MySQL Protocol Features You Should Be Aware Of

Daniël van Eeden

PingCAP

pre-FOSDEM MySQL Belgian Days, February 2026

Agenda

- **Introduction**
- **Protocol Features**
  - Session Tracking
  - Compression
  - Connection Attributes
  - Query Attributes
  - Secure connections
  - Version info
  - X Protocol
  - Cloning
  - Replication
  - MariaDB enhancements
  - Timeouts
- **Q&A**

Who am I?

Daniël van Eeden.

Working for PingCAP on TiDB (MySQL Compatible database, written in Go).
Long time MySQL user.

Who am I?

Daniël van Eeden.

Working for PingCAP on TiDB (MySQL Compatible database, written in Go).
Long time MySQL user. Interested in the MySQL protocol because of contributions to:

- Wireshark

Who am I?

Daniël van Eeden.

Working for PingCAP on TiDB (MySQL Compatible database, written in Go).
Long time MySQL user. Interested in the MySQL protocol because of contributions to:

- Wireshark
- go-mysql

Who am I?

Daniël van Eeden.

Working for PingCAP on TiDB (MySQL Compatible database, written in Go).
Long time MySQL user. Interested in the MySQL protocol because of contributions to:

- Wireshark
- go-mysql
- TiDB

Who am I?

Daniël van Eeden.

Working for PingCAP on TiDB (MySQL Compatible database, written in Go).
Long time MySQL user. Interested in the MySQL protocol because of contributions to:

- Wireshark
- go-mysql
- TiDB
- MySQL

Who am I?

Daniël van Eeden.

Working for PingCAP on TiDB (MySQL Compatible database, written in Go).
Long time MySQL user. Interested in the MySQL protocol because of contributions to:

- Wireshark
- go-mysql
- TiDB
- MySQL
- DBD::mysql (Perl)

Session Tracking

Session Tracking adds extra information to OK packets.
Available info:

- The default schema
- Session specific variables
- User-defined variables
- Temporary tables
- Prepared statements

Session Tracking

How to enable:

- `session_track_state_change=ON`
- `session_track_system_variables='*'`
- `session_track_schema=ON`
- `session_track_gtids=ALL_GTIDS` (Or `OWN_GTID`)
- `session_track_transaction_info=CHARACTERISTICS`

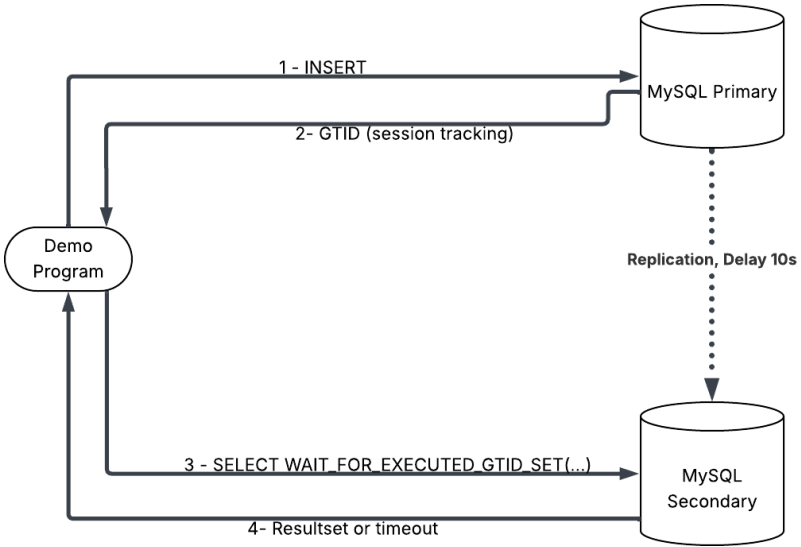
Session Tracking

Usecases

- Getting the `statement_id` for later lookup in `performance_schema`
- Getting the GTID(s) for consistent reads from replicas
- `SET NAMES latin1` → Changes in:
  `characters_set_client`,
  `character_set_connection`,
  `character_set_results`
- Getting transaction info that is useful for proxies (session migration)

Note that MySQL Shell displays the GTID and Statement ID. MySQL Client does not.

Session Tracking: Consistend replica reads



Session Tracking: Consistend replica reads

MySQL/3310 → MySQL/3311 (with `SOURCE_DELAY=10` for 10s delay).

```
Connected to MySQL on port 3310
  Creating test.t1
  Inserting record into test.t1
  Primary GTID: 859bfcf9-f1f7-11f0-ab7b-ee172f58920a:63 (from session tracking)
  SELECT SUM(val) FROM t1 returned 882
-----------------------------------------------

Connected to MySQL on port 3311
  SELECT SUM(val) FROM t1 returned 840
  Running SELECT WAIT_FOR_EXECUTED_GTID_SET('859bfcf9-f1f7-11f0-ab7b-ee172f58920a:63', 3) (try 0/10)
  Running SELECT WAIT_FOR_EXECUTED_GTID_SET('859bfcf9-f1f7-11f0-ab7b-ee172f58920a:63', 3) (try 1/10)
  Running SELECT WAIT_FOR_EXECUTED_GTID_SET('859bfcf9-f1f7-11f0-ab7b-ee172f58920a:63', 3) (try 2/10)
  Running SELECT WAIT_FOR_EXECUTED_GTID_SET('859bfcf9-f1f7-11f0-ab7b-ee172f58920a:63', 3) (try 3/10)
  Replica executed GTID successfully
  SELECT SUM(val) FROM t1 returned 882
```

Compression

Compression trades CPU time against bandwidth.

- Slow links (Doing DBA work over hotel Wi-Fi?)
- Expensive bandwidth (e.g. Cloud)

Compression

- Compression is **disabled** in the MySQL protocol by default.
- Compression used to only support **zlib**.
- Recently support for **zstandard** was added.
- Example: `mysql --compression-algorithms=zlib,zstd ...` (replaces the legacy `--compress`)
- New with `zstd`: specify the compression level (default: 3)
- Compression can also be used for replication via `SOURCE_COMPRESSSION_ALGORITHMS` and `SOURCE_ZSTD_COMPRESSION_LEVEL`
- For replication you may want to use Binary Log Transaction Compression instead.

Compression

```
mysql-9.5.0 [test]> status
--------------
mysql  Ver 9.5.0 for Linux on x86_64 (MySQL Community Server - GPL)

Connection id:          19
Current database:       test
Current user:           root@10.89.0.2
SSL:                    Cipher in use is TLS_AES_128_GCM_SHA256
Current pager:          stdout
Using outfile:          ''
Using delimiter:        ;
Server version:         9.5.0 MySQL Community Server - GPL
Protocol version:       10
Connection:             127.0.0.1 via TCP/IP
Server characterset:    utf8mb4
Db     characterset:    utf8mb4
Client characterset:    utf8mb4
Conn.  characterset:    utf8mb4
TCP port:               3310
Protocol:               Compressed, algorithms:  zstd, zstd level:  3
Binary data as:         Hexadecimal
Uptime:                 2 hours 40 min 44 sec

Threads: 3  Questions: 68  Slow queries: 0  Opens: 165  Flush tables: 3  Open tables: 85  Queries per second avg: 0.007
--------------
```

Compression

- A compressed packet can contain multiple MySQL packets
- A single MySQL packet can be split into multiplie compressed packets
- The maximum size of a compressed packet is just under 16MiB.
- Note that regular MySQL packets are also limited to just under 16MiB.
- `max_allowed_packet` does not set the maximum allowed packet size. It set the limit for a "message", which could consist of many packets.
- A compressed packet has a payload that can be compressed or uncompressed.
- If the compressed version of a payload is *larger* than the uncompressed version, then the uncompressed version is sent.
- If the payload is less than 50 bytes (`MIN_COMPRESS_LENGTH`) then it is sent uncompressed.
- The compression is done on MySQL packets of <16MiB, which limits compression.

Compression drawbacks

- Overhead: 7 byte header per packet. Especially when mostly sending small (uncompressed) packets.
- CPU usage. Especially on a MySQL primary that might already be heavily loaded.

Compression

So what happens if we run `SELECT REPEAT('fosdem____',5000)`?

Query: 46 bytes. Payload: not compressed.

Consisting of:

7	byte	compressed header
4	byte	regular header
1	byte	command
2	byte	parameter info (only if query attributes are enabled)
32	byte	statement

- With compression enabled: 46 bytes
- Without compression enabled: 39 bytes
- Compression **adds** 18%

Compression: zlib result 1/2

Compressed packet 1: 124 bytes compressed, 16384 bytes uncompressed.
    Column count: 1
    Field packet: describing column 1
    Row packet: part 1
Compressed packet 2: 99 bytes compressed, 33679 bytes uncompressed.
    Row packet: part 2
Compressed packet 3: not compressed, 11 bytes.
    OK packet

Row packet: 50_003 bytes (10 byte string x 5_000 + 3 byte header)
Note that `net_buffer_length` (default: 16384) sets the size of the first packet.

Compression: zlib result 2/2

Now with `net_buffer_length=100000`

Compressed packet 1: 197 bytes compressed, 50074 bytes uncompressed.

- Column count: 1
- Field packet: describing column 1
- Row packet
- OK packet

Row packet: 50_003 bytes (10 byte string x 5_000 + 3 byte header)

Compression: zstd result

Also with `net_buffer_length=100000`

Compressed packet 1: 96 bytes compressed, 50074 bytes uncompressed.

- Column count: 1
- Field packet: describing column 1
- Row packet
- OK packet

Row packet: 50_003 bytes (10 byte string x 5_000 + 3 byte header)

Compression level 1: 96 bytes
Compression level 22: 94 bytes

Compression: conclusions

Test	Packets	Uncompressed	Compressed
zlib with defaults	3	50074	234
zlib with large net buffer (level 3, default)	1	50074	197
zstd with large net buffer (level 1, min)	1	50074	96
zstd with large net buffer (level 22, max)	1	50074	94

1. zstd compressed better than zlib
2. increasing the `net_buffer_size` can really help
3. seeing replication efficiency is easy with Wireshark. Note that Wireshark can also uncompress zlib and zstd.

Note: Repeating text is easier to compress than something more random.

Note: The uncompressed size is excluding the 7-byte compressed packet header.

Compression monitoring

You can monitor `Bytes_sent` and `Bytes_received` and compare these with your OS counters.

```
mysql-9.5.0 [(none)]> SHOW STATUS LIKE 'Bytes\_%';
+----------------+-------+
| Variable_name  | Value |
+----------------+-------+
| Bytes_received | 760   |
| Bytes_sent     | 2264  |
+----------------+-------+
2 rows in set (0.002 sec)
```

For X Protocol there are more things you can monitor.

```
mysql-9.5.0 [(none)]> SHOW STATUS LIKE 'Mysqlx\_bytes\_%';
+---------------------------------------+-------+
| Variable_name                         | Value |
+---------------------------------------+-------+
| Mysqlx_bytes_received                 | 0     |
| Mysqlx_bytes_received_compressed_payload | 0     |
| Mysqlx_bytes_received_uncompressed_frame | 0     |
| Mysqlx_bytes_sent                     | 0     |
| Mysqlx_bytes_sent_compressed_payload  | 0     |
| Mysqlx_bytes_sent_uncompressed_frame  | 0     |
+---------------------------------------+-------+
6 rows in set (0.002 sec)
```

Compression monitoring

```
mysql-9.5.0 [performance_schema]> SHOW STATUS LIKE 'Bytes\_%';
+----------------+-------+
| Variable_name  | Value |
+----------------+-------+
| Bytes_received | 5730  |
| Bytes_sent     | 44749 |
+----------------+-------+
2 rows in set (0.002 sec)

mysql-9.5.0 [performance_schema]> pager wc -c
PAGER set to 'wc -c'
mysql-9.5.0 [performance_schema]> SELECT REPEAT('fosdem____',5000);
201049
1 row in set (0.001 sec)

mysql-9.5.0 [performance_schema]> pager
Default pager wasn't set, using stdout.
mysql-9.5.0 [performance_schema]> SHOW STATUS LIKE 'Bytes\_%';
+----------------+-------+
| Variable_name  | Value |
+----------------+-------+
| Bytes_received | 5817  |
| Bytes_sent     | 45011 |
+----------------+-------+
2 rows in set (0.002 sec)
```

So `wc` told us we received 201049 bytes. (includes table drawing etc, disable with `mysql -Nsr ...`)

The server told us it sent 45011 - 44749 = 262 bytes.

Compression monitoring

Systemtap can be useful:

```
probe process("/usr/bin/mysql").function("my_uncompress*").call {
    cl = @cast($complen, "size_t")
    printf("\nReceiving compressed packet with len=%u, complen=%u\n", $len, cl)
}
```

If you create trace files (`mysqld --debug`) you get some info:

```
...
T@18: | | | | | | | | note: Packet got longer on compression; Not compressed
...
T@18: | | | | | | | | note: Packet too short: Not compressed
...
T@18: | | | | | | | | note: Packet got longer on zstd compression; Not compressed
```

This will tell you when the payload was less than 50 bytes or when either zlib or zstd compressed data was longer than the original data.

Compression

Possible future work for compression

1. More connectors/drivers with zstd support
2. eBPF instrumentation for compression
3. Improved metrics in the server and/or client
4. Open source otel trace/span integration

Connection Attributes

Connection attributes are key-value pairs that are set per connection. This is useful for troubleshooting. They key-value pairs are metadata and are visible in `performance_schema`.

```
mysql-9.5.0 [(none)]> SELECT * FROM performance_schema.session_connect_attrs;
+----------------+-----------------+-------------+------------------+
| PROCESSLIST_ID | ATTR_NAME       | ATTR_VALUE  | ORDINAL_POSITION |
+----------------+-----------------+-------------+------------------+
|             20 | _pid            | 1526206     |                0 |
|             20 | _platform       | x86_64      |                1 |
|             20 | _os             | Linux       |                2 |
|             20 | _client_name    | libmysql    |                3 |
|             20 | os_user         | dvaneeden   |                4 |
|             20 | _client_version | 9.5.0       |                5 |
|             20 | program_name    | mysql       |                6 |
+----------------+-----------------+-------------+------------------+
7 rows in set (0.001 sec)
```

These are set by

- Connectors/Drivers (convention: the name starts with a _)
- Applications (`os_user` and `program_name` in this example)

Connection Attributes

Things you may want to set in your application.

- Application: (e.g. `report-generator`, `slackbot`, `paymentgw`)
- Application version (e.g. `myapp_commit: 214056e`).
- Environment (prod/acc/text/...)
- Job name (e.g. `daily-etl-iot`)
- Connection Pool: Thread/Connection ID and Pool name (when using multiple pools)
- Managed by (e.g. `managed_by_team: Chaos Creation Team 6`)

What you should not: Anything that changes over the connection lifetime. So do not set the name of the page you are rendering if you are rendering multiple pages with the same connection.

Query Attributes

- Query Attributes are per query.
- Each of them has a type, a flag a name and a value.
- Extra functionality is available via
  `INSTALL COMPONENT "file://component_query_attributes".`

Query Attributes

```
import os
import mysql.connector

c = mysql.connector.connect(
        host='127.0.0.1',
        user='root',
        ssl_disabled=True,
)

cur = c.cursor()
cur.add_attribute("proxy_user", "myuser")
cur.add_attribute("user_id", os.getuid())
cur.execute(
    """
    SELECT %s,
    mysql_query_attribute_string('proxy_user')
    """,
    ("hello",)
)
for row in cur:
    print(row)  # Output: ('hello', 'myuser')
cur.close()

c.close()
```

Sent with the query:

Type	Unsigned	Name	Value
254 (string)	false	proxy_user	myuser
8 (longlong)	false	user_id	1000

The `mysql_query_attribute_string()` function shows one of these. The same function can be use for the other attribute as well.

Query Attributes

Query Attributes can be used to send metadata about the query to the server. Examples: Webpage, End-user, Application file and line number, TenantID, ExperimentID, etc Basically the same as:

```
SELECT /* tenant_id=5,page=index.html,location=foo.php:123 */ foo FROM bar WHERE baz=123
```

But this is:

1. Can be parsed reliably
2. Has type information
3. Does not result in hard to read queries (e.g. slow query top-n)

Query Attributes

So with Query Attributes we can send a query and a set of key/type/flag/value's?

Query Attributes

So with Query Attributes we can send a query and a set of key/type/flag/value's?

Isn't that similar to prepared statements?

Query Attributes

So with Query Attributes we can send a query and a set of key/type/flag/value's?

Isn't that similar to prepared statements?

Yes, but prepared statements arguments don't have a name/key.

Query Attributes

So with Query Attributes we can send a query and a set of key/type/flag/value's?

Isn't that similar to prepared statements?

Yes, but prepared statements arguments don't have a name/key.

And prepared statements take more roundtrips.

Query Attributes

So with Query Attributes we can send a query and a set of key/type/flag/value's?

Isn't that similar to prepared statements?

Yes, but prepared statements arguments don't have a name/key.

And prepared statements take more roundtrips.

Many Connectors/Drivers implement client-side prepared statements to avoid the roundtrips.

Query Attributes

So with Query Attributes we can send a query and a set of key/type/flag/value's?

Isn't that similar to prepared statements?

Yes, but prepared statements arguments don't have a name/key.

And prepared statements take more roundtrips.

Many Connectors/Drivers implement client-side prepared statements to avoid the roundtrips.

But Query Attributes could form the bases for single roundtrip prepared statements.

Query Attributes

So with Query Attributes we can send a query and a set of key/type/flag/value's?

Isn't that similar to prepared statements?

Yes, but prepared statements arguments don't have a name/key.

And prepared statements take more roundtrips.

Many Connectors/Drivers implement client-side prepared statements to avoid the roundtrips.

But Query Attributes could form the bases for single roundtrip prepared statements.

This could simplify the connectors/drivers.

Query Attributes

So with Query Attributes we can send a query and a set of key/type/flag/value's?

Isn't that similar to prepared statements?

Yes, but prepared statements arguments don't have a name/key.

And prepared statements take more roundtrips.

Many Connectors/Drivers implement client-side prepared statements to avoid the roundtrips.

But Query Attributes could form the bases for single roundtrip prepared statements.

This could simplify the connectors/drivers.

This was already suggested in `https://archive.fosdem.org/2021/schedule/event/mysql_protocl/` by Joro Kodinov.

Query Attributes

So with Query Attributes we can send a query and a set of key/type/flag/value's?

Isn't that similar to prepared statements?

Yes, but prepared statements arguments don't have a name/key.

And prepared statements take more roundtrips.

Many Connectors/Drivers implement client-side prepared statements to avoid the roundtrips.

But Query Attributes could form the bases for single roundtrip prepared statements.

This could simplify the connectors/drivers.

This was already suggested in `https://archive.fosdem.org/2021/schedule/event/mysql_protocl/` by Joro Kodinov.

Note that X Protocol can also do prepared statements with a single roundtrip.

Secure connections

Secure connections: TLS or something local like UNIX domain socket. Practically required for `caching_sha2_password`.

MySQL does TLS with a STARTTLS like approach on the same port. When the ServerGreeting and Login both have `CLIENT_SSL` set in their capability flags, then both switch to TLS.

This opens up the risk for TLS-stripping if a man-in-the-middle would just unset the `CLIENT_SSL` bit.

By default MySQL clients and connectors don't check if the hostname it connects to matches the one in the certificate.

By default MySQL clients don't check if server certificate is signed by a known certificate authority.

Note that `caching_sha2_password` considers any TLS connection secure and sends the plain-text password over the connection.

Consider setting `--ssl-mode=VERIFY_IDENTITY`.

Pre-auth version info in Go

```go
package main

import (
        "bytes"
        "encoding/binary"
        "fmt"
        "net"
        "slices"
)

func main() {
        // Connect
        conn, _ := net.Dial("tcp", "127.0.0.1")

        // Read Header
        header := make([]byte, 4)
        conn.Read(header)

        // Decode packet length
        plen := binary.LittleEndian.Uint32(slices.Concat(header[:3], []byte{0x0}))

        // Read the rest of the packet
        greeting := make([]byte, plen)
        conn.Read(greeting)

        // Print the version
        version := bytes.Split(greeting[1:], []byte{0x0})[0]
        fmt.Printf("version: %s\n", version)
}
```

Pre-auth version info in Ada

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Streams; use Ada.Streams;
with Ada.Streams.Stream_IO;
with Interfaces;  use Interfaces;
with GNAT.Sockets;

procedure Mysqlversion is
   package Sockets renames GNAT.Sockets;
   Sock           : Sockets.Socket_Type;
   Addr           : Sockets.Sock_Addr_Type;
   Header         : Stream_Element_Array (1 .. 4);
   Last           : Stream_Element_Offset;
   PacketLength   : Unsigned_32;
   type Stream_Element_Array_Access is access all
        Stream_Element_Array;
   Greeting       : Stream_Element_Array_Access;
   Version_Start : constant Stream_Element_Offset := 2;
   Version_End    : Stream_Element_Offset;
```

```
begin
   Sockets.Create_Socket (Sock);
   Addr.Addr := Sockets.Inet_Addr ("127.0.0.1");
   Addr.Port := 3306;
   Sockets.Connect_Socket (Sock, Addr);
   Sockets.Receive_Socket (Sock, Header, Last);
   PacketLength :=
     Unsigned_32 (Header (1))
     or Shift_Left (Unsigned_32 (Header (2)), 8)
     or Shift_Left (Unsigned_32 (Header (3)), 16);
   Greeting :=
     new Stream_Element_Array (1 ..
           Stream_Element_Offset (PacketLength));
   Sockets.Receive_Socket (Sock, Greeting.all, Last);
   Version_End := Greeting'Last;
   for I in Version_Start .. Greeting'Last loop
      if Greeting (I) = 0 then
         Version_End := I - 1;
         exit;
      end if;
   end loop;
   Put ("version: ");
   for I in Version_Start .. Version_End loop
      Put (Character'Val (Greeting (I)));
   end loop;
   New_Line;
   Sockets.Close_Socket (Sock);
end Mysqlversion;
```

Pre-auth

```
$ ./mysqlversion 127.0.0.1:3306
version: 9.5.0
$ ./mysqlversion 127.0.0.1:4000
version: 8.0.11-TiDB-v8.5.5
$ ./mysqlversion 127.0.0.1:3307
version: 11.8.5-MariaDB-ubu2404-log
```

Ways to get the server version:

1. Initial packet (a.k.a. Server Greeting, HandshakeV10)
2. `SELECT VERSION()`
3. `SHOW VARIABLES LIKE 'version'`

What if they don't match? What if you use a proxy?

Pre-auth

1. This can be great if you want to scan for MySQL versions in your network.
2. Especially if you want to scan for forgotten or unauthorized servers.
3. This doens't require authentication, so evil hackers can do the same.
4. Checking for the version in your code? Make sure it uses the right source to get the version.
5. Want to avoid extra network checks for `SELECT VERSION()`? That's possible.

X Protocol

1. Can do prepared statements in a single roundtrip.
2. Can do pipelining

Cloning

Cloning via `CLONE INSTANCE FROM...` uses the MySQL Protocol.

Note that this is InnoDB only.

Cloning can not use X Protocol.

Cloning can use many existing features: Authentication, TLS, etc.

Replication

1. **Heartbeat**: Heartbeat Events are in the binlog stream, but not in the binlog files. They serve two purposes:
   - Replace ignored events in the stream (already seen GTIDs) and keep the same positions.
   - Improve detection of failed connections.
2. **Semisync**: Wait for OK on Commit until there are n acknowledgements. Acknowledger doesn't have to be a replica...

MariaDB enhancements

CLIENT_PROGRESS

```
MariaDB [test]> ALTER TABLE t1 ENGINE=aria;
Stage: 2 of 4 'Repair by sorting'   34.6% of stage done
```

This is similar to the Notice frames in MySQL X Protocol.

timeout reason

```
sql> SET SESSION autocommit=ON;
ERROR: 4031 (HY000): The client was disconnected by the server because of inactivity. See wait_timeout and
      interactive_timeout for configuring this behavior.

sql>
```

Questions?

Thank you!

Daniel.van.Eeden@pingcap.com

https://gitlab.com/wireshark/wireshark/
https://www.wireshark.org/
https://dev.mysql.com/doc/dev/mysql-server/latest/PAGE_PROTOCOL.html