Booking.com

MySQL X Protocol

Talking to MySQL Directly over the Wire

Simon J Mudd <simon.mudd@booking.com>

Percona Live Europe Amsterdam – 5th October 2016

Content

- What is MySQL X protocol
- How does it work
- Building Drivers
- Pipelining
- Why we need a proper protocol specification
- X thoughts – things I noticed
- Conclusion

Disclaimer

- Not involved in the design
- I have not looked at how the old protocol works
- Information obtained from docs, code and observation
- Incorrect descriptions of behaviour are my own

Focus

- A developer should not have to care about this as he or she will be using a driver and will therefore not see the details
- The focus of this presentation is for a driver writer or someone interested in knowing how the communication between client and server works

Focus

- Booking.com uses 2 languages which do not currently have X protocol support: Perl and Go
- We already do special things with MySQL
  - Process binlogs with the binlog router and for sending data to Hadoop
  - We wanted to see if the new protocol would be beneficial to use in our current use cases

What is the MySQL X Protocol

What is the MySQL X Protocol?

In April MySQL 5.7.12 introduces MySQL DocumentStore

- noSQL API to access JSON data in MySQL
- MySQL x plugin in the server
- MySQL shell to provide command line access
- X DevAPI client libraries for: Java, C, dot Net, node.js and Python

What is the MySQL X Protocol?

X protocol: more flexible connectivity between client and server

- asynchronous API, *command pipelining*
- uses tcp port 33060 rather than 3306
- transport uses wrapped Google protobuf messages
- Supports both SQL and new noSQL API
- meant as a foundation for future features

What is the MySQL X Protocol?

Are there any ”buts”?

- The name: wikipedia says the X protocol was created in 1984… I tend to use *MySQL X protocol*
- Support missing in any other “MySQL-like” products
- Drivers missing for other languages
  - network-based protocol specification: users can use to write their own drivers
- However, this is still very new…

How does it work?

How does the X protocol work?

Messages exchanged between client and server are *wrapped* Google protobuf messages

- *Wrapped* means prefixing each message with a 4-byte length and a 1-byte message type indicator
- Protobuf descriptions are buried in the server code!
- Mysqlx_max_allowed_packet: default 1MB
  - Limits the size of a query or single row returned to client
  - In practice this setting may need to be increased

How does the X protocol work?

Message flow consists of the following phases

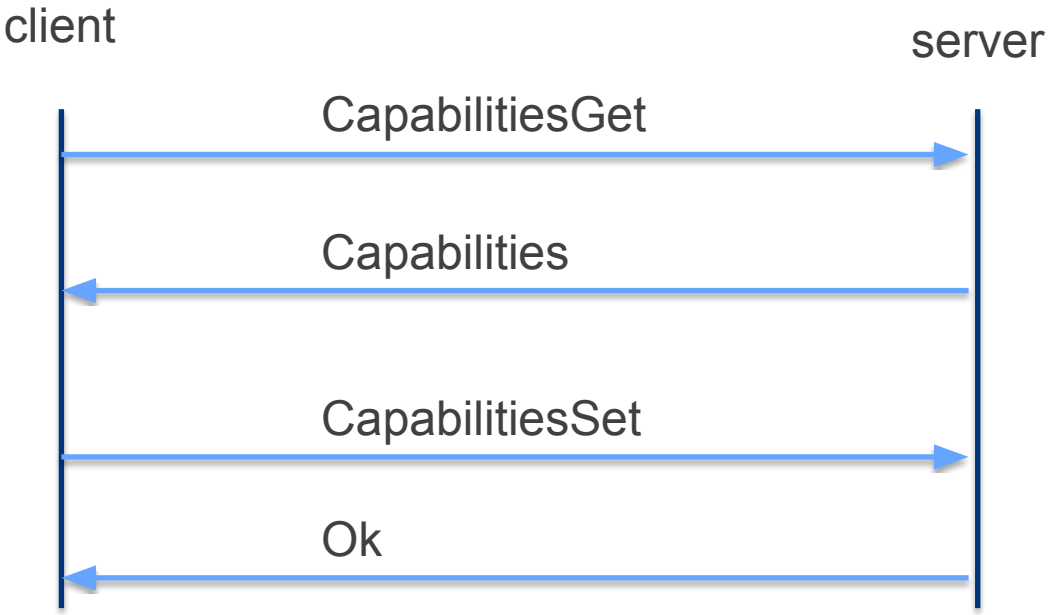
- Connect to server
- Capabilities exchange (optional)
- Authentication
- Querying server (optional)
- Disconnect from server

Capabilities Exchange

Name/value based configuration exchange

- Request/Set some server settings *prior* to authentication
- Used to initiate TLS
- Used to determine which authentication mechanisms are available to the client
- “value” can in theory be *any arbitrary type* though currently single scalar values or a list of scalars
  - this should be formally restrained to keep things simple

Capabilities Exchange



Current Capabilities:

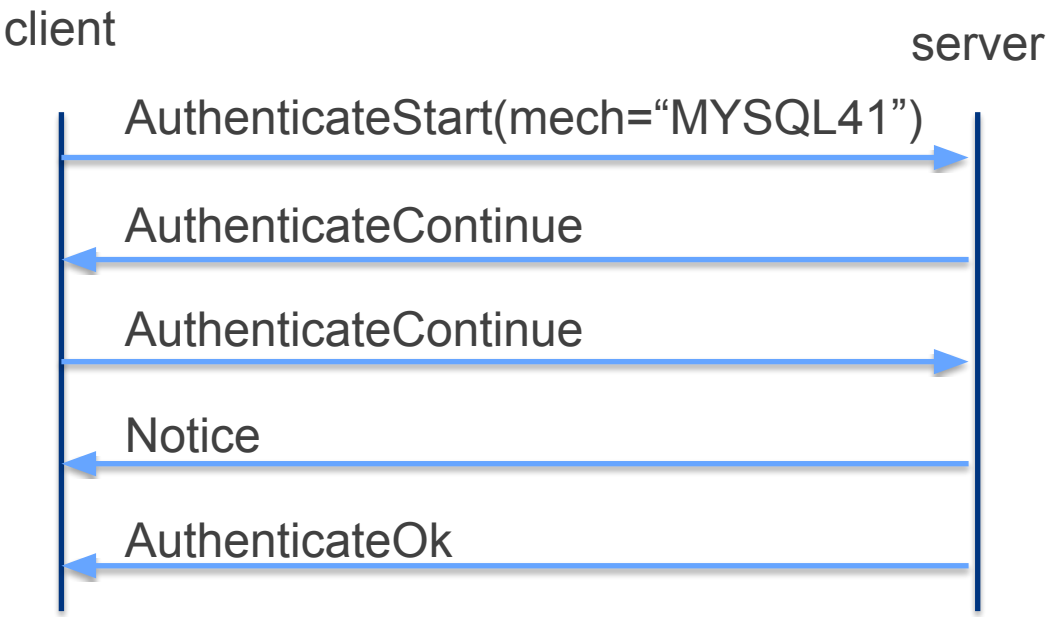
- tls (if TLS is configured)
- authentication.mechanisms
- doc.formats
- node_type
- plugin.version
- client.pwd_expire_ok

Can be used before authenticating client

Authentication

- MYSQL41 by default
- If using TLS other options are available:
  - PLAIN (safe as transport is encrypted)
  - EXTERNAL
- It would be good to define which authentication options are available when and why

Authentication



AuthenticateStart in this case just provides the mech name

Second AuthenticateContinue provides username plus scrambled password but also *database to connect to*

Notice provides a CLIENT_ID

Query Server (noSQL)

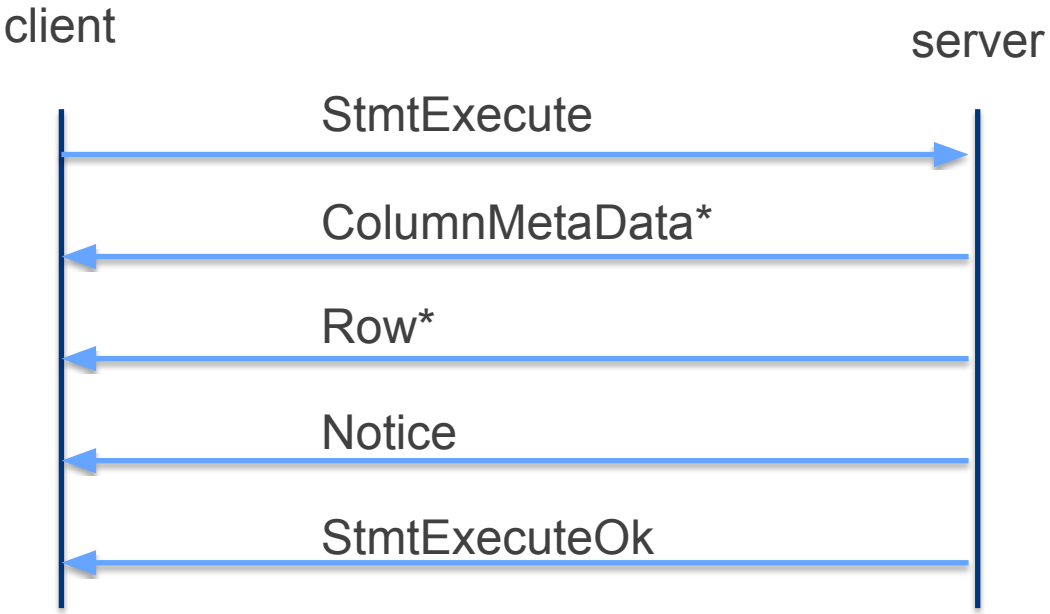
- DocumentStore stuff
- JSON stored in tables and use of CRUD type messages
  - Find, Insert, Update, Delete messages
- Not covered in this presentation

Query Server (SQL)

Client requests data from the server.

- Prepared statements are not available (5.7.15)
- Documentation indicates they are available in sample message flows (see Figure 15.11 Messages for SQL)
- The messages sql::StmtPrepare, and PreparedStmt::ExecuteIntoCursorIt do not appear to exist, but there is a StmtExecute
- Future functionality? Should be indicated more clearly

Query Server (SQL)



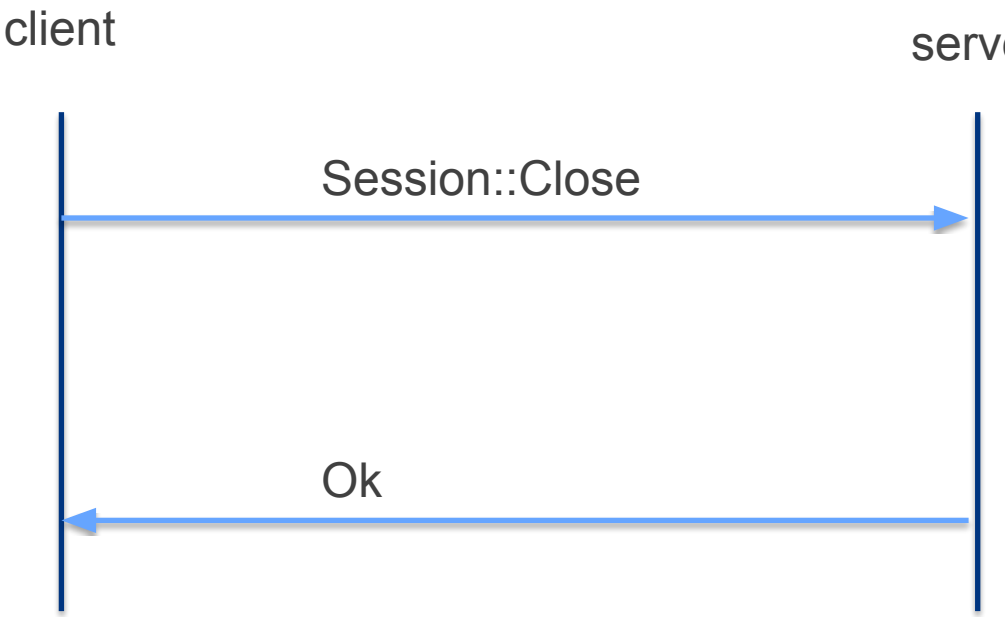
Query:
Contains query and optionally parameters to be used with placeholders

Results:
One ColumnMetaData message per column in result set
One Row message per row in result set
Notice returns rows affected

Disconnect

- Tell MySQL we have finished and then disconnect

Disconnect



Not much to say.

Client free to disconnect from server after receiving Ok

Building Drivers

Building Drivers

Usually drivers are built below a standard high-level interface for the language concerned

- e.g. Go: database/sql, Perl: DBI
- Client can only use API provided by high-level driver
- X protocol wants to use pipelining: may not be available
- To get “all features”: need full custom driver

Building Drivers

- We had a look at Go and Perl
- Harder than expected
- Documentation was not as complete as desired
  - Protobuf files are not enough
  - No explanation of expected behaviour under error conditions
  - Few examples of complete message exchanges
  - Incorrect or misleading documentation
  - Resorted to reading source code or source code tests

Building Drivers

Results of our proof of concept:

- Learnt about message flows
- Achieved authentication
- Able to send queries to the server and get back results
- Look at edge cases
- Work in progress

Building Drivers

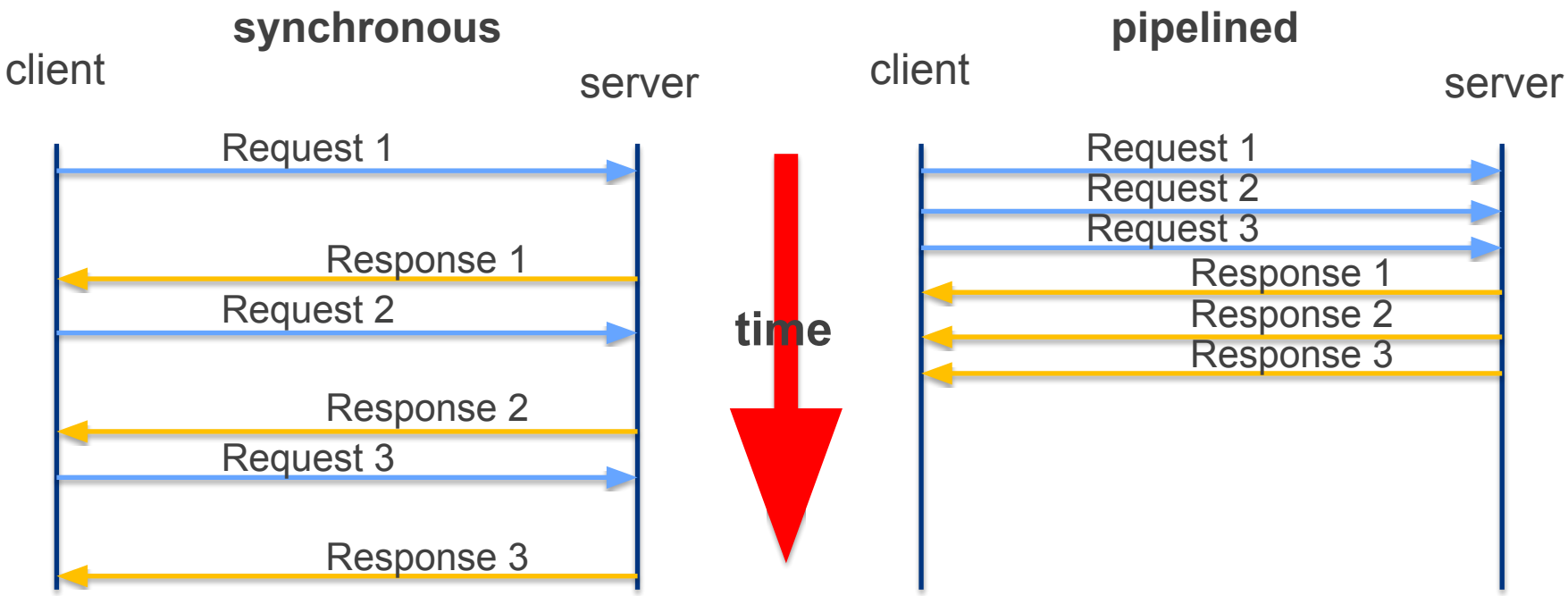
Results of what we did can be seen here:

- Go driver: https://github.com/sjmudd/go-mysqlx-driver
- Perl: https://github.com/slanning/perl-mysql-xprotocol

- But more work to do

Pipelining

Pipelining



X protocol message responses are one or more messages

Pipelining

- Most MySQL X messages are quite small
- Network layer can piggyback more than one message into a single packet when sending
- Useful for session startup as several messages exchanged
- Helpful if you have several independent queries to send
- Avoids the synchronous round trip time wait
- *But* pipelined messages **are not** queued on the server

Pipelining

Servers in more than one data centre:

- cross-dc latency is higher (e.g. ~15 ms vs < 1ms)
- Applications which serialise access to the db may have problems if accessing a remote db when talking locally *runs fine*
- MySQL X protocol here looks interesting

Pipelining

Results of some SQL benchmarking in perl[1]

- 100 primary key SELECTs

Benchmark	Same DC	Cross DC	Latency Affect
Perl DBI:	34ms	1248ms	36x
MySQL X pipelined:	44ms	59ms	1.34x
MySQL X non-pipelined:	89ms	982ms	11x

Conclusion

- Same DC: DBI still faster
- Cross DC: pipelining much faster
- Change application logic to remove serialisation

[1] Scott Lanning: https://github.com/slanning/perl-mysql-xprotocol

Pipelining

Example: Orchestrator

- Currently uses “*legacy*” driver: go-sql-drivers/mysql
- Driver by default sends prepared statements (2x slower)
- We have had to disable prepared statements for performance reasons.
- With MySQL X protocol the pipelining would allow the client to send the prepared statement and execute it together by default – so simpler

Pipelining

- Pipelining will work quite well on higher latency links
- Depends on query execution time vs network latency time
- X protocol is quite noisy (many messages): could be optimised further
- No current support (yet?) for asynchronous queries

Why we need a protocol specification

Why we need a protocol specification

First: Oracle have made a very solid first implementation

- Server side X plugin
- Client libraries
- New shell
- Documentation
- Supports both SQL and noSQL access
- Intended to be production quality on release

Why we need a protocol specification

The MySQL ecosystem is very large

- Everyone using the *classic* or *legacy* protocol
- Moving to a new protocol will only work if it is worthwhile and if players see the benefit
- The benefit can only be gained if everyone jumps on board

Why we need a protocol specification

- Today we have complex use cases:
  - Sharding
  - “external” connectivity (Hadoop, Vitess, Kafka, …)
  - “proxy” connectivity (MySQL router, MaxScale, ProxySQL)
  - may not be in languages supported by Oracle
- Current documentation while improving is still incomplete
- Migration to the X protocol needs to be **easy**
- Other MySQL-like vendors must come on board

Why we need a protocol specification

- Reading the source code of the current X plugin or client libraries does not count as documentation as this is a moving target
- The docs are only available online or in EPUB as my request for a pdf failed. bug#81128

Why we need a protocol specification

- Easy to download document showing full specification
- Driver writers only have one place to look
  - Examples and test cases included
  - Should avoid the need to look at source code
- Ensures that enhancements will not break backwards compatibility
- More likely to get buy-in from the community
- Helps avoid fragmentation

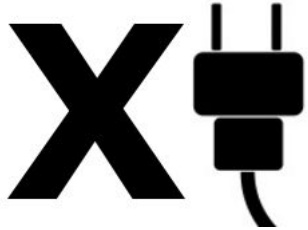
Why we need a protocol specification

I have tried to start writing one myself

- Very much work in progress
- RFC style
- I would appreciate support from others who might be interested in helping
- See: https://github.com/sjmudd/xprotocol-notes

X thoughts - things I noticed

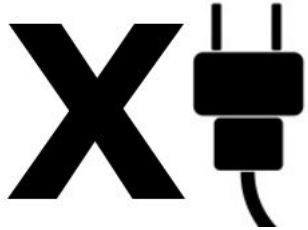
X plugin variable names



mysqlx_max_connections vs max_connections

- I might prefer to limit all connections globally
- mysqlx_max_connections = 0 disable MySQL X connections ?
- mysqlx_max_connections = -1 use max_connections to limit connections ?

X plugin variable names



`mysqlx_min_worker_threads` vs `thread_cache_size`

- Inconsistent naming: maybe better `mysqlx_thread_cache_size` ?

Booking.com

X plugin variable names

`mysqlx_ssl_*` vs `ssl_*`

- `mysqlx_ssl_*` settings need to go away.
- See bug#81528

X plugin

Need more information for monitoring

- Counters for normal and error conditions
- Session and global metrics
- Need timing metrics
- Probably work in progress

X protocol

Initial State

- Character set being used: client and server
- Minimum/default `mysqlx_max_allowed_packet`

Error Handling

- definition of behaviour under different error conditions?

Missing

- Optional idle heartbeat (in both directions), timers
- Checksums: needed? binlog events?

X protocol π Я 音 æ∞

Character sets

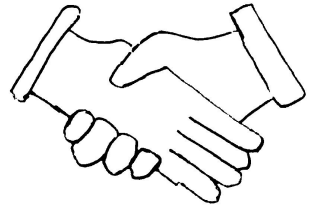
- Expectation of client/server character sets is unclear
- Expectation of data transferred over the wire? Utf8? Which one?
- What about storage and retrieval with character set based columns?
- Where is conversion done when there is a difference?

X protocol π Я 音 æ∞

Character sets

- MySQL 5.7 by default uses utf8 (3-bytes)
- MySQL 8.0 to use utf8mb4 (4-bytes) by default?
- Many people configure things other than the default
- Column data can use different character sets
- Needs to fit together in an **unambiguous** way

X protocol



Initial session setup lengthy

- If you want to check and set things mentioned before and go to TLS then the session setup is lengthy
- Not good for “fast” single query connection types
- Pipelining can help but does not completely solve the problem
- Add **Capabilities Exchange** *during* authentication?

X protocol

Capabilities

- Values can be any type, (avoid nested types)
- No way to see which capabilities are readable or writeable
- No specification of the values that can be applied
- Limit the capabilities which are exposed prior to authentication
- Remove version specific values (e.g. plugin.version)

X protocol

ColumnMetaData received in each query response

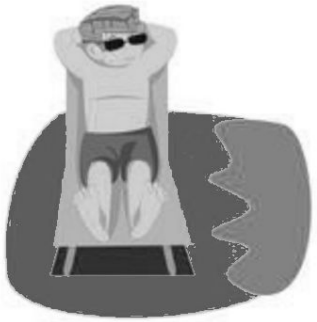
- the X protocol sends one message per column rather than a singe message including all column meta data
- This generates a network overhead of 5 bytes per column (length plus message type) and adds to code complexity as each message processed separately
- Inefficient for single row responses

X protocol

Notice messages too overloaded

- Unsolicited messages from server to client
- Responses of change
- Warnings response to queries
- Session variable changes
- Binlog events …
- In theory can be *ignored* (according to documentation)
- Sometimes unneeded (so overhead)

X protocol



Idle behaviour issues:

- “Server gone away” errors
- “Server still executing query” of a disconnected client
- Tcp keepalive might not see a *stuck* mysqld

Possible Solution

- Optional heartbeat server to client and/or client to server
- Used only if no activity in the direction concerned

X protocol

No unique message ids

- Client and server message ids overlap
- “Sniffers” need to know the direction of the message to be able to decode.

X protocol

Performance given as focus: see WL#8639

- Would be good to see comparisons from the MySQL team.
- Under some use cases the X protocol can be faster. (high latency between client and server with high message rate)

X protocol

Extensibility

- Likely to undergo rapid change
  - Binlog routing? (docs imply this)
  - Sharding (comments from OOW 2016 keynote)
- Do not forget backward compatibility
  - Will help early adopters, driver writers etc
- Proper specifications help

Conclusion

Conclusion

- MySQL X protocol looks good and stable
- If you use the supported languages you will be fine
- A formal specification will ease adoption by third parties, clarify current behaviour and ensure compatibility as the protocol evolves
- Simple drivers to hook into existing SQL infrastructure should be easier to write, but if you want to use the new features such as pipelining more specialised drivers will be needed

References

Oracle

- http://dev.mysql.com/doc/internals/en/x-protocol.html

My work:

- https://github.com/sjmudd/go-mysqlx-driver
- https://github.com/sjmudd/mysql-x-protocol-specification

Thank you