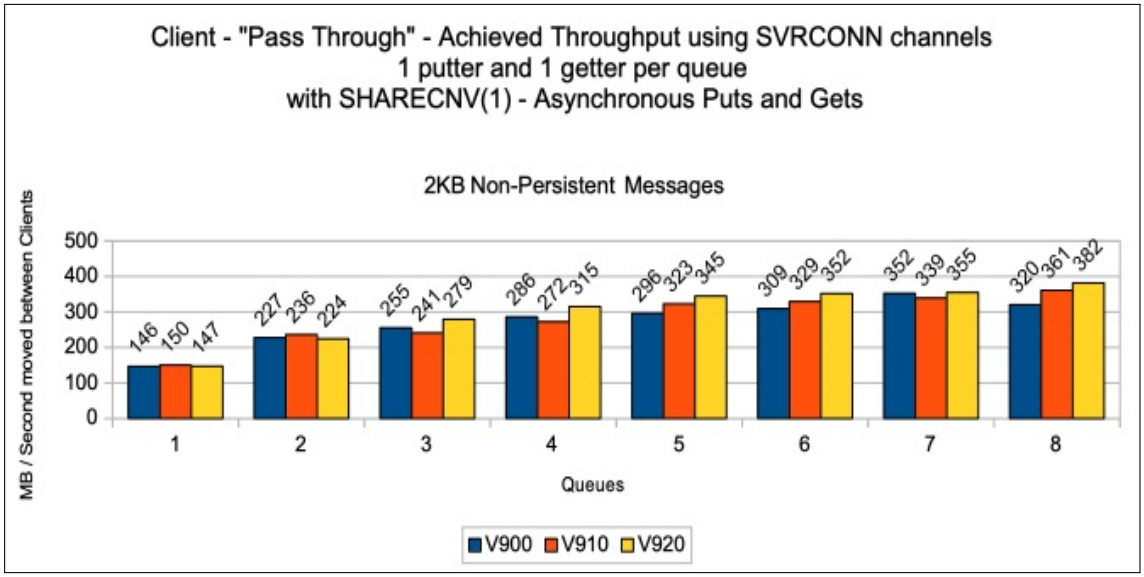
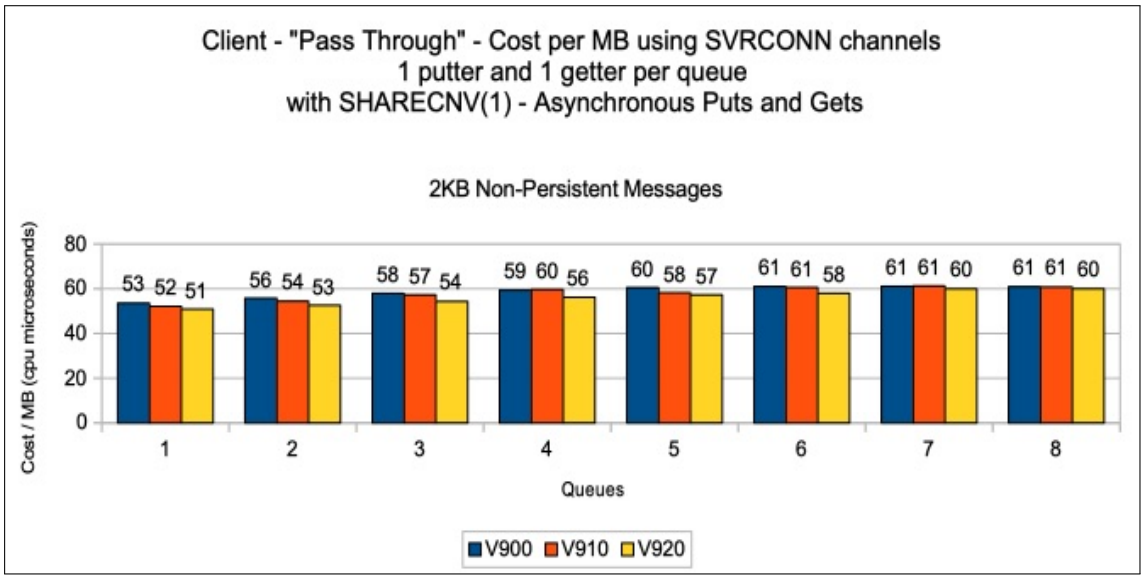


Chart: Cost per MB of client pass through tests with SHARECNV(1)



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

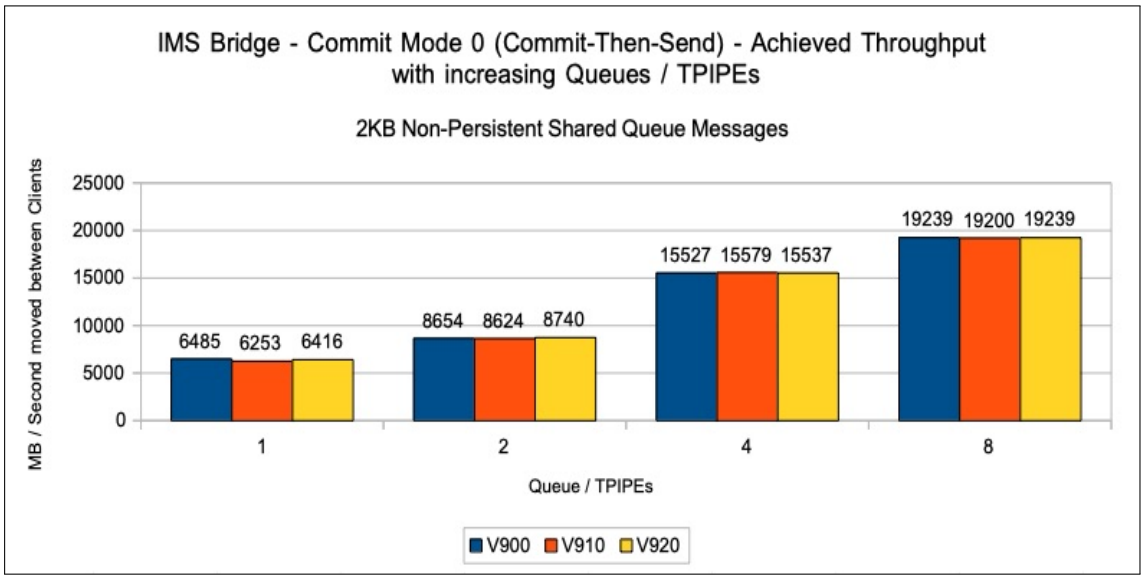
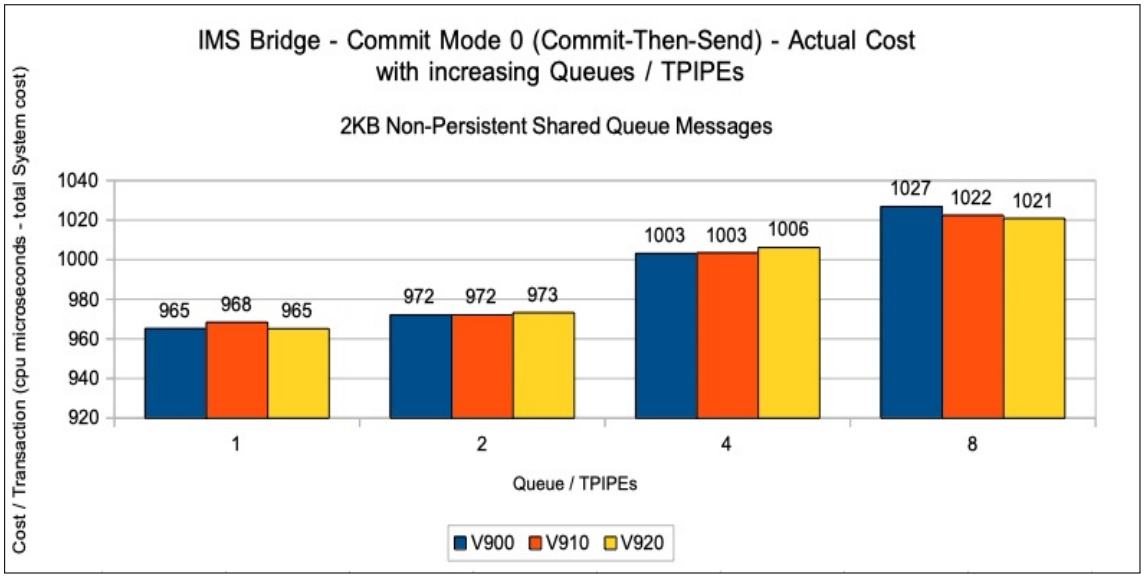


Chart: IMS Bridge commit mode 0 - Transaction cost



Commit mode 1 (send-then-commit)

Chart: IMS Bridge commit mode 1 - Throughput rate

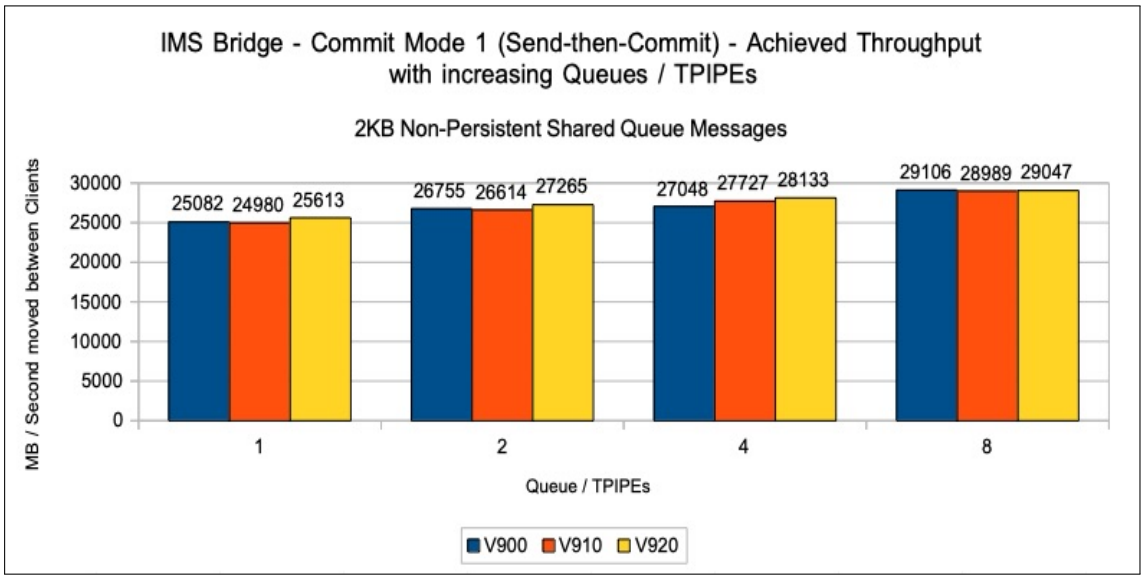
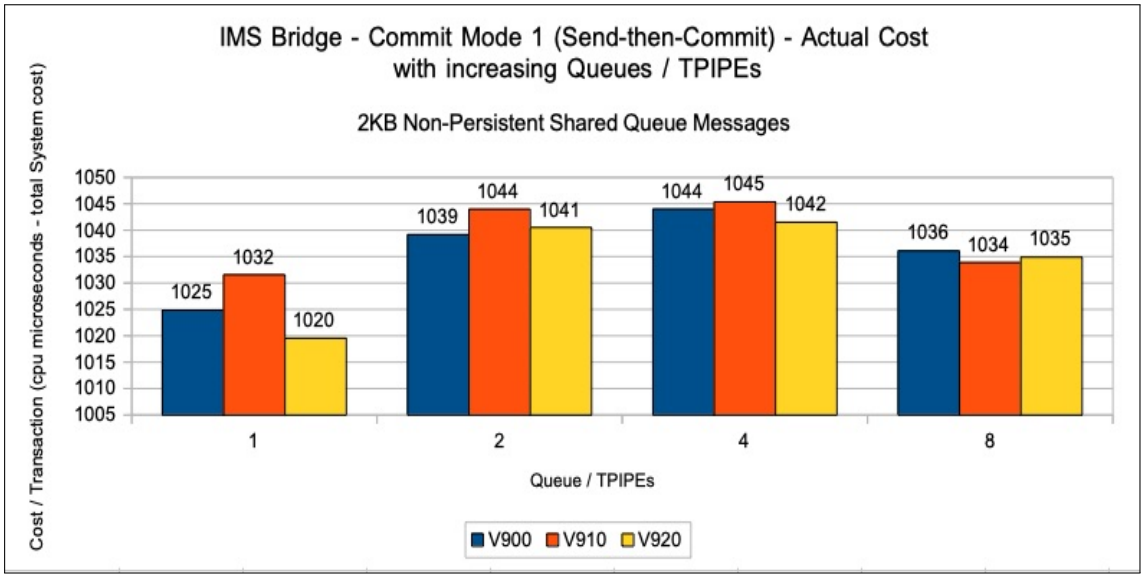


Chart: IMS Bridge commit mode 1 - Transaction cost



Trace

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

Since the performance for V900, V910 and V920 is comparable, the charts in this section show only V920 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

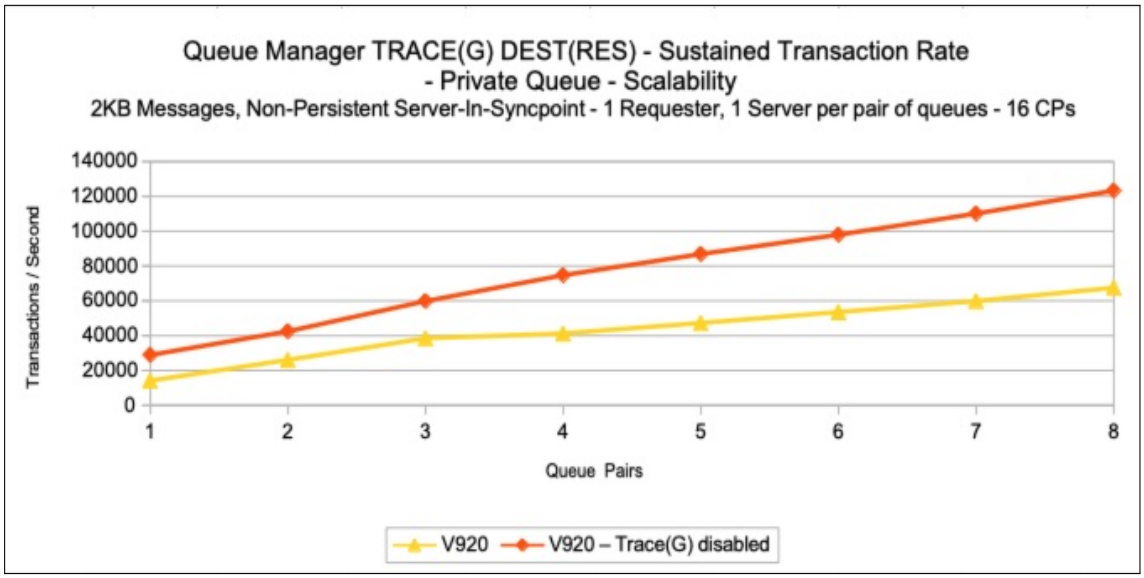
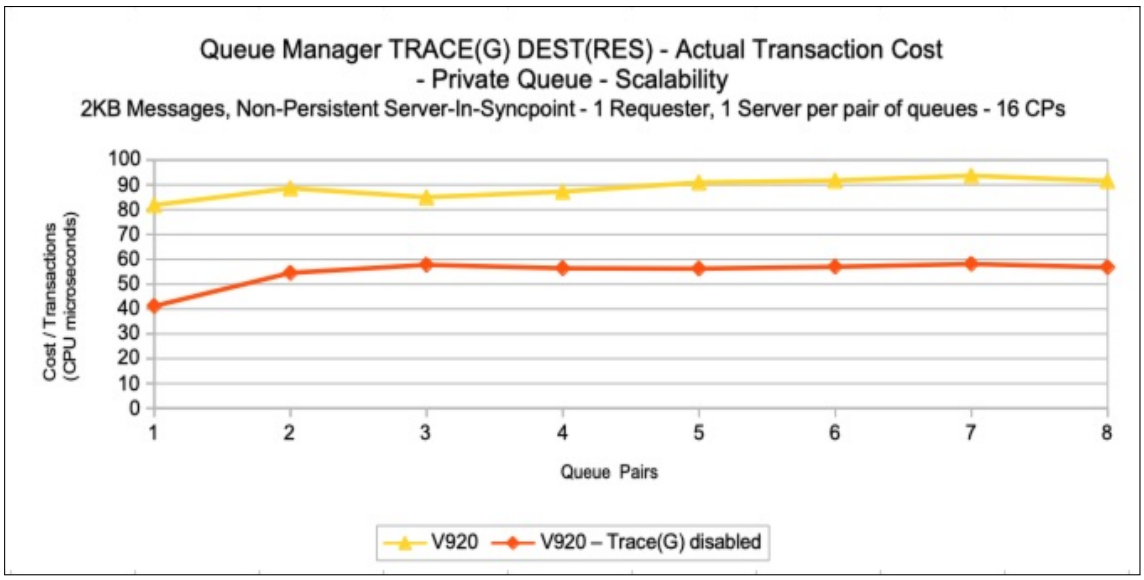


Chart: Queue manager TRACE(G) DEST(RES) - transaction cost



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

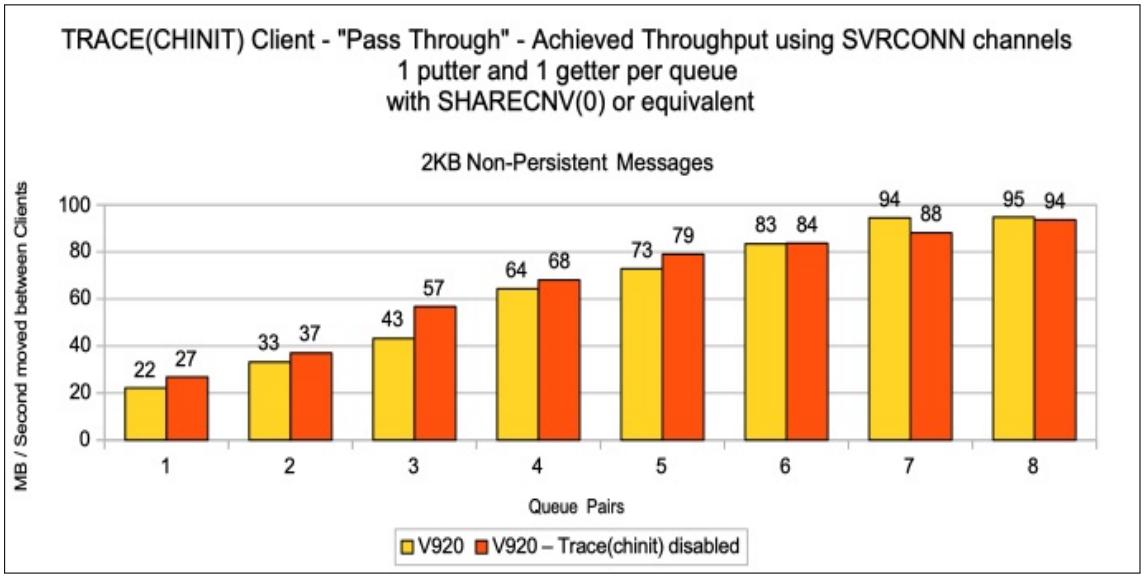
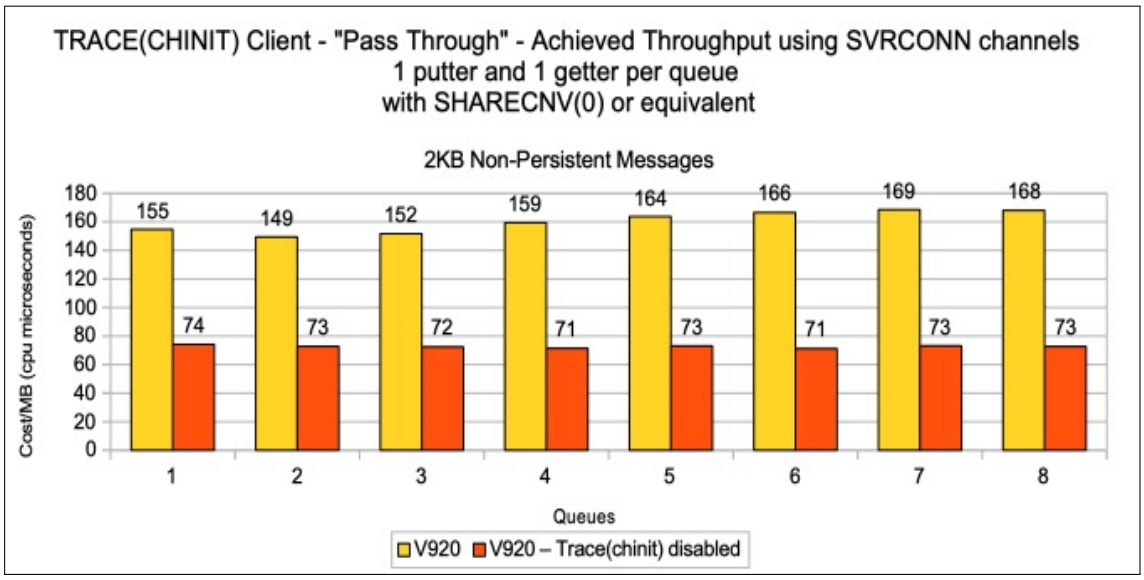


Chart: Channel initiator TRACE(CHINIT) - cost per MB



Note that there was sufficient CPU capacity such that the measurements with TRACE CHINIT enabled were able to achieve similar 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 version 9.2 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 9.0.0 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 version 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

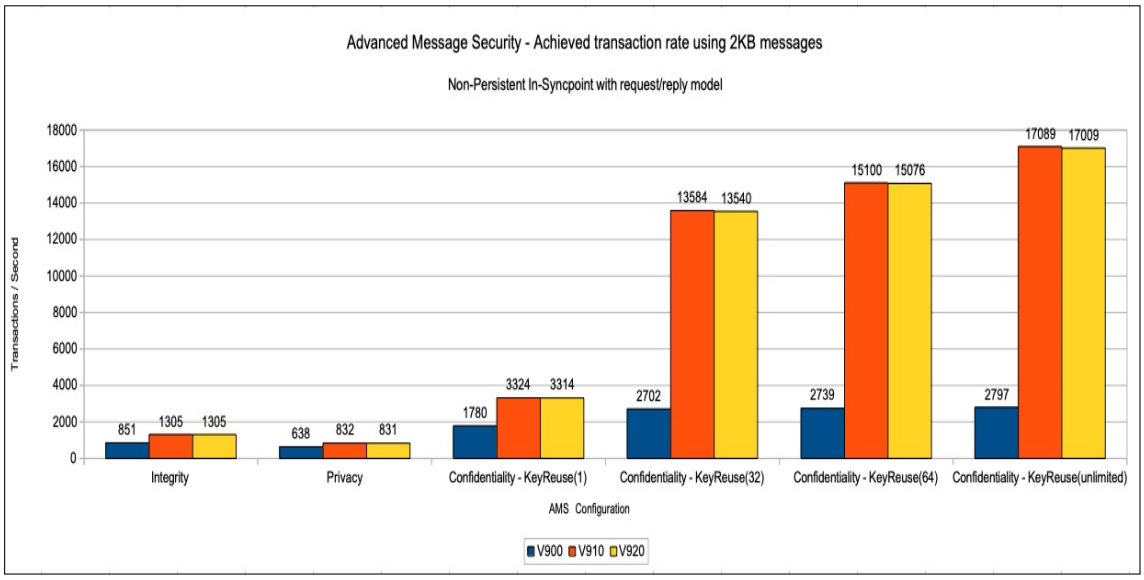
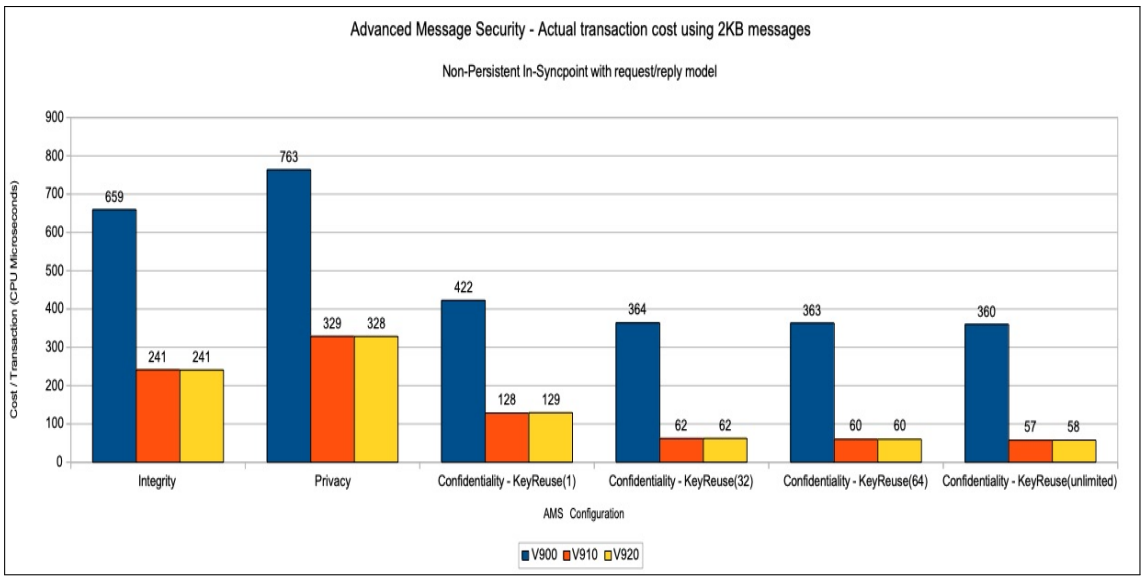


Chart: Transaction cost for 2KB non-persistent workload with AMS protection applied to request and reply queues



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

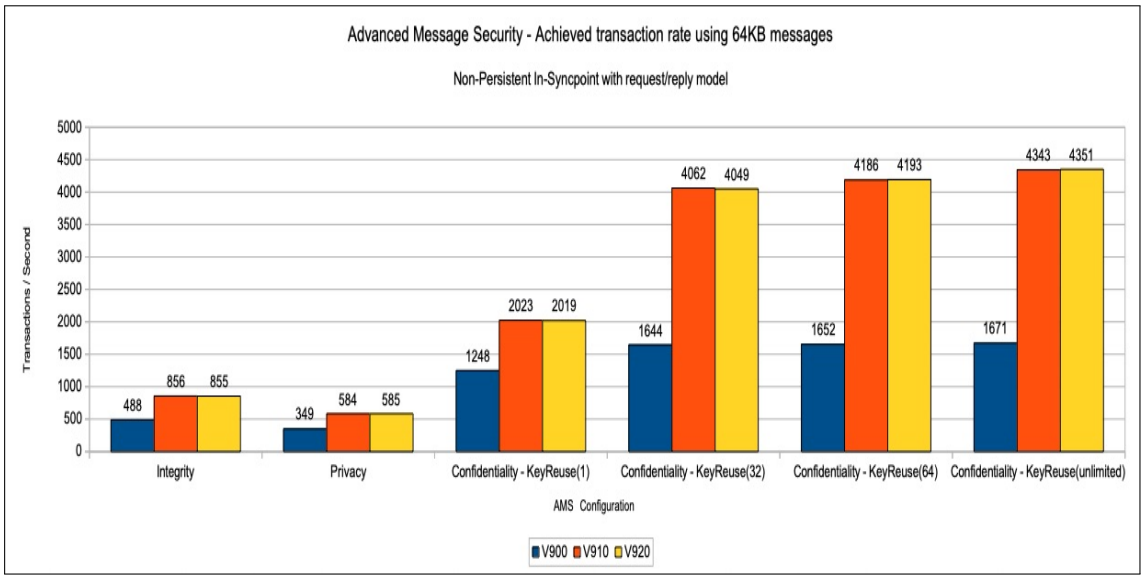
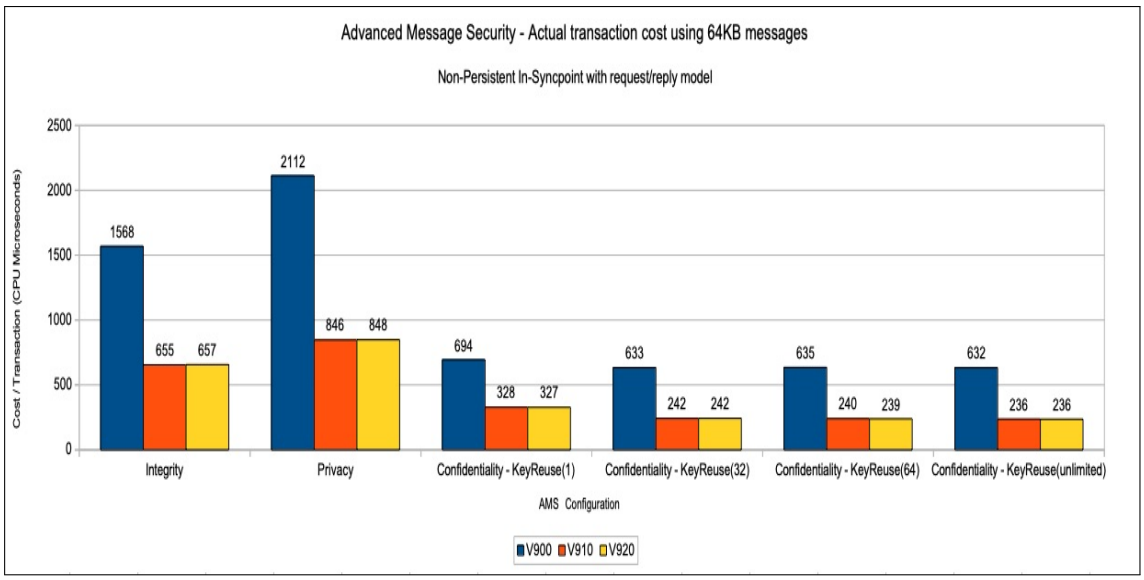


Chart: Transaction cost for 64KB non-persistent workload with AMS protection applied to request and reply queues



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

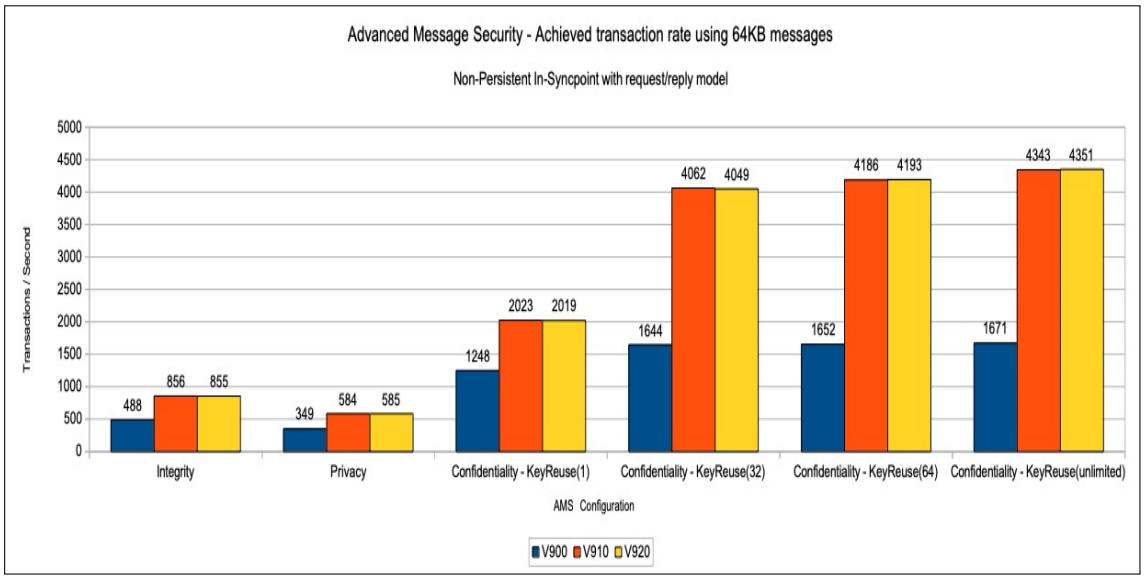
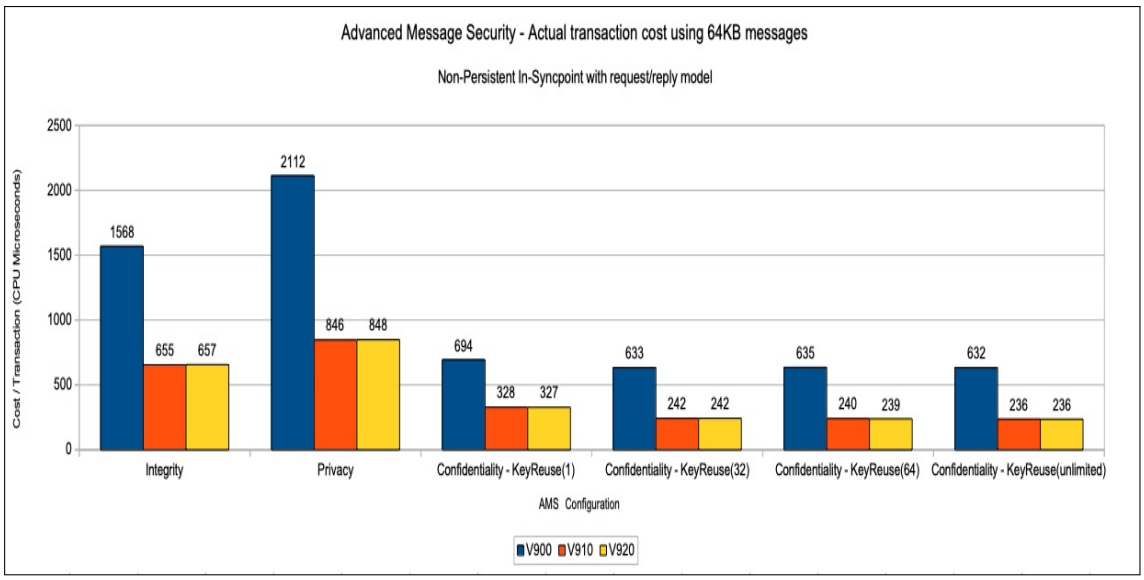


Chart: Transaction cost for 4MB non-persistent workload with AMS protection applied to request and reply queues



Appendix B

System configuration

IBM MQ Performance Sysplex running on z15 (8561-7J0) configured thus:

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

LPAR 2: 1-3 dedicated CP processors, 44GB of real storage.

LPAR 3: 1-10 dedicated CP processors, plus 2 zIIP, 44GB of real storage.

Default Configuration:

3 dedicated processors on each LPAR, where each LPAR running z/OS v2r4 FMID HBB77C0.

Coupling Facility:

- Internal coupling facility with 4 dedicated processors.
- Coupling Facility running CFCC level 24 service level 00.18.
- Dynamic CF Dispatching off.
- 3 x ICP links between each z/OS LPAR and CF

DASD:

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

System settings:

- zHPF disabled by default.
- HIPERDISPATCH enabled by default.
- LPARs 1 and 3 configured with different subnets such that tests moving messages over channels send data over 10GbE performance network.
    - SMC-R enabled by default between LPARs 1 and 3.
    - 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.
  - IBM MQ for z/OS version 9.1 with latest service applied as of August 2020.
  - IBM MQ for z/OS version 9.0 with latest service applied as of August 2020.