MySQL Network Protocol: A walkthrough

Daniël van Eeden

PingCAP

FOSDEM, February 2025

Agenda

- **Introduction**
- **Connection Setup**
  - Handshake
  - TLS & Compression
  - Authentication
- **Command Phase**
  - Queries & resultsets
  - Prepared statements
- **Replication**

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

Why you should care about the MySQL Protocol

- **Troubleshooting**: protocol related bugs happen
- **Performance**: reduce roundtrips
- **Cost**: reduce bandwidth cost
- **Contribute**: add protocol support to new languages, tools, etc

Definitions & Scope

The MySQL Protocol is the protocol that is used between a client application and a MySQL or MySQL Compatible server.

This is nowadays known as the "Classic" protocol. There is also a "X Protocol", which is newer and based on protobuf, but this is out of scope of this presentation.

The protocol can be used over a UNIX socket or over TCP.

Protocol information

The documentation for the protocol is made available by Oracle/MySQL on `https://dev.mysql.com/doc/dev/mysql-server/latest/PAGE_PROTOCOL.html` and is using doxygen.

The MySQL Protocol does not have a formal specification. But it is documented. And MySQL Server, Client, Connectors, etc can be seen as refrence implementations.

So when working with the protocol it is often useful to look at the documentation, but often you have to look at network traces with tools like Wireshark as well. And in a few cases it helps to look at the MySQL server and client source code.

Who implements the protocol?

- MySQL Server
- TiDB
- Vitess (`vtgate`)
- MariaDB Server
- ProxySQL (including the admin interface)

... and much more ...

- MySQL Connector/J
- PHP with `mysqlnd`
- MySQL Connector/Python
- MySQL C API (libmysqlclient)
  - MySQL Client
  - DBD::mysql (Perl)
  - `mysqlclient` (Python)
- Wireshark
- `go-mysql` (both client and server)
- `go-sql-driver`
- ClickHouse

Using Wireshark with MySQL

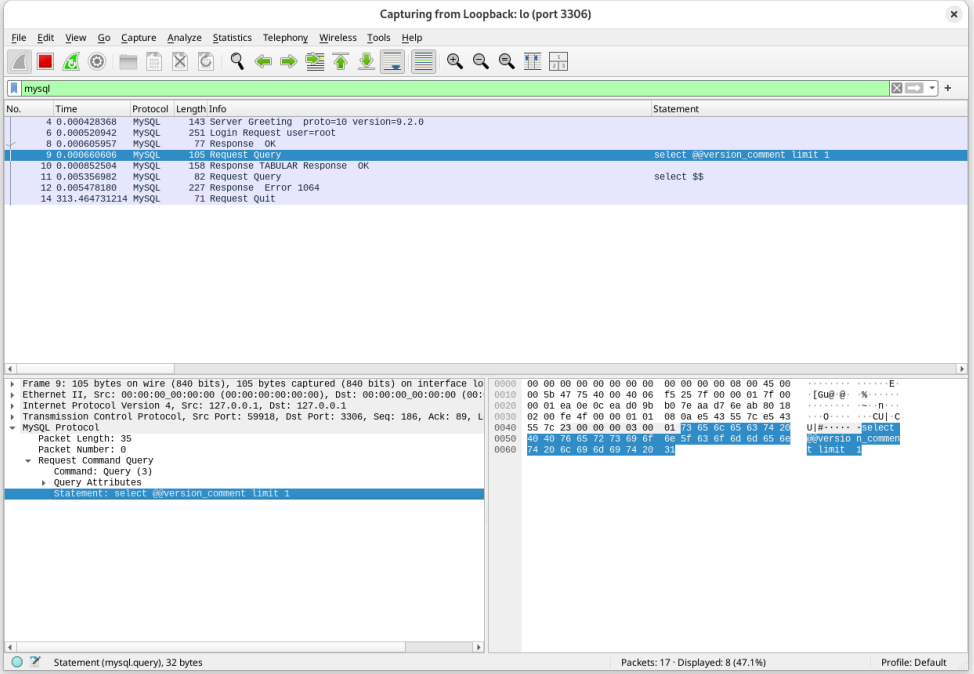
Capture traffic and analyze

- Capture pcap file with `tcpdump` and analyze with Wireshark.
- Capture and analyze with Wireshark directly.
- Or use `tshark` (Wireshark CLI), especially with automation.
- Set `-s 65535` with older `tcpdump` versions to capture complete packets.

Hints for the MySQL side

- Use `-h 127.0.0.1`, not `localhost` to avoid using a UNIX socket.
- Use `--ssl-mode=DISABLED`
- Use port 3306, or use *Decode As ...* in Wireshark to set the protocol
- Use `mysql` as display filter to hide TCP info.
- Start capturing from the beginning of the connection to allow Wireshark to see what protocol features are in use.

Wireshark

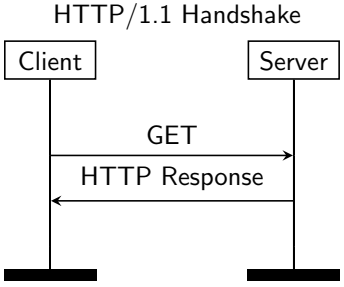

Capturing from Loopback: lo (port 3306)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
mysql
No. Time Protocol Length Info Statement
4 0.000428368 MySQL 143 Server Greeting proto=10 version=9.2.0
6 0.000520942 MySQL 251 Login Request user=root
8 0.000605957 MySQL 77 Response OK
9 0.000660606 MySQL 105 Request Query select @@version_comment limit 1
10 0.000852504 MySQL 158 Response TABULAR Response OK
11 0.005356982 MySQL 82 Request Query select $$
12 0.005478180 MySQL 227 Response Error 1064
14 313.464731214 MySQL 71 Request Quit
Frame 9: 105 bytes on wire (840 bits), 105 bytes captured (840 bits) on interface lo
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 59918, Dst Port: 3306, Seq: 186, Ack: 89, L
MySQL Protocol
Packet Length: 35
Packet Number: 0
Request Command Query
Command: Query (3)
Query Attributes
Statement: select @@version_comment limit 1
0000 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00 ....E.
0010 00 5b 47 75 40 00 40 06 f5 25 7f 00 00 01 7f 00 .[Gu@.@. .%.....
0020 00 01 ea 0e 0c ea d0 9b b0 7e aa d7 6e ab 80 18 ..... .~..n...
0030 02 00 fe 4f 00 00 01 01 08 0a e5 43 55 7c e5 43 ...O.... ...CU|.C
0040 55 7c 23 00 00 00 03 00 01 73 65 6c 65 63 74 20 U|#..... .select
0050 40 40 76 65 72 73 69 6f 6e 5f 63 6f 6d 6d 65 6e @@versio n_comen
0060 74 20 6c 69 6d 69 74 20 31 t limit 1
Statement (mysql.query), 32 bytes
Packets: 17 · Displayed: 8 (47.1%)
Profile: Default

Connection Setup

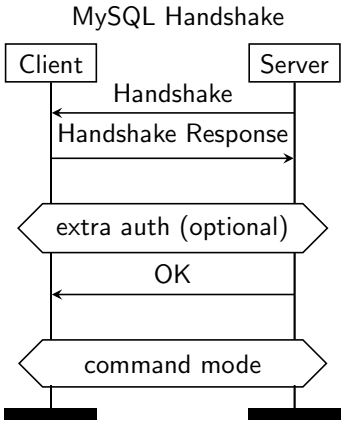
Who sends the first message?
With HTTP the client sends the first message. With MySQL the server sends the first message.

Versioning
The Classic protocol has versions: V9 and V10. And V10 is used since MySQL 3.21.0 (1998). So basically the version number is useless.
The MySQL protocol instead uses capability flags to indicate features.

Handshake



Handshake



The TCP connection is established by the client, but it is the server that sends the first message.

This is more complicated than HTTP because it does include authentication.

In case of authentication failure `ERR` is returend instead of `OK` and the connection is closed.

Handshake

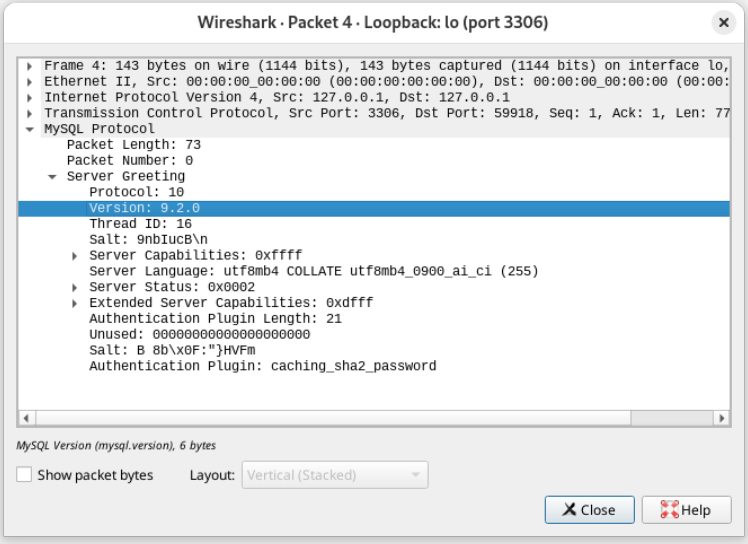
HandshakeV10 (a.k.a. Server Greeting)
This contains:

1. Server Version (string)
2. Connection ID
3. Capability flags (4 bytes)
4. Authentication Scramble

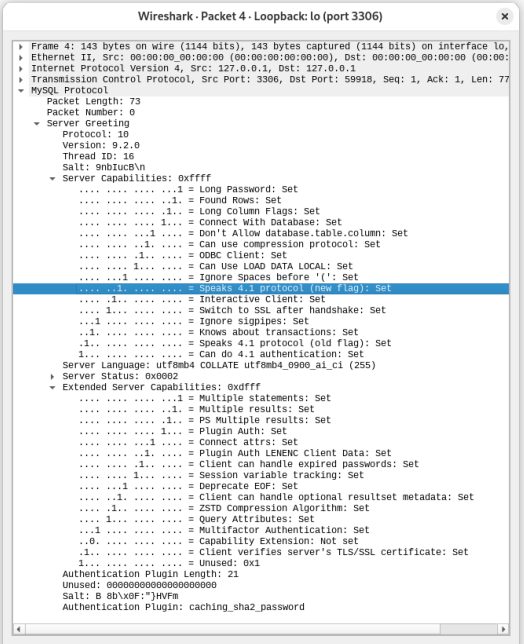
Capabilities

```
CLIENT_LONG_PASSWORD
CLIENT_FOUND_ROWS
CLIENT_LONG_FLAG
CLIENT_CONNECT_WITH_DB
CLIENT_NO_SCHEMA
CLIENT_COMPRESS
CLIENT_ODBC
CLIENT_LOCAL_FILES
CLIENT_IGNORE_SPACE
CLIENT_PROTOCOL_41
CLIENT_INTERACTIVE
CLIENT_SSL
CLIENT_IGNORE_SIGPIPE
CLIENT_TRANSACTIONS
CLIENT_RESERVED
CLIENT_RESERVED2
CLIENT_MULTI_STATEMENTS
CLIENT_MULTI_RESULTS
CLIENT_PS_MULTI_RESULTS
CLIENT_PLUGIN_AUTH
CLIENT_CONNECT_ATTRS
CLIENT_PLUGIN_AUTH_LENENC_CLIENT_DATA
CLIENT_CAN_HANDLE_EXPIRED_PASSWORDS
CLIENT_SESSION_TRACK
CLIENT_DEPRECATE_EOF
CLIENT_OPTIONAL_RESULTSET_METADATA
CLIENT_ZSTD_COMPRESSION_ALGORITHM
CLIENT_QUERY_ATTRIBUTES
CLIENT_CAPABILITY_EXTENSION
CLIENT_SSL_VERIFY_SERVER_CERT
CLIENT_REMEMBER_OPTIONS
CLIENT_MULTI_QUERIES
```

Server Greeting



Capabilies


Wireshark · Packet 4 · Loopback: lo (port 3306)
Frame 4: 143 bytes on wire (1144 bits), 143 bytes captured (1144 bits) on interface lo,
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 3306, Dst Port: 59918, Seq: 1, Ack: 1, Len: 77
MySQL Protocol
Packet Length: 73
Packet Number: 0
Server Greeting
Protocol: 10
Version: 9.2.0
Thread ID: 16
Salt: 9nBIuc8\n
Server Capabilities: 0xffff
....1 = Long Password: Set
....1. = Found Rows: Set
.... .1.. = Long Column Flags: Set
.... 1... = Connect With Database: Set
....1 .... = Don't Allow database.table.column: Set
....1. .... = Can use compression protocol: Set
.... .1.. .... = ODBC Client: Set
.... 1... .... = Can Use LOAD DATA LOCAL: Set
....1 .... = Ignore Spaces before '(': Set
.... ..1. .... = Speaks 4.1 protocol (new flag): Set
.... .1.. .... = Interactive Client: Set
...1 .... = Switch to SSL after handshake: Set
..1. .... = Ignore sigpipes: Set
.1.. .... = Knows about transactions: Set
1... .... = Speaks 4.1 protocol (old flag): Set
Server Language: utf8mb4 COLLATE utf8mb4_0900_ai_ci (255)
Server Status: 0x0002
Extended Server Capabilities: 0xdfff
....1 = Multiple statements: Set
....1. = Multiple results: Set
.... .1.. = PS Multiple results: Set
.... 1... = Plugin Auth: Set
....1 .... = Connect attrs: Set
....1. .... = Plugin Auth LENENC Client Data: Set
.... .1.. .... = Client can handle expired passwords: Set
.... 1... .... = Session variable tracking: Set
....1 .... = Deprecate EOF: Set
.... ..1. .... = Client can handle optional resultset metadata: Set
.... .1.. .... = ZSTD Compression Algorithm: Set
.... 1... .... = Query Attributes: Set
...1 .... = Multifactor Authentication: Set
..0. .... = Capability Extension: Not set
.1.. .... = Client verifies server's TLS/SSL certificate: Set
1... .... = Unused: 0x1
Authentication Plugin Length: 21
Unused: 0000000000000000
Salt: B 8b\x0F:"}HVFm
Authentication Plugin: caching_sha2_password

Handshake Response

HandshakeResponse41 (a.k.a. Client Login)
This contains:

1. username
2. authentication data
3. database (if `CLIENT_CONNECT_WITH_DB` is set)
4. collation
5. max packet size
6. client plugin (if `CLIENT_PLUGIN_AUTH` is set)
7. Connection Attributes

Connection attributes

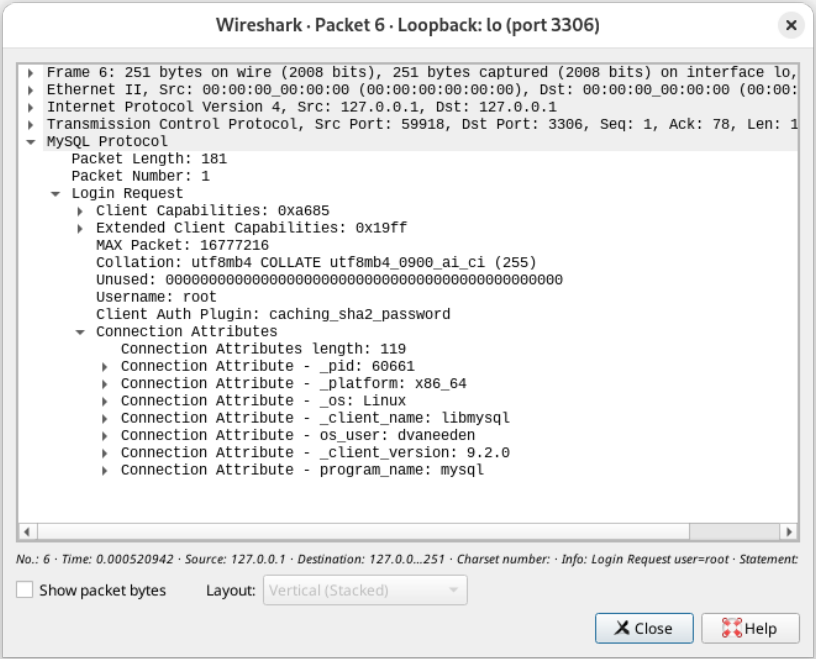
Connection Attributes are key-value pairs that the client sends to the server.
Some are set by connectors, some are set by applications.

Connection attributes

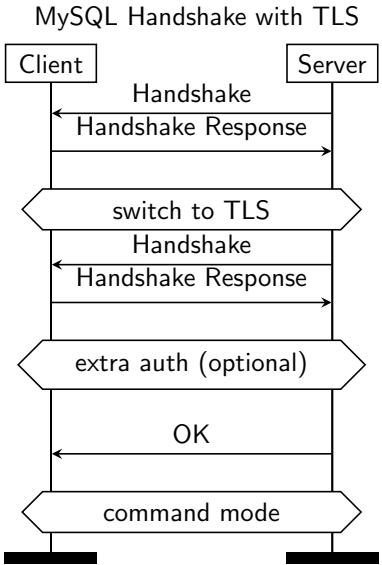
There is a performance_schema table that shows these.

```
mysql> TABLE performance_schema.session_connect_attrs;
+----------------+-----------------+-------------+------------------+
| PROCESSLIST_ID | ATTR_NAME       | ATTR_VALUE | ORDINAL_POSITION |
+----------------+-----------------+-------------+------------------+
|             27 | _pid            | 360214      |                0 |
|             27 | _platform       | x86_64      |                1 |
|             27 | _os             | Linux       |                2 |
|             27 | _client_name    | libmysql    |                3 |
|             27 | os_user         | dvaneeden   |                4 |
|             27 | _client_version | 9.1.0       |                5 |
|             27 | program_name    | mysql       |                6 |
+----------------+-----------------+-------------+------------------+
7 rows in set (0.01 sec)
```

Login


Wireshark · Packet 6 · Loopback: lo (port 3306)
Frame 6: 251 bytes on wire (2008 bits), 251 bytes captured (2008 bits) on interface lo,
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 59918, Dst Port: 3306, Seq: 1, Ack: 78, Len: 1
MySQL Protocol
Packet Length: 181
Packet Number: 1
Login Request
Client Capabilities: 0xa685
Extended Client Capabilities: 0x19ff
MAX Packet: 16777216
Collation: utf8mb4 COLLATE utf8mb4_0900_ai_ci (255)
Unused: 00000000000000000000000000000000000000000000000000
Username: root
Client Auth Plugin: caching_sha2_password
Connection Attributes
Connection Attributes length: 119
Connection Attribute - _pid: 60661
Connection Attribute - _platform: x86_64
Connection Attribute - _os: Linux
Connection Attribute - _client_name: libmysql
Connection Attribute - os_user: dvaneeden
Connection Attribute - _client_version: 9.2.0
Connection Attribute - program_name: mysql
No.: 6 · Time: 0.000520942 · Source: 127.0.0.1 · Destination: 127.0.0...251 · Charset number: · Info: Login Request user=root · Statement:
Show packet bytes
Layout:
Vertical (Stacked)
Close
Help

TLS



With HTTP and HTTPS the server listens on two ports (80 and 443) and port 443 directly starts the TLS negotiation.

With MySQL both secure (TLS) and insecure connections use the same port (3306). The connection always starts without TLS and then switches to TLS if both client and server have `CLIENT_SSL` set.

OpenSSL knows how to do this if you use `openssl s_client -connect 127.0.0.1:3306 -starttls mysql`.

Wireshark and TLS

Did you know Wireshark can decode TLS traffic if you give it the private key and select a suitable ciphersuite?

Some ciphersuites use Diffie-Hellman key exchange to get a session key, this requires export of the session key if you want to decode the traffic.

Compression

zlib based compression has been in the protocol since MySQL 3.22.3.
zstandard based compression was added in 8.0.18.
Protocol flags:

- `CLIENT_COMPRESS`
- `CLIENT_ZSTD_COMPRESSION_ALGORITHM`

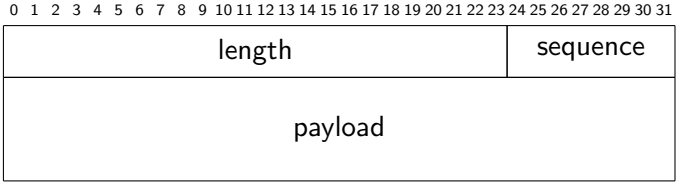
The HandshakeResponse41 contains the zstd compression level if the flag is set.

zlib or zstd?

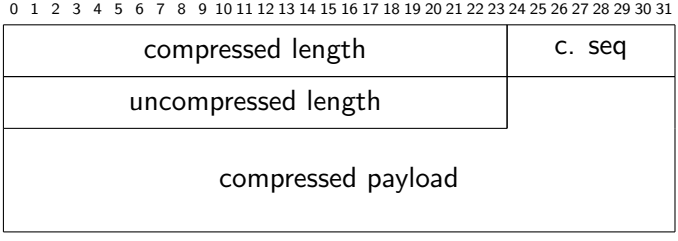
```
$ mysql -u root -h 127.0.0.1 --compression-algorithms=zstd \
> -e "SHOW STATUS LIKE 'Compression_algorithm'"
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Compression_algorithm | zstd  |
+-----------------------+-------+

$ mysql -u root -h 127.0.0.1 --compression-algorithms=zlib \
> -e "SHOW STATUS LIKE 'Compression_algorithm'"
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Compression_algorithm | zlib  |
+-----------------------+-------+

$ mysql -u root -h 127.0.0.1 --compression-algorithms=zlib,zstd \
> -e "SHOW STATUS LIKE 'Compression_algorithm'"
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Compression_algorithm | zlib  |
+-----------------------+-------+

$ mysql -u root -h 127.0.0.1 --compression-algorithms=zstd,zlib \
> -e "SHOW STATUS LIKE 'Compression_algorithm'"
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Compression_algorithm | zlib  |
+-----------------------+-------+
```

Compression Details

Regular MySQL Packet:



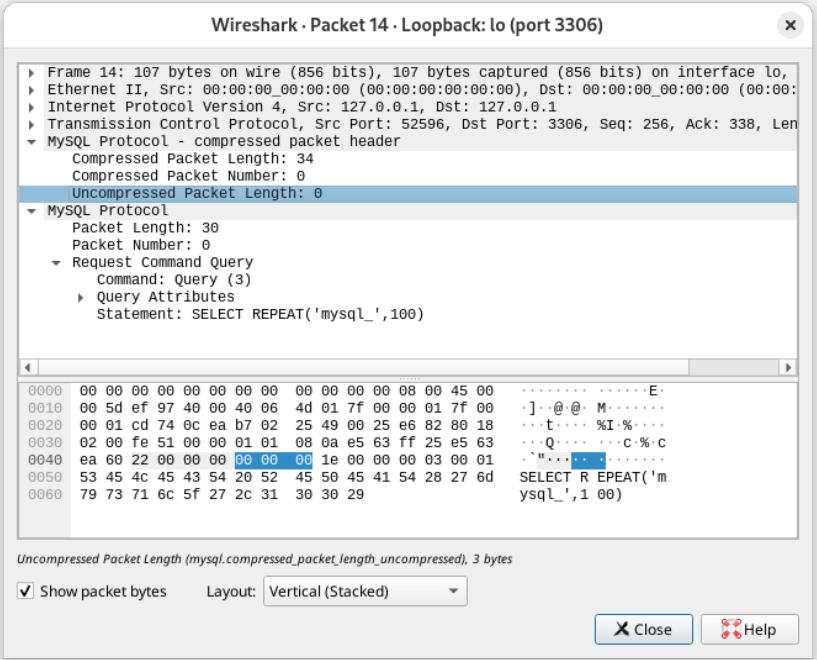
Compressed MySQL Packet:



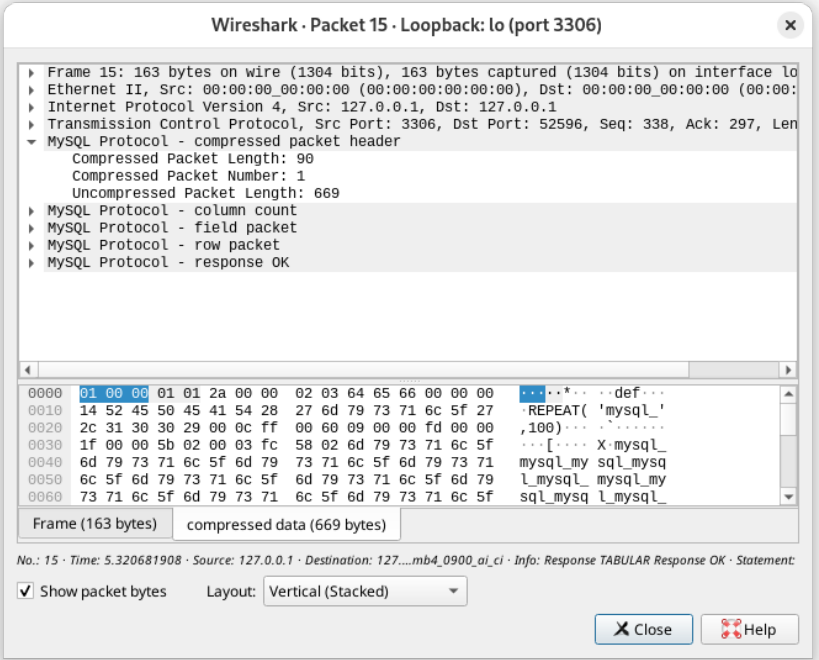
Payload is not compressed if **uncompressed length** is set to 0. Compression is done per packet.

Some operations may be split into multiple packets.

Compression



Compression



Authentication

- The server sends a default authentication method in the Server Greeting.
- The client then *may* then use this or any other method to authenticate.
- Then the server sends a AuthSwitch back if the account needs a different auth method.
- Eventually the response is a `OK` packet or `ERR` packet.
- Authentication methods are free to add extra roundtrips.

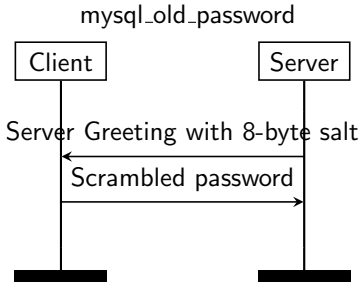
Note that mysql does do authentication based on accounts, this is the combination of username and host. So `user@somehost` is a different account to user@otherhost and can have a different password, permissions, etc.

Authentication

When migrating to a newer authentication method the server announced default and the client default might be wrong for a specific account. This increases the AuthSwitch roundtrips. Especially for Perl, PHP, etc this might be costly as they tend not to use persistent connections.

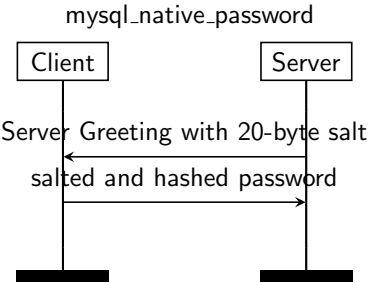
mysql_old_password

Default prior to MySQL 4.1.
Deprecated in MySQL 5.6.
16 byte hash.
The `secure_auth` setting was used to disable this.
The scramble, which is a hash of the password which is then XORed with the salt from the server, makes it "safe" to use this on an insecure connection. This doesn't protect statements later on that might include the password or password hash.
As the hashes are stored unsalted, this allows the use of rainbow tables.



mysql_native_password

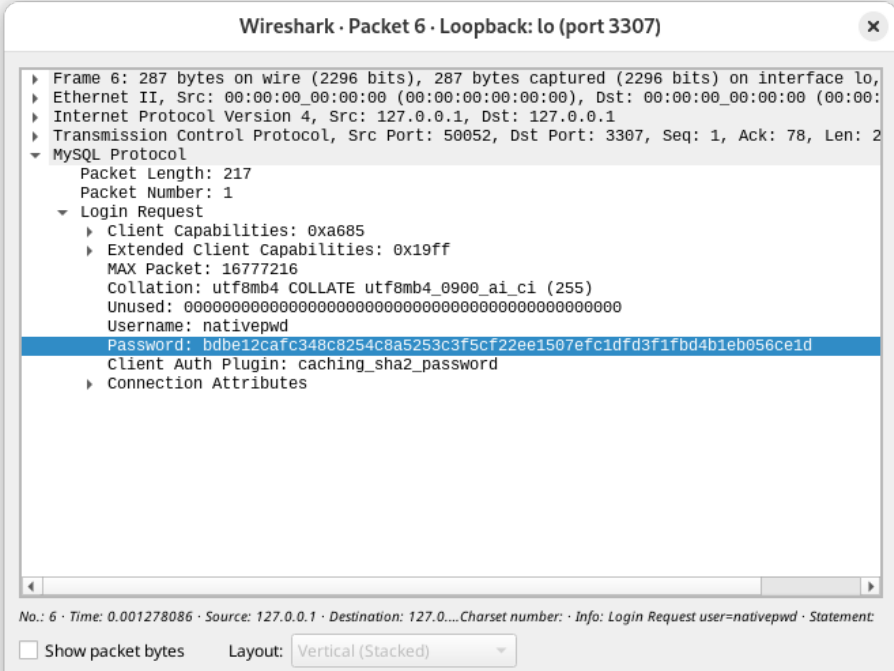
Introduced in MySQL 4.1 as default.
Deprecated as of MySQL 8.0.34.
Removed in MySQL 9.0.
Uses SHA1 hashes (standardised, 160-bit (20 byte))
Server stores salted hashes.



Native Auth

No.	Time	Protocol	Length	Info
4	0.000836252	MySQL	143	Server Greeting proto=10 version=8.4.3
6	0.001278086	MySQL	287	Login Request user=nativepwd
8	0.001599987	MySQL	114	Auth Switch Request
9	0.001898867	MySQL	90	Auth Switch Response
10	0.002286893	MySQL	77	Response OK
11	0.002360381	MySQL	105	Request Query
12	0.002919810	MySQL	158	Response TABULAR Response OK
13	0.007432589	MySQL	82	Request Query
14	0.007661946	MySQL	227	Response Error 1064

Native Auth


Wireshark · Packet 6 · Loopback: lo (port 3307)
Frame 6: 287 bytes on wire (2296 bits), 287 bytes captured (2296 bits) on interface lo,
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 50052, Dst Port: 3307, Seq: 1, Ack: 78, Len: 2
MySQL Protocol
Packet Length: 217
Packet Number: 1
Login Request
Client Capabilities: 0xa685
Extended Client Capabilities: 0x19ff
MAX Packet: 16777216
Collation: utf8mb4 COLLATE utf8mb4_0900_ai_ci (255)
Unused: 0000000000000000000000000000000000000000000000000000
Username: nativepwd
Password: bdbe12cafc348c8254c8a5253c3f5cf22ee1507efc1dfd3f1fbd4b1eb056ce1d
Client Auth Plugin: caching_sha2_password
Connection Attributes
No.: 6 · Time: 0.001278086 · Source: 127.0.0.1 · Destination: 127.0....Charset number: · Info: Login Request user=nativepwd · Statement:
Show packet bytes
Layout:
Vertical (Stacked)

Native Auth

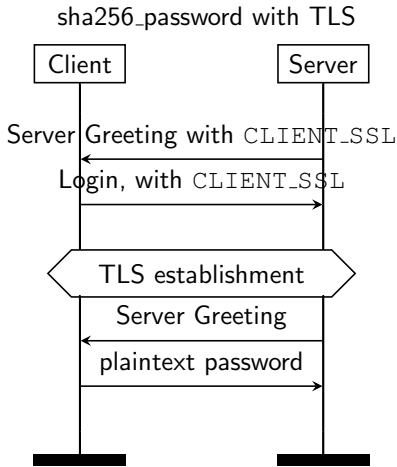

Wireshark · Packet 8 · Loopback: lo (port 3307)
Frame 8: 114 bytes on wire (912 bits), 114 bytes captured (912 bits) on interface lo, i
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 3307, Dst Port: 50052, Seq: 78, Ack: 222, Len:
MySQL Protocol - authentication switch request
Packet Length: 44
Packet Number: 2
Response Code: EOF Packet (0xfe)
EOF marker: 254
Auth Method Name: mysql_native_password
Auth Method Data: 2c7f5f7d6d3177502e4a4e261e7c553a604a6a2b00
No.: 8 · Time: 0.001599987 · Source: 127.0.0.1 · Destination: 127.0.0...th: 114 · Charset number: · Info: Auth Switch Request · Statement:
Show packet bytes
Layout:
Vertical (Stacked)

Native Auth


Wireshark · Packet 9 · Loopback: lo (port 3307)
Frame 9: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface lo, id
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 50052, Dst Port: 3307, Seq: 222, Ack: 126, Len
MySQL Protocol
Packet Length: 20
Packet Number: 3
Auth Method Data: 55452c69ce04be3ae877547651799edd3b15d86c
No.: 9 · Time: 0.001898867 · Source: 127.0.0.1 · Destination: 127.0.0...th: 90 · Charset number: · Info: Auth Switch Response · Statement:
Show packet bytes
Layout:
Vertical (Stacked)

sha256_password

Uses SHA-256 (part of SHA2 family).
Uses a 256-bit (32-byte) hash.
Requires TLS or a RSA keypair to keep the password secure during authentication.
Deprecated as of MySQL 8.0.16.



RSA keypair

Automatically generated.
How to distribute the public keys of your server to the clients securely is up to the user.
So is key rollover, etc.
Only protects the authentication part of the connection.
There is the `--get-server-public-key` option to request the public key over a MySQL connection. But this connection is not secure and open to a man-in-the-middle attack.
A TOFU (Trust On First Use) would have somewhat improved this.
No hostname validation.
TLS takes care of all of this.

MySQL and TLS

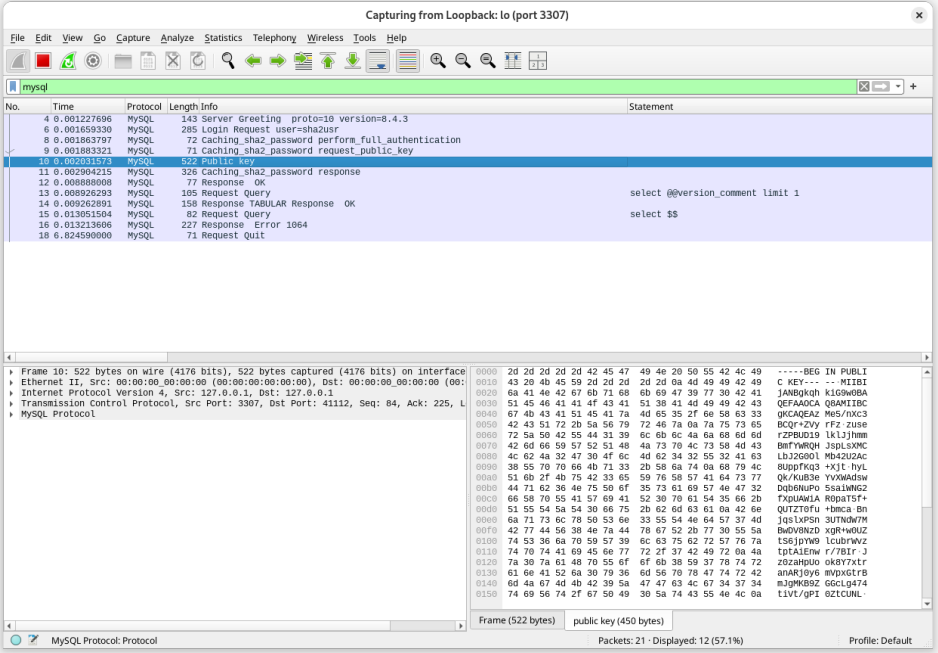
- MySQL never supported SSLv2 or SSLv3
- In most cases SSL doesn't mean SSL, it means TLS...
- MySQL used to have YaSSL or OpenSSL, now it only has OpenSSL.
- YaSSL didn't support TLSv1.2 an was slower than OpenSSL
- OpenSSL and GPL didn't mix well.
- MySQL doesn't do hostname validation (`--ssl-mode=VALIDATE_IDENTITY`) by default.
- MySQL doesn't do CA validation (`--ssl-mode=VALIDATE_CA`)by default either.
- The default doesn't protect against man-in-the-middle attacks.
- Both `caching_sha2_password` and `sha256_password` send the plaintext password over the TLS connection.
- So by default MySQL clients/connectors will happily send the password to the man in the middle...

caching_sha2_password

Uses SHA-256 (part of SHA2 family).
Uses a 256-bit (32-byte) hash.
Requires TLS or a RSA keypair to keep the password secure during authentication.
Has these authentication paths:

- **Quick**: When the password is empty.
- **Fast**: Validate the (cached) scramble.
- **Full**: No cached scramble. Requires TLS or RSA keypair.

Caching SHA2 Password



auth_socket

Only supports UNIX domain sockets, so no TCP connections.
Uses `SO_PEERCRED` to get the username of the other side of the socket.
Relies on OS authentication.
Usually the MySQL username and UNIX username should match. But does support mapping.

Multi Factor Authentication

AuthNextFactor packets are sent by the server after each method.

mysql_clear_password

Only a client-side plugin.
Useful with PAM and simple LDAP authentiction.

Queries & resultsets

- Text Protocol: `COM_QUERY`
- Binary Protocol: Prepared statements

COM_QUERY

```
COM_QUERY
<query>
```

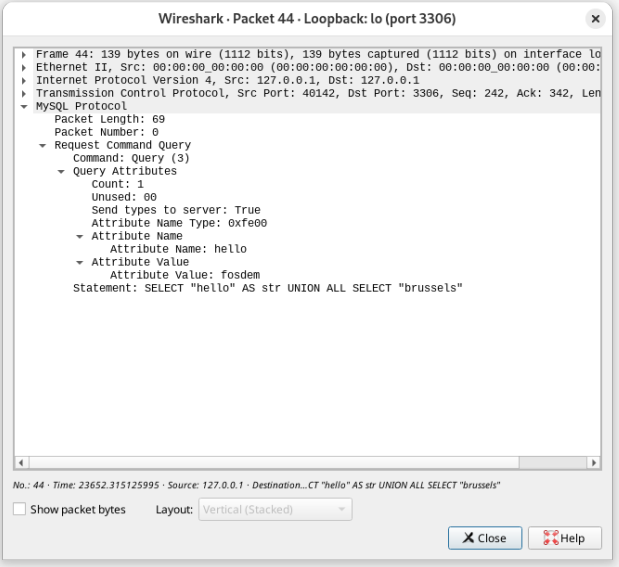
COM_QUERY, with Query Attributes

```
COM_QUERY
if QUERY_ATTRIBUTES {
        <parameter_count>
        <parameter_set_count>  // always 1
        if parameter_count > 0 {
                <null_bitmap>
                <new_params_bind_flag> // always 1
                if new_params_bind_flag >0 {
                        for each param {
                                <param_type>
                                <param_flag>
                                <parameter_name>
                        }
                        for each param {
                                <param_value>
                        }
                }
        }
}
<query>
```

Query Attributes

```
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.execute(
    "SELECT %s, mysql_query_attribute_string('proxy_user')",
    ("hello",)
)
for row in cur:
    print(row)  # Output: ('hello', 'myuser')
cur.close()

c.close()
```

Query


Wireshark · Packet 44 · Loopback: lo (port 3306)
Frame 44: 139 bytes on wire (1112 bits), 139 bytes captured (1112 bits) on interface lo
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 40142, Dst Port: 3306, Seq: 242, Ack: 342, Len
MySQL Protocol
Packet Length: 69
Packet Number: 0
Request Command Query
Command: Query (3)
Query Attributes
Count: 1
Unused: 00
Send types to server: True
Attribute Name Type: 0xfe00
Attribute Name
Attribute Name: hello
Attribute Value
Attribute Value: fosdem
Statement: SELECT "hello" AS str UNION ALL SELECT "brussels"
No.: 44 · Time: 23652.315125995 · Source: 127.0.0.1 · Destination...CT "hello" AS str UNION ALL SELECT "brussels"
Show packet bytes
Layout:
Vertical (Stacked)
Close
Help

Resultsets

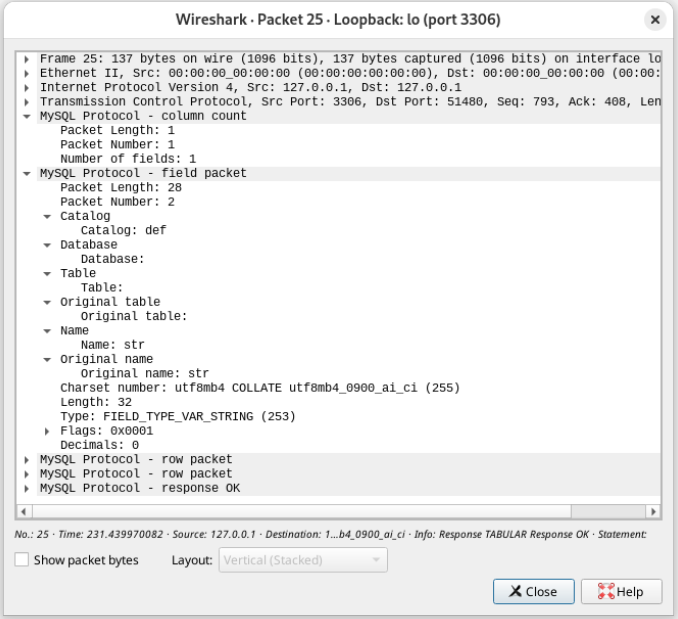
Result is one of these:

- ERR-packet
- OK-packet
- `LOCAL INFILE` request
- Text resultset

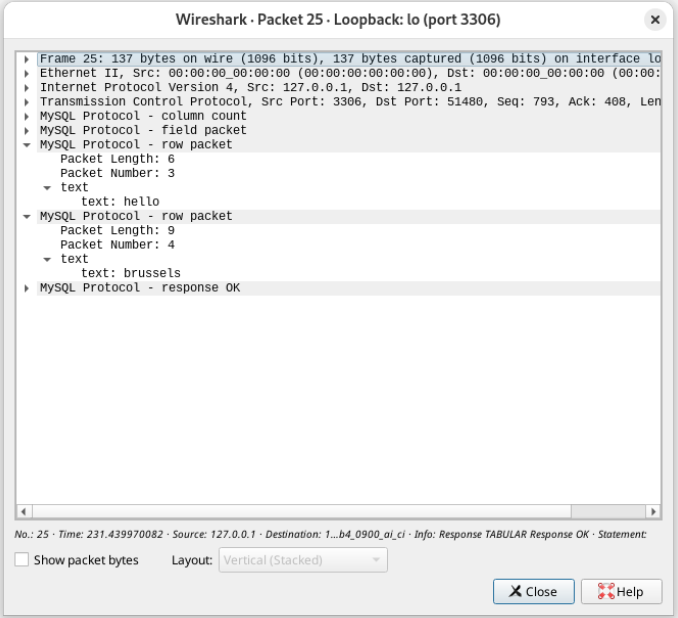
Text Resultset

- Field count
- List of fields
- Intermediate EOF
- List of rows

Resultset


Wireshark · Packet 25 · Loopback: lo (port 3306)
Frame 25: 137 bytes on wire (1096 bits), 137 bytes captured (1096 bits) on interface lo
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 3306, Dst Port: 51480, Seq: 793, Ack: 408, Len
MySQL Protocol - column count
Packet Length: 1
Packet Number: 1
Number of fields: 1
MySQL Protocol - field packet
Packet Length: 28
Packet Number: 2
Catalog
Catalog: def
Database
Database:
Table
Table:
Original table
Original table:
Name
Name: str
Original name
Original name: str
Charset number: utf8mb4 COLLATE utf8mb4_0900_ai_ci (255)
Length: 32
Type: FIELD_TYPE_VAR_STRING (253)
Flags: 0x0001
Decimals: 0
MySQL Protocol - row packet
MySQL Protocol - row packet
MySQL Protocol - response OK
No.: 25 · Time: 231.439970082 · Source: 127.0.0.1 · Destination: 1...b4_0900_ai_ci · Info: Response TABULAR Response OK · Statement:
Show packet bytes
Layout:
Vertical (Stacked)
Close
Help

Resultset


Wireshark · Packet 25 · Loopback: lo (port 3306)
Frame 25: 137 bytes on wire (1096 bits), 137 bytes captured (1096 bits) on interface lo
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 3306, Dst Port: 51480, Seq: 793, Ack: 408, Len
MySQL Protocol - column count
MySQL Protocol - field packet
MySQL Protocol - row packet
Packet Length: 6
Packet Number: 3
text
text: hello
MySQL Protocol - row packet
Packet Length: 9
Packet Number: 4
text
text: brussels
MySQL Protocol - response OK
No.: 25 · Time: 231.439970082 · Source: 127.0.0.1 · Destination: 1...b4_0900_ai_ci · Info: Response TABULAR Response OK · Statement:
Show packet bytes
Layout:
Vertical (Stacked)
Close
Help

Prepared statements

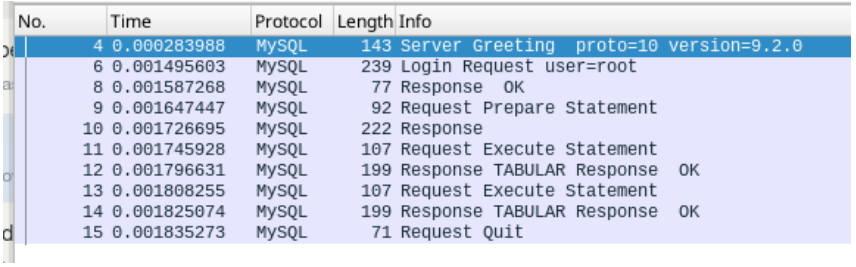
- Prepare: Just `COM_STMT_PREPARE` with the query.
- Response: Metadata about number of parameters, etc
- Execute: `COM_STMT_EXECUTE` with the ID of the prepare and a list of parameters.

Note that some drivers might do a client side prepare emulation to avoid the roundtrip.

Prepared Statements

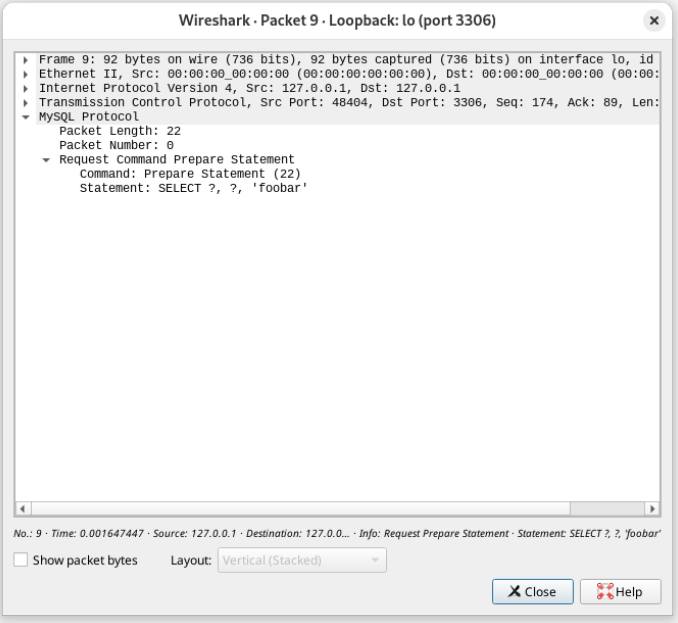
```perl
#!/bin/perl
use v5.40;
use DBI;

my $dsn = 'dbi:mysql:host=127.0.0.1;mysql_server_prepare=false';
my $dbh = DBI->connect($dsn, 'root', '');
my $sth = $dbh->prepare("SELECT ?, ?, 'foobar'");
$sth->execute(123, "one-two-three");

$sth->bind_param(1, 456);
$sth->bind_param(2, "four-five-six");
$sth->execute;

$sth->finish;
$dbh->disconnect;
```

Prepared Statements

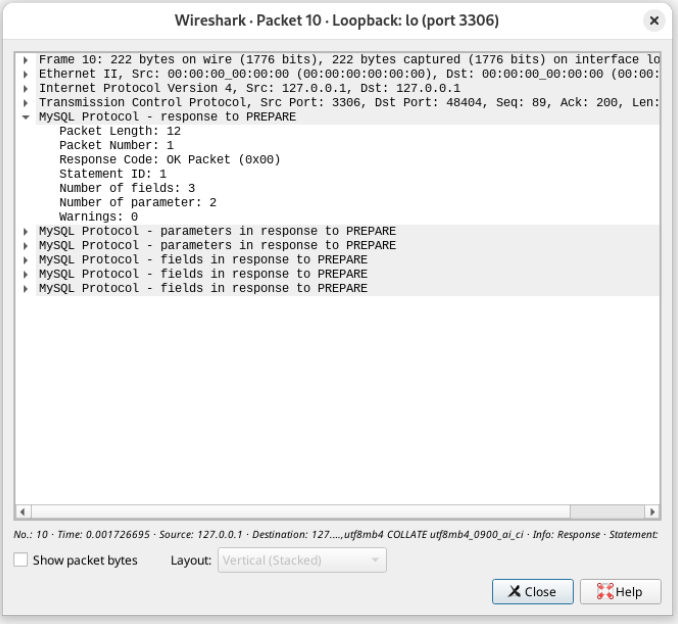


No.	Time	Protocol	Length	Info
4	0.000283988	MySQL	143	Server Greeting proto=10 version=9.2.0
6	0.001495603	MySQL	239	Login Request user=root
8	0.001587268	MySQL	77	Response OK
9	0.001647447	MySQL	92	Request Prepare Statement
10	0.001726695	MySQL	222	Response
11	0.001745928	MySQL	107	Request Execute Statement
12	0.001796631	MySQL	199	Response TABULAR Response OK
13	0.001808255	MySQL	107	Request Execute Statement
14	0.001825074	MySQL	199	Response TABULAR Response OK
15	0.001835273	MySQL	71	Request Quit

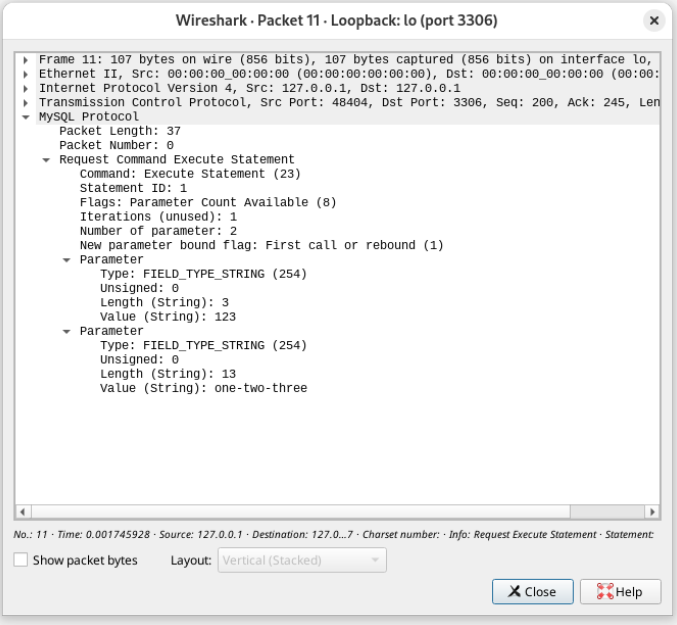
Prepared Statements



Prepared Statements



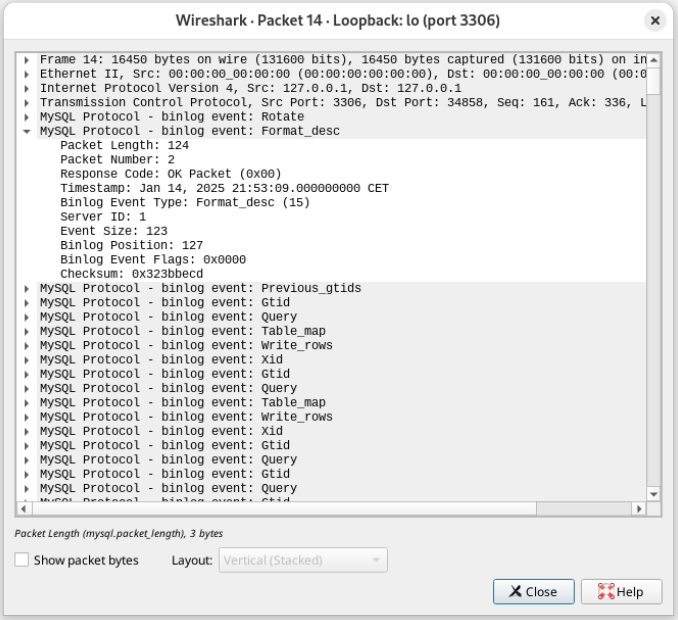
Prepared Statements



Replication

- Connection setup as usual
- Optional: `COM_REGISTER_REPLICA` to register the relica (for `SHOW REPLICAS`)
- `COM_BINLOG_DUMP` or `COM_BINLOG_DUMP_GTID` to start binlog stream.
- binlog stream is *mostly* identical to the binlog files on disk
- binlog stream consistes of a series of events.

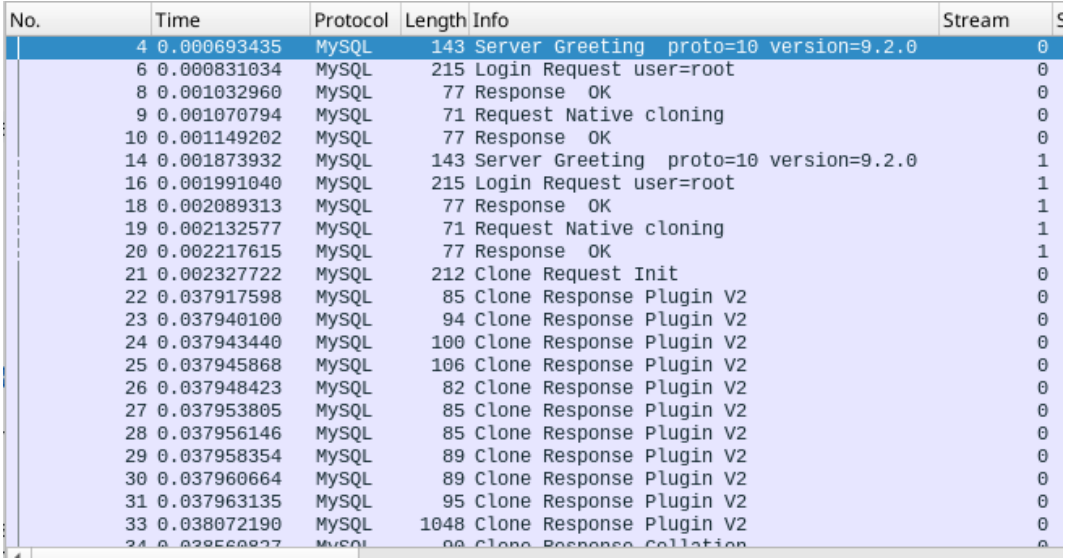
Replication



Clone

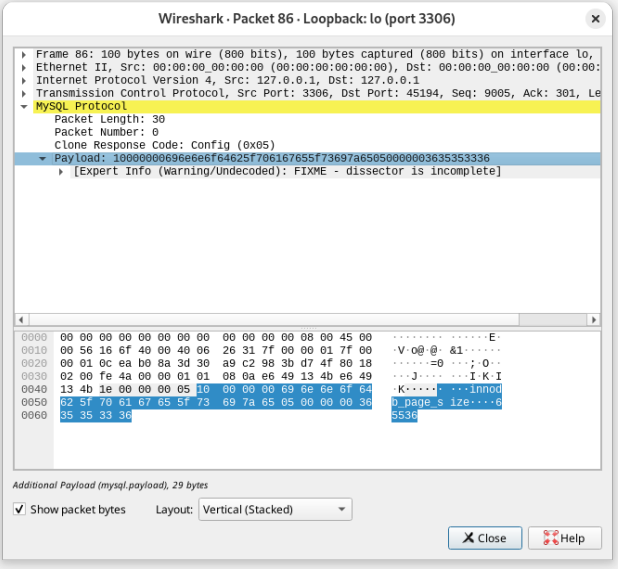
- Clones the data files of a MySQL instance
- Uses multiple connections

Clone



No.	Time	Protocol	Length	Info	Stream
4	0.000693435	MySQL	143	Server Greeting proto=10 version=9.2.0	0
6	0.000831034	MySQL	215	Login Request user=root	0
8	0.001032960	MySQL	77	Response OK	0
9	0.001070794	MySQL	71	Request Native cloning	0
10	0.001149202	MySQL	77	Response OK	0
14	0.001873932	MySQL	143	Server Greeting proto=10 version=9.2.0	1
16	0.001991040	MySQL	215	Login Request user=root	1
18	0.002089313	MySQL	77	Response OK	1
19	0.002132577	MySQL	71	Request Native cloning	1
20	0.002217615	MySQL	77	Response OK	1
21	0.002327722	MySQL	212	Clone Request Init	0
22	0.037917598	MySQL	85	Clone Response Plugin V2	0
23	0.037940100	MySQL	94	Clone Response Plugin V2	0
24	0.037943440	MySQL	100	Clone Response Plugin V2	0
25	0.037945868	MySQL	106	Clone Response Plugin V2	0
26	0.037948423	MySQL	82	Clone Response Plugin V2	0
27	0.037953805	MySQL	85	Clone Response Plugin V2	0
28	0.037956146	MySQL	85	Clone Response Plugin V2	0
29	0.037958354	MySQL	89	Clone Response Plugin V2	0
30	0.037960664	MySQL	89	Clone Response Plugin V2	0
31	0.037963135	MySQL	95	Clone Response Plugin V2	0
33	0.038072190	MySQL	1048	Clone Response Plugin V2	0
34	0.038560827	MySQL	90	Clone Response Collation	0

Clone


Wireshark · Packet 86 · Loopback: lo (port 3306)
Frame 86: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface lo,
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 3306, Dst Port: 45194, Seq: 9005, Ack: 301, Le
MySQL Protocol
Packet Length: 30
Packet Number: 0
Clone Response Code: Config (0x05)
Payload: 1000000069e6e6f64625f706167655f73697a6505000000363535333 6
[Expert Info (Warning/Undecoded): FIXME - dissector is incomplete]
0000 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00
0010 00 56 16 6f 40 00 40 06 26 31 7f 00 00 01 7f 00
0020 00 01 0c ea b0 8a 3d 30 a9 c2 98 3b d7 4f 80 18
0030 02 00 fe 4a 00 00 01 01 08 0a e6 49 13 4b e6 49
0040 13 4b 1e 00 00 00 05 10 00 00 00 69 6e 6e 6f 64
0050 62 5f 70 61 67 65 5f 73 69 7a 65 05 00 00 00 36
0060 35 35 33 36
Additional Payload (mysql.payload), 29 bytes
Show packet bytes
Layout:
Vertical (Stacked)
Close
Help

Query Cache & DEPRECATE_EOF

The (now removed) Query Cache in MySQL did not cache queries.

The Query Cache cached network packets with results. The query matching was very simple. Different case? Different comment? Different query!

The `DEPRECATE_EOF` flag changed how result packets looked like. This flag is per connection. The cache lookup didn't check if the connection was using the same setting for this flag.

The results that were cached were not respecting the setting of this flag and didn't properly follow the protocol.

`https://bugs.mysql.com/bug.php?id=83346`

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