relay log. It decompresses the transaction payloads to apply the transactions, and then writes them uncompressed to its own binary log, if it has one. Any downstream replicas receive the uncompressed transaction payloads.

When a source in a replication topology does not have binary log transaction compression enabled but its replica does, if the replica has a binary log, it compresses the transaction payloads after applying them, and writes the compressed transaction payloads to its binary log. Any downstream replicas receive the compressed transaction payloads.

When a MySQL server instance has no binary log, if it is at a release from MySQL 8.0.20, it can receive, handle, and display compressed transaction payloads regardless of its value for `binlog_transaction_compression`. Compressed transaction payloads received by such server instances are written in their compressed state to the relay log, so they benefit indirectly from compression that was carried out by other servers in the replication topology.

A replica at a release before MySQL 8.0.20 cannot replicate from a source with binary log transaction compression enabled. A replica at or above MySQL 8.0.20 can replicate from a source at an earlier release that does not support binary log transaction compression, and can carry out its own compression on transactions received from that source when writing them to its own binary log.

Monitoring Binary Log Transaction Compression

You can monitor the effects of binary log transaction compression using the Performance Schema table `binary_log_transaction_compression_stats`. The statistics include the data compression ratio for the monitored period, and you can also view the effect of compression on the last transaction on the server. You can reset the statistics by truncating the table. Statistics for binary logs and relay logs are split out so you can see the impact of compression for each log type. The MySQL server instance must have a binary log to produce these statistics.

The Performance Schema table `events_stages_current` shows when a transaction is in the stage of decompression or compression for its transaction payload, and displays its progress for this stage. Compression is carried out by the worker thread handling the transaction, just before the transaction is committed, provided that there are no events in the finalized capture cache that exclude the transaction from binary log transaction compression (for example, incident events). When decompression is required, it is carried out for one event from the payload at a time.

`mysqlbinlog` with the `--verbose` option includes comments stating the compressed size and the uncompressed size for compressed transaction payloads, and the compression algorithm that was used.

You can enable connection compression at the protocol level for replication connections, using the `SOURCE_COMPRESSION_ALGORITHMS` | `MASTER_COMPRESSION_ALGORITHMS` and `SOURCE_ZSTD_COMPRESSION_LEVEL` | `MASTER_ZSTD_COMPRESSION_LEVEL`options of the `CHANGE REPLICATION SOURCE TO` statement (from MySQL 8.0.23) or `CHANGE MASTER TO` statement (before MySQL 8.0.23), or the deprecated `slave_compressed_protocol` system variable. If you enable binary log transaction compression in a system where connection compression is also enabled, the impact of connection compression is reduced, as there might be little opportunity to further compress the compressed transaction payloads. However, connection compression can still operate on uncompressed events and on message headers. Binary log transaction compression can be enabled in combination with connection compression if you need to save storage space as well as network bandwidth. For more information on connection compression for replication connections, see Section 4.2.8, “Connection Compression Control”.

For Group Replication, compression is enabled by default for messages that exceed the threshold set by the `group_replication_compression_threshold` system variable. You can also configure