IBM MQ for z/OS on Channel Compression

Version 1.0 – December 2022

Tony Sharkey

IBM MQ Performance
IBM UK Laboratories
Hursley Park
Winchester
Hampshire

Notices

DISCLAIMERS
The performance data contained in this report was 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 upon 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 using channel compression.* The information is not intended as the specification of any programming interface that is provided by IBM MQ. It is assumed that the reader is familiar with the concepts and operation of IBM MQ for z/OS.

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 of their respective companies in the United States, other countries, or both:

- **IBM Corporation:** IBM
- **Intel Corporation:** Intel, Xeon
- **Red Hat:** Red Hat, Red Hat Enterprise Linux

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.

Preface

In this paper, I will be looking at MQ for z/OS channel compression performance on IBM z16.

This paper is split into several parts:
Part one - What is data compression?
Part two - IBM zSystems and compression.
Part three - Compression and IBM MQ for z/OS.
Part four - Summary of performance measurements.
Part five - What else to consider

Part one offers a brief overview of compression and why you might consider using it.

Part two discusses IBM zSystems and the history of hardware compression as well as system level options for monitoring.

Part three discusses where compression might be used on MQ for z/OS, how to monitor the success of compressing your MQ data and how to ensure you have enough MQ resources to avoid impacting other workloads. We also discuss how channel compression fits into the process of sending and receiving messages over MQ channels and which compression option might be best for you.

Part four looks at the configurations that we used and how compression affected the cost of the transactions and throughput.

Part five considers the impact of applying Advanced Message Security policies to messages, the fasp.io gateway on high latency networks and short-lived MQ channels.

Table of Contents

1. What is data compression?

Data compression is the process of encoding, re-structuring or otherwise modifying data in to reduce its size. Fundamentally, it involves re-encoding information using fewer bits than the original representation.

When might data compression be useful?

The main advantages of compression are reductions in storage hardware, data transmission time and communication bandwidth. This can result in significant cost savings. Compressed data requires less storage capacity than uncompressed files, meaning less expense in storage. Additionally compressed data requires less time for transfer while consuming less network bandwidth.

The main disadvantage of compression is there are increased use of computing resources to apply compression and decompression to the relevant data.

2. IBM zSystems and compression

Hardware compression has been available on IBM zSystems since zEC12 for zlib data compression.

The zlib data compression library provides in-memory compression and decompression functions, including integrity checks of the uncompressed data. A modified version of the zlib compression library is used by zEnterprise Data Compression (zEDC). The IBM-provided zlib compatible C library provides a set of wrapper functions that use zEDC compression when appropriate and when zEDC is not appropriate, software-based compression services are used.

From zEC12, zEDC was available as an optional PCIe feature.
However, from z15 zEDC is provided as on-chip integrated compression.

This document will primarily discuss zEDC on IBM z16, but a summary of the differences is shown following.

zEDC on PCIe (zEC12 to z14)

- The minimum size of data that can processed by zEDC on PCIe is 4KB.
- Thresholds can be altered using the PARMLIB(IQPPRMxx) member
    - `ZEDC,INFMINREQSIZE=4,DEFMINREQSIZE=4`
- The RMF PCIE report details the level of compression achieved.

Integrated zEDC (on-chip) compression

- The minimum size of data that can be processed is 1KB.
- Thresholds can no longer be changed.
- On-chip compression runs in 2 modes – synchronous and asynchronous
    - There are no RMF reports for synchronous compression.
    - Asynchronous compression performance is reported by RMF using the EADM (Extended Asynchronous Data Mover) report.

How do I know if zEDC is available?

The "`D IQP`" command reports whether the zEDC feature is enabled and what thresholds are set to determine whether compression is via software or hardware.

For IBM z15 and later hardware, the output of the command is:

```
zEDC Information
DEFMINREQSIZE: 1K (STATIC)
INFMINREQSIZE: 1K (STATIC)
Feature Enablement: Enabled
```

For IBM z15 and later hardware, it is no longer possible to change the thresholds of compression or decompression using the IQPPRMxx PARMLIB member.

Attempting to alter either the DEFMINREQSIZE or INFMINREQSIZE will return:
`IQP062I REQUEST REJECTED - OPTIONS IGNORED`

How do I know if zEDC compression is using hardware?

IBM documentation "Integrated Accelerator for z/OS – z/OS applications" discusses support for in-application compression through zlib, similar to that used by MQ, and provides options for assessing compression.

- SMF type 113 records – hardware capacity, reporting and statistics will record synchronous compression at a *system* level.
- SMF type 30 records include zEDC metrics at a job level. However as the "zEDC usage statistics section" indicates some of the fields are not set on z15 and later.
    - On IBM z15 or later, only authorised compression requests are included – C/Java are not tracked.
        - As such, the SMF 30 zEDC data reported for the MQ channel initiator does not include accurate counts of hardware compression usage.

3. Compression and MQ for z/OS

Whilst this paper is primarily aimed at MQ for z/OS and channel compression, there are several areas that MQ for z/OS can make use of compression routines.

MQ generally limits the compression types supported to:

- **RLE** (Run Length Encoding), which is a form of lossless data compression in which repeated characters are stored as a single data value and count, rather than as the original repeating character.
- ZLIB (and variations **ZLIBFAST** and **ZLIBHIGH**), which is a library for data compression, and supports the DEFLATE algorithm.
    - ZLIBFAST will where possible, attempt to compress and decompress using zEDC hardware. If the data to be compressed is not eligible for hardware compression, the compression or decompression will be performed in software.
    - ZLIBHIGH will attempt to compress the data as much as possible, but will be performed in software, thus adding to MQ address space costs.

Compression of active logs

The queue manager attribute `COMPLOG` can be set to `RLE` to compress data written to the MQ active log datasets. There will be additional CPU cost associated with compression, and if required during recovery, decompression) but this may be offset by the I/O savings (subject to the data being suitable for RLE compression).

Compression of archive logs

Compression of archive logs uses the z/OS compression features which are discussed in detail in the blog "Reducing storage occupancy with IBM zEnterprise Data Compression (zEDC) on IBM MQ for z/OS".

From IBM z15, this form of compression using zEDC uses the asynchronous compression and can be monitored using the RMF EADM (Extended Asynchronous Data Mover) report.

MQ Channel Compression, including SVRCONN

MQ channels offer 2 options to compress data flowing over channels:

COMPHDR

This attribute is a list of header data compression techniques supported by the channel.

For sender, server, cluster-sender, cluster-receiver, and client-connection channels the values specified are in order of preference with the first compression technique supported by the remote end of the channel being used. The channels' mutually supported compression techniques are passed to the sending channel's message exit where the compression technique used can be altered on a per message basis. Compression alters the data passed to send and receive exits.

Possible values are:

`NONE`
: No header data compression is performed. This value is the default value.

`SYSTEM`
: Header data compression is performed in software.

COMPMSG

This attribute is a list of message data compression techniques supported by the channel.

For sender, server, cluster-sender, cluster-receiver, and client-connection channels the values specified are in order of preference. The first compression technique supported by the remote end of the channel is used. The channels' mutually supported compression techniques are passed to the sending channel's message exit where the compression technique used can be altered on a per message basis. Compression alters the data passed to send and receive exits.

The possible values are:

`NONE`
: No message data compression is performed. This value is the default value.

`RLE`
: Message data compression is performed using run-length encoding.

`ZLIBFAST`
: Message data compression is performed using the zlib compression technique. A fast compression time is preferred.
ZLIBFAST can optionally be offloaded to the zEnterprise® Data Compression facility. See zEDC Express facility for further information.

`ZLIBHIGH`
: Message data compression is performed using the zlib compression technique. A high level of compression is preferred.

`ANY`

Allows the channel to support any compression technique that the queue manager supports. Only supported for Receiver, Requester and Server-Connection channels.

This attribute is valid for all channel types.

COMPMSG(ZLIBFAST)

MQ channel compression using COMPMSG(ZLIBFAST) can use zEDC **synchronous** compression, and as such there are no specific RMF reports to indicate whether the data compressed using ZLIBFAST used hardware or software compression in the specific MQ channel initiator address space.

Does MQ offer monitoring options for channel compression?

Yes – there are two options, which both require `MONCHL(LOW|MEDIUM|HIGH)` to be enabled. On z/OS there is no difference between low, medium, and high for the MONCHL attribute.

MQSC's "DISPLAY CHSTATUS" command returns the `COMPRATE` and `COMPTIME` values.

COMPRATE:

The compression rate achieved displayed to the nearest percentage; that is, a rate of 25 indicates messages are being compressed to 75% of their original length. Two values are displayed:

- The first value based on recent activity over a short period.
- The second value based on activity over a longer period.

These values are reset every time the channel is started and are displayed only when the STATUS of the channel is RUNNING. If monitoring data is not being collected, or if no messages have been sent by the channel, the values are shown as blank.

COMPTIME:

On z/OS, COMPTIME is the amount of time for each message, provided that the message does not have to be processed in segments. This segmenting of the message on z/OS occurs when the message is:

- 32 KB or larger, or
- 16 KB or larger, and the channel has TLS encryption.

If the message is split into segments, COMPTIME is the time spent compressing each segment. This means that a message that is split into 8 segments spends "COMPTIME multiplied by 8" microseconds during compression or decompression.

In the following examples the message size is 63KB and the channels have TLS encryption.

ZLIBFAST	**ZLIBHIGH**
CHSTATUS(VTS1_VTS2_0001) MONCHL(HIGH) COMPTIME(9,9) COMPRATE(76,74)	CHSTATUS(VTS1_VTS2_0001) MONCHL(HIGH) COMPTIME(116,106) COMPRATE(76,75)

As the message is 63KB, there are four segments – to the total time spent compressing the message can be determined thus:
ZLIBFAST: 36 microseconds
ZLIBHIGH: 464 microseconds

For both ZLIBFAST and ZLIBHIGH, the message was compressed to approximately 25% of the original size.

Note: The `DISPLAY CHSTATUS` command can also report the `COMPMSG` setting using on the channel.

MQ class(4) accounting trace provides the compression rate, which can be formatted using the MQSMF program provided as part of supportPac MP1B "Interpreting accounting and statistics data". The DCHS report shows the compression rate achieved, for example:

```
VTS1_VTS2_0001 10.20.20.20 Compression rate 76
```

How does MQ channel compression work?

The MQ channel initiator has several different types of tasks including:

- **Adaptor** provides the interface between the channel initiator and the queue manager.
- **Dispatchers** manipulate and send/receive the message over the network.
- **SSL task** provides secure environment for secret key negotiation, encryption of data etc.

The **dispatcher** task is responsible for any channel compression activity.

Sample flows for MQ channel with COMPMSG(ZLIBFAST)

Sender-side:

1. Message is put to a queue which results in the message arriving on a transmit queue.
2. Channel initiator adaptor task gets the message and holds the message in a buffer, notifying the dispatcher.
3. The dispatcher will assess the message to determine whether it can be processed in a single chunk.
    a. For channels with SSLCIPH specified, the maximum 'chunk' size is 16KB
    b. For all other channels, the maximum chunk size is 32KB.
    c. Each chunk of data will be compressed individually.
    d. A 33KB message on an encrypted channel would be 3 chunks of 16, 16 and 1KB.
    e. On IBM z15 or later, each chunk would be compressed in hardware.
    f. On IBM z14, the 16KB chunks would use hardware and the 1KB would be in software.
4. Once each chunk is compressed, if SSLCIPH was set the message would be encrypted.
5. The compressed (and optionally encrypted) chunk is sent.

Receiver-side:

1. Dispatcher receives the chunk of data from the socket.
2. If required, decrypts the data.
3. Inflate the message – if the message is below threshold, inflate in software.
4. Reconstruct the message chunks into the full message.
5. When the full message has been re-assembled, notify adaptor task.
6. Adaptor puts the message to the desired queue.

Should I use compression with my MQ channels?

Consider why you might want to use compression.

Are you attempting to reduce cost on z/OS?

Channel compression occurs at non-zero cost. To reduce the overall cost, this additional cost of compressing and decompressing the data needs to be offset.
Using MQ channels with TLS protection may offer some opportunity to offset the increased cost of compression by reducing the cost of both encryption and more significantly secret key negotiation at the interval controlled by `SSLRKEYC`.

Are you attempting to reduce amount of data flowing over the network?

If your network is limited on bandwidth or is high latency or unreliable, or indeed charged for usage, channel compression may provide some benefit in reducing the amount of data flowing, provided the data is compressible.
In the case of usage charges, whether the reduction in network usage offsets the increased CPU cost from compression is something for you to determine.

Are you attempting to improve throughput rate?

Compressing messages may result in improved throughput provided the compression option selected is sufficiently efficient (COMPTIME and COMPRATE). Does the data compress sufficiently that network or SSL encryption times are reduced?
On IBM z14, we found that whilst ZLIBFAST running on zEDC (PCIe) was efficient, there was added latency from the switch to/from the PCIe feature that made the use of compression less desirable.

What data does your message contain? Are your messages compressible?

If your data is already compressed, for example you are sending a ZIP file across your channel, attempting to re-compress the message may not provide any further benefit but may add additional cost from attempting to compress.

If you are sending a report, perhaps with many repeating space characters, RLE might offer a simple and cheap compression routine. Similarly, the ZLIB-prefixed compression types can compress RLE data and depending on the size of the data to be compressed, you may find ZLIBFAST using hardware compression is more effective than regular RLE compression.

Which compression option should I use?

Ultimately you know your data – it is impossible for MQ to determine whether your data is compressible until the dispatcher attempts to compress it. Similarly, MQ cannot determine which compression type might best suit the message payload for any channel.

All we can offer is guidelines on cost and benefits for compression types and compressibility of data on our low-latency networks.

The trade off – dispatcher or SSL tasks

MQ channel compression will increase the cost of the work performed by the dispatcher task, regardless of the compression type specified.

How much that cost increases will depend on the compression type as well as the size and compressibility of the data.

Attempting to compress data that is either incompressible or largely incompressible may see the cost of compression increase significantly. Data that is easily compressible, such as data with large numbers of repeating characters may be ideal for RLE compression, but both ZLIBHIGH and ZLIBFAST can also perform RLE compression.

Whilst compressing data will increase MQ dispatcher usage, for channels protected using TLS ciphers, the compressed data may result in two benefits:

1. Less data to encrypt
2. More messages can flow over the channel before the `SSLRKEYC` threshold for secret key re-negotiation is reached.

For workloads where the messages are compressible, the reduction in SSL task cost for certain TLS ciphers with secret key negotiation enabled can offset the additional dispatcher task cost such that the overall transaction cost is reduced.

The following table offers an example analysis of the costs attributed to MQ channel initiator tasks for a requester-side request/reply flow of a 16KB message that was 40% compressible flowing over a channel protected with cipher `ECDHE_RSA_256_CBC_SHA384`. Compression is provided using ZLIBFAST.

CPU by task	**Non-Compressed**	**Compressed**
Adaptor	10	10
Dispatcher	35	68
SSL Task	75	52
Total	*120 microseconds*	*130 microseconds*

By compressing the data, the SSL task has reduced cost from 75 to 52 CPU microseconds, a reduction of 23 microseconds.
However, the dispatcher task has increased cost by 33 microseconds, so overall there is a net increase of 10 microseconds per transaction in the channel initiator.

Do I have enough dispatcher tasks?

When using channel compression, there may be increased use of the dispatcher tasks.

It is advisable to monitor the class(4) statistics trace data for dispatcher usage. This can be viewed in MQSMF's "DISP" report.

```
Task,Type,Requests,Busy %,     CPU used, CPU %,"avg CPU","avg ET"
    ,    ,        ,      ,      Seconds,      , uSeconds,uSeconds
   0,DISP,  166073,  97.6,    58.111662, 96.9,       350,      353
   1,DISP,  164092,   3.7,     2.308928,  3.8,        14,       14
Summ,DISP,  330189,   2.0,    60.420713,  2.0,       183,      184
   0,DISP, number of channels on this TCB,    1
   1,DISP, number of channels on this TCB,    1
Summ,DISP, number of channels on all TCBs,    2 task MQSMF's
```

In this example report the channel initiator is running with just 2 dispatcher tasks and 2 active channels.

The channels have been configured with COMPMSG(ZLIBHIGH) and the workload is 100KB non-persistent messages.

Dispatcher 0 is being used for the outbound channel – where compression is occurring. Dispatcher 1 is being used for the inbound channel – where decompression is occurring.

Even with a single outbound channel, dispatcher 0 is 97.6% busy through the interval, and as such would be unlikely to support additional channels without affecting the performance of the channels.

In this environment, it would be advisable to have additional dispatcher tasks available and to ensure that the ratio of dispatchers to maximum channels is set appropriately. The relationship between dispatchers and maximum channels is discussed in performance report MP16 "Capacity Planning and Tuning guide" in the "CHIDISPS and MAXCHL" section.

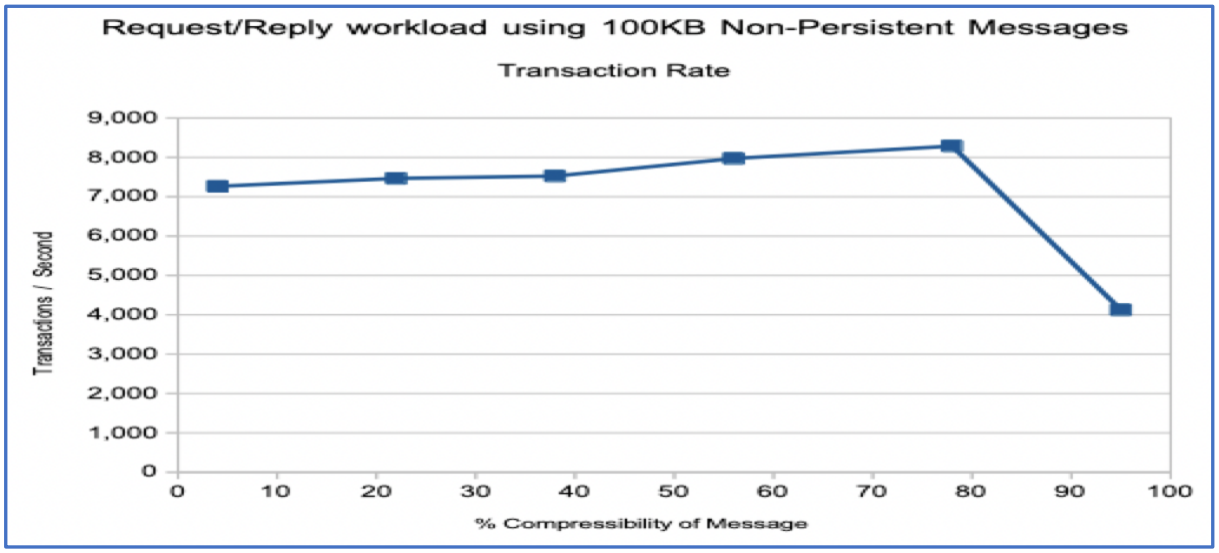
Highly compressible messages and ZLIBFAST

For messages that are highly compressible and flow over MQ channels where `COMPMSG(ZLIBFAST)` is set, the inflate may occur in software which can significantly reduce the benefits of compressing the message data.

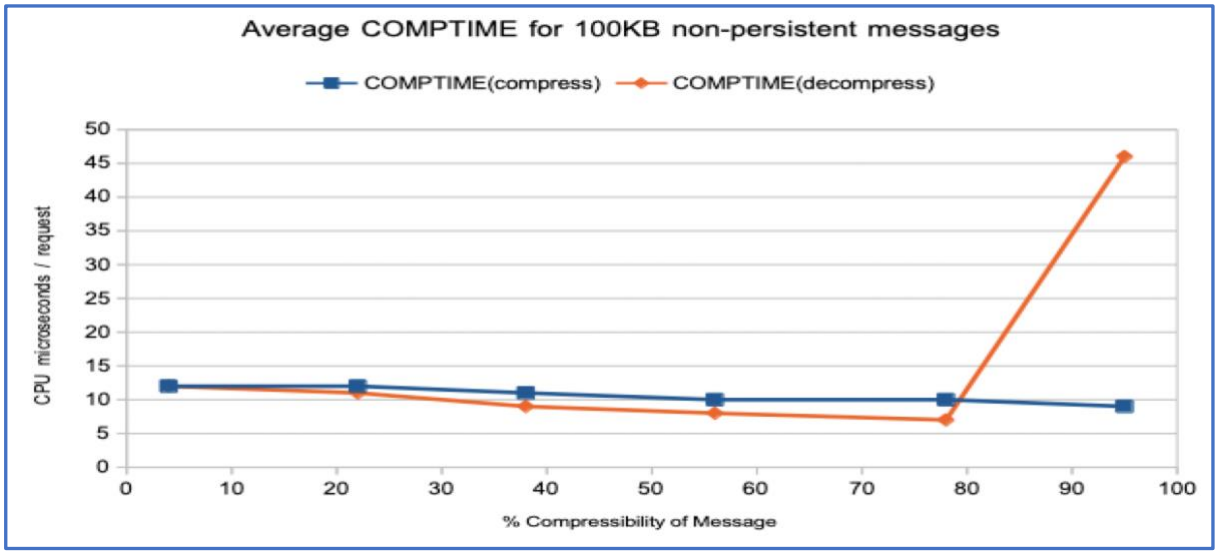
This inflate in software may occur due to the size of the compressed data being below the 1KB "`INFMINREQSIZE`" threshold.

To demonstrate the impact of this software inflation, we ran the following measurements:
Request/reply workload between 2 z/OS-based queue managers using sender-receiver channels configured with `COMPMSG(ZLIBFAST)`.
The measurement used 100KB non-persistent messages of varying compressibility.

This first chart shows that when the 100KB message is highly compressible (95%), the transaction rate is 50% of when the message is 78% compressible.



If we look at the COMPTIME data from the sender channel (compression) and the receiver channel (decompression) we see the reason why the transaction rate dropped:



As the message becomes more compressible, the data received falls below the 1KB threshold and must use software to decompress. This COMPTIME value increases from 7 microseconds for a 100KB message that was 78% compressible, up to 46 microseconds for a message that was 95% compressible.

Additionally, due to the size of the message, it is processed in 32KB chunks, which means there are multiple decompress calls per received message – for a 100KB message we would expect 4 decompress calls. This means that the compression time goes up from 28 microseconds for the 78% compressible message to 188 microseconds for the 95% compressible message.

It is worth noting that neither ZLIBHIGH nor RLE would see a similar increase in decompress costs, but that is because all their decompression is already performed in software.

ZLIBHIGH or ZLIBFAST?

ZLIBHIGH is more aggressive at compressing the message data than ZLIBFAST, but this comes at additional cost.

This is true regardless of whether the ZLIBFAST compression is performed by zEDC or in software.

As such, the question is, what is my aim from compressing the data – is maximum compression the aim, regardless of cost?

If the *only* requirement is maximum compression, then use ZLIBHIGH.

COMPMSG(ZLIBFAST) using software

Enabling IBM MQ for z/OS channels with COMPMSG(ZLIBFAST) on IBM z15 or later will attempt to use zEDC hardware compression.

Should it be necessary to disable hardware compression for COMPMSG(ZLIBFAST), discuss this with your IBM MQ Service representative, who can provide MQ configuration options.

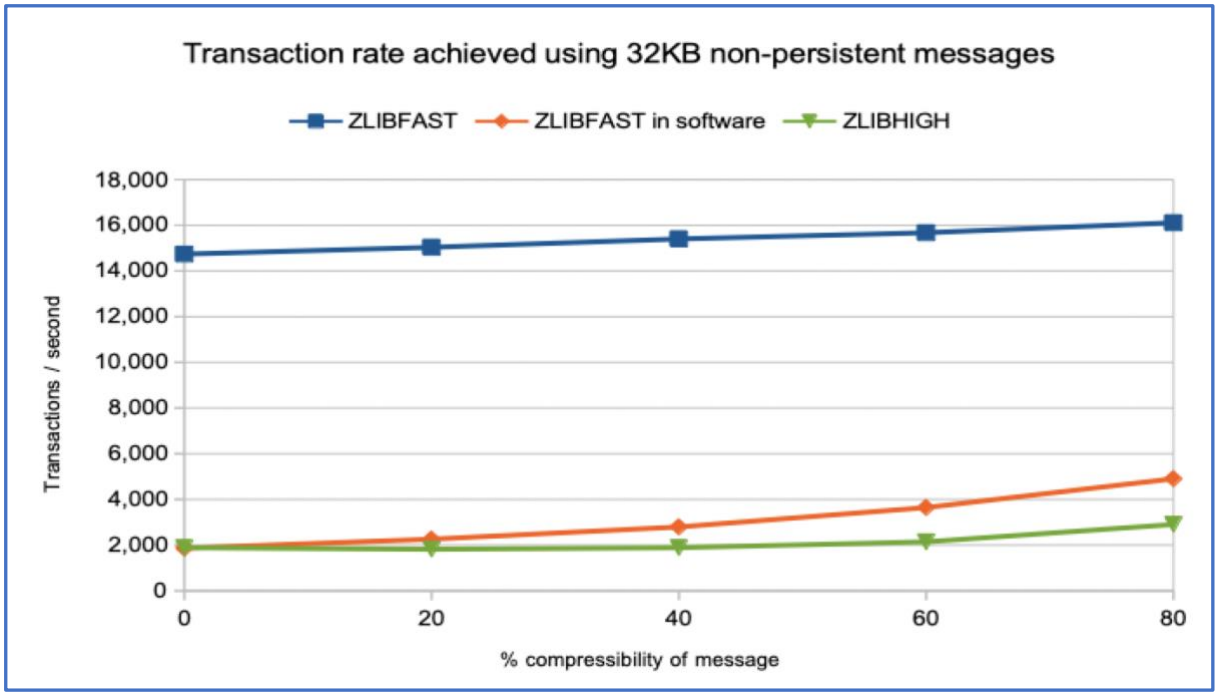
How does ZLIBFAST in hardware compare with ZLIBFAST and ZLIBHIGH in software?

To demonstrate the impact of this forcing ZLIBFAST to use software inflation, we ran the following measurements:
32KB non-persistent request/reply workload between 2 z/OS-based queue managers using sender-receiver channels configured with:

- `COMPMSG(ZLIBFAST)` using hardware.
- `COMPMSG(ZLIBFAST)` using software.
- `COMPMSG(ZLIBHIGH)`.

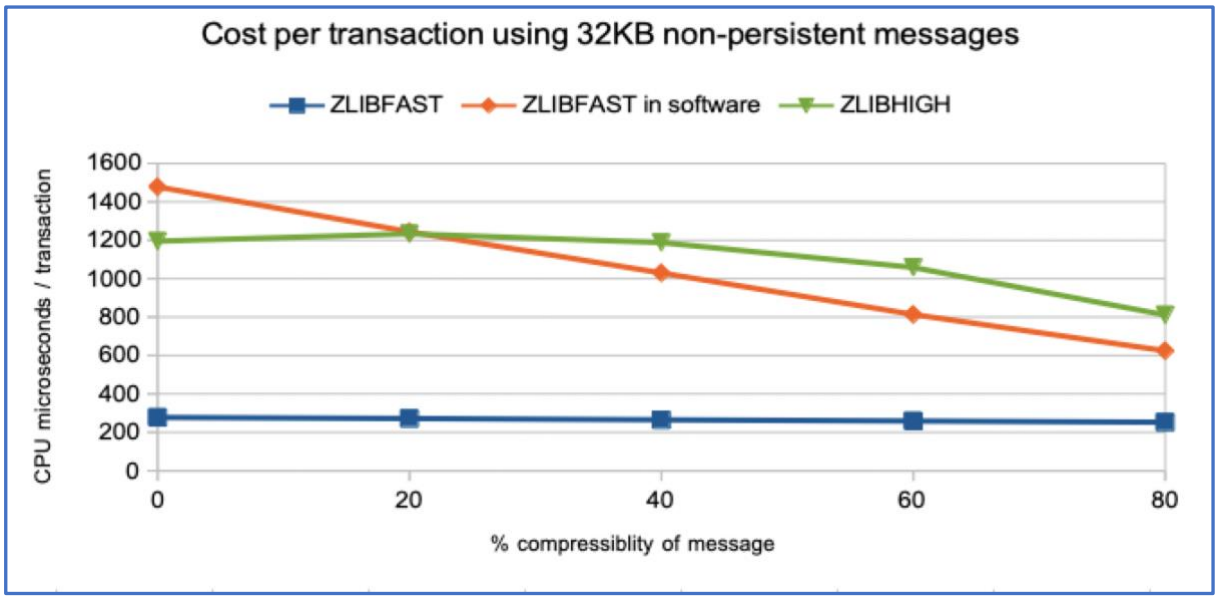
The workload is run using messages of increasing compressibility.

Transaction rate



In this configuration ZLIBFAST in hardware can achieve up to 7 times the transaction rate of either ZLIBFAST in software or ZLIBHIGH for a 32KB non-persistent message.

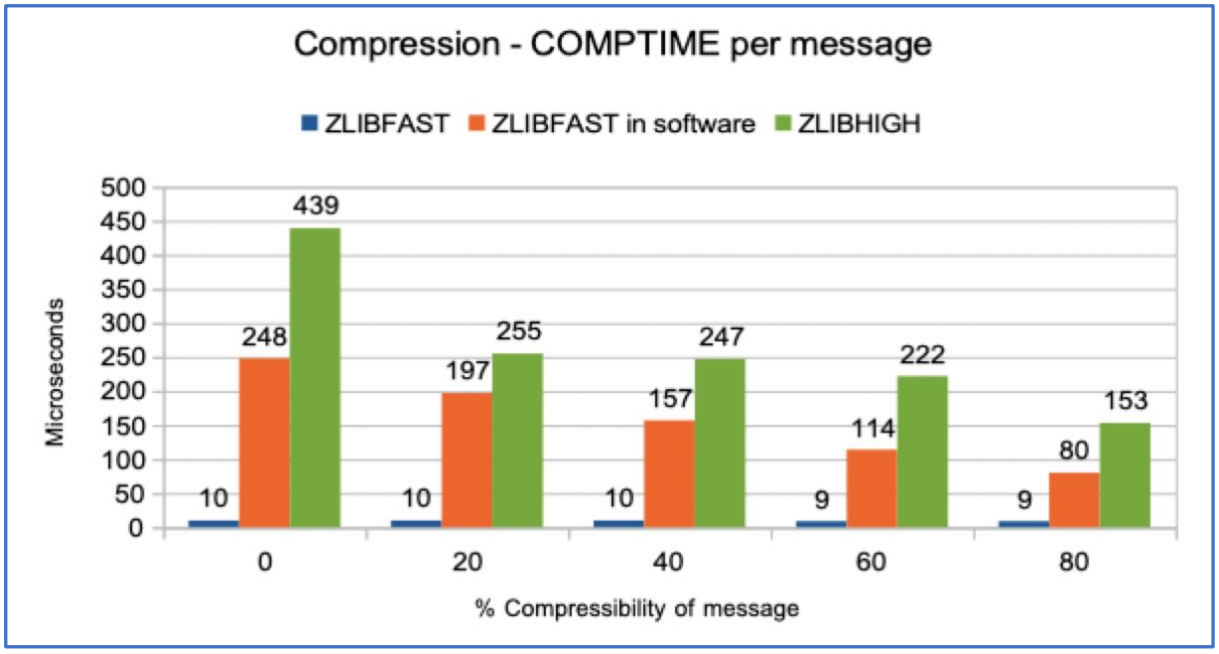
Transaction cost



For a 32KB message compressed using ZLIBFAST where zEDC is available, the transaction cost does not change significantly regardless of how compressible the message may be.

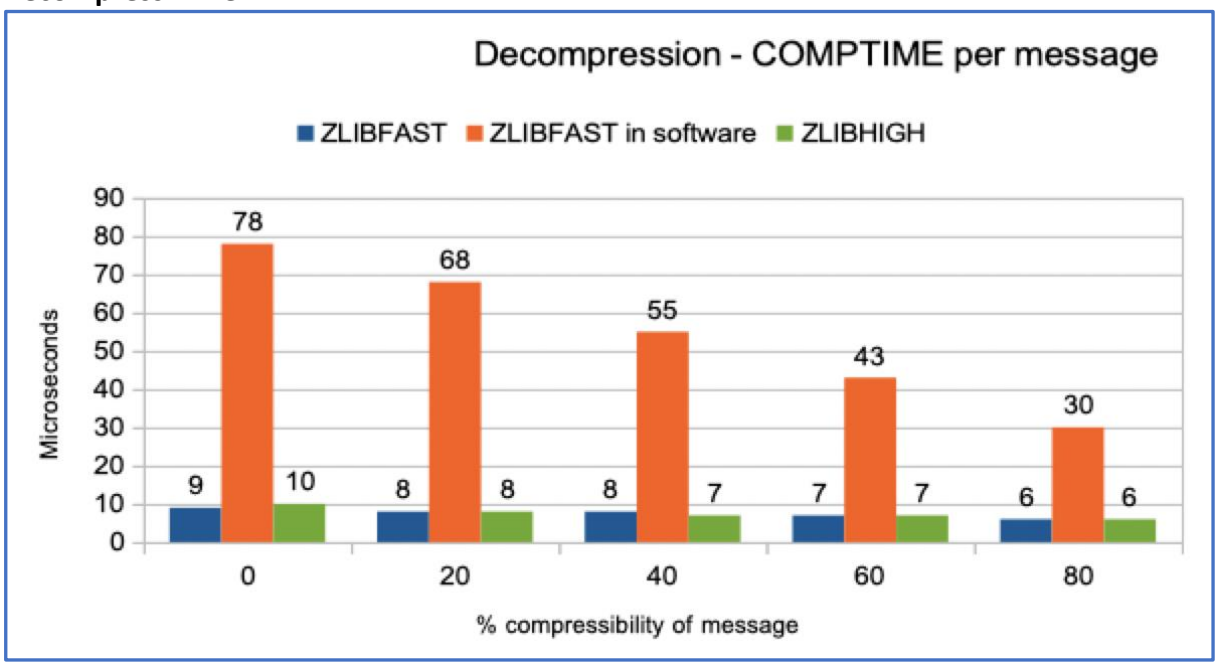
When ZLIBFAST is prevented from using zEDC hardware, the compression rate achieved of the message significantly affects the cost of the transaction – where a highly compressible message may be 50% of the cost of an incompressible message.

Compression Time



ZLIBFAST using zEDC can compress the message data much more quickly than when the compression is performed in software.
ZLIBFAST in software can compress the message data faster than ZLIBHIGH but does not attempt to compress the message so aggressively.

Decompress Time



Inflating compressed data using ZLIBFAST in software is considerably more expensive than inflating either using zEDC or even ZLIBHIGH.

It is not clear why ZLIBHIGH is so much better at decompressing data than ZLIBFAST in software.

Using IBM's Application Performance Analyser (APA) against the channel initiator suggests:

ZLIBHIGH spending time in:		**ZLIBFAST (software)** spends time in:	
CSQXZDEF	41%	CSQXZDEF	63%
CSQXZTRE	24%	CSQXZTRE	24%
CSQXZIFF	19%		
CSQXZADL	5%	CSQXZADL	2.5%

ZLIBFAST using zEDC will see CPU usage in FPZINLPA – the proportions will depend upon the workload.

4. Using MQ message compression

The measurements used in this paper are based on simple request/reply workloads between two z/OS queue managers using two pairs of sender-receiver channels – one channel for outbound and one channel for inbound messages on each queue manager.

The channels are configured with COMPMSG(NONE|RLE|ZLIBFAST|ZLIBHIGH).

Messages range from 2KB to 100KB and are non-persistent.

Messages are generated to range from 0 (incompressible) to 80% (highly compressible). For the messages we have classified as “incompressible”, ZLIB is able to achieve a small degree of compression - up to 3% compressible.

The applications are lightweight and contain minimal processing and are used in our standard micro-benchmark measurements.
Multiple request applications generate the messages, put to a common request queue, and wait for the specific reply messages.
The multiple server applications get-wait for a message and generate the reply message using the message contents.

For the purposes of this paper, we run in three configurations:

1. Channel compression over non-TLS enabled channels.
2. Channel compression over TLS 1.2 protected channels. In this configuration the secret key negotiation is configured to run at 1MB intervals.
3. Channel compression over TLS 1.3 protected channels. As discussed in the MQ for z/OS 9.2 performance report , TLS 1.3 ciphers have secret key re-negotiation included as part of the protocol and therefore measurements are run with `SSLRKEYC(0)`.

What is clear is that when using TLS-protected channels, the cipher selected makes a significant difference to whether the cost of compression is offset by reducing the impact of encryption and secret key negotiation on each transaction.

How does channel compression affect non-TLS enabled channels?

In our measurements, once message compression was enabled, the cost per transaction **exceeded** that of workloads run with message compression disabled for **all** message sizes and compressibility of messages.

Similarly on our low-latency networks, the time taken to compress and decompress the messages meant that we were never able to match the transaction rates achieved when no compression was enabled.

Appendix A shows the sets of data collected in the non-TLS enabled channel configuration, including the transaction cost, transaction rate and compress/decompress times.

With regards to transaction cost, generally ZLIBFAST offered the lowest overhead across the range of message sizes, but for 2KB messages RLE was able to offer a lower cost than ZLIBFAST.

In all message sizes, ZLIBHIGH was prohibitively expensive (2-3 times the cost) when compared to ZLIBFAST but did provide a slightly (1-2%) greater amount of compression.

How does channel compression affect TLS 1.2 protected channels

The TLS 1.2 cipher used to protect the MQ channels does make a difference as to when or indeed whether the added cost of compression is offset by the reduction in cost of encryption and secret key negotiation.

For all TLS 1.2 ciphers the cost of encrypting the data is relatively similar, regardless of actual cipher, and this is shown in the MQ for z/OS on z16 report as part of the "no renegotiation of secret key" section.

Where the transaction cost was most affected was with secret key negotiation, such that TLS_RSA prefixed ciphers were approximately 33% lower cost than 50% of the cost of ECDHE prefixed ciphers.

For the purposes of this paper, there are 2 sections reporting the data for TLS 1.2 protected ciphers:

- Appendix B shows the performance of compression when using channels protected with cipher ECDHE_RSA_256_CBC_SHA384.
- Appendix C shows the performance of compression when using channels protected with cipher TLS_RSA_WITH_AES_256_CBC_SHA256.

For ECDHE prefixed ciphers the compressed configurations frequently resulted in an overall reduction in transaction cost, occasionally where the message was compressed by 40% but more often when compressing the message by 60%.

For the TLS_RSA prefixed ciphers, the reduced cost of SSL processing was rarely sufficient to offset the increase in cost from compression, such that only highly compressible messages achieved an overall decrease in transaction cost compared to the baseline runs where compression was not used.

It is worth re-iterating that TLS-protected channels process messages in chunks of 16KB whereas non-TLS protected channels process messages up to 32KB in a chunk. The value reported by COMPTIME is the average time per compression request, and therefore there may be additional compression requests for a 32KB message over a TLS-enabled channel than over a non-TLS enabled channel.

As the message compression occurs before an TLS-protection is applied, the cipher used has no impact to the COMPTIME or COMPRATE for either compressing or decompressing the message.

How does channel compression affect TLS 1.3 protected channels

As TLS 1.3 ciphers have secret key re-negotiation included as part of the protocol, there is less opportunity for the cost of compression to be cancelled out solely by reducing the cost of encryption.

Currently MQ for z/OS 9.2 onwards supports 3 TLS 1.3 ciphers:

- TLS_AES_128_GCM_SHA256
- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256

Of the 3 ciphers, the encryption costs for the TLS_AES prefixed ciphers are similar, with TLS_CHACHA20_POLY1305_SHA256 cipher being significantly more expensive as it is unable to make full use of CPACF.

As a result, there are 2 sections reporting the data for TLS 1.3 protected ciphers:

- Appendix D shows the performance of compression when using channels protected with cipher TLS_AES_128_GCM_SHA256.
- Appendix E shows the performance of compression when using channels protected with cipher TLS_CHACHA20_POLY1305_SHA256.

For the TLS_AES prefixed ciphers, the reduced cost of encryption processing was **never** sufficient to offset the increase in cost from compression, resulting in an overall increase in transaction cost, regardless of compressibility or size.

For the TLS_CHACHA20_POLY1305_SHA245 cipher, compression can demonstrate benefits to both transaction cost and throughput achieved. The relatively high cost of encrypting data using this cipher means that compressing prior to encryption can reduce the overall transaction cost for all sizes of messages, even messages that are just a few percent compressible.

5. What else to consider

Short-lived MQ channels

The report has discussed the benefits of channel compression particularly over TLS-enabled channels due to reducing the number of secret key negotiations.

Channel start of a TLS-enabled channel also include the relatively expensive process of secret key negotiation and use of channel compression will not reduce the frequency of key negotiation if the channel is running for only a short time.

Advanced Message Security (AMS)

Policies applied to messages may result in encrypted data – which may be largely incompressible.

As this encryption occurs at the time the message is put, channel compression is unlikely to provide any benefit, and indeed may degrade performance as there will be cost incurred from attempting to compress a message that is incompressible.

This differs to channels protected by TLS-encryption, where the message is compressed before encryption.

Aspera fasp.io gateway

TCP/IP does not perform particularly well over large distances, due to its relatively conversational mode of operation.

Since MQ 9.2, MQ for z/OS supports the Aspera fasp.io gateway which can improve the flow of data between geographically remote partners.

The fasp.io gateway can run on a Linux on Z LPAR or from z/OS v2r4 on a zCX “container extension” address space.

The performance of MQ channels over high latency networks using the fasp.io gateway is discussed in "MQ with zCX”.

Messages compressed using `COMPMSG(ZLIBFAST)` can further improve the rate of transfer.

Summary

As we have discussed throughout this paper, compression is not free, even when MQ is able to use zEDC hardware compression. To have the possibility of saving CPU cycles in the MQ channel initiator whilst using compression, that additional cost needs to be offset by saving cycles elsewhere.

For channels configured with ZLIBFAST where zEDC may be used there is still some additional cost in setting up the environment to request compression. The actual decision to compress ZLIBFAST in zEDC or in software cycles is taken at a lower level than MQ for z/OS.

The most likely candidate for saving cycles is for channels using TLS ciphers to encrypt the data. With constant improvements to encryption processing, largely performed by CPACF on z/OS, the best opportunity comes from compressing the data so that the secret key negotiation happens less frequently (TLS 1.2 ciphers).

Whether the cost of compression is offset by the lower cost incurred from fewer secret key negotiations will depend on several variables:

- Value of SSLRKEYC.
- Compression type used.
- The contents of the message - How compressible is the data?
- The size of the messages.

TLS 1.3 ciphers do not use the traditional secret key renegotiation process, but the TLS_CHACHA20_POLY1305_SHA256 cipher is not able to encrypt data using CPACF, so does offer an opportunity for compression to assist in reducing the overall cost within the channel initiator address space.

Ultimately, whether you can see performance benefits from compression will depend on the type of data you are sending/receiving over MQ channels and the network between the queue managers, or indeed the queue manager and client(s).

When considering compression, it is worth:

- Monitor your channel performance – whether using the DISPLAY CHSTATUS command or using MQ's Accounting trace – class(4).
- Know your data – is it suitable for compression, and if so, what type of compression.
- Know your why – what are you trying to achieve, i.e., is it reducing the z/OS cost, improving transfer rate or something else?

From a general performance perspective, you should always be monitoring your systems, so that you can identify bad performance from good performance, but particularly around compression. Firmware / microcode updates can affect the tipping point for compression versus encryption – at IBM z15 GA with CryptoExpress7S, it was more likely that TLS-protected channels would benefit from compression. By the time our systems migrated to IBM z16, encryption costs had improved such that the benefits of compression were harder to identify.

Appendix A – Channel Compression over unencrypted channels

This section provides a summary of the data collected when comparing message compression options over unencrypted channels, i.e., where SSLCIPH was not configured.

1. Transaction cost in CPU microseconds. Cost includes MQ MSTR and CHIN address spaces plus TCP/IP and lightweight application cost.
2. Achieved transaction rate.
3. Time spent in compression, as per COMPTIME.
4. Time spent in decompression, as per COMPTIME.

Table 1: Transaction cost in CPU microseconds

Message Size	Compress	0%	20%	40%	60%	80%
2KB	NONE	107.81	107.81	107.81	107.81	107.81
	RLE	122.03	120.88	118.19	116.85	114.43
	ZLIBFAST	136.24	135.49	135.44	152.61	149.72
	ZLIBHIGH	216.86	236.66	251.81	283.87	294.33
8KB	NONE	117.40	117.40	117.40	117.40	117.40
	RLE	164.57	159.80	155.36	147.52	140.71
	ZLIBFAST	152.85	150.83	149.14	146.59	145.31
	ZLIBHIGH	365.25	427.83	470.23	490.89	433.44
16KB	NONE	125.79	125.79	125.79	125.79	125.79
	RLE	217.24	205.81	194.38	180.59	169.35
	ZLIBFAST	172.07	168.74	164.77	161.62	158.78
	ZLIBHIGH	624.82	731.64	785.86	690.71	562.94
32KB	NONE	183.80	183.80	183.80	183.80	183.80
	RLE	283.47	344.03	319.59	296.52	272.42
	ZLIBFAST	277.67	271.88	264.99	259.07	252.64
	ZLIBHIGH	1194.68	1233.32	1187.76	1058.11	810.86
64KB	NONE	260.17	260.17	260.17	260.17	260.17
	RLE	392.43	577.26	524.88	475.94	429.44
	ZLIBFAST	416.68	412.37	397.50	378.19	367.58
	ZLIBHIGH	2590.26	2403.52	1944.41	1683.54	1323.54
100KB	NONE	323.13	323.13	323.13	323.13	323.13
	RLE	483.28	817.84	737.62	659.18	581.15
	ZLIBFAST	553.40	541.18	528.95	498.51	480.96
	ZLIBHIGH	3948.07	3536.07	2795.96	2311.06	1813.80

Table 2: Transaction rate

Message Size	Compress	0%	20%	40%	60%	80%
2KB	NONE	37433	37433	37433	37433	37433
	RLE	33728	34231	35253	35741	36483
	ZLIBFAST	30396	30720	30592	26399	27456
	ZLIBHIGH	13364	11752	10855	10148	9108
8KB	NONE	33276	33276	33276	33276	33276
	RLE	23160	24449	25784	28131	29929
	ZLIBFAST	27067	27689	27866	28557	28854
	ZLIBHIGH	6935	5684	5054	4788	5548
16KB	NONE	29858	29858	29858	29858	29858
	RLE	16297	17633	19377	21975	24532
	ZLIBFAST	23625	24378	25133	25855	26485
	ZLIBHIGH	3720	3091	2845	3284	4139
32KB	NONE	21006	21006	21006	21006	21006
	RLE	11277	10063	11175	12217	14782
	ZLIBFAST	14737	15034	15399	15672	16108
	ZLIBHIGH	1886	1814	1885	2142	2903
64KB	NONE	12738	12738	12738	12738	12738
	RLE	8136	5727	6522	7537	8806
	ZLIBFAST	9710	9527	9941	10582	10973
	ZLIBHIGH	831	905	1138	1330	1740
100KB	NONE	10287	10287	10287	10287	10287
	RLE	6714	3984	4497	5334	6405
	ZLIBFAST	7264	7466	7525	7971	8286
	ZLIBHIGH	543	610	785	965	1259

Table 3: Compression time – using COMPTIME

Message Size	Compress	0%	20%	40%	60%	80%
2KB	RLE	4	4	3	2	1
	ZLIBFAST	7	7	7	7	7
	ZLIBHIGH	48	59	67	75	84
8KB	RLE	16	14	12	9	6
	ZLIBFAST	8	8	8	8	8
	ZLIBHIGH	7	6	6	5	5
16KB	RLE	33	28	23	18	12
	ZLIBFAST	10	10	10	9	9
	ZLIBHIGH	245	295	324	278	216
32KB	RLE	43	28	23	20	12
	ZLIBFAST	10	10	10	9	9
	ZLIBHIGH	439	255	247	222	153
64KB	RLE	61	37	30	24	16
	ZLIBFAST	13	11	10	10	9
	ZLIBHIGH	374	379	272	230	170
100KB	RLE	70	42	35	24	26
	ZLIBFAST	12	12	11	10	10
	ZLIBHIGH	438	412	358	247	192

Table 4: De-compression time – using COMPTIME

Message Size	Compress	0%	20%	40%	60%	80%
2KB	RLE	1	1	1	1	1
	ZLIBFAST	5	4	4	15	13
	ZLIBHIGH	5	5	4	15	8
8KB	RLE	6	6	6	5	4
	ZLIBFAST	7	6	6	5	5
	ZLIBHIGH	7	6	6	5	5
16KB	RLE	13	12	11	10	8
	ZLIBFAST	9	8	7	7	6
	ZLIBHIGH	9	8	7	7	6
32KB	RLE	0	12	11	9	8
	ZLIBFAST	9	8	8	7	6
	ZLIBHIGH	10	8	7	7	6
64KB	RLE	0	16	14	12	10
	ZLIBFAST	11	10	9	8	7
	ZLIBHIGH	11	10	9	8	7
100KB	RLE	0	18	16	14	12
	ZLIBFAST	12	11	9	8	7
	ZLIBHIGH	13	11	37	8	7

Appendix B – Channel Compression over TLS1.2 encrypted channels (ECDHE_RSA_256_CBC_SHA384)

This section provides a summary of the data collected when comparing message compression options over TLS 1.2 encrypted channels where the cipher used was ECDHE_RSA_256_CBC_SHA384.

As we saw with the non-encrypted channels, ZLIBHIGH was prohibitively expensive and provided little benefit for the small increase in compression achieved, so only values for `COMPMSG(NONE|RLE|ZLIBFAST)` are included.

As with Appendix A, there are 4 tables representing the data collected for this configuration.

Table 5: Transaction cost in CPU microseconds

Message Size	Compress	0%	20%	40%	60%	80%
2KB	NONE	162.56	162.56	162.56	162.56	162.56
	RLE	178.35	174.93	165.10	**160.78**	**155.02**
	ZLIBFAST	190.16	186.98	175.00	194.71	184.57
8KB	NONE	228.70	228.70	228.70	228.70	228.70
	RLE	267.24	250.58	231.52	**211.71**	**192.66**
	ZLIBFAST	257.25	244.80	**223.23**	**211.99**	**196.45**
16KB	NONE	376.97	376.97	376.97	376.97	376.97
	RLE	420.18	426.86	377.93	**346.09**	**304.62**
	ZLIBFAST	442.94	414.77	**370.91**	**348.39**	**321.37**
32KB	NONE	602.16	602.16	602.16	602.16	602.16
	RLE	655.85	703.16	624.57	**532.37**	**457.30**
	ZLIBFAST	711.75	652.00	**590.25**	**515.95**	**464.63**
64KB	NONE	1087.03	1087.03	1087.03	1087.03	1087.03
	RLE	1127.21	1249.08	1081.76	**904.42**	**768.41**
	ZLIBFAST	1210.74	1106.25	**954.61**	**874.13**	**755.41**
100KB	NONE	1545.73	1545.73	1545.73	1545.73	1545.73
	RLE	1573.83	1819.16	1570.27	**1302.00**	**1020.30**
	ZLIBFAST	1766.84	1582.82	**1402.95**	**1209.30**	**988.83**

Table 6: Transaction rate

Message Size	Compress	0%	20%	40%	60%	80%
2KB	NONE	9188	9188	9188	9188	9188
	RLE	**13104**	**13681**	**15514**	**16697**	**18188**
	ZLIBFAST	8902	**9310**	**9600**	**9713**	**10670**
8KB	NONE	5025	5025	5025	5025	5025
	RLE	**6786**	**7703**	**9006**	**10578**	**12859**
	ZLIBFAST	**5129**	**5504**	**6405**	**7334**	**9134**
16KB	NONE	3108	3108	3108	3108	3108
	RLE	**3753**	**4123**	**5096**	**5864**	**7361**
	ZLIBFAST	**3112**	**3382**	**4036**	**4526**	**5855**
32KB	NONE	1927	1927	1927	1927	1927
	RLE	**2105**	**2324**	**2774**	**3478**	**4666**
	ZLIBFAST	1862	**2157**	**2518**	**3041**	**3770**
64KB	NONE	1024	1024	1024	1024	1024
	RLE	**1102**	**1252**	**1550**	**2023**	**2669**
	ZLIBFAST	**1129**	**1317**	**1636**	**1976**	**2722**
100KB	NONE	707	707	707	707	707
	RLE	**774**	**849**	**1050**	**1379**	**2044**
	ZLIBFAST	**758**	**909**	**1095**	**1381**	**2054**

Table 7: Compression time – using COMPTIME

Message Size	Compress	0%	20%	40%	60%	80%
2KB	RLE	4	4	3	2	1
	ZLIBFAST	7	7	7	7	7
8KB	RLE	16	14	12	9	6
	ZLIBFAST	8	8	8	8	8
16KB	RLE	19	14	11	8	6
	ZLIBFAST	8	8	8	8	8
32KB	RLE	29	18	15	11	8
	ZLIBFAST	9	9	9	8	8
64KB	RLE	36	22	16	11	9
	ZLIBFAST	9	9	9	8	8
100KB	RLE	39	23	18	12	9
	ZLIBFAST	10	9	9	8	8

Table 8: Decompression time – using COMPTIME

Message Size	Compress	0%	20%	40%	60%	80%
2KB	RLE	2	1	1	1	1
	ZLIBFAST	5	5	4	15	13
8KB	RLE	6	6	6	5	4
	ZLIBFAST	7	6	6	5	5
16KB	RLE	0	6	5	5	4
	ZLIBFAST	6	6	6	5	5
32KB	RLE	0	8	7	6	5
	ZLIBFAST	7	7	6	6	5
64KB	RLE	0	9	8	6	6
	ZLIBFAST	8	7	6	6	7
100KB	RLE	0	10	9	7	6
	ZLIBFAST	8	8	7	6	6

Appendix C – Channel Compression over TLS1.2 encrypted channels (TLS_RSA_WITH_AES_256_CBC_SHA256)

This section provides a summary of the data collected when comparing message compression options over TLS 1.2 encrypted channels where the cipher used was TLS_RSA_WITH_AES_256_CBC_SHA256.

As with cipher ECDHE_RSA_256_CBC_SHA384, due to the prohibitive cost of ZLIBHIGH, we have only included values for `COMPMSG(NONE|RLE|ZLIBFAST)`.

Table 9: Transaction cost in CPU microseconds

Message Size	Compress	0%	20%	40%	60%	80%
2KB	NONE	148.24	148.24	148.24	148.24	148.24
	RLE	163.47	160.69	**148.09**	**144.94**	**138.06**
	ZLIBFAST	176.60	174.95	165.04	186.94	178.31
8KB	NONE	182.61	182.61	182.61	182.61	182.61
	RLE	231.67	220.81	203.43	191.52	**179.77**
	ZLIBFAST	216.24	209.92	195.50	192.77	184.43
16KB	NONE	288.05	288.05	288.05	288.05	288.05
	RLE	341.53	353.59	313.85	303.08	**284.39**
	ZLIBFAST	359.61	343.02	316.31	310.83	295.21
32KB	NONE	425.15	425.15	425.15	425.15	425.15
	RLE	481.38	559.59	517.00	466.00	**419.38**
	ZLIBFAST	542.44	512.81	485.78	444.42	**424.42**
64KB	NONE	723.10	723.10	723.10	723.10	723.10
	RLE	768.16	960.86	867.08	759.21	**691.76**
	ZLIBFAST	915.52	870.95	759.68	**710.19**	**673.67**
100KB	NONE	1003.55	1003.55	1003.55	1003.55	1003.55
	RLE	1039.95	1375.33	1240.64	1069.83	**905.10**
	ZLIBFAST	1290.33	1212.04	1117.46	1025.92	**863.29**

Table 10: Transaction rate

Message Size	Compress	0%	20%	40%	60%	80%
2KB	NONE	11001	11001	11001	11001	11001
	RLE	10706	**11107**	**11309**	**11537**	**11893**
	ZLIBFAST	10556	10848	10974	10718	**11527**
8KB	NONE	7109	7109	7109	7109	7109
	RLE	**7250**	**7655**	**8257**	**9046**	**10781**
	ZLIBFAST	**7268**	**7468**	**8441**	**9090**	**10678**
16KB	NONE	4836	4836	4836	4836	4836
	RLE	**4811**	**4858**	**5712**	**5805**	**7017**
	ZLIBFAST	4707	**5025**	**5691**	**5412**	**6988**
32KB	NONE	3503	3503	3503	3503	3503
	RLE	**4158**	**4189**	**4688**	**5330**	**6279**
	ZLIBFAST	3228	3317	**3833**	**4369**	**4400**
64KB	NONE	2018	2018	2018	2018	2018
	RLE	**2369**	**2397**	**2816**	**3392**	**3724**
	ZLIBFAST	1758	**2102**	**2557**	**2836**	**3134**
100KB	NONE	1438	1438	1438	1438	1438
	RLE	**1747**	**1717**	**1983**	**2409**	**3018**
	ZLIBFAST	1365	**1443**	**1740**	**1966**	**2705**

Table 11: Compression time – using COMPTIME

Message Size	Compress	0%	20%	40%	60%	80%
2KB	RLE	4	4	3	2	1
	ZLIBFAST	7	7	7	7	7
8KB	RLE	16	14	12	9	6
	ZLIBFAST	8	8	8	8	8
16KB	RLE	19	14	11	8	6
	ZLIBFAST	8	8	8	8	7
32KB	RLE	27	18	15	11	8
	ZLIBFAST	9	9	8	8	8
64KB	RLE	36	22	16	12	9
	ZLIBFAST	9	9	9	8	8
100KB	RLE	39	23	18	13	9
	ZLIBFAST	10	9	9	8	8

Table 12: Decompression time – using COMPTIME

Message Size	Compress	0%	20%	40%	60%	80%
2KB	RLE	2	1	1	1	1
	ZLIBFAST	5	5	4	15	13
8KB	RLE	6	6	6	5	4
	ZLIBFAST	7	6	6	5	5
16KB	RLE	0	6	5	5	4
	ZLIBFAST	7	6	6	5	5
32KB	RLE	0	8	7	6	5
	ZLIBFAST	7	7	6	6	5
64KB	RLE	0	9	8	6	6
	ZLIBFAST	8	7	6	7	6
100KB	RLE	0	10	8	7	7
	ZLIBFAST	8	8	7	6	6

Appendix D – Channel Compression over TLS 1.3 encrypted channels (TLS_AES_128_GCM_SHA256)

This section provides a summary of the data collected when comparing message compression options over TLS 1.3 encrypted channels where the cipher used was TLS_AES_128_GCM_SHA256.

Whilst ZLIBHIGH was remains expensive and offers little benefit in terms of additional compression, this section includes the following `COMPMSG` values of NONE, RLE, ZLIBFAST and ZLIBHIGH.

Table 13: Transaction cost in CPU microseconds

Message Size	Compress	0%	20%	40%	60%	80%
2KB	NONE	150.22	150.22	150.22	150.22	150.22
	RLE	162.69	161.95	155.90	154.51	152.17
	ZLIBFAST	173.12	171.80	164.09	186.44	179.35
	ZLIBHIGH	254.87	275.34	289.52	316.11	327.54
8KB	NONE	162.30	162.30	162.30	162.30	162.30
	RLE	206.23	200.33	194.52	186.68	181.01
	ZLIBFAST	195.61	192.52	185.93	184.58	181.29
	ZLIBHIGH	415.13	474.27	511.91	534.36	473.59
16KB	NONE	253.26	253.26	253.26	253.26	253.26
	RLE	295.52	325.99	312.69	298.81	289.37
	ZLIBFAST	326.46	319.40	299.18	301.42	293.62
	ZLIBHIGH	764.89	913.22	963.30	882.48	746.62
32KB	NONE	349.84	349.84	349.84	349.84	349.84
	RLE	413.71	506.00	475.08	447.56	418.64
	ZLIBFAST	479.13	467.38	458.77	430.41	428.56
	ZLIBHIGH	1355.22	1402.91	1372.93	1268.05	1051.58
64KB	NONE	554.08	554.08	554.08	554.08	554.08
	RLE	645.58	862.78	800.02	737.52	685.89
	ZLIBFAST	796.74	774.67	700.43	668.07	681.83
	ZLIBHIGH	2569.49	2454.56	2254.65	2044.26	1738.36
100KB	NONE	754.43	754.43	754.43	754.43	754.43
	RLE	860.73	1225.91	1133.28	1030.69	930.60
	ZLIBFAST	1101.55	1065.49	1009.04	928.45	853.40
	ZLIBHIGH	3771.99	3477.68	3138.71	2825.40	2382.41

Table 14: Transaction rate

Message Size	Compress	0%	20%	40%	60%	80%
2KB	NONE	23498	23498	23498	23498	23498
	RLE	23053	22653	**24777**	**24969**	**25257**
	ZLIBFAST	12397	12987	11617	11704	12305
	ZLIBHIGH	11060	10129	9403	8613	8024
8KB	NONE	21806	21806	21806	21806	21806
	RLE	18187	19026	20029	21015	20552
	ZLIBFAST	11007	11242	11929	12471	12219
	ZLIBHIGH	6288	5276	4669	4458	5178
16KB	NONE	12924	12924	12924	12924	12924
	RLE	11660	11003	11721	12232	11884
	ZLIBFAST	8330	8463	8963	9203	9645
	ZLIBHIGH	3298	2614	2424	2658	3274
32KB	NONE	9220	9220	9220	9220	9220
	RLE	8379	6850	7505	8126	8231
	ZLIBFAST	6082	6594	6937	7434	7449
	ZLIBHIGH	1779	1687	1724	1859	2339
64KB	NONE	5392	5392	5392	5392	5392
	RLE	5449	3896	4303	4813	5006
	ZLIBFAST	4094	4172	4984	4766	4781
	ZLIBHIGH	917	959	1047	1151	1402
100KB	NONE	4084	4084	4084	4084	4084
	RLE	4022	2704	2970	3379	3891
	ZLIBFAST	3015	3045	3253	3377	4126
	ZLIBHIGH	606	661	740	832	1014

Table 15: Compression time – using COMPTIME

Message Size	Compress	0%	20%	40%	60%	80%
2KB	RLE	4	4	3	2	1
	ZLIBFAST	7	7	7	7	7
	ZLIBHIGH	49	59	70	72	86
8KB	RLE	16	14	12	9	6
	ZLIBFAST	8	8	8	8	7
	ZLIBHIGH	7	6	6	5	5
16KB	RLE	19	14	11	8	6
	ZLIBFAST	8	8	8	8	8
	ZLIBHIGH	118	158	196	155	123
32KB	RLE	28	18	15	11	8
	ZLIBFAST	9	9	9	8	8
	ZLIBHIGH	156	165	165	153	114
64KB	RLE	36	22	16	11	10
	ZLIBFAST	9	9	9	8	8
	ZLIBHIGH	189	184	170	157	118
100KB	RLE	39	23	20	13	11
	ZLIBFAST	10	10	9	8	8
	ZLIBHIGH	222	227	187	157	114

Table 16: De-compression time – using COMPTIME

Message Size	Compress	0%	20%	40%	60%	80%
2KB	RLE	2	1	1	1	1
	ZLIBFAST	5	5	4	15	13
	ZLIBHIGH	5	5	4	14	8
8KB	RLE	6	6	6	4	4
	ZLIBFAST	7	6	6	5	5
	ZLIBHIGH	7	6	6	5	5
16KB	RLE	0	6	5	5	4
	ZLIBFAST	6	6	6	5	5
	ZLIBHIGH	7	6	6	6	5
32KB	RLE	0	8	7	6	5
	ZLIBFAST	7	7	6	6	5
	ZLIBHIGH	7	7	6	6	5
64KB	RLE	0	9	8	6	6
	ZLIBFAST	8	7	6	6	5
	ZLIBHIGH	8	8	6	6	5
100KB	RLE	0	10	8	7	6
	ZLIBFAST	8	8	7	6	6
	ZLIBHIGH	8	8	7	6	6

Appendix E – Channel Compression over TLS 1.3 encrypted channels (TLS_CHACHA20_POLY1305_SHA256)

This section provides a summary of the data collected when comparing message compression options over TLS 1.3 encrypted channels where the cipher used was TLS_CHACHA20_POLY1305_SHA256.

Table 17: Transaction cost in CPU microseconds

Message Size	Compress	0%	20%	40%	60%	80%
2KB	NONE	334.53	334.53	334.53	334.53	334.53
	RLE	344.07	**324.69**	**295.95**	**270.59**	**244.07**
	ZLIBFAST	358.31	338.79	**317.56**	**313.98**	**285.45**
	ZLIBHIGH	438.98	438.62	432.04	441.99	428.24
8KB	NONE	738.25	738.25	738.25	738.25	738.25
	RLE	775.22	**674.21**	**566.90**	**455.44**	**348.53**
	ZLIBFAST	741.58	**650.51**	**553.29**	**454.05**	**356.47**
	ZLIBHIGH	957.97	928.27	877.85	802.61	646.80
16KB	NONE	1380.22	1380.22	1380.22	1380.22	1380.22
	RLE	1421.66	**1251.31**	**1033.17**	**813.29**	**602.01**
	ZLIBFAST	1392.35	**1206.96**	**1006.49**	**807.35**	**613.18**
	ZLIBHIGH	1833.52	1800.94	1662.13	1393.81	1066.63
32KB	NONE	2547.71	2547.71	2547.71	2547.71	2547.71
	RLE	2605.90	**2308.82**	**1855.91**	**1421.76**	**981.78**
	ZLIBFAST	2571.31	**2192.09**	**1788.46**	**1380.56**	**984.82**
	ZLIBHIGH	3445.08	3126.46	2701.60	**2219.32**	**1610.79**
64KB	NONE	4909.42	4909.42	4909.42	4909.42	4909.42
	RLE	4965.35	**4411.42**	**3517.00**	**2593.33**	**1748.37**
	ZLIBFAST	**4902.05**	**4144.61**	**3312.46**	**2503.40**	**1716.72**
	ZLIBHIGH	6674.91	5818.71	4853.57	**3879.92**	**2770.51**
100KB	NONE	7359.07	7359.07	7359.07	7359.07	7359.07
	RLE	7420.09	**6608.97**	**5249.72**	**3883.16**	**2527.67**
	ZLIBFAST	**7341.04**	**6170.27**	**4930.07**	**3681.28**	**2451.90**
	ZLIBHIGH	10086.96	8643.16	**7101.98**	**5584.07**	**3922.67**

Table 18: Transaction rate

Message Size	Compress	0%	20%	40%	60%	80%
2KB	NONE	11283	11283	11283	11283	11283
	RLE	11068	**11684**	**12863**	**13921**	**15452**
	ZLIBFAST	10606	11219	**11950**	**11604**	**12782**
	ZLIBHIGH	7348	7097	7000	6961	6710
8KB	NONE	2852	2852	2852	2852	2852
	RLE	**4923**	**5703**	**6784**	**5002**	**10977**
	ZLIBFAST	**5149**	**5923**	**6972**	**8479**	**10758**
	ZLIBHIGH	**3332**	**3256**	**3265**	**3415**	**4175**
16KB	NONE	2724	2724	2724	2724	2724
	RLE	2662	**3053**	**3671**	**4636**	**6025**
	ZLIBFAST	**2730**	**3164**	**3772**	**4649**	**5979**
	ZLIBHIGH	1717	1635	1685	1968	2541
32KB	NONE	1482	1482	1482	1482	1482
	RLE	**1469**	**1650**	**2057**	**2660**	**3668**
	ZLIBFAST	1472	**1721**	**2116**	**2718**	**3705**
	ZLIBHIGH	905	962	1082	1273	**1720**
64KB	NONE	774	774	774	774	774
	RLE	**779**	**866**	**1083**	**829**	**2101**
	ZLIBFAST	**782**	**916**	**1155**	**1493**	**2121**
	ZLIBHIGH	465	522	613	744	**1008**
100KB	NONE	517	517	517	517	517
	RLE	**522**	**577**	**724**	**973**	**1483**
	ZLIBFAST	**523**	**618**	**769**	**1021**	**1534**
	ZLIBHIGH	306	353	422	**521**	**715**

Table 19: Compression time – using COMPTIME

Message Size	Compress	0%	20%	40%	60%	80%
2KB	RLE	4	4	3	2	1
	ZLIBFAST	7	7	7	7	7
	ZLIBHIGH	48	58	67	72	86
8KB	RLE	16	14	12	9	6
	ZLIBFAST	8	8	8	8	7
	ZLIBHIGH	7	6	6	5	5
16KB	RLE	19	14	11	8	6
	ZLIBFAST	8	8	8	8	7
	ZLIBHIGH	120	158	175	155	121
32KB	RLE	27	18	15	11	8
	ZLIBFAST	9	9	9	8	8
	ZLIBHIGH	156	168	166	154	115
64KB	RLE	36	23	16	11	9
	ZLIBFAST	9	9	9	8	8
	ZLIBHIGH	189	183	166	147	118
100KB	RLE	39	23	17	12	9
	ZLIBFAST	10	9	9	8	8
	ZLIBHIGH	213	190	171	157	113

Table 20: De-compression time – using COMPTIME

Message Size	Compress	0%	20%	40%	60%	80%
2KB	RLE	2	1	1	1	1
	ZLIBFAST	5	4	4	15	13
	ZLIBHIGH	5	5	4	14	8
8KB	RLE	6	6	6	5	4
	ZLIBFAST	7	6	6	5	5
	ZLIBHIGH	7	6	6	5	5
16KB	RLE	0	6	5	5	4
	ZLIBFAST	7	6	7	5	5
	ZLIBHIGH	6	6	6	5	5
32KB	RLE	0	8	7	6	5
	ZLIBFAST	7	7	6	6	5
	ZLIBHIGH	7	7	6	6	5
64KB	RLE	0	9	8	6	6
	ZLIBFAST	8	7	6	6	6
	ZLIBHIGH	8	7	6	6	5
100KB	RLE	0	10	8	7	6
	ZLIBFAST	8	8	7	6	6
	ZLIBHIGH	8	8	7	6	6

Appendix F – Useful Links

The use of environment variable "`_HZC_COMPRESSION_METHOD`" is discussed in:
https://www.ibm.com/docs/en/zos/2.5.0?topic=compression-running-zlib

SMF Records:
Type 113 records – hardware capacity, reporting and statistics.
Type 30 zEDC usage records - "zEDC usage statistics section".

MQ supportPac MP1B "Interpreting accounting and statistics data".

MQ Performance reports:
Landing page - https://ibm-messaging.github.io/mqperf/
General MQ for z/OS performance - MP16 "Capacity Planning and Tuning guide"
MQ for z/OS 9.2 - https://ibm-messaging.github.io/mqperf/MQ_for_zOS_V920_Performance.pdf
MQ for z/OS 9.3 - https://ibm-messaging.github.io/mqperf/MQ%20for%20zOS%209.3%20Performance.pdf

Reduce storage occupancy using IBM zEDC compression links:
Blog - https://community.ibm.com/community/user/integration/viewdocument/reducing-storage-occupancy-with-ibm
Redbook - https://www.redbooks.ibm.com/redbooks/pdfs/sg248259.pdf

Hosting the fasp.io gateway on zCX to assist performance MQ channel performance over high latency networks is discussed in "MQ with zCX".

Appendix E – Test Environment

Measurements were performed using:

The IBM MQ performance sysplex ran measurements on:

- **IBM z16 (3931-7x1) – 4 CPC drawers**

The sysplex was configured thus:

- LPAR 1:
  - 1-32 dedicated CP plus 2 zIIP with 144 GB of real storage.
- LPAR 2:
  - 1-10 dedicated CP plus 2 zIIP with 48 GB of real storage.
- LPAR 3:
  - 1-3 dedicated CP with 48 GB of real storage.
- z/OS v2r5.
- Db2 for z/OS version 12 configured for MQ using Universal Table spaces.
- IMS 15.3
- IBM CICS CTS 6.2
- MQ queue managers:
  - configured at MQ 9.3.
  - configured with dual logs and dual archives.

Coupling Facility:

- Internal Coupling Facility with 4 dedicated processors
- Coupling Facility running latest CFCC level.
- Dynamic CF dispatching off
- 3 x ICP links between z/OS LPARs and CF.

DASD:

- FICON Express 16S connected DS8950F
- 4 dedicated channel paths
- HYPERPAV enabled
- zHPF disabled unless otherwise specified.

Network:

- 10GbE network configured with minimal hops to distributed partner machines
- 1GbE network available

Applications written in a mixture of:

- C
- COBOL compiled with Enterprise COBOL for z/OS 6.3 with options ARCH(13) and OPT(1).