IBM MQ for z/OS - Managed File Transfer Performance

Version 1.02 – March 2021

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 - Managed File Transfer*. The information is not intended as the specification of any programming interface that is provided by 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 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.

Summary of Amendments

Date	Version	Changes
2021-February	1.0	Initial document
2021-February	1.01	Minor changes to report
2021-March	1.02	• Clarification to performance to transfer of text files to xLinux. • Future tuning of MFT agent • Message to file transfer performance

Preface

In this paper, I will be looking at the current performance of IBM MQ for z/OS with the Managed File Transfer (MFT) feature enabled.

This paper is intended to replace the original performance evaluation provided in performance report FP11.

This paper is split into several parts:

Part one - Overview of approach to testing.
Part two - Performance highlights.
Part three - Performance "out of the box".
Part four - Alternative MQ configurations.
Part five - Network implications.
Part six - Tuning your z/OS system.
Part seven - MFTs use of MQ resources.
Part eight - Guidance when using MFT on z/OS.
Part nine - Guidance for monitoring your MQ queue manager.

Part one describes the basic configuration used and how costs have been determined.

Part two presents the performance highlights, offering indication of the performance obtained in our performance test environment.

Part three discusses the performance observed when running MFT with the agents and queue managers with no specific tuning applied.

Part four looks at alternative configurations, such as using shared queue for the agents, TLS channel protection and AMS policy protection on the MFT queues.

Part five discusses the impact of a network with more capacity as well as the implications of latency (distance) on the file transfer. The section also discusses how this impact can be mitigated using the fasp.io gateway that is available with IBM MQ Advanced for z/OS VUE.

Part six looks at some of the z/OS system concepts that can affect the performance of the file transfer.

Parts seven and eight offer explanation of the queue usage and good practice when using MFT on z/OS.

Part nine describes those areas of the queue manager that are important for good performance with regards to file transfer workloads.

Table of Contents

1. Overview

The Managed File Transfer (MFT) feature, is a file transfer product that uses IBM MQ as its transport layer.

The MFT product is written in Java™ and as such is largely eligible for offload to zIIP, which can significantly reduce the cost of the file transfers, whether sending to or from z/OS.

This report is based on measurements taken from the IBM MQ Performance sysplex that consists of 3 logical partitions, plus an internal Coupling Facility on an IBM z15 running z/OS v2r4.

The costs reported are gathered from the Resource Management Facility (RMF) reports and are based on the total cost across the Sysplex.

These measurements were run in a controlled environment, where only required subsystems were active during the measurements.

Typically the performance measurements run on the z/OS systems aim to ensure that there is no more than a 6% variability between runs of any configuration. With the relatively light system load that a set of MFT transfers has on our system, we have found that there is considerably more variability between MFT runs.

The measurements for the performance headlines are based on the time taken to transfer a set of files and the associated CPU cost on z/OS.

The measurements use 4 different file sizes:

- 1MB
- 10MB
- 100MB
- 1GB

Each transfer measurement moves between 1 and 10GB of files from the source to the destination system. For example, if a test is using 1MB files, then it will transfer a minimum of 1024 files in a single measurement.

This document provides information as to the achieved transfer rate and the cost of transferring the data on our systems for a number of configurations. Unlike performance report FP11, this report will not review the tuning options, instead relying on the default configurations.

There are multiple factors that affect the transfer rates achieved in this document, including network capacity and disk response times.

- The default network used is rated at 1Gb, so we cannot expect transfer rates to significantly exceed 100MB/second.
- Disk response times are a significant factor in the achieved transfer rate – the disk subsystems that the z15 are now reaching end of service and are not indicative of the latest performance capabilities. The xLinux partner machine has an 8Gb SAN card connected to IBM SAN Volume controller plus SSD storage array, but is also relatively old, and better performance may be achieved with more modern CPU.

Since the original performance report was published, we found that a number of configuration changes to our run time environment could offer some benefit to the performance of file transfers both in and out of z/OS.

These included:

- Large page memory allocation – in particular we set:
  - `-Xlp:objectheap:pagesize=2g,warn,nonpageable`
  - `-Xlp:codecache:pagesize=1m,pageable`
- Garbage collection using the pause less option e.g.
  - `-Xgc:concurrentScavenge`
- Disabling IBM Health Center data gathering in "headless" mode, from both the agent and file transfer requests.

Subsequent releases of the performance report will apply these options as the default configuration, but compared to our measurements to date, we saw improvements to the performance of up to 30% as a result of applying all 3 changes – both in terms of a reduction on transfer cost and an improvement to throughput rate.

Additionally we found that the distributed partner performance was affected by insufficient warm-up time, which resulted in highly variable results. For some reason the distributed MFT agent required a larger volume of work to sufficiently build an optimized level of code. As a bypass to this we have now configured the distributed MFT agent with:

- `-Xjit:optLevel=scorching`

This was partly brought on by our testing process which does not keep the MFT agents running for extended periods of time. As with the z/OW tuning options, subsequent releases of this performance report will use the `-Xjit:optLevel=[]` option for all file transfers involving a distributed MFT agent.

2. Performance highlights

In this latest version of the MFT for z/OS performance report, where the measurements have been run on an IBM z15, we have seen a number of areas which are worth highlighting:

When sufficient CPU, disk subsystem and network is available, the file transfer rates achieved can exceed:

- 200MB/second when using z/OS datasets
- 250MB/second when using Unix System Service files.

Performance of high latency file transfers can be assisted with the use of the fasp.io gateway, improving throughput by up to 22 times that of equivalent transfers using TCP/IP.

MQ channel compression, supported by on-chip compression on IBM z15, can improve throughput by up to 50%.

Protecting the file transfer payload in MQ using AMS qualities of protection has minimal impact on throughput, provided there is sufficient CPU and cryptographic resource.

3. How does MFT perform “out of the box”?

To demonstrate the performance characteristics of MFT on z/OS, the initial measurements are run using default options.

There are 3 basic configurations used:

- xLinux to z/OS
- z/OS to xLinux
- z/OS to z/OS

Additionally the z/OS end of the transfer is configured to read / write from z/OS data sets and Unix System Services files.

4 sizes of files are transferred – 1MB, 10MB, 100MB and 1GB.

In each measurement, a minimum of 1GB’s worth of data will be transferred. For files larger than 1MB, the measurement involves transferring between 4 and 10GB of data.

For transfers involving different platforms, both binary and text transfers are measured. The cost of the text transfer includes translating the data into the appropriate code page for the receiving machine.

For transfers run z/OS to z/OS, whether using z/OS datasets or Unix System Service (USS) files, the measurements using a binary transfer only, as no data conversion is required.

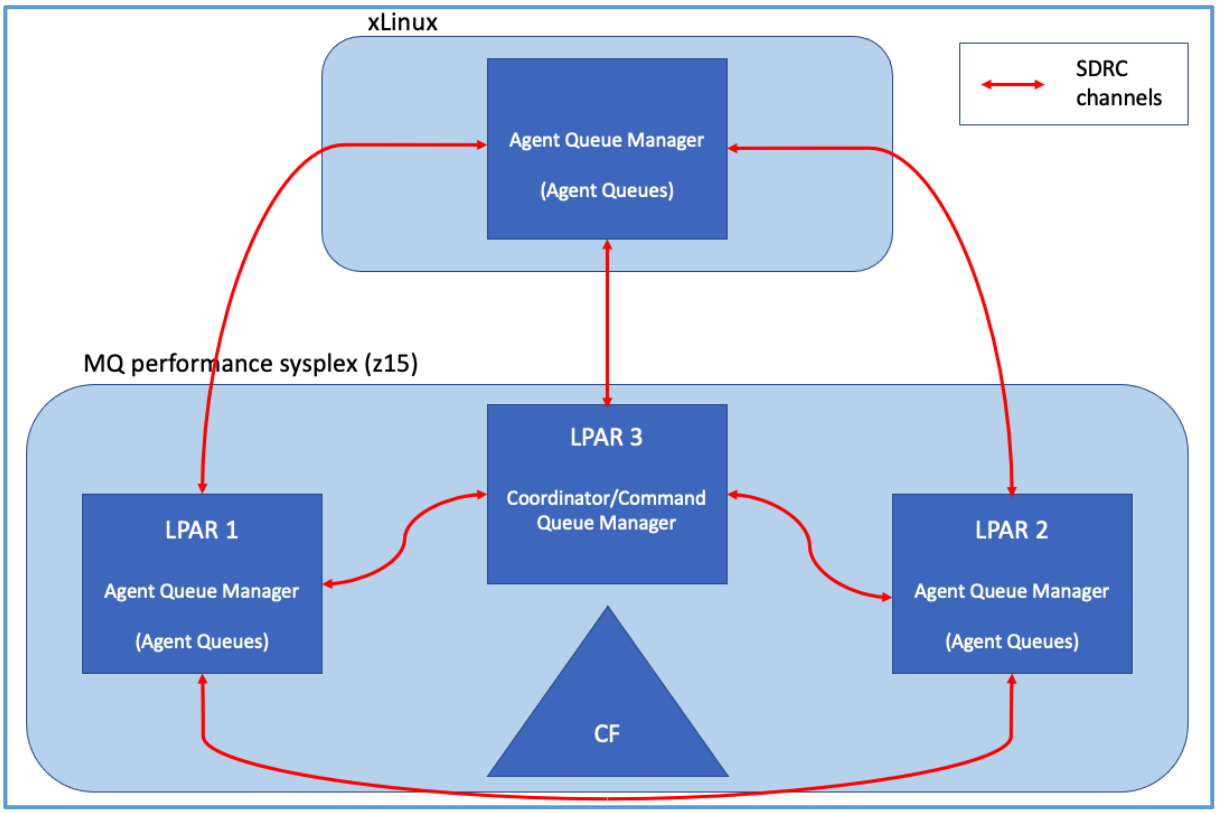
The costs reported are the costs across the Sysplex as determined from the RMF reports.

The initial configuration uses local queues with queue manager to queue manager communication using server/receiver channel pairs.

All file transfer for the initial measurements will be run using the 1Gb performance network.

Basic test topology

The following topology is used for these initial measurements:



The topology shows that each agent queue manager has locally defined MFT queues and are connected to all other queue managers in the network via sender-receiver channels.

LPARs 1 and 3 are on different subnets of the same physical z15, which has been configured such that traffic is routed outside of the z15 such that communication is over the physical network infrastructure.

Inbound to z/OS datasets

Chart 1: Achieved transfer rate between xLinux and z/OS datasets

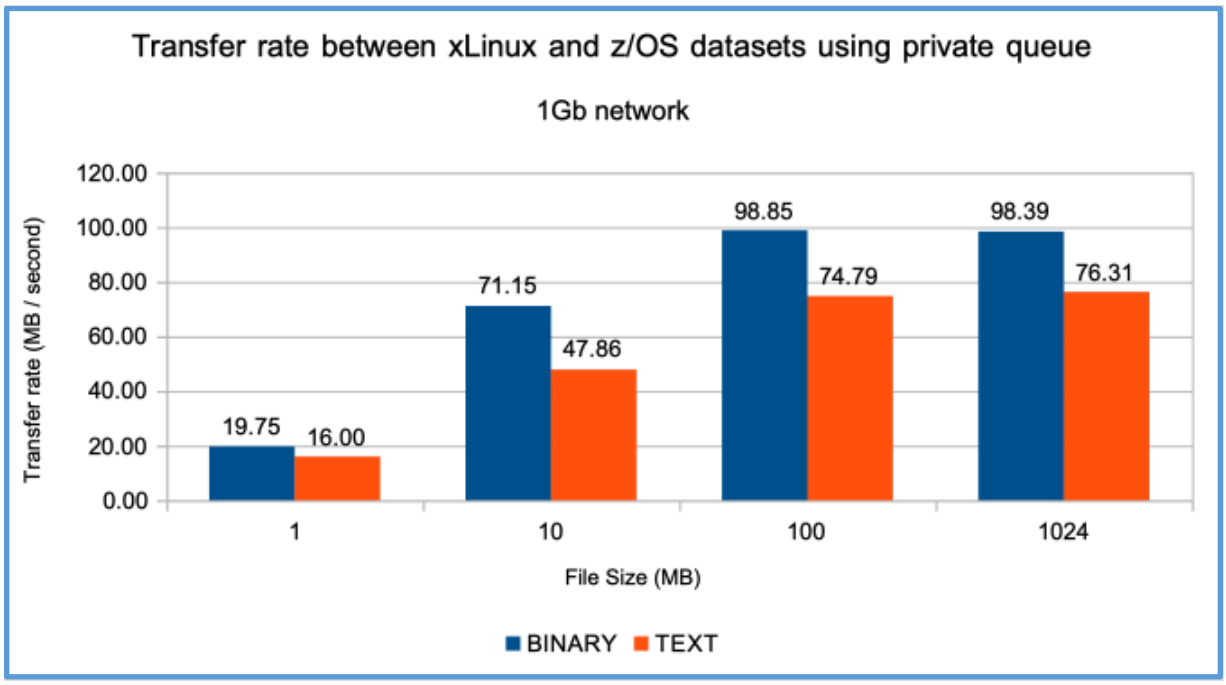
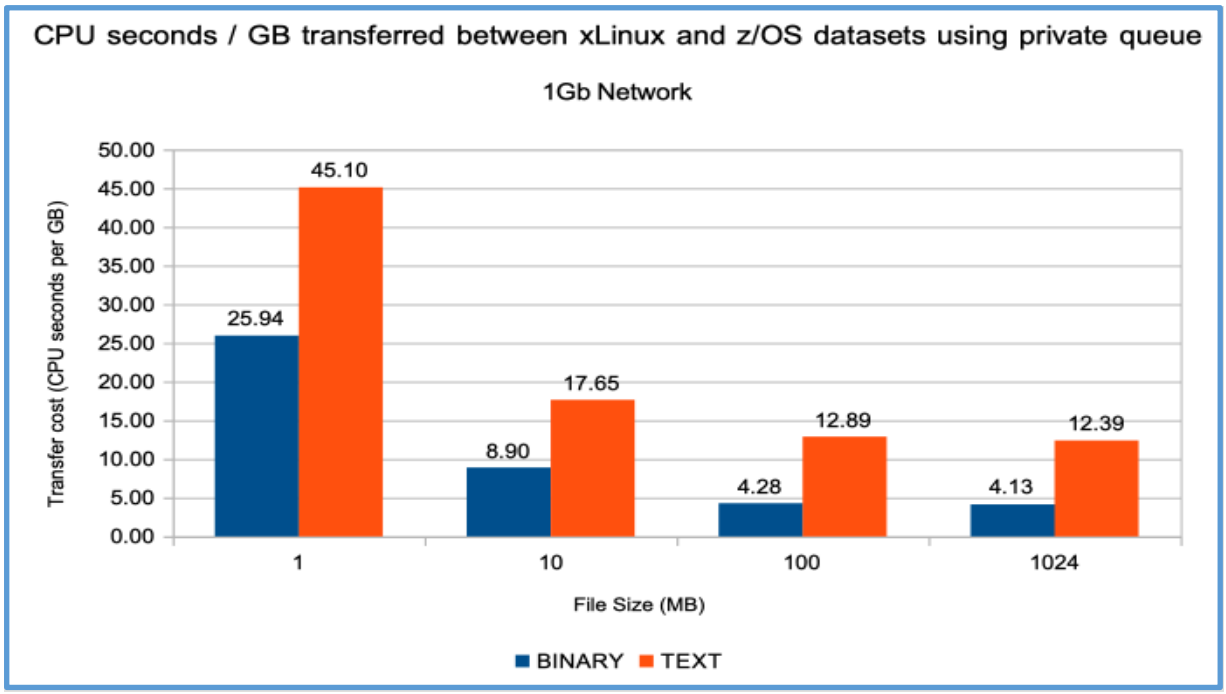


Chart 2: Cost per GB of data transferred between xLinux and z/OS datasets



The difference in the performance of the binary and text transfers, results from the z/OS agent receiving the file transfer having to perform data conversion in the text transfer, converting ASCII to EBCDIC, which results in increased cost in the Agent task and reduced transfer rate.

Inbound to USS files

Chart 3: Achieved transfer rate between xLinux and Unix System Services files

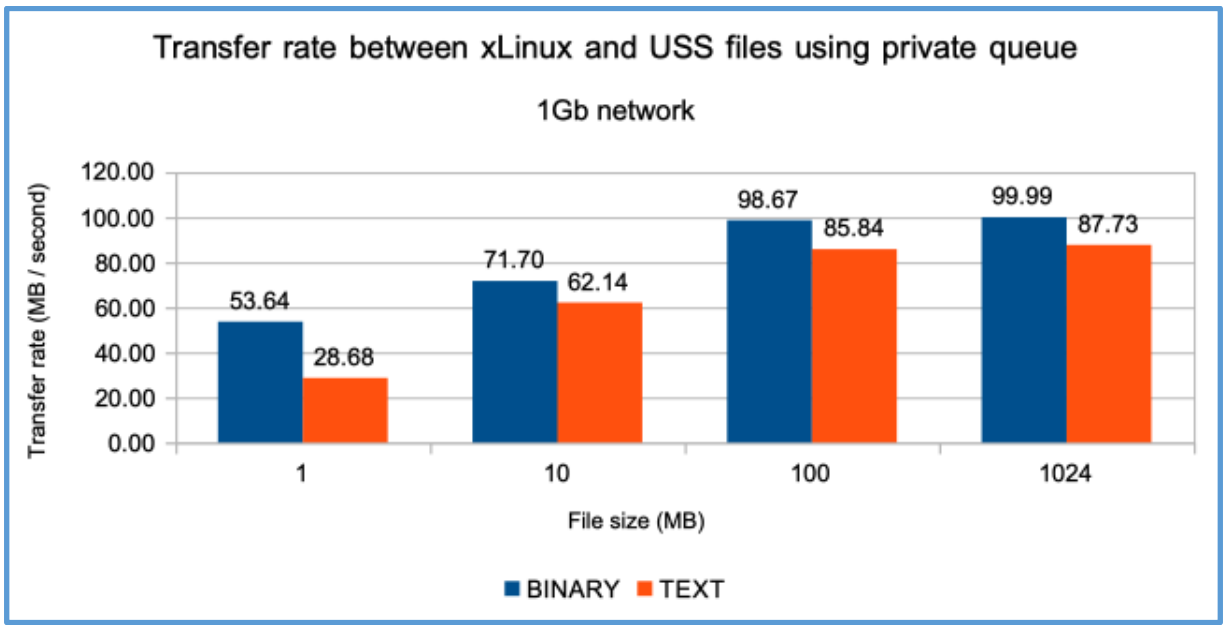
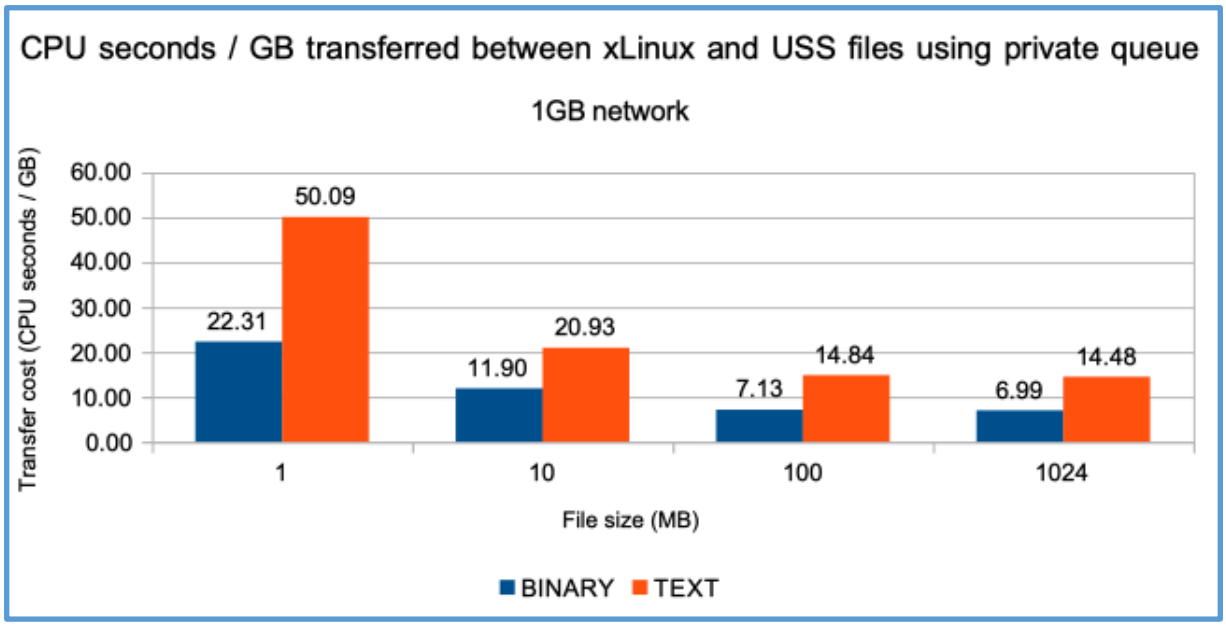


Chart 4: Cost per GB of data transferred between xLinux and z/OS datasets



As with files written to datasets, files written to Unix System Services directories, show an increased cost for text files due to performing the data conversion on z/OS.

Outbound from z/OS datasets

Chart 5: Achieved transfer rate between z/OS datasets and xLinux

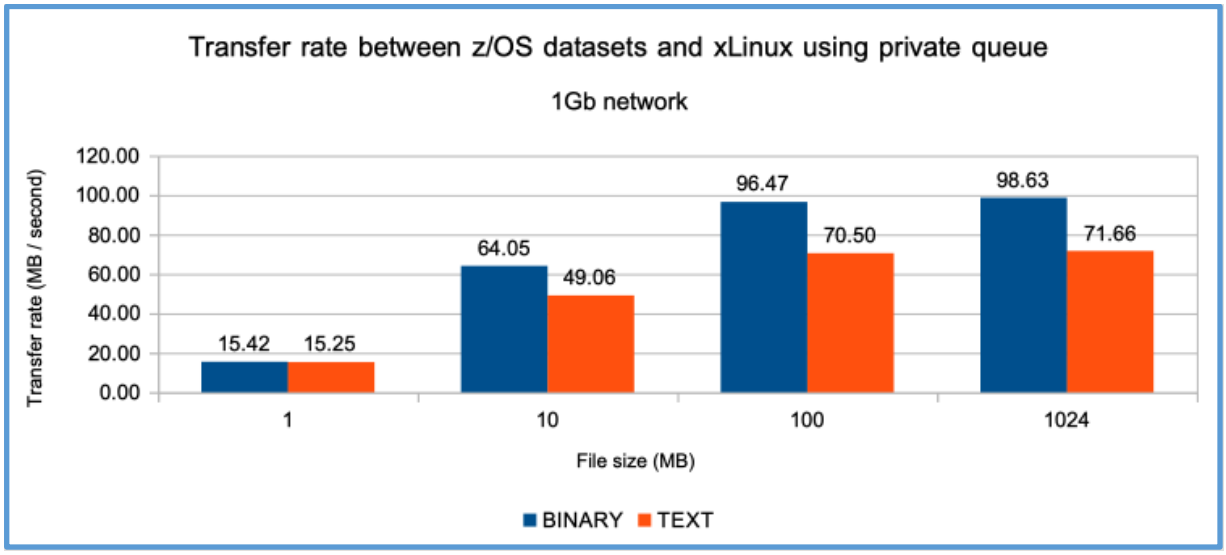
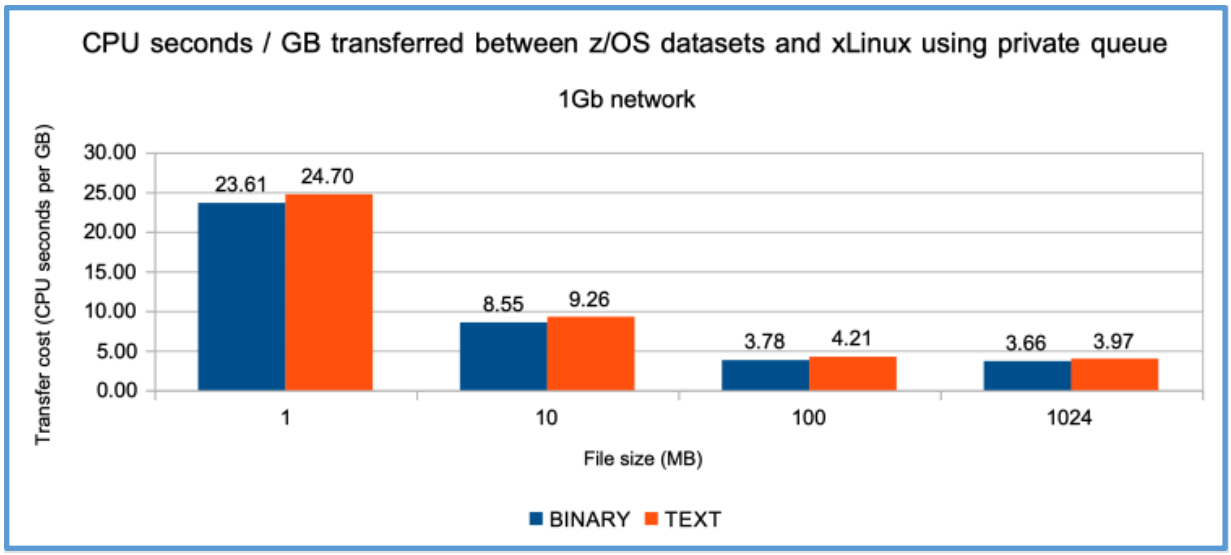


Chart 6: Cost per GB of data transferred between z/OS datasets and xLinux



The cost of transferring text files appears slightly higher than the equivalent transfer of binary files, which can be explained due to the approach used to calculate the costs.

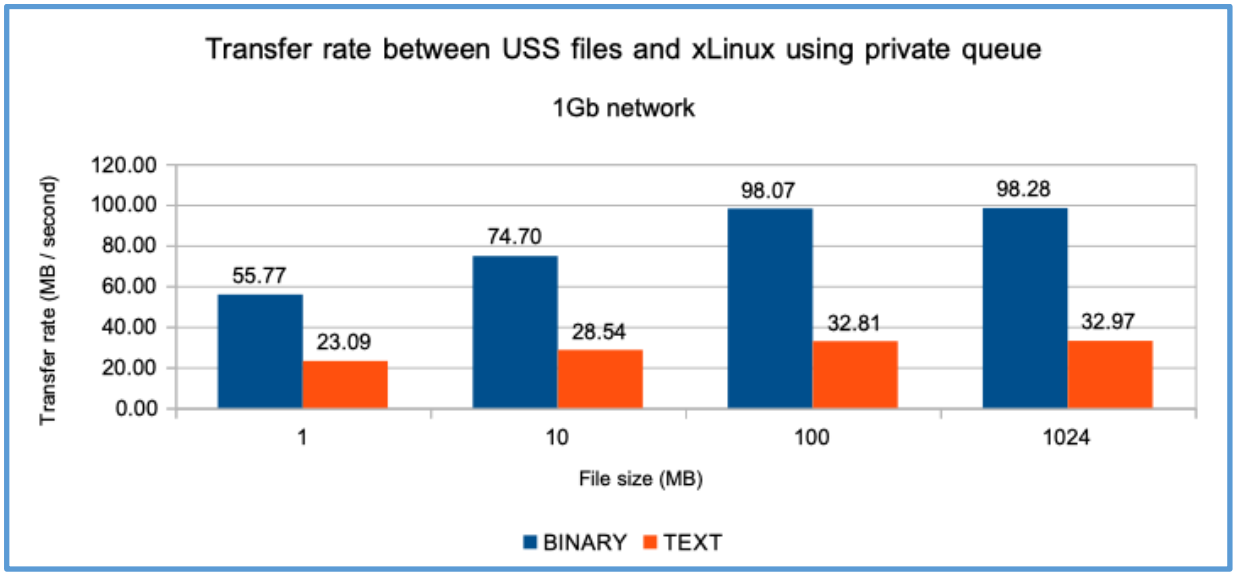
The z/OS LPAR has an inherent cost of being online which we will call “system overheads”, and might equate to 1-2% CPU utilisation even in our performance environment. Since the cost per GB is being calculated based on the **total system** cost, the “system overhead” cost is included in these calculations.

The binary file transfer for files of 10MB or larger completes 30% faster than the text file transfer, and so results in less “system overhead” cost, and it is this difference that results in the outbound text file transfer appearing to cost more per GB of data transferred.

The text file transfer takes longer primarily due to the receiving Agent having to perform data conversion, which is CPU intensive.

Outbound from USS files

Chart 7: Achieved transfer rate between USS files and xLinux



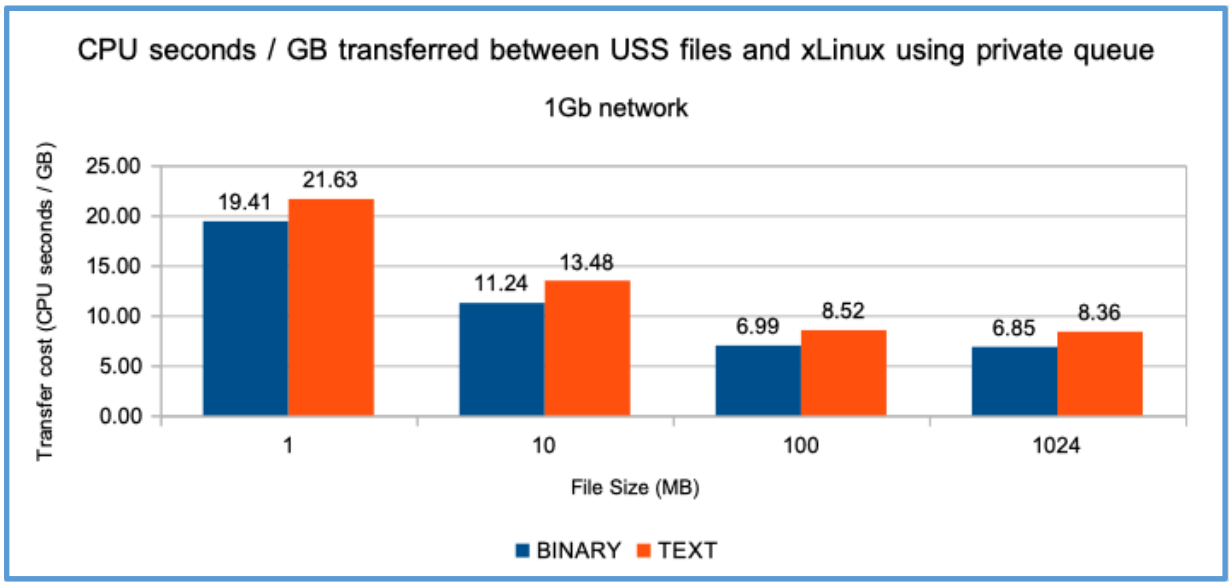
The performance of the text transfers, particularly for files of 100MB and 1GB is surprisingly low, and is the primary reason for revisiting in this version of the performance report.

For example:

- z/OS datasets containing the same 1Gb data, sent to the same partner configuration achieved a transfer rate of 71.66 MB/second.
- Switching the destination from the SAN to the internal HDD results in the 1GB text transfer achieving a rate of 41MB/second, an increase of 9MB/second.

This is discussed further in the section "Trying to understand the variable performance", which follows the chart demonstrating the file transfer costs for this configuration.

Chart 8: Cost per GB of data transferred between USS files and xLinux



Trying to understand the variable performance

Given the files being transferred from z/OS and USS are essentially the same, including code page, record length, and the resulting data conversion required on the remote system, we might expect similar performance from the two configurations, and yet as reported earlier, the z/OS file transfer is *sometimes* significantly out-performing the USS transfer.

So the questions are, what is causing the difference and can it be prevented?

In order to try to answer both of these questions, we went through the following question and answer process:

- Is the warm-up phase and workload similar in each case?

Yes. In order to try to ensure the agents are suitably warmed-up, the MFT agents are given at least 4 individual transfers each consisting of transferring 1024 files of 1MB in the same direction as the measured workload. The file transfers are then run in the same order – binary transfers for each of the specified file sizes, followed by text transfer.

- Does the variability remain when the network is not a constraining factor?

Yes. Sending the file transfers over the 10Gb network did not reduce the variability of the workloads.

- Is the source system the cause of the variability?

In all cases, there was sufficient CPU and network available, and the network response times and MQ channel batch sizes were comparable in both the high and low transfer rate measurements.

This left the DASD response times, to read the data from file. RMF 42 subtype(6) records suggested that the performance was consistent.
Despite this consistent I/O performance, we decided to try to remove as much potential for variability as possible, and investigated message-to-file transfers – but this did not resolve the variability issues.

- Does the destination file system affect the performance?

Following guidance in the "SAN tuning" section of the "Persistent Messaging Performance" document, we looked at the impact of write merges. With minimal write merges occurring on the writes to the SAN, disabling the feature had little impact.

In an attempt to rule out the impact of the file system on the xLinux partner, we configured a RAM disk – which allowed us to prove that the variability was not as a result of file system variabilities.

- Is the destination agent processing causing the variability?

When running the file transfers, we monitored the destination system using the following command to poll the system: `"top -S -u mqperf -d 5 -n 24 -b"`

  - Where `mqperf` is the user id running the MFT agent.

This allowed us to monitor the CPU utilisation of the Java™ process performing the data conversion.

For each interval during the text-type transfer, the Java™ process was using 100-108% of a CPU, for example:

```
   PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
404574 mqperf    20   0 5686512 686052  16404 S 105.6  2.1  10:17.27 java
```

Given the xLinux partner has 12 cores and is dedicated for this workload, there is CPU capacity to spare, should the workload require it.

This might suggest that whilst the xLinux partner has capacity for more work, the particular workload may be CPU bound on a single thread.

- What is the java workload doing on the xLinux partner?

For this we looked at several options – enabling trace as well as sampling using the IBM Health Center.

Enabling trace by specifying `trace=com.ibm.wmqfte=all` in the `agent.properties` file, resulted in the performance degrading to such an extent as to be of little benefit. Targeted trace would have had less impact on the file transfer but at this stage it was difficult to determine which classes should be traced.

Enabling the IBM Health Center sampling in "headless" mode as part of the agent process created the HCD files for use in the Health Center client.

To enable sampling we configured:

```
export BFG_JVM_PROPERTIES="-Xmx1024M -Xhealthcenter:level=headless
-Dcom.ibm.java.diagnostics.healthcenter.headless.files.to.keep=0"
```

Enabling Health Center requires the MFT agent process to be restarted, with the HCD files being written once the agent was stopped. These files could be found in the agent log directory.

Viewing the data in the Health Center client indicated that 80% of the CPU used was in the data conversion routines, whether converting the source data to unicode byte-by-byte, or subsequently converting each byte to a character.

From a processing perspective, the time spent in data conversion and the methods to perform the data conversion does make sense, and also further suggests that the process is somewhat constrained by the clock speed on the xLinux partner.

Up to now, we have been able to demonstrate that the transfer is limited by CPU, rather than network or file system performance, but this does not explain why we can see such variability in transferring the text files.

Is it Java™?

If the source and destination systems are not changing between high and low achieved throughput file transfers, and the file contents are constant, then surely the variability must be due to Java differences?

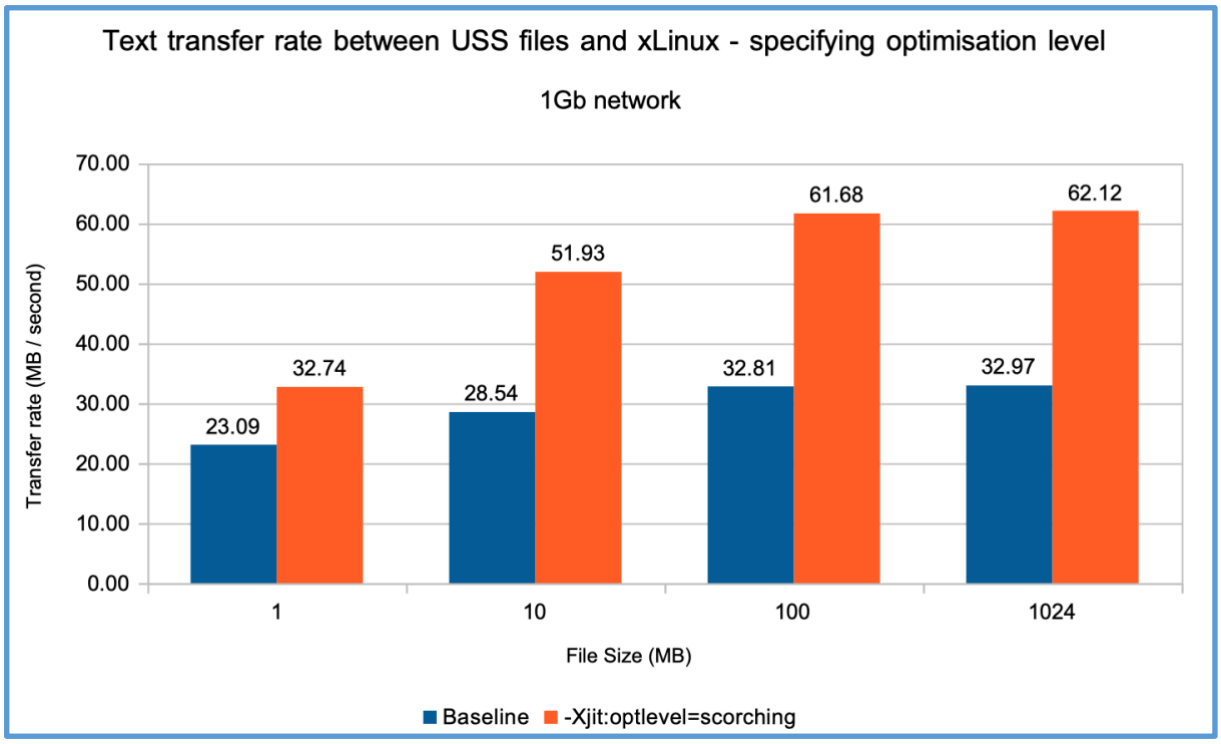
To see whether the JIT would affect the performance, negatively or otherwise, we looked at the JIT options.

- Disable JIT using **`-Xnojit`**
  Unsurprisingly this option did not result in well performing code.
- Configure the JIT to compile the method once it has been called a fixed number of times using **`-Xjit:count=10`**
  This option did result in more consistent performance but at the lower end of the throughput range.
- Force the JIT compiler to compile all methods at a specific optimization level, for example **`-Xjit:optlevel=scorching`**

  We selected option "scorching", with the intention to give Java a nudge that performance was critical, which resulted in the 1GB text transfer achieving 62 MB/second and perhaps more importantly from a performance perspective, this configuration achieved consistent and repeatable results.

  The "veryHot" option also resulted in consistent performance results but peaked at 58 MB/second – however this is within acceptable boundaries for test variability for these workloads.

  As the following chart indicates, by specifying the "scorching" level of optimisation on the xLinux MFT agent we were able to nearly double the throughput rates for files of 10MB and larger in a consistent manner.



With regards to the cost of transferring the data, by specifying the JIT optimization level on the partner, the z/OS costs were up to 20% lower than in the baseline, primarily because we are recording less "z/OS overheads" due to the reduced run time.

Of course, by specifying the JIT optimization level on the xLinux partner, we are increasing slightly increasing the load on that partner machine:

Early stages of the file transfer showed the usage, where the JIT was aggressively optimizing the code, the 'top' output shows:

```
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 632991 mqperf    20   0 5697764 639272  16356 S 231.1  2.0   7:18.24 java
```

Once the JIT process was complete, the output from the 'top' command showed:

```
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 632991 mqperf    20   0 5683432 686644  16392 S 111.8  2.1  12:28.78 java
```

Does this apply to the MFT agent running on z/OS?

When we apply the same level of JIT optimization to the z/OS agent, we do not see a benefit in terms of throughput but do see an increase in the transfer costs and as such would not use this option with the MFT performance tests.

Should I override the JIT optimization level?

As the Java documentation for -Xjit:optlevel=[] suggests, this may have an unexpected effect on performance, including reduced overall performance, so it is advisable to test this in your own environment before implementing in your production environment.

z/OS to z/OS transfers

The following 2 charts compare transfer requests for z/OS to z/OS dataset transfers with USS file to USS file transfers.

Chart 9: Achieved transfer rate between z/OS LPARs

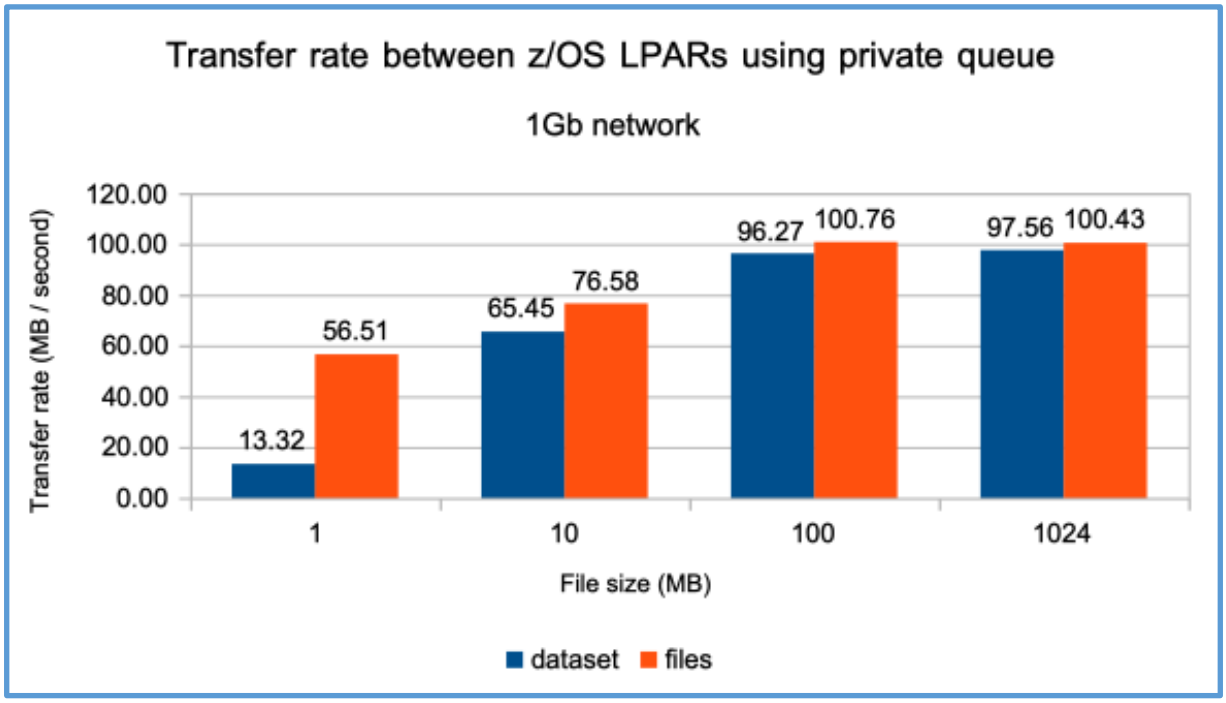
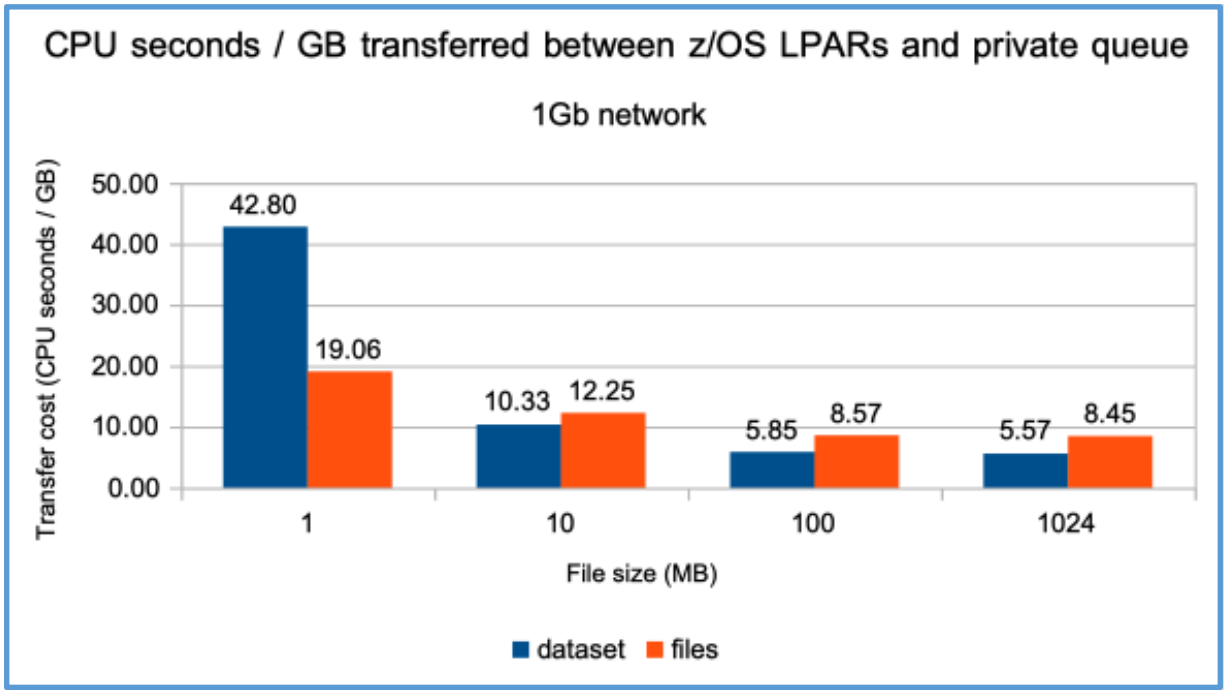


Chart 10: Cost per GB of data transferred between z/OS LPARs



The performance of file transfers whether using z/OS datasets or USS files is similar where the file size is 100MB or larger, however the transfer rate of small USS files is significantly higher than small z/OS datasets, with some of the additional cost a result of the increased transfer time and increased impact on system overheads from that longer time.

Additionally, when transferring into z/OS datasets, the STATE queue contains more information about the destination file, including allocation data such as block size and record format, which results in larger persistent messages on the queue.

Setting the channel BATCHINT attribute to a value of larger than 0 does ensure the channel batches are held open for longer but did not have a significant impact to the overall throughput achieved.

The most significant factor affecting the 1MB file transfer performance is that Java file deletion and creation of z/OS datasets is more expensive than the equivalent actions with USS files.

With larger files being transferred, a smaller proportion of the time is spent deleting and creating the destination dataset, which explains why there is less difference between z/OS datasets and USS files performance.

There is still some difference in transfer rate and cost between z/OS datasets and USS files, and this is partly because USS files do not have the same constraints of record length and block size. As a result, more disk I/O requests may be required to write to the z/OS dataset, and as such the disk response time becomes an additional factor in the achieved transfer rate when writing to z/OS datasets.

4. Alternative MQ Configurations

Shared queues

An alternative MFT configuration on z/OS is to host the agent queues on shared queues.

The default configuration of MFT is to use 256KB messages.

In a shared queue environment, 256KB messages will be stored in either Db2 tables or Shared Message Data Sets (SMDS).

As the channels are defined with `NPMSPEED(FAST)` and the messages are processed out of syncpoint, they are eligible for "put to waiting getter", meaning that they are able to bypass the queues and either Db2 or SMDS.

The impact of "put to waiting getter" is discussed later.

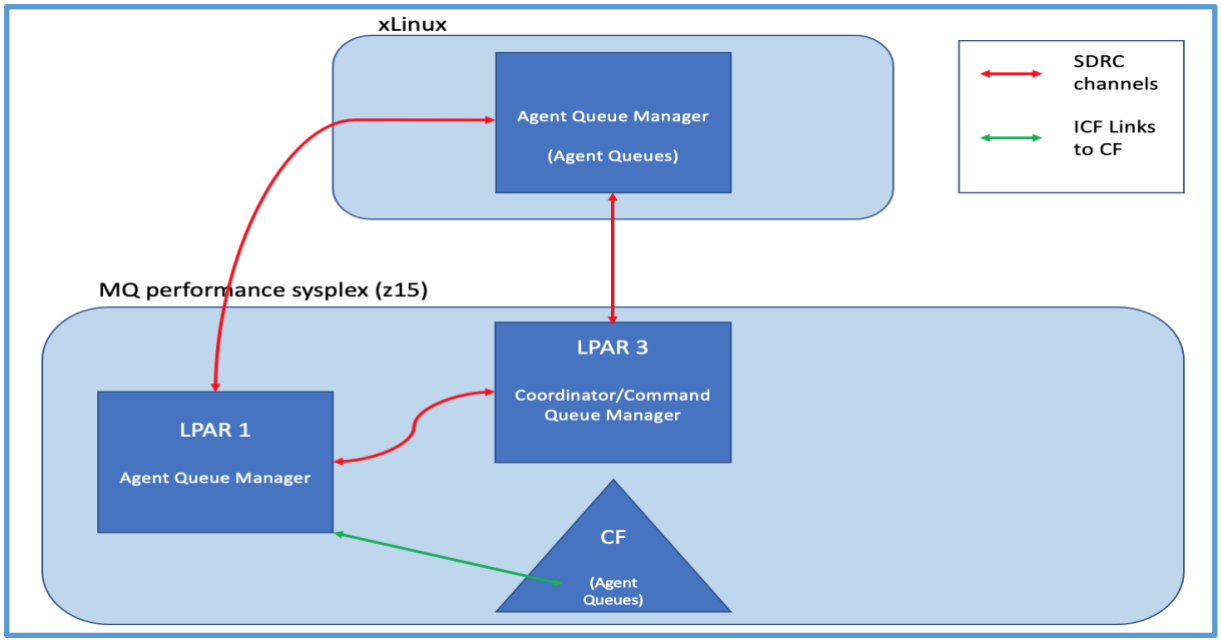
When defining an MFT agent to a z/OS queue manager, the `fteCreateAgent` script provides MQSC to define 5 queues.

These 5 queues are prefixed `SYSTEM.FTE`, followed by the role of the queue and then suffixed by the agent name, e.g. `SYSTEM.FTE.COMMAND.QM01`.

When defining these queues, the `MAXMSGL` is 4MB, so the queues need to be defined to structures of CF level 4 or greater.

Whilst MFT sends non-persistent messages to the `DATA` queue, other queues may use persistent messages, so they need to be in a structure defined with `RECOVER(YES)`.

The topology used for the shared queue configuration is as follows



To give a representation of the cost of using shared queues, both Db2 and SMDS measurements are offered in this section.

In each configuration, all agent queues are defined in a single application structure defined at CF level 5 with the offload destination configured as either:

- Db2.
- SMDS configured to offload all messages.

Inbound to z/OS dataset

Chart 11: Achieved transfer rate between xLinux and z/OS datasets

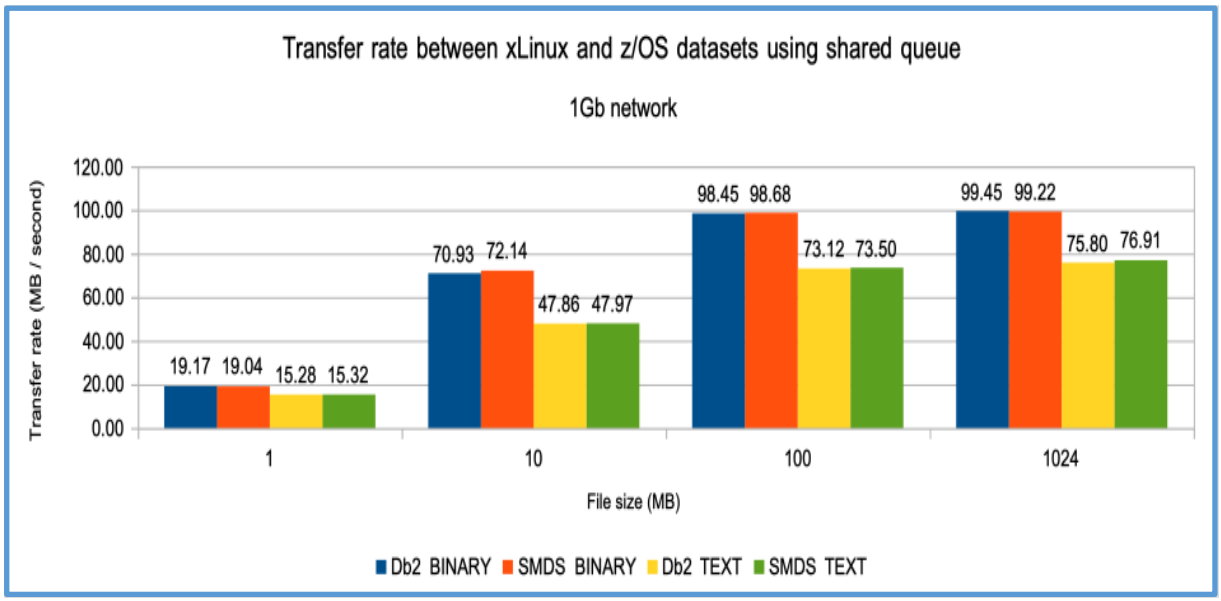
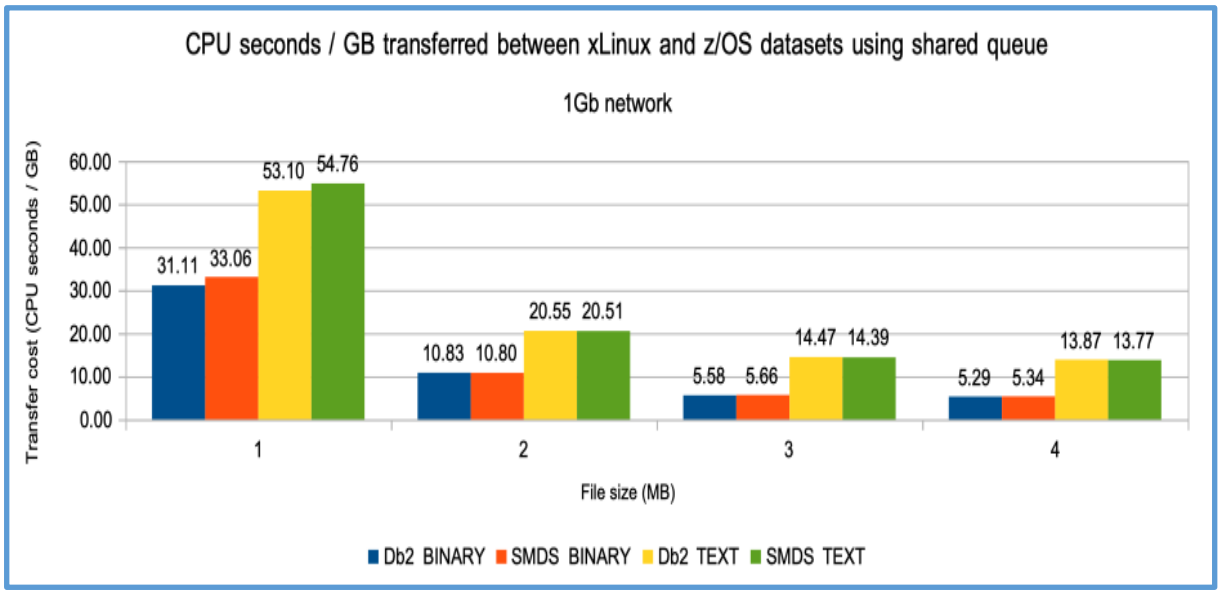


Chart 12: Cost per GB of data transferred between xLinux and z/OS datasets



Outbound from z/OS datasets

Chart 13: Achieved transfer rate between z/OS datasets and xLinux

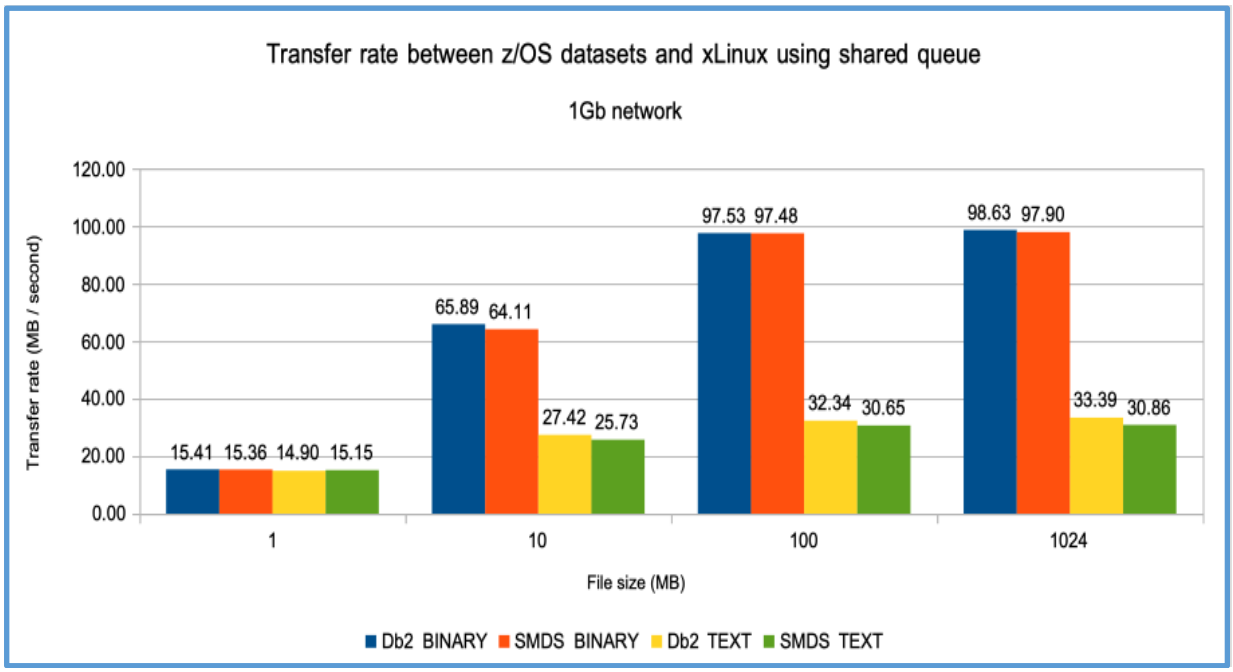
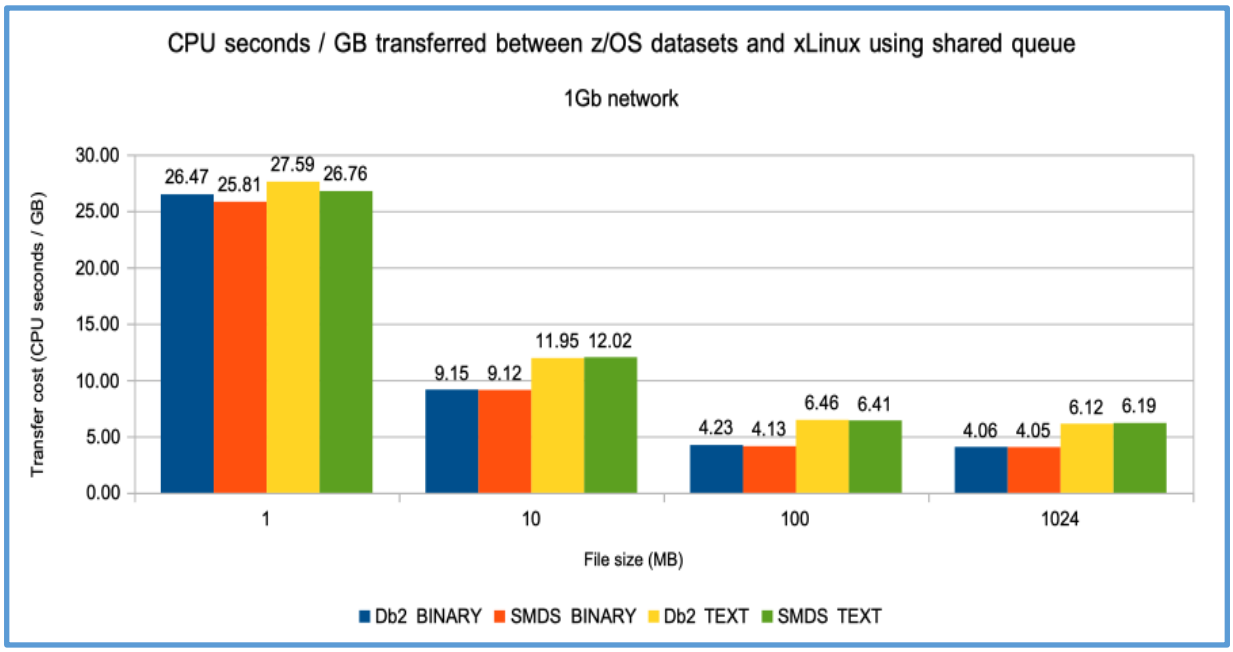


Chart 14: Cost per GB of data transferred between z/OS datasets and xLinux



TLS protection on MQ channels

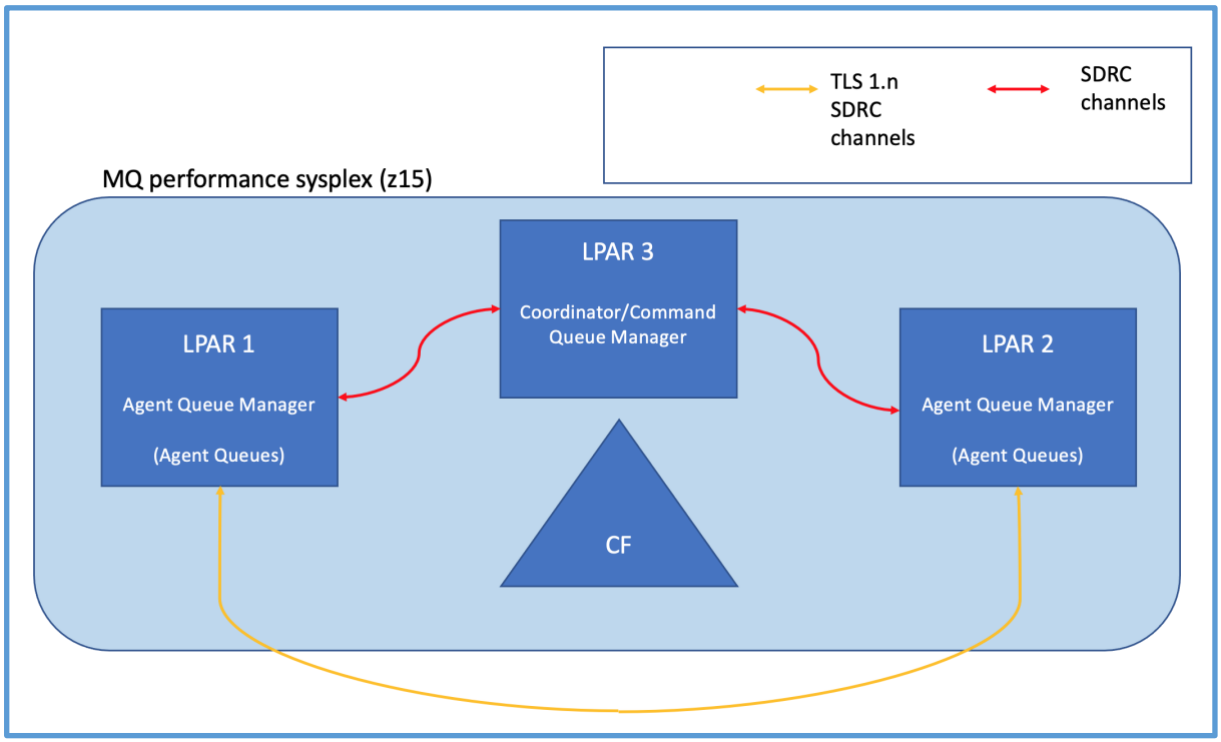
It is typical to run MQ channels with TLS cipher protection enabled. This will incur additional cost in the MQ channel initiator address space, although some of the cost can be offloaded onto the available cryptographic hardware, whether CryptoExpress or CPACF.

For the purposes of these measurements, the MQ channels have been configured with TLS cipher protection but SSLRKEYC queue manager is set to 0, such that the secret key is not re-negotiated by MQ post-channel start. Note that TLS 1.3 ciphers will re-negotiate the secret key at self-determined intervals.

The ciphers used in these measurements are:

- TLS 1.2: `009D TLS_RSA_WITH_AES_256_GCM_SHA384`
- TLS 1.3: `1302 TLS_AES_256_GCM_SHA384`

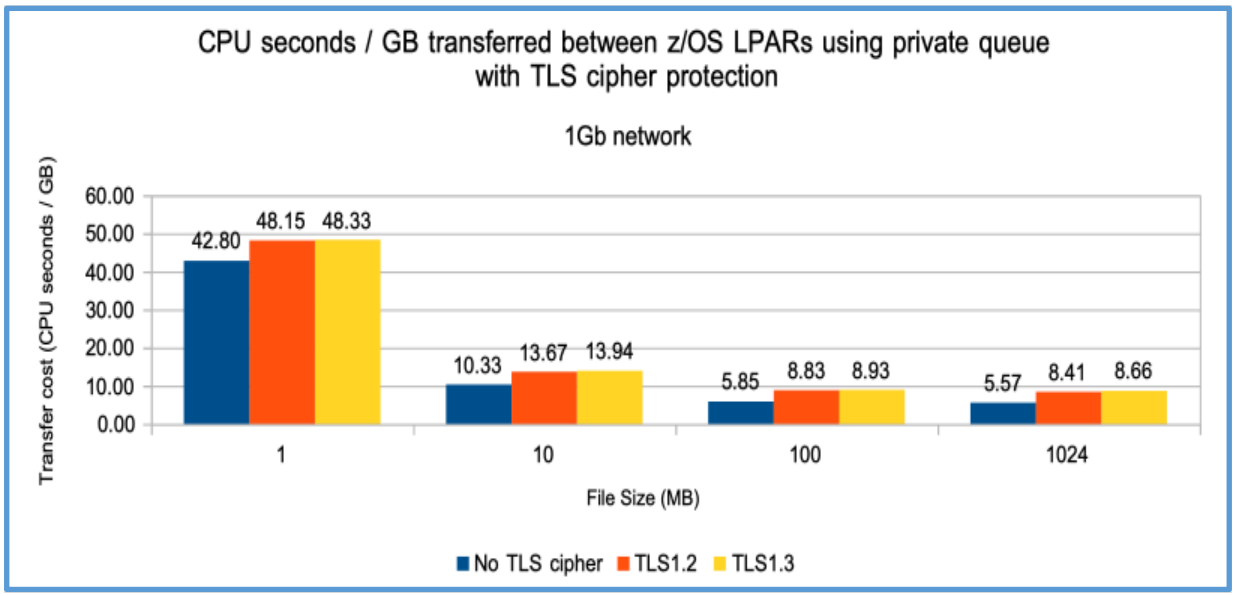
The topology used for these TLS protection configuration is as follows:



Note that for the purposes of demonstrating the impact of TLS cipher protection, only z/OS to z/OS transfers are measured. This allows the maximum use of available cryptographic hardware, namely CPACF, which is used for encryption and decryption of data on z/OS.

In our performance system, applying TLS cipher protection to the MQ channels between MFT Agents resulted in minimal impact to the file transfer rate, but as indicated earlier, there was additional cost. In a system where CPU is already limited, the additional cost may have a more notable impact on the transfer rate.

Chart 15: Cost per GB of data transferred between z/OS LPARs



When applying TLS cipher protection to the MQ channels, we observed an increase to the cost per GB of data transferred to be of the order of 3 to 5.5 CPU seconds. This increase is to the total cost but the increase is relatively evenly distributed between the source and the destination LPARs.

Can channel compression offer benefit?

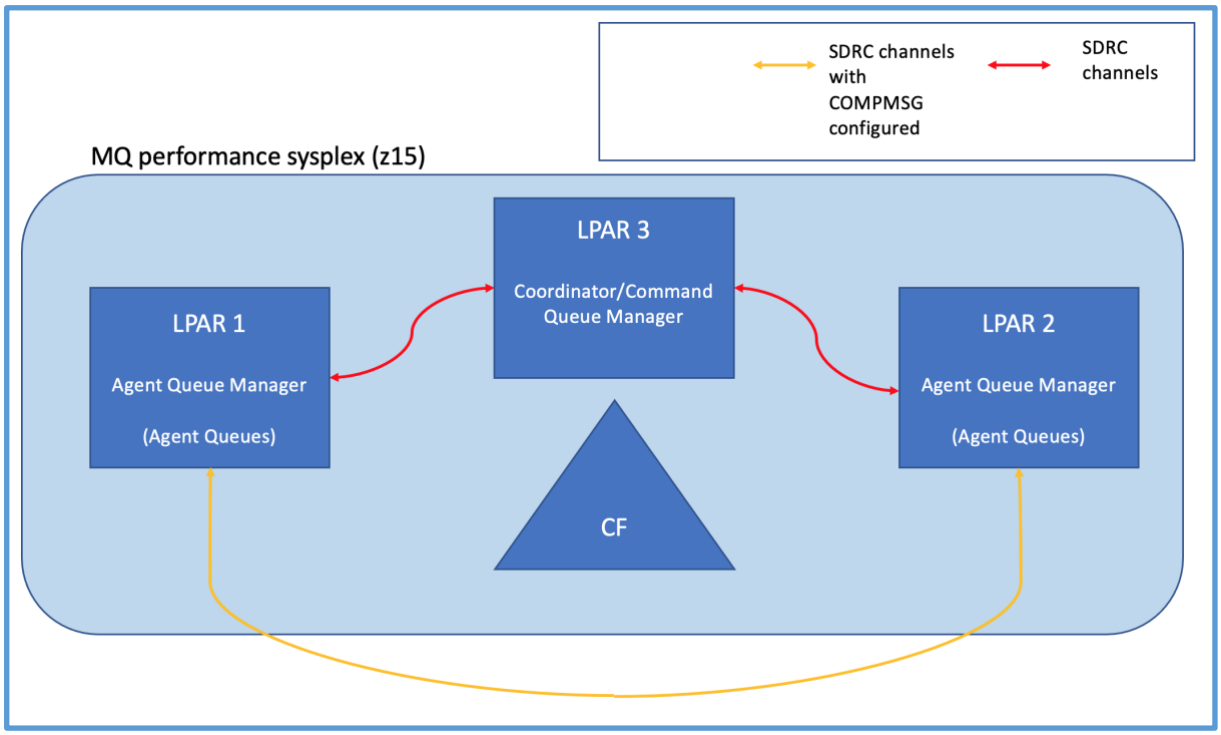
MFT typically uses 256KB non-persistent messages to transfer the file data between MFT agents. Depending on the contents of the data being transferred and network capacity, some benefit may be achieved by compressing the data using the `COMPMSG` channel attribute.

The `COMPMSG` channel attribute supports the following options on z/OS:

- RLE - software-based run length encoding.
- ZLIBHIGH - software-based ZLIB compression
- ZLIBFAST - hardware-based (where available) ZLIB compression.

Compressing the data will result in less data being transferred over the network, but typically will come with additional cost. With sufficiently high compression being achieved, the savings on network cost may out-weigh the cost of compression, particularly if the MQ channels are protected with TLS ciphers – this is discussed further in the performance report "MQ for z/OS on z15".

The topology used for the channel compression measurements is as follows:



Note that the transfers reported in this section are only between z/OS LPARs. This allows the workload to fully exploit hardware compression via the `COMPMSG(ZLIBFAST)` option on the z15. Hardware compression on the optional zEDC (zEnterprise Data Compression) PCIe card is also available on z14 and earlier hardware but in that configuration, compression performance is affected by the additional latency of moving the data to and from the PCIe card.

Whether compression is performed in software (as per RLE or ZLIBHIGH) or in hardware (ZLIBFAST), the re-inflation of the compressed data is typically performed in software and is relatively similar in cost regardless of compression option.

The following table show the percentage of compression achieved by the channels when processing each of the file sizes by the compression type specified:

COMPMSG	1MB	10MB	100MB	1GB
RLE	53	25	48	43
ZLIBHIGH	63	44	60	57
ZLIBFAST	64	44	60	57

Note that the compression numbers shown are the rate achieved to the nearest percentage; that is, a rate of 25 indicates messages are being compressed to 75% of their original length.

Despite the test files being relatively simple files containing repeating strings of data, the ZLIB compression algorithms were able to compress the messages more.

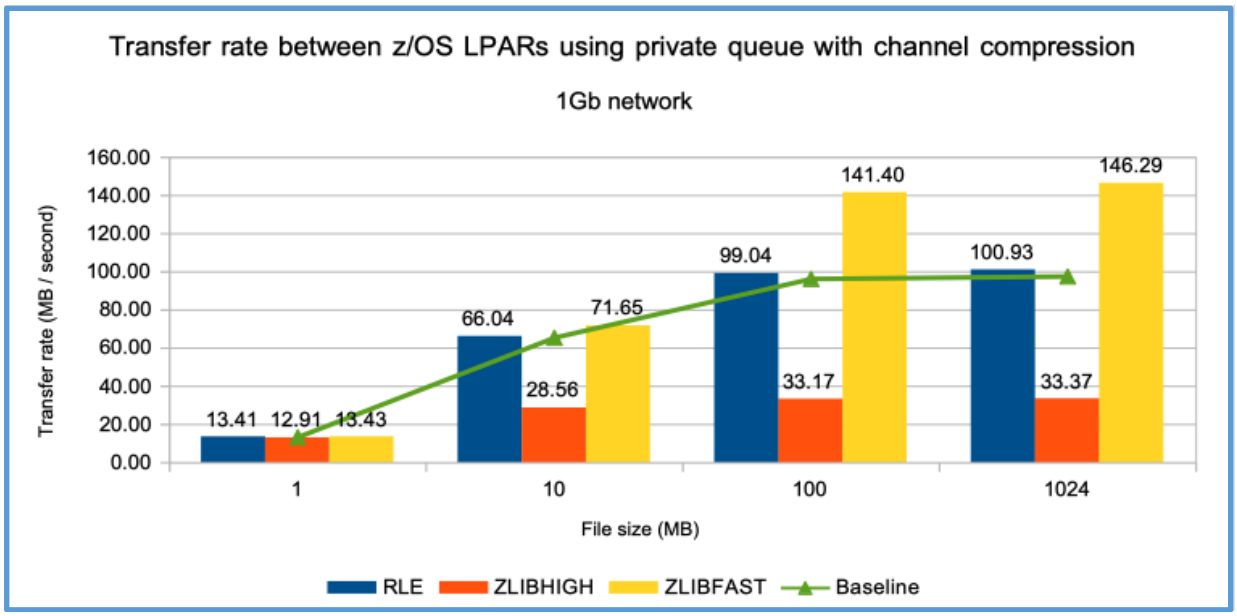
This second table shows the time spent in compression in microseconds by the sending-side of the channel.

COMPMSG	1MB	10MB	100MB	1GB
RLE	23	35	24	28
ZLIBHIGH	304	489	335	363
ZLIBFAST	10	11	11	11

It is clear that when compression hardware is available, such as on z15 or via the optional zEDC on PCIe on earlier systems, ZLIBFAST offers a more efficient compression option than both RLE and ZLIBHIGH for both how much compression can be achieved and the cost of that compression.

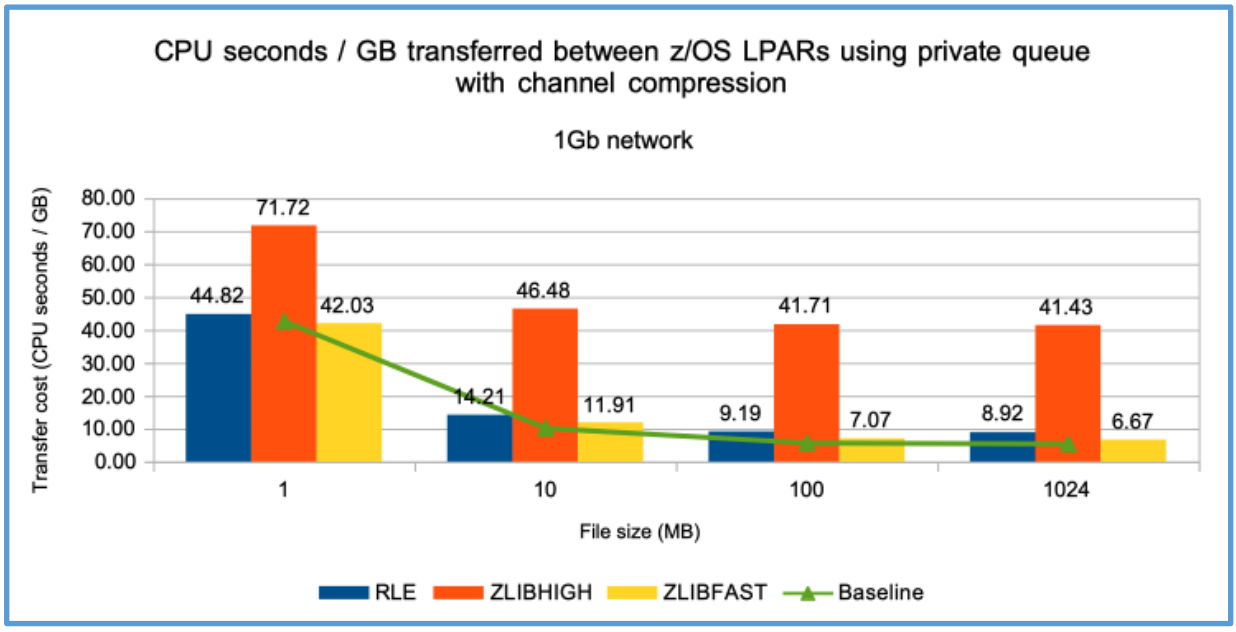
The following two charts show how compressing the messages affects the transfer rate and the cost of transferring the data.

Chart 16: Achieved transfer rate between z/OS LPARs with COMPMSG set



In our low latency measurement environment, applying channel compression typically does not tend to improve overall throughput unless there is significant compression achieved. However with ZLIBFAST on the larger file sizes, there is a significant improvement in throughput over the uncompressed file transfer, such that the transfer rate equates to 146 MB/second.

Chart 17: Cost per GB of data transferred between z/OS LPARs with COMPMSG set



In terms of the cost of compressing the files being transferred, both RLE and ZLIBFAST result in a small increase over the uncompressed measurements, with ZLIBHIGH being significantly higher.

Does AMS Protection impact the transfer?

What is AMS?

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.

There are three qualities of protection:

1. *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.
2. *Privacy* protection is provided by a combination of digital signing and encryption. Encryption ensures that the message data is only viewable by the intended recipient, or recipients.
3. *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 Express7S, 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.

Which MFT queues need AMS policies?

There is a small anomaly when using AMS protection with MFT on z/OS in that the transmit queue also needs to be protected.

For the purposes of these measurements, we have protected the following queues:

Source agent
- SYSTEM.FTE.DATA.<destination QM>
- SYSTEM.FTE.COMMAND.<destination QM>
- Transmit Queue which in our environment is named <destination QM>.

Destination agent
- SYSTEM.FTE.DATA.<destination QM>
- SYSTEM.FTE.COMMAND.<destination QM>

The other MFT queues could have AMS policies applied, but only the STATE queue would potentially hold sensitive data, i.e. file names.

AMS configurations with MFT

In this section, the following configurations were monitored:
- z/OS LPAR to z/OS LPAR binary file transfers using the 1Gb performance network.

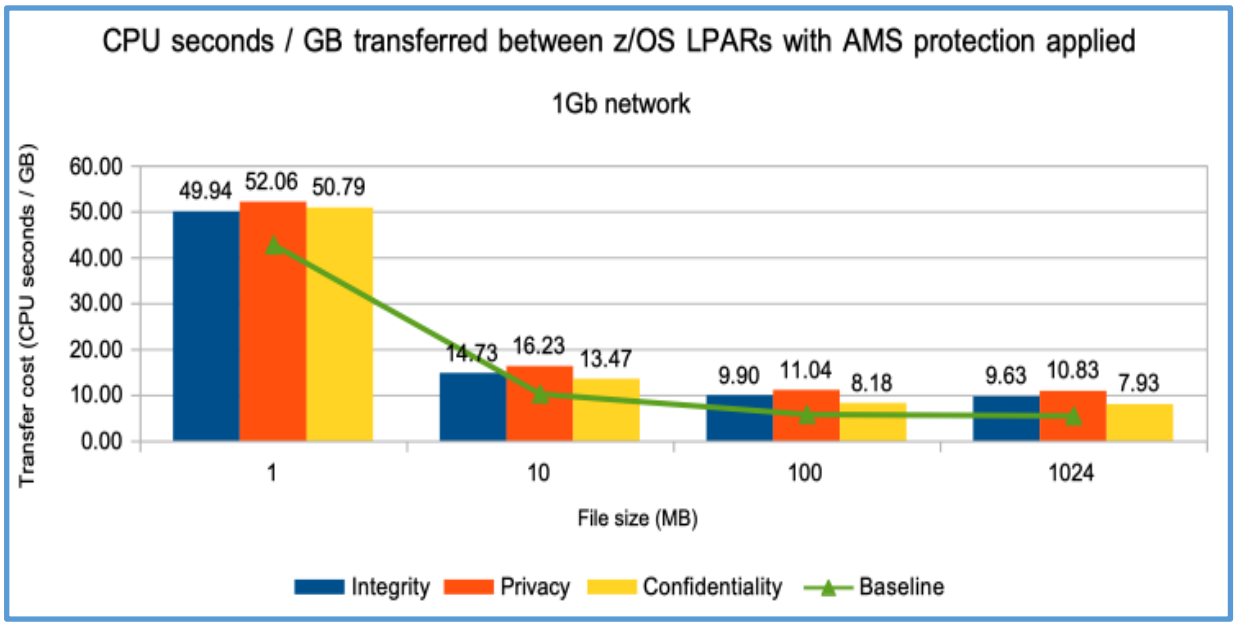
AMS policies used were:
- Integrity
- Privacy
- Confidentiality with key reuse of 32.

Note that applying AMS protection to the DATA queue will prevent MQ from being able to use “put to waiting getter” and this will result in higher MQ costs due to putting and getting the message from the queue.

In terms of transfer rate, the policy type had minimal impact on the measured file sizes, with less than a 6% difference to the throughput achieved in the reported z/OS to z/OS transfer section.

There is however an impact to the cost of transferring data when AMS policy protection is applied.

Chart 18: Cost per GB of data transferred between z/OS LPARs with AMS policies defined.



Note that cost of the 1MB file transfers is primarily impacted by the cost of the file I/O, and any difference in the cost is due to the variable nature of the workload.

Message to file transfers

The **message-to-file** feature of MFT enables you to transfer data from one or more messages on an IBM MQ queue to a file, a data set (on z/OS) or a user file space.

Conversely the file-to-message feature allows a file to be transferred from a file to a single or multiple messages on an IBM MQ queue.

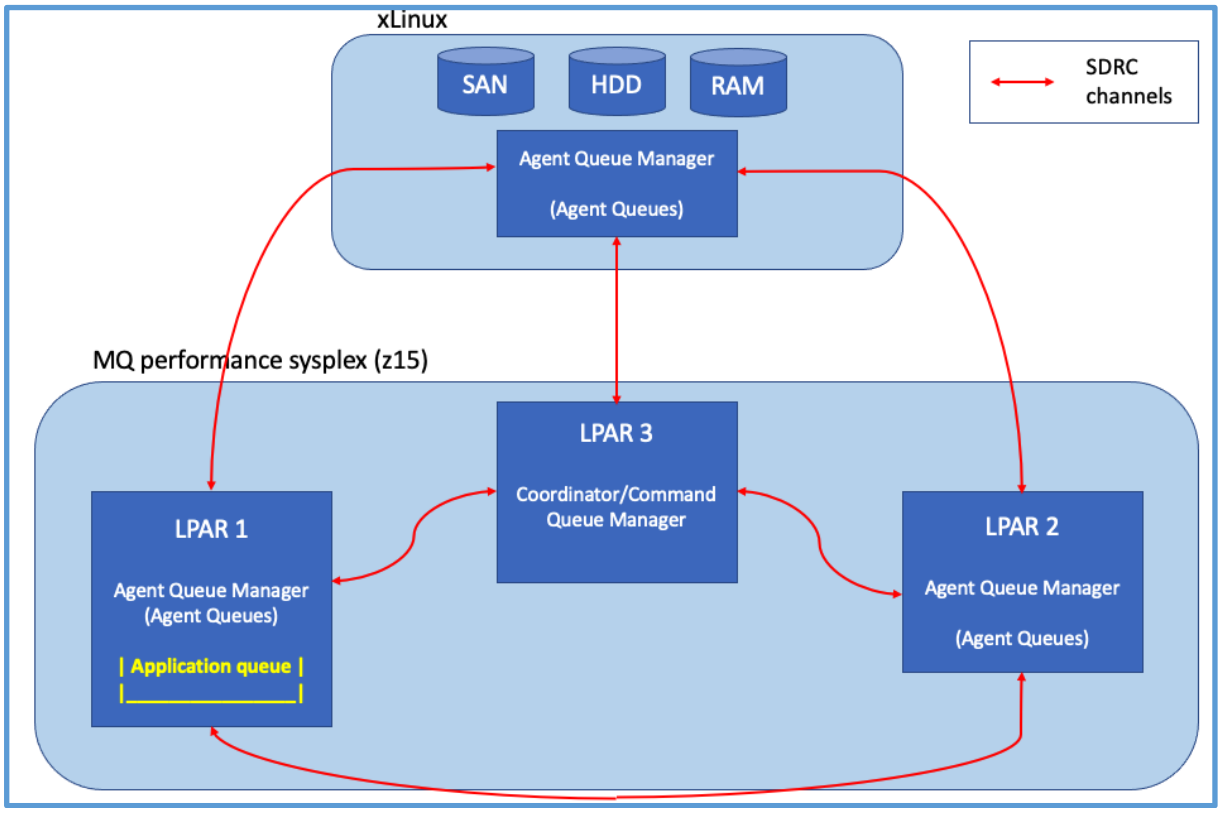
For the purposes of this section, we will discuss the performance of the message-to-file feature where the queue is hosted on a z/OS queue manager.

In order for the source and destination agents to support file-to-message, the source agent must be configured with `enableQueueInputOutput=true` in the `agent.properties` file at start of agent.

In this section, the transfer will send a pre-loaded MQ queue on the z/OS queue manager to a single file on a range of destinations:

- xLinux – where the file system is local disk, SAN and RAM disk.
- Unix System Services directory.

The topology used for this configuration is as follows:



In all measurements in this section, the queue on the queue manager in LPAR 1 is pre-loaded with 1024 non-persistent messages each with a 1MB payload.

The MQ queue hosting the 1GB of message data is hosted on a 4GB buffer pool such that the data can be gotten without page set I/O.

Message-to-file binary transfers

The performance of the message-to-file binary transfers typically is better than the performance achieved when text transfers are requested, simply because the binary transfer does not need to perform any code conversion of the data to be written.

As such the performance that was achieved on our systems is as follows:

Configuration	Transfer rate (MB / second)	z/OS Cost per GB (CPU seconds)
Messages to xLinux SAN	65.35	14.9
Messages to xLinux HDD	59.40	14.9
Messages to xLinux RAM disk	72.47	15.0
Messages to Unix System Services	81.40	16.4

Note that the cost of the messages to Unix System Services includes the cost on both the source and destination LPARs, whereas the transfer of messages to xLinux only includes the cost of the z/OS source LPAR.

The RAM disk measurement is only included to demonstrate the impact of the file system response times on the xLinux end-point, and is not suggested as a suitable production implementation.

Message-to-file text transfers

As mentioned previously, the performance of message-to-file text transfers is typically worse than the equivalent binary transfer, in terms of both cost and transfer rate.

From a z/OS perspective, as the code conversion is taking place on the destination system, the message-to-file transfers which target xLinux should see similar transfer costs as the binary transfers, although we do see a slight increase as the transfer takes longer to complete.

For the text transfer between the message on z/OS to the Unix System Services file, where data conversion is taking place on the destination LPAR, we see additional cost to the transfer, which is largely eligible for offload to zIIP.

Configuration	Transfer rate (MB / second)	z/OS Cost per GB (CPU seconds)
Messages to xLinux SAN	38.25	16.5
Messages to xLinux HDD	25.00	17.9
Messages to xLinux RAM disk	29.71	17.6
Messages to Unix System Services	31.28	39.1

File to message transfers

The **file-to-message** feature, which is not in the scope of this document, as detailed by the MQ knowledge, states that the messages put to the queue are persistent by default.

This may impact the performance of other persistent workload using the destination queue manager.

How much benefit comes from "put to waiting getter"?

IBM MQ has an optimisation for out-of-syncpoint non-persistent messages that can be MQPUT directly to a waiting out-of-syncpoint MQGET, rather than placed onto the queue and then read from the queue.

Receiving channels which have been defined with `NPMSPEED(FAST)` use out-of-syncpoint MQPUTs for non-persistent messages; applications which issue out-of-syncpoint MQGETs against these messages are therefore eligible to benefit from this improvement.

The larger the message, the greater the CPU reduction – and since MFT transports file transfer data in 256KB messages, the savings can be significant.

When running with "put to waiting getter" disabled so that every message was put to queue, the transfer costs per GB increased by up to 30% those of the baseline measurements.

5. Network

Does a network with more capacity help?

Yes – in certain circumstances.

Certainly there will be benefit if the existing network is running at capacity at any time the file transfer is occurring, but in our performance system, we are typically the only significant user of the network links at the time of the measurements.

To determine the impact of transporting the files over a network with greater capacity, the MQ channel definitions were updated to route the workloads over the 10Gb performance network and the following configurations were repeated:

- xLinux to z/OS datasets
- z/OS datasets to xLinux
- USS files to xLinux
- z/OS to z/OS
- USS files to USS files

Typically, binary file transfers of larger (100MB and 1GB) achieved improved transfer rates with increased network bandwidth.

Smaller files and files where data conversion is required (text-type), saw minimal impact from the additional network bandwidth.

Binary files transferred from **xLinux to z/OS datasets** achieved transfer rates of up to 190MB/second.

Files transferred from **z/OS datasets to xLinux** saw significant improvement for larger binary-type files being transferred, with transfer rates up to 250MB/second.

USS files transferred **to xLinux** saw transfer rates of up to 220MB/second.

z/OS to z/OS files transfers also saw significant improvements to throughput for the larger file configurations (100MB and larger) – with 1GB files transferring at a rate of 200MB/second.

USS files to USS files demonstrated a peak transfer rate reached 223MB/second for 1GB files as indicated in the following table showing the achieved transfer rates in MB per second:

Details of the throughput achieved can be found in Appendix C – Summary of results.

Multi-channel support

In the previous section, we saw that the performance of USS file transfer was improved with a better network. In the majority of measurements in this report, we have used a 1Gb network and have been achieving transfer rates of up to 100MB/second, but simply by using the 10Gb network, the peak transfer rates has increased to 293MB/second.

In addition, the MFT agent supports multi-channel configurations using the `agentMultipleChannelsEnabled` property in the `agent.properties` file.

This allows MFT to use multiple MQ channels to transfer files between the source and destination agent queue managers and potentially exploit additional unused bandwidth to enable the file transfers to complete more quickly.

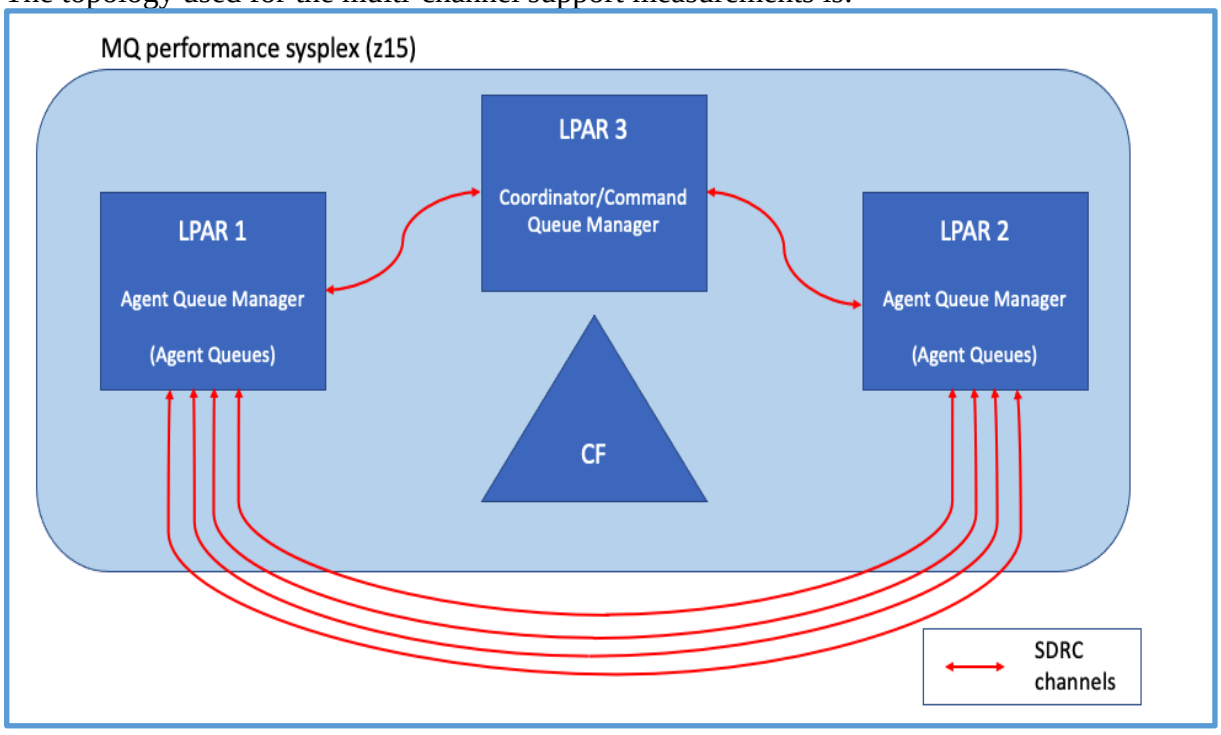
The IBM MQ Knowledge centre section "Configuring an MFT agent for multiple channels non-clustered" provides an example configuration.

For this set of measurements, the file transfers were between 2 z/OS LPARs using:

- z/OS dataset to z/OS dataset
- USS files to USS files

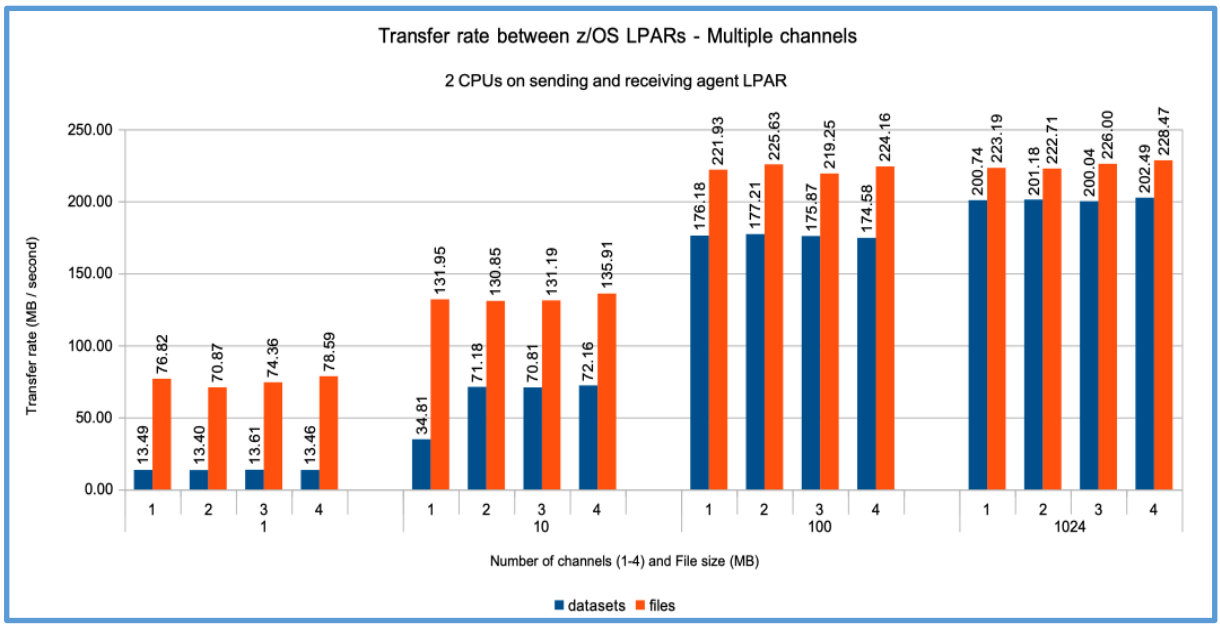
As the transfers are between z/OS LPARs, they use binary transfers only.

The topology used for the multi-channel support measurements is:



Transfers are run using 1 to 4 MQ channels, and the differences are shown in the following two charts.

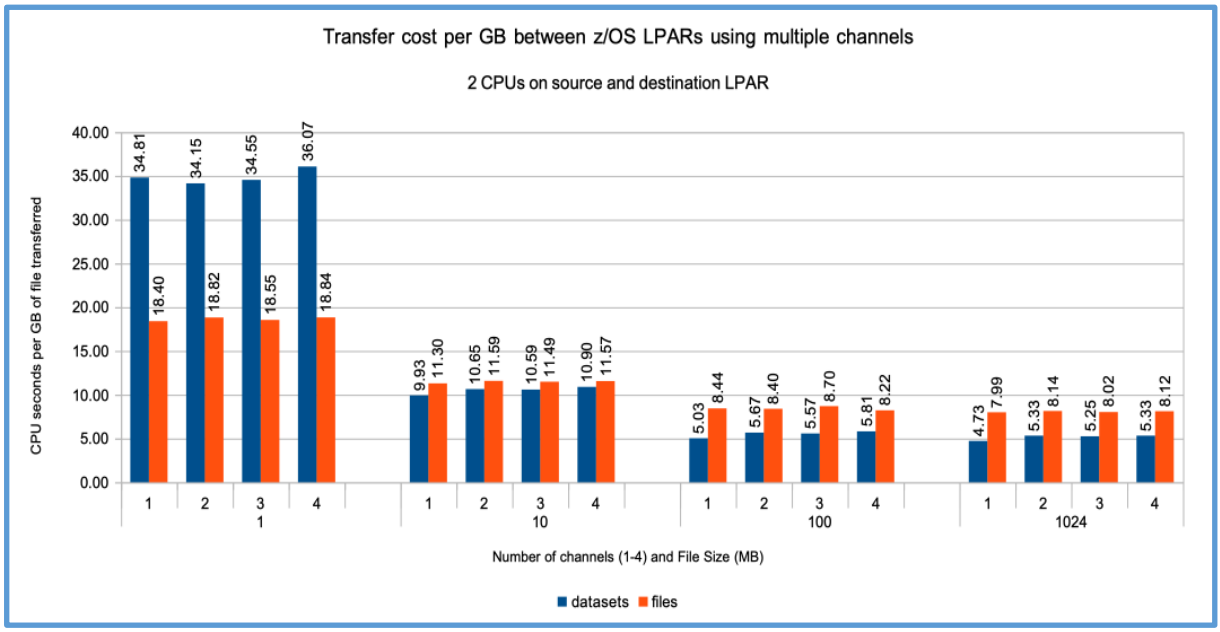
Chart 19: Achieved transfer rate between z/OS LPARs with multiple MQ channels.



In our measurements, file transfers using multiple MQ channels did not result in improved transfer rates.

When multiple MQ channels are available for MFT transfers, all data for any particular file will flow over a single channel. Therefore multiple channels will not offer benefit to single file transfers.

Chart 20: Cost per GB of data transferred between z/OS LPARs with multiple MQ channels.



Using multiple channels to transfer the file workload resulted in minimal overall impact to the overall cost.

All of the disk I/O either to or from z/OS datasets uses a storage group with only a single volume, and as such does seem some extended I/O times when running multiple channel configurations.

How does MFT perform over high latency networks?

When transferring files using MFT over extended distance, both the achieved transfer rate and the cost of the transfer can be affected.

Since IBM MQ Advanced for z/OS VUE version 9.2, the IBM® fasp.io gateway is available which provides a fast TCP/IP tunnel that can significantly increase network throughput for IBM MQ.

Details on fasp.io performance for general MQ workloads is available in the MQ for z/OS version 9.2 performance report.

The fasp.io 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 fasp.io gateway, so it is important to check network performance before and after defining a fasp.io gateway connection.

A queue manager running on any entitled platform can connect through a fasp.io 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).

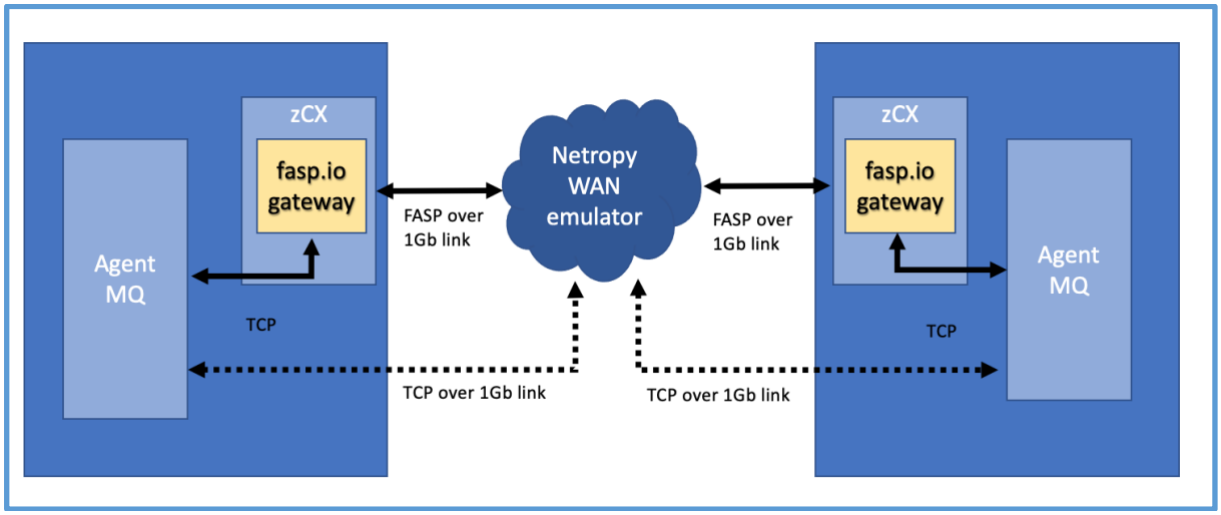
When deploying the fasp.io gateway in a z/OS Container Extension, the CPU used is largely eligible for offload to zIIP. In our measurements, it was typical to see in excess of 98% of the zCX costs to be eligible for zIIP offload.

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.

fasp.io gateway test configuration

The 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.

The topology between agents is shown in the diagram following:



Note:

- All communication between agent queue managers, whether using the fasp.io gateway or not, is routed though the Netropy WAN emulator.
- All communication is limited to 1Gb / second.
- The fasp.io gateway on each LPAR is running in a zCX.
- Due to the availability of connections to the Netropy appliance, communications with the Coordinator/Command queue manager are using channels with minimal latency.

As with other file transfer measurements, those running over extended simulated distance will use 1MB, 10MB, 100MB and 1GB files.

All transfers will be run using z/OS LPAR to LPAR using datasets without any data conversion.

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

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

Chart 21: Achieved transfer rate between z/OS LPARs over extended distance

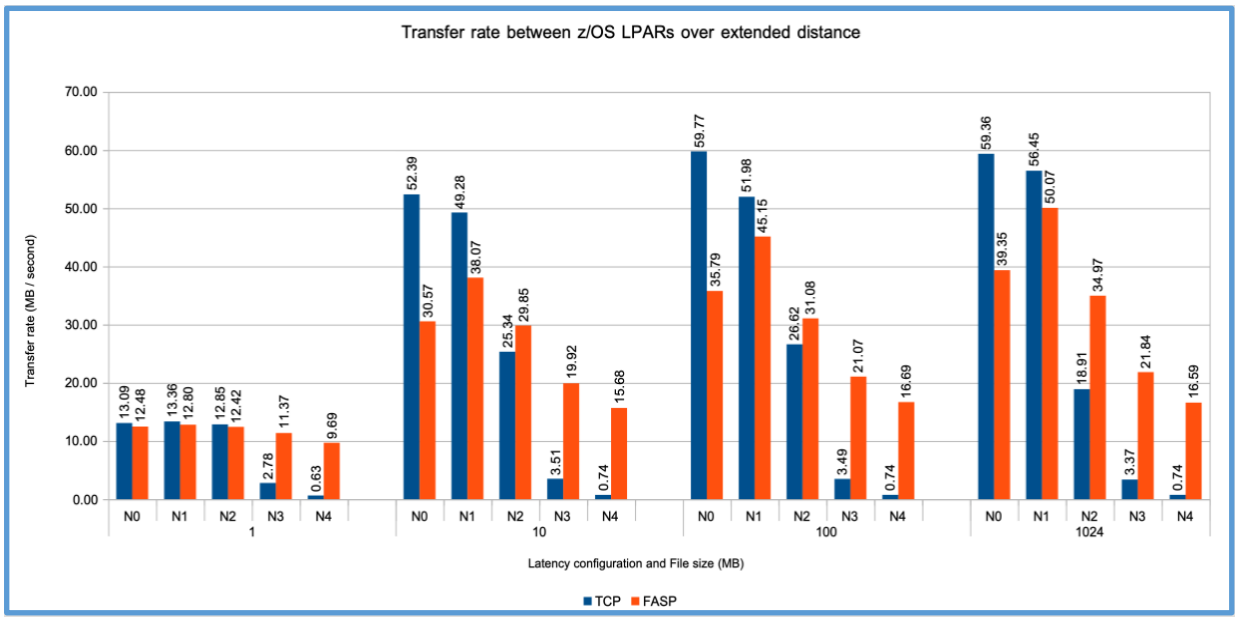
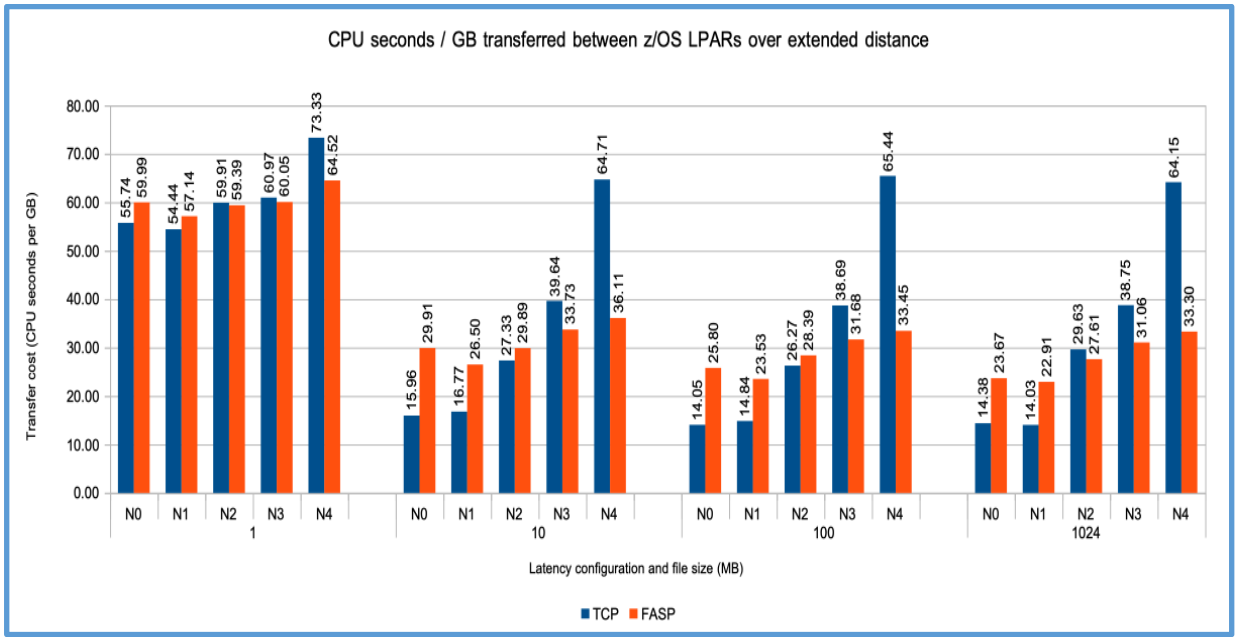


Chart 22: Cost per GB of data transferred between z/OS LPARs over extended distance



For short distance transfers with the default settings MQ channel settings and Agent configurations, we found that no benefit in using the fasp.io gateway.

As the latency increased, the benefit of using the fasp.io gateway increased such that:

- N2 – with larger file sizes, throughput up to 1.8 times that of the TCP connection.
- N3 – Improvements in throughput ranged between 4 to 7 times that of TCP.
- N5 – fasp.io saw up to 22 times the throughput of the equivalent TCP transfer.

Increased channel batch size, coupled with tuning of the Agent configuration, such as `AgentWindowSize` and `AgentFrameSize`, which may result in more messages flowed

in a single batch. This may help improve throughput on a file transfer that is constrained by the network response time due to the distance between agents – particularly on a reliable network or one using a fasp.io gateway.

An overview of some of the Agent configuration options is provided in Appendix B.

6. Tuning your system for MFT

What benefit is there from zHPF?

zHPF is a performance and reliability, availability, serviceability enhancement of the z/Architecture and the FICON channel architecture.

Exploitation of zHPF by the FICON channel, the z/OS operating system and the control unit is designed to help reduce the FICON channel overhead.

Enabling zHPF can reduce the load on the I/O subsystem, and as such we would typically encourage MQ for z/OS users to run with zHPF enabled.

The disk subsystems used for the measurements in this report are dedicated for performance test and as such as consistently lightly loaded, and zHPF has less impact than it might on a busier disk subsystem. Despite this, disabling zHPF resulted in the file transfer rate decreasing by up to 55%.

DASD – Optimising BLKSIZE

When transferring data into MVS datasets, MFT allows you to specify the LRECL and BLKSIZE attributes for the destination dataset.

It is recommended that an optimum BLKSIZE is used for the data being transferred.

For example, consider a 1MB file that is written to a file with a record length of 80 bytes but varying the BLKSIZE:

- BLKSIZE(27920) uses 18.9 tracks (1.02MB)
- BLKSIZE(6400) uses 21.0 tracks (1.135MB)

Using a BLKSIZE that is less than optimum size has resulted in using 11% more disk space.

For best performance, aim for 2 blocks per track.

Processor choices

The IBM® z Integrated Information Processor (zIIP) is a dedicated processor designed to operate asynchronously with the general purpose processors in a mainframe to process new workloads; manage containers and hybrid cloud interfaces; facilitate system recovery; and assist with several types of analytics, systems monitoring and other applications, including Java™-based workloads.

The IBM MFT product is largely written in Java™ and as such is eligible for benefitting from zIIP-offload.

IBM MQ for z/OS is not written in Java and therefore does not benefit directly from the use of zIIPs, however offload of MFT work to zIIP relieves some of the workload on the general purpose processors for other work.

The measurements in this report were run on z/OS LPARs where there were zIIP configured but varied offline, with the intent to report the file transfer costs when zIIPs are not available.

The RMF workload report can be used to display the proportion of the total workload that is eligible for zIIP offload.

To ensure the maximum offload of the MFT workload is achieved, sufficient zIIP should be available. The zIIP-eligible portion of the workloads run in this report are typically contained within a single processor.

If there is only a single zIIP available, running the processors in multi-threading mode using the `MT_ZIIP_MODE=2` enabled would help ensure maximum offload to zIIP is achieved.

For the measurements in this report, the following zIIP-eligibility was reported:

- With smaller files (less than 100MB), where the agent running on z/OS does not perform data conversion, approximately 60% of the total cost is eligible for offload to zIIP.
- For larger files (100MB+) where the agent running on z/OS does not perform data conversion, approximately 40% of the total cost can be run on zIIP.
- Where the agent running on z/OS does perform data conversion, i.e. text-type transfers, approximately 90% of the total cost is eligible for offload to zIIP.

7. MFT queue usage

When using the `fteCreateAgent` script to define an Agent, it is necessary to define a number of MQ queues. These are prefixed "SYSTEM.FTE" and suffixed with the Agent name:-

1. **COMMAND**
   This queue is used when the transfer is being initiated / terminated
2. **DATA**
   This queue is used when the transfer is in progress to hold the files being transferred. By default these will hold 256KB non-persistent messages.
3. **REPLY**
   This queue is used during the transfer process to allow agents to exchange acknowledgment of progress.
4. **STATE**
   The STATE queue is maintained to allow MFT to track the progress of the file transfer. The size of the message put to the queue depends on:
   - The number of files being transferred
   - The path length to the file(s) being transferred.

   When transferring a large number of files or transferring files with long path names, the size of the message put to the STATE queue will increase.

   If the STATE queue has been defined using the default MFT definitions, the maximum message length will be 4MB. Transferring a large number of files with long path names can result in a STATE message being larger than 4MB – it may be necessary to increase the value of MAXMSGL for the STATE queue.

5. **EVENT**
   This queue is used when monitors are triggered and for other non-regular transfer functions.

8. MFT recommendations

The following are a list of bullet-pointed recommendations when planning your IBM MQ Managed File Transfer network when z/OS is involved:

- When file sizes are small, send them over multiple concurrent transfers rather than a single large transfer. This increases the efficiency of the I/O involved in transferring the files as well as driving the MQ channel more efficiently.

- When transferring files that require translation, i.e. text-mode, it is advisable to run multiple concurrent threads.

- Consider whether a text transfer is required between Agents. Even if the Agents are moving data from compatible platforms, the cost of attempting unnecessary translations is significant.

- Performance of the file transfer is significantly improved when the destination file does not already exist, such that the pre-transfer delete does not need to take place.

- Transferring large numbers of files with long path names can result in large messages being put to the STATE queue – it is suggested that when transferring large numbers of files, the files are identified by a short path and file name.

- Multiple smaller files place the agent under strain due to the Operating System open/close costs associated with more files. Where possible, configure your file creation processes to generate archives of smaller files, enabling MFT to use less open/close calls.

- Reading and writing to physical disk is often going to be the performance constraint. For agents that will see a large number of incoming and outgoing transfers, it would be best if high performance disks were used to read data from and write data to.

- Ensure your Agent has sufficient memory available. On z/OS, prior to starting the Agent, we specified:
  `export BFG_JVM_PROPERTIES="-Xmx1024M -Xms1024M"`

- As discussed in the Overview section, since the original publishing of this document we have measured with other Java™ configuration options, and subsequent releases of this report will implement the following options in the `BFG_JVM_PROPERTIES` variable:
  `-Xlp:objectheap:pagesize=2g,warn,nonpageable`
  `-Xlp:codecache:pagesize=1m,pageable`
  `-Xgc:concurrentScavenge`

- Default MQ channel settings allow MFT to work in its optimal manner. Overriding attributes such as NPMSPEED can have a significant detrimental effect.

- If NPMSPEED(NORMAL) is required on the MQ channel, it is advisable to test a range of typical transfers whilst varying the advanced tuning option. MFT has been optimised for use with NPMSPEED(FAST).

- As mentioned in the "trying to understand variable performance" section, in future releases we will configure the MFT agent on the distributed partner with:
  ```
  export BFG_JVM_PROPERTIES="-Xmx2048M -Xms2048M -Xjit:optlevel=scorching"
  ```

9. IBM MQ Considerations

When using MFT with IBM MQ for z/OS, it is advisable to ensure that the queue managers in use are monitored on a regular basis.

Given the nature of the MFT workload, it is not necessary to configure a dedicated z/OS queue manager solely for the MFT transfer, but it is still worth ensuring that:

- For agents using private queues, ensure that there are sufficient buffers allocated to be able to store a full batch of MFT messages.
- For agents using shared queues with SMDS offload, ensure there are sufficient DSBUFFS such that as much of the MFT data can be read from buffers, thus avoiding disk I/O on MQGET.
- Channels used for MFT should be defined with `NPMSPEED(FAST)` to allow "put to waiting getter" to be used.

Review performance report MP16 "Capacity Planning and Tuning Guide" for advice on configuring your queue manager on z/OS.

Appendix A – Test Environment

The IBM MQ performance sysplex run on z15 (8561-7J1) configured thus:

LPAR 1: 2 dedicated general purpose processors with 144 GB of real storage.
LPAR 2: 1 dedicated general purpose processors with 48 GB of real storage.
LPAR 3: 1 dedicated general purpose processors with 48 GB of real storage.

LPARs 1 and 3 have zIIP configured but varied offline. This allows us to use RMF to predict the percentage of workload eligible for offload, whilst reporting total costs when zIIPs are not available.

- z/OS v2r4.
- Db2 for z/OS version 12 configured for MQ using Universal Table spaces.
- MQ queue managers:
  - configured at MQ V9.2.0 with latest maintenance.
  - configured with dual logs and dual archives.
- MFT agents and transfer requests on z/OS configured with IBM Health Center enabled in headless mode e.g. `-Xhealthcenter:level=headless`

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 LPAR and CF.

DASD:

- FICON Express 16S connected DS8870
- 4 dedicated channel paths
- HYPERPAV enabled

Network:

- 1GbE network used by default.
- 10GbE network available.

System settings:

- zHPF enabled, unless otherwise specified.
- HIPERDISPATCH enabled by default.
- On-chip compression available by default, used with MQ channel attribute COMPMSG(ZLIBFAST).
- Crypto Express7 features enabled, configured thus:
  - 1 x Accelerator, shared between LPARs 1, 2 and 3.
  - 2 x Coprocessor on LPAR 1.
  - 2 x Coprocessor on LPAR 2.
  - 1 x Coprocessor on LPAR 3.

IBM MQ trace status:

- TRACE(GLOBAL) disabled.
- TRACE(CHINIT) disabled.

- TRACE(S) CLASS(1,3,4) enabled.
- TRACE(A) CLASS(3,4) enabled.

Client machines:

- 2 x IBM System X5660 each with 12 x 2.6Ghz Processor, 32GB memory.
    - 8Gb SAN card connected to IBM SAN Volume Controller plus SSD storage array (IBM FlashSystem 900).

Appendix B – MFT Advanced tuning options

This section briefly describes some of the tuning options that can be applied to the `agent.properties` file – for further detail, refer to the IBM Knowledge section "MFT agent.properties file".

Each agent has its own properties files that contain the information that an agent uses to connect to its queue manager. This `agent.properties` file can be altered to use advanced agent properties. Some of these agent properties are shown below with their default value.

- agentChunkSize 262144 bytes (256KB)
- agentWindowSize 10
- agentFrameSize 5
- agentCheckpointInterval 1

For each `agentWindowSize` messages received on the DATA queue, a reply message is sent to the sending agent.

If the sending agent sends `agentWindowSize * agentFrameSize` messages and does not receive acknowledgement for the first `agentWindowSize` batch of messages, the sending agent will stop sending any more messages until acknowledgement is received.

For every `agentWindowSize * agentFrameSize * agentCheckpointInterval` messages received on the DATA queue, an update is made to the STATE queue, which involves an MQGET and MQPUT of a persistent message.

For the default `agentCheckpointInterval` the total amount of data moved is:

```
agentChunkSize    * agentWindowSize       * agentFrameSize
256KB             * 10                    * 5               * 1   = 12.5MB
```

This means that 12.5MB of data will be transferred per checkpoint when the `agentCheckpointInterval` is 1.

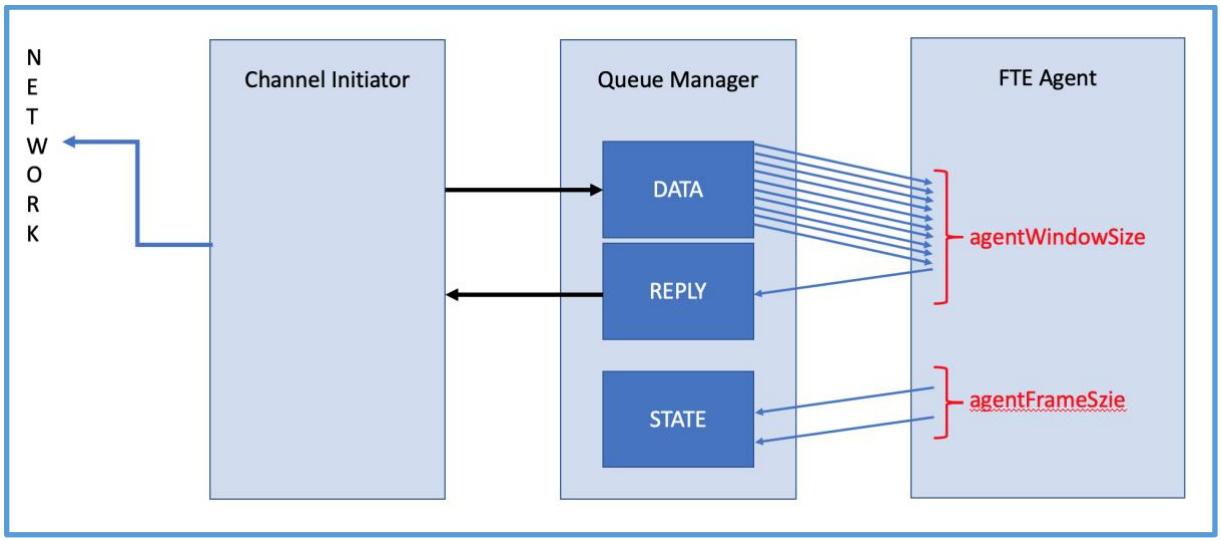
When altering the advanced tuning options, consider the relationship between attributes. For example consider the following table:

Configuration	agentWindowSize	agentFrameSize	agentCheckpoint Interval
1	10	5	2
2	10	10	1

In the above table, for each configuration the STATE queue is updated every 100 messages, i.e.: `agentWindowSize * agentFrameSize * agentCheckpointInterval`

In configuration 2, the increased `agentFrameSize` means that the sending agent will send twice as many batches of data before pausing in the event of the acknowledgement message not being received from the first batch.

The following diagram shows the flow of messages and the interaction with MFT queues when files are transferred into a z/OS agent queue manager.



The messages flow in from the network to the channel initiator and for every 10 (`agentWindowSize`) messages put to the `DATA.<receivingAgent>` queue, a single acknowledgement message is sent to the `REPLY.<sendingAgent>` queue.

For every 5 (`agentFrameSize * agentCheckpointInterval`) acknowledgement messages sent, an MQGET (destructive) followed by an MQPUT is made to the `STATE.<receivingAgent>` queue.

Depending on your network, you may find it beneficial to increase or decrease the agent properties mentioned previously in this section.

agentChunkSize

Varying the size of the message transmitted over the network may offer some benefit to the rate at which the data is transferred between agents.

agentWindowSize

The `agentWindowSize` property can be used to control the amount of syncpoints committed, as well as the number of acknowledgements sent between two agents when transferring files.

Each `agentWindowSize` batch requires an acknowledgement message but the agent will allow `agentFrameSize` windows to be sent before the first acknowledgement message must be received else the transfer will pause until the message is received. Once the first acknowledgment is received, the next batch can be sent, at which point the second acknowledgment must be received before the subsequent batch can be sent, and so on.

The `agentWindowSize` property has a default value of 10. This means that for every 10 chunks of data sent over WebSphere MQ, the sending agent will take an internal checkpoint.

Increasing this property increases the amount of data that could potentially need to be re-transmitted if a recovery is required and is not recommended for unreliable networks.

agentFrameSize

The `agentFrameSize` property can be used to control the number of windows for the transfer frame.

agentCheckpointInterval

Varying the size of the checkpoint interval affects the number of complete frames of data at which a checkpoint is taken for recovery purposes. If a transfer fails, the transfer can only recover at checkpoint boundaries. Hence the larger the value of `agentCheckpointInterval`, the longer the agent will take to recover failed transfers.

With a reliable network and system, there may be benefit in increasing the size of the `agentCheckpointInterval`.

When a frame is complete, the receiving agent will apply an update to its STATE queue.

Appendix C – Summary of results

Binary transfers – Achieved transfer rate in MB/second

Source	Target	NETWORK	QUEUE	ZHPF	TLS	AMS	COMPMSG	P2WG	BATCHINT	1MB	10MB	100MB	1GB
xLinux	z/OS	1Gb	Private	Y	N	N	N	Y	0	19.75	71.15	98.85	98.39
z/OS	xLinux	1Gb	Private	Y	N	N	N	Y	0	15.42	64.05	96.47	98.63
xLinux	USS	1Gb	Private	Y	N	N	N	Y	0	53.64	71.70	98.67	99.99
USS	xLinux	1Gb	Private	Y	N	N	N	Y	0	55.77	74.70	98.07	98.28
z/OS	zOS	1Gb	Private	Y	N	N	N	Y	0	13.32	65.45	96.27	97.56
z/OS	zOS	1Gb	Private	Y	N	N	N	Y	1000	13.30	63.35	94.67	97.60
USS	USS	1Gb	Private	Y	N	N	N	Y	0	56.51	76.58	100.76	100.43
xLinux	z/OS	10Gb	Private	Y	N	N	N	Y	0	19.01	78.79	176.80	190.72
z/OS	xLinux	10Gb	Private	Y	N	N	N	Y	0	15.36	77.07	224.87	251.60
z/OS	z/OS	10Gb	Private	Y	N	N	N	Y	0	13.49	71.78	176.18	200.74
USS	USS	10Gb	Private	Y	N	N	N	Y	0	76.82	131.95	221.93	223.19
USS	xLinux	10Gb	Private	Y	N	N	N	Y	0	68.86	124.85	217.58	220.36
z/OS	z/OS	1Gb	Private	Y	TLS 1.2	N	N	Y	0	13.37	64.84	95.76	97.59
z/OS	z/OS	1Gb	Private	Y	TLS 1.3	N	N	Y	0	13.24	65.53	95.37	97.22
xLinux	z/OS	1Gb	Private	N	N	N	N	Y	0	18.31	72.14	99.30	99.44
z/OS	xLinux	1Gb	Private	N	N	N	N	Y	0	14.49	64.09	97.32	98.79
z/OS	z/OS	1Gb	Private	N	N	N	N	Y	0	12.69	65.59	96.40	97.95
z/OS	z/OS	1Gb	Private	Y	N	Integrity	N	Y	0	13.56	65.76	95.89	97.09
z/OS	z/OS	1Gb	Private	Y	N	Privacy	N	Y	0	13.59	63.83	95.36	97.85
z/OS	z/OS	1Gb	Private	Y	N	Confidentiality	N	Y	0	13.33	62.54	95.25	98.20
xLinux	z/OS	1Gb	Shared – Db2	Y	N	N	N	Y	0	19.17	70.93	98.45	99.45
xLinux	z/OS	1Gb	Shared – SMDS	Y	N	N	N	Y	0	19.04	72.14	98.68	99.22
z/OS	xLinux	1Gb	Shared – Db2	Y	N	N	N	Y	0	15.41	65.89	97.53	98.63
z/OS	xLinux	1Gb	Shared – SMDS	Y	N	N	N	Y	0	15.36	64.11	97.48	97.90
xLinux	z/OS	1Gb	Private	Y	N	N	N	N	0	19.51	72.19	99.27	98.90
xLinux	z/OS	1Gb	Shared – Db2	Y	N	N	N	N	0	19.44	71.65	97.04	97.95
xLinux	z/OS	1Gb	Shared – SMDS	Y	N	N	N	N	0	19.33	72.16	98.53	98.36
z/OS	z/OS	10Gb	Private	Y	N	N	RLE	Y	0	13.41	66.04	99.04	100.93
z/OS	z/OS	10Gb	Private	Y	N	N	ZLIBHIGH	Y	0	12.91	28.56	33.17	33.37

z/OS	z/OS	10Gb	Private	Y	N	N	ZLIBFAST	Y	0	13.43	71.65	141.40	146.29

Text transfers – Achieved transfer rate in MB/second

Source	Target	NETWORK	QUEUE	ZHPF	TLS	AMS	COMPMSG	P2WG	BATCHINT	1MB	10MB	100MB	1GB
xLinux	z/OS	1Gb	Private	Y	N	N	N	Y	0	16.00	47.86	74.79	76.31
z/OS	xLinux	1Gb	Private	Y	N	N	N	Y	0	15.25	49.06	70.50	71.66
xLinux	USS	1Gb	Private	Y	N	N	N	Y	0	28.68	62.14	85.84	87.73
USS	xLinux	1Gb	Private	Y	N	N	N	Y	0	23.09	28.54	32.81	32.97
xLinux	z/OS	**10Gb**	Private	Y	N	N	N	Y	0	16.26	48.36	74.52	76.99
z/OS	xLinux	**10Gb**	Private	Y	N	N	N	Y	0	15.00	27.17	32.51	32.53
USS	xLinux	**10Gb**	Private	Y	N	N	N	Y	0	23.04	28.62	32.71	32.82
xLinux	z/OS	1Gb	Private	**N**	N	N	N	Y	0	14.17	43.11	66.44	69.09
z/OS	xLinux	1Gb	Private	**N**	N	N	N	Y	0	14.40	26.29	31.71	31.90
xLinux	z/OS	1Gb	Shared – Db2	Y	N	N	N	Y	0	15.28	47.86	73.12	75.80
xLinux	z/OS	1Gb	Shared – SMDS	Y	N	N	N	Y	0	15.32	47.97	73.50	76.91
z/OS	xLinux	1Gb	Shared – Db2	Y	N	N	N	Y	0	14.90	27.42	32.34	33.39
z/OS	xLinux	1Gb	Shared – SMDS	Y	N	N	N	Y	0	15.15	25.73	30.65	30.86
xLinux	z/OS	1Gb	Private	Y	N	N	N	N	0	14.56	44.63	74.12	76.40
xLinux	z/OS	1Gb	Shared – Db2	Y	N	N	N	N	0	16.02	47.26	73.58	76.37
xLinux	z/OS	1Gb	Shared – SMDS	Y	N	N	N	N	0	15.15	46.97	74.14	77.56

Binary transfers - Cost per GB transferred in CPU seconds.

Source	Target	NETWORK	QUEUE	ZHPF	TLS	AMS	COMPMSG	P2WG	BATCHINT	1MB	10MB	100MB	1GB
xLinux	z/OS	1Gb	Private	Y	N	N	N	Y	0	25.94	8.90	4.28	4.13
z/OS	xLinux	1Gb	Private	Y	N	N	N	Y	0	23.61	8.55	3.78	3.66
xLinux	USS	1Gb	Private	Y	N	N	N	Y	0	22.31	11.90	7.13	6.99
USS	xLinux	1Gb	Private	Y	N	N	N	Y	0	19.41	11.24	6.99	6.85
z/OS	zOS	1Gb	Private	Y	N	N	N	Y	0	42.80	10.33	5.85	5.57
z/OS	zOS	1Gb	Private	Y	N	N	N	Y	1000	44.48	10.50	5.89	5.69
USS	USS	1Gb	Private	Y	N	N	N	Y	0	19.06	12.25	8.57	8.45
xLinux	z/OS	10Gb	Private	Y	N	N	N	Y	0	23.18	8.96	4.22	3.73
z/OS	xLinux	10Gb	Private	Y	N	N	N	Y	0	25.81	8.29	2.93	2.73
z/OS	z/OS	10Gb	Private	Y	N	N	N	Y	0	34.81	9.93	5.03	4.73
USS	USS	10Gb	Private	Y	N	N	N	Y	0	18.40	11.30	8.44	7.99
USS	xLinux	10Gb	Private	Y	N	N	N	Y	0	17.50	9.82	5.57	5.43
z/OS	z/OS	1Gb	Private	Y	TLS 1.2	N	N	Y	0	48.15	13.67	8.83	8.41
z/OS	z/OS	1Gb	Private	Y	TLS 1.3	N	N	Y	0	48.33	13.94	8.93	8.66
xLinux	z/OS	1Gb	Private	N	N	N	N	Y	0	30.61	9.83	5.14	4.86
z/OS	xLinux	1Gb	Private	N	N	N	N	Y	0	29.54	10.05	4.55	4.41
z/OS	z/OS	1Gb	Private	N	N	N	N	Y	0	48.99	11.26	6.47	6.37
z/OS	z/OS	1Gb	Private	Y	N	Integrity	N	Y	0	49.94	14.73	9.90	9.63
z/OS	z/OS	1Gb	Private	Y	N	Privacy	N	Y	0	52.06	16.23	11.04	10.83
z/OS	z/OS	1Gb	Private	Y	N	Confidentiality	N	Y	0	50.79	13.47	8.18	7.93
xLinux	z/OS	1Gb	Shared - Db2	Y	N	N	N	Y	0	31.11	10.83	5.58	5.29
xLinux	z/OS	1Gb	Shared - SMDS	Y	N	N	N	Y	0	33.06	10.80	5.66	5.34
z/OS	xLinux	1Gb	Shared - Db2	Y	N	N	N	Y	0	26.47	9.15	4.23	4.06
z/OS	xLinux	1Gb	Shared - SMDS	Y	N	N	N	Y	0	25.81	9.12	4.13	4.05
xLinux	z/OS	1Gb	Private	Y	N	N	N	N	0	28.50	11.02	4.81	4.63
xLinux	z/OS	1Gb	Shared - Db2	Y	N	N	N	N	0	26.83	10.86	6.40	6.33
xLinux	z/OS	1Gb	Shared - SMDS	Y	N	N	N	N	0	26.74	9.63	5.03	4.90
z/OS	z/OS	10Gb	Private	Y	N	N	RLE	Y	0	44.82	14.21	9.19	8.92
z/OS	z/OS	10Gb	Private	Y	N	N	ZLIBHIGH	Y	0	71.72	46.48	41.71	41.43
z/OS	z/OS	10Gb	Private	Y	N	N	ZLIBFAST	Y	0	42.03	11.91	7.07	6.67

Text transfers - Cost per GB transferred in CPU seconds.

Source	Target	NETWORk	QUEUE	ZHPF	TLS	AMS	COMPMSG	P2WG	BATCHINT	1MB	10MB	100MB	1GB
xLinux	z/OS	1Gb	Private	Y	N	N	N	Y	0	45.10	17.65	12.89	12.39
z/OS	xLinux	1Gb	Private	Y	N	N	N	Y	0	24.70	9.26	4.21	3.97
xLinux	USS	1Gb	Private	Y	N	N	N	Y	0	50.09	20.93	14.84	14.48
USS	xLinux	1Gb	Private	Y	N	N	N	Y	0	21.63	13.48	8.52	8.36
xLinux	z/OS	**10Gb**	Private	Y	N	N	N	Y	0	42.68	18.04	13.19	12.64
z/OS	xLinux	**10Gb**	Private	Y	N	N	N	Y	0	26.73	11.19	5.96	5.85
USS	xLinux	**10Gb**	Private	Y	N	N	N	Y	0	20.88	12.84	8.16	8.09
xLinux	z/OS	1Gb	Private	**N**	N	N	N	Y	0	56.14	24.09	14.51	13.85
z/OS	xLinux	1Gb	Private	**N**	N	N	N	Y	0	30.26	13.49	7.70	7.43
xLinux	z/OS	1Gb	Shared - Db2	Y	N	N	N	Y	0	53.10	20.55	14.47	13.87
xLinux	z/OS	1Gb	Shared - SMDS	Y	N	N	N	Y	0	54.76	20.51	14.39	13.77
z/OS	xLinux	1Gb	Shared - Db2	Y	N	N	N	Y	0	27.59	11.95	6.46	6.12
z/OS	xLinux	1Gb	Shared - SMDS	Y	N	N	N	Y	0	26.76	12.02	6.41	6.19
xLinux	z/OS	1Gb	Private	Y	N	N	N	N	0	53.03	22.98	13.56	13.07
xLinux	z/OS	1Gb	Shared - Db2	Y	N	N	N	N	0	47.33	21.46	14.90	14.35
xLinux	z/OS	1Gb	Shared - SMDS	Y	N	N	N	N	0	48.90	20.44	13.83	13.00

Appendix D – Summary of impact from latency

All measurements in this section are configured with:

- Binary transfers between zOS LPARs using datasets
- 1Gb network
- Private queue
- zHPF enabled
- No TLS protection on MQ channels
- No AMS policy protection
- `COMPMSG(NONE)`
- Put to waiting getter available
- `BATCHINT(0)`

Achieved transfer rate in MB/second

Protocol	Latency (KM)	Loss (%)	1MB	10MB	100MB	1GB
TCP	0	0	13.09	52.39	59.77	59.36
TCP	1000	0	13.36	49.28	51.98	56.45
TCP	5000	0	12.85	25.34	26.62	18.91
TCP	8000	0.1	2.78	3.51	3.49	3.37
TCP	10000	0.5	0.63	0.74	0.74	0.74
FASP	0	0	12.48	30.57	35.79	39.35
FASP	1000	0	12.80	38.07	45.15	50.07
FASP	5000	0	12.42	29.85	31.08	34.97
FASP	8000	0.1	11.37	19.92	21.07	21.84
FASP	10000	0.5	9.69	15.68	16.69	16.59

Cost per GB transferred in CPU seconds

Protocol	Latency (KM)	Loss (%)	1MB	10MB	100MB	1GB
TCP	0	0	55.74	15.96	14.05	14.38
TCP	1000	0	54.44	16.77	14.84	14.03
TCP	5000	0	59.91	27.33	26.27	29.63
TCP	8000	0.1	60.97	39.64	38.69	38.75
TCP	10000	0.5	73.33	64.71	65.44	64.15
FASP	0	0	59.99	29.91	25.80	23.67
FASP	1000	0	57.14	26.50	23.53	22.91
FASP	5000	0	59.39	29.89	28.39	27.61
FASP	8000	0.1	60.05	33.73	31.68	31.06
FASP	10000	0.5	64.52	36.11	33.45	33.30

Appendix E – Summary of impact from multiple MQ channels

All measurements in this section are configured with:

- Binary transfers between zOS LPARs using either datasets or USS files
- 1 to 4 MQ channels between source and destination
- 10Gb network
- Private queue
- zHPF enabled
- No TLS protection on MQ channels
- No AMS policy protection
- `COMPMSG(NONE)`
- Put to waiting getter available
- `BATCHINT(0)`

In our measurements we saw little benefit from setting the agent property `agentMultipleChannelsEnabled=true`, but that may be due to the test environment or the file transfer workload.

Achieved transfer rate in MB/second

Type	Channels	1MB	10MB	100MB	1GB
Dataset	1	13.49	71.78	176.18	200.74
Dataset	2	13.40	71.18	177.21	201.18
Dataset	3	13.61	70.81	175.87	200.04
Dataset	4	13.46	72.16	174.58	202.49
USS files	1	76.82	131.95	221.93	223.19
USS files	2	70.87	130.85	225.63	222.71
USS files	3	74.36	131.19	219.25	226.00
USS files	4	78.59	135.91	224.16	228.47

Cost per GB transferred in CPU seconds

Type	Channels	1MB	10MB	100MB	1GB
Dataset	1	34.81	9.93	5.03	4.73
Dataset	2	34.15	10.65	5.67	5.33
Dataset	3	34.55	10.59	5.57	5.25
Dataset	4	36.07	10.90	5.81	5.33
USS files	1	18.40	11.30	8.44	7.99
USS files	2	18.82	11.59	8.40	8.14
USS files	3	18.55	11.49	8.70	8.02
USS files	4	18.84	11.57	8.22	8.12