Implementing High-Performance Database Driver

Pavel Tupitsyn
Lead .NET engineer at GridGain
.NET client maintainer at Apache Ignite
ptupitsyn@apache.org, ptupitsyn.github.io

What is a DB driver (client) and what does it do?

```
using Npgsql;

using var conn = new NpgsqlConnection("db-host");
conn.Open();

using var cmd = new NpgsqlCommand(
    "SELECT Name FROM User WHERE CompanyId = @cid", conn)
{
   Parameters = { new("cid", 123L) }
};

using var reader = cmd.ExecuteReader();

while (reader.Read())
   Console.WriteLine(reader.GetString(0));
```

What is a DB driver and what does it do?

```
using StackExchange.Redis;

using var multiplexer = ConnectionMultiplexer.Connect("127.0.0.1");
IDatabase db = multiplexer.GetDatabase()!;
db.StringGet("1");

var channel = multiplexer.GetSubscriber().Subscribe("messages");
channel.OnMessage(message =>
    Console.WriteLine((string)message.Message));
```

Why am I talking about this?

- Author and maintainer of thin client protocol and drivers in Apache Ignite
  - Distributed K/V + SQL db
  - Initially Java-only API, closed protocol
  - Open protocol => lightweight clients, any language (C#, Java, C++, Python, ...)
  - 2.x protocol exists for 5 years, we learned a few things

- This talk is mostly language-agnostic, with only a few .NET specific topics

Do you *really* want your own protocol?

Existing protocol = free implementations

- ODBC, JDBC
- CockroachDB uses PostgreSQL protocol
- ClickHouse and SingleStore (former MemSQL) - MySQL protocol

Risks:

- Limited capabilities
- Licensing
- Uncontrolled evolution / feature deprecation
- Random quirks and features

https://www.cockroachlabs.com/blog/why-postgres/

Roadmap

Part 1: Essentials

- Communication
- Messaging
- Serialization
- Cursors

Part 2: Recommendations

- Cross-language Compat
- Schema Caching
- Object Mapping
- Multiplexing
- Retry, Failover, Keepalive
- Protocol Evolution and Compatibility
- Compression & Encryption

Communication

- Low level API: Socket
- gRPC, HTTP, etc - too much complexity and overhead

Greeting benchmark, new connection

Method	Mean	Ratio	Allocated
Socket	77.25 us	1.00	5 KB
Http	179.15 us	2.32	48 KB
gRPC	321.17 us	4.16	4 KB

Existing connection

Method	Mean	Ratio	Allocated
Socket	19.43 us	1.00	644 B
gRPC	86.27 us	4.44	3,632 B

- UDP vs TCP
  - QUIC & HTTP/3 use UDP to solve head-of-line blocking

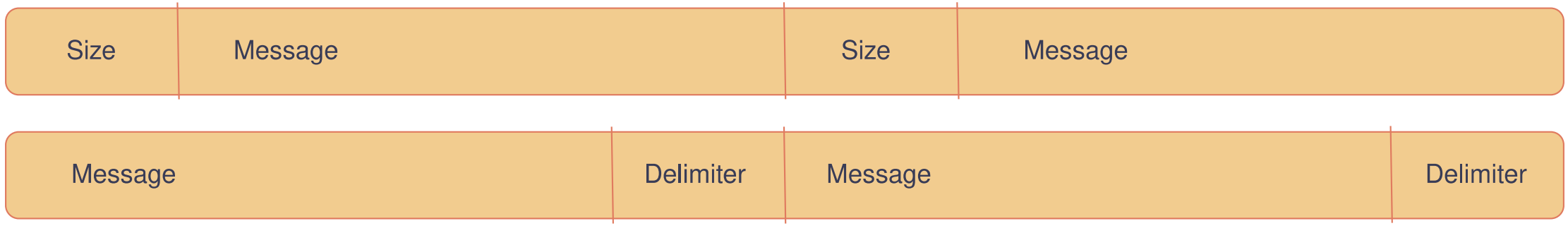
* Low allocations in gRPC benchmark: C# grpc library wraps a C++ binary

Communication

```
var socket = new Socket(SocketType.Stream, ProtocolType.Tcp)
{
   NoDelay = true
};

socket.Connect("127.0.0.1", 10800);

socket.Send(new byte[] { 1, 2, 3 });

var buf = new byte[3];
int received = socket.Receive(buf);
```

Messages

- Socket = stream of bytes
- DB drivers operate on *Messages*, request/response style
  - Client: send SQL
  - Server: send result
- Extract messages from stream
  - Delimiters
  - Size prefix



Serialization: Format

Request

```
{
 "operation": "query",
 "text": "select Id, Name from Customer where
CompanyId = @companyId",
 "args": {
   "companyId": 12345
 }
}
```

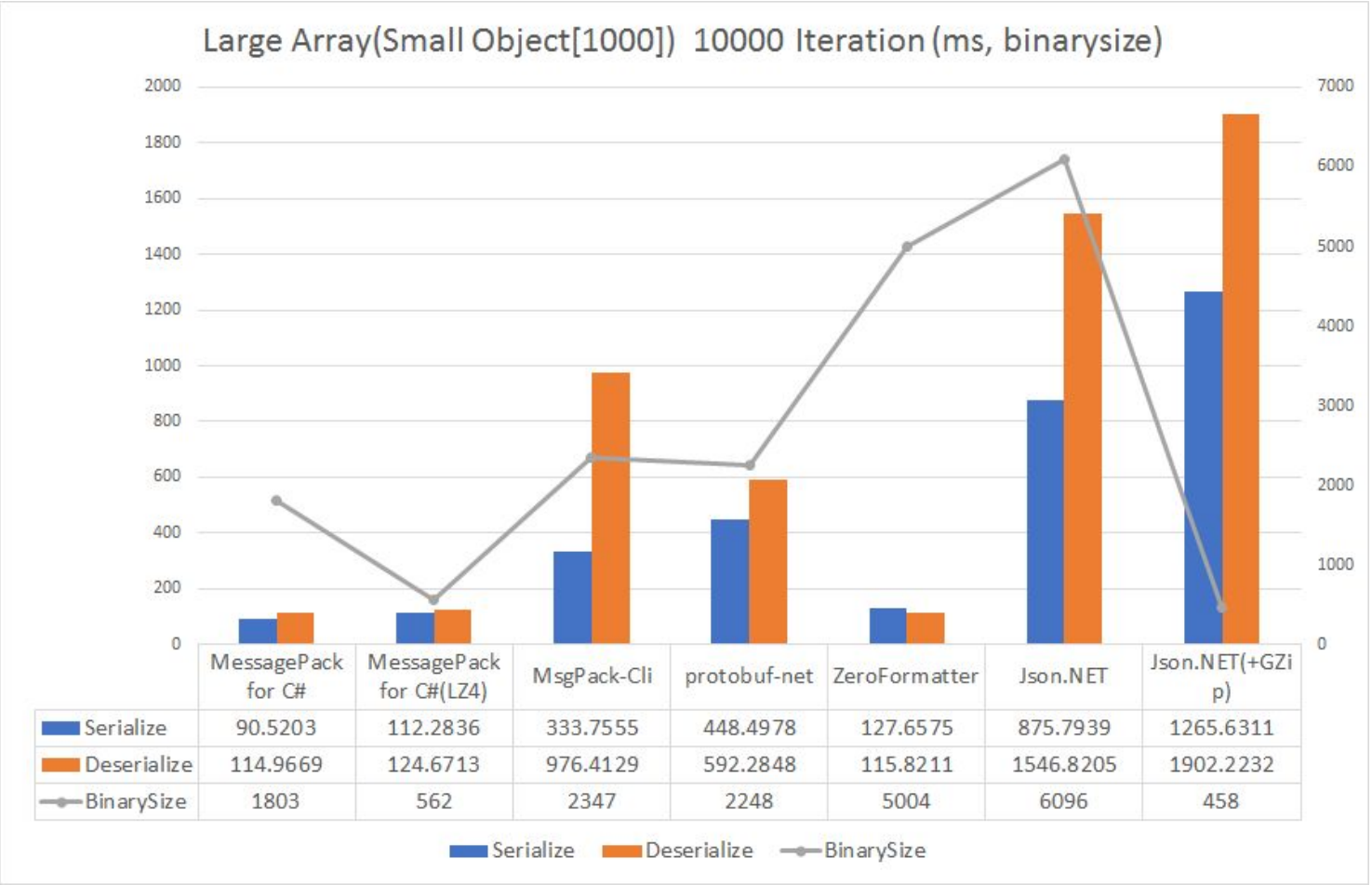
Response

```
{
 "columns": {
   "Id": "int64",
   "Type": "string"
 },
 "rows": [
   { "Id": 1, "Name": "John" },
   { "Id": 2, "Name": "Jack" }
 ]
}
```

Requirements

- Schemaless
- Extensible
- Binary
- Existing implementations
- Permissive license

Serialization: Format



	MessagePack for C#	MessagePack for C#(LZ4)	MsgPack-Cli	protobuf-net	ZeroFormatter	Json.NET	Json.NET(+GZip)
Serialize	90.5203	112.2836	333.7555	448.4978	127.6575	875.7939	1265.6311
Deserialize	114.9669	124.6713	976.4129	592.2848	115.8211	1546.8205	1902.2232
BinarySize	1803	562	2347	2248	5004	6096	458

MessagePack / CBOR

Honorable mention:
Apache Avro

Serialization: Handshake

We have a connection, but:

- Is it the right server?
- Is the version compatible?
- Is authentication required?
- …

Start with a handshake message exchange:

- “Magic bytes” to detect invalid usage early
- Server and client versions (are they compatible?)
- Config flags (auth, compression, metadata)

Serialization: Request Header

- Op Code
- Request Id (Interlocked.Increment)

```
private readonly ConcurrentDictionary<long, TaskCompletionSource> _requests = new();
private long _requestId;

public Task SendAsync()
{
    var tcs = new TaskCompletionSource();
    _requests[Interlocked.Increment(ref _requestId)] = tcs;
    return tcs.Task;
}

private async Task ReceiveLoop()
{
    while (true)
    {
        var msg = await ReceiveNextAsync();
        if (_requests.TryRemove(msg.Id, out var tcs))
            tcs.SetResult();
    }
}
```

Serialization: Response Header

- Message Type (response or event)
- Request Id
- Error Code (0 = success)
- Error Message
- Response Data

```
try
{
    await dbContext.SaveChangesAsync();
}
catch (DbUpdateException ex) when (ex.InnerException is PostgresException
postgresException)
{
    if (postgresException.SqlState == PostgresErrorCodes.UniqueViolation)
        // Retry
    else
        throw;
}
```

Cursors

- Result set can be huge (millions of rows) - exceeds message size limit
- Server return CursorId instead
- Client fetches results in pages by CursorId
- Close cursor on server:
  - All data fetched
  - Client disconnected
  - Explicit (client stops enumeration early)

Conclusion (Part 1)

Basics are covered:

- Connect with TCP Socket
- Split byte stream into messages with length prefix
- Use MessagePack for serialization
- Exchange handshakes to verify the other side, check compatibility
- Send query, receive results with Cursor

Serialization: Cross-language Compatibility

Numeric

- C#: byte (unsigned), sbyte (signed), int (signed), uint (unsigned)
- Java: byte (signed), int (signed)
- JavaScript: number 🤷
- MsgPack: varint (signed/unsigned)

Unique ID

- C#: Guid
- Java: UUID
- JavaScript: string 🤷
- MsgPack: custom type (16 bytes)

Dates, Times, Time zones

- C#: DateTime, DateTimeOffset, DateOnly, TimeOnly
- Java: LocalDate, LocalTime, LocalDateTime, Date, Timestamp
- JavaScript: string 🤷
- MsgPack: custom types

Serialization: Schema Caching

- SELECT Id, Name, Email from User
  - Result is 3 columns
  - Client does not parse SQL and does not know names or types of the result set
  - Server returns Schema Id = 1
  - Schema may contain column names, types, attributes (nullable, default value, etc)
  - ALTER TABLE => new schema id from server

Serialization: object mapping (“micro-ORM”)

```
using NpgsqlCommand cmd = new NpgsqlCommand("SELECT Id, Name FROM Person", conn);
using NpgsqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())
   // Range checks, buffer seek, potential allocations
   yield return new Person(Id: reader.GetInt64(0), Name: reader.GetString(1));
```

- Query results are mapped to objects sooner or later
- Generated mapper (IL.Emit) can outperform handcrafted code
  - Known field order => read sequentially without range checks
  - Known field types => avoid boxing and type casts

```
yield return reader.Get<Person>();
```

```
// Ignite.NET 3 object mapping (IL.Emit) vs hand-coded data reader

      Method |        Mean | Allocated |
  ReadObject |    257.5 ns |      80 B |
   ReadTuple |    561.0 ns |     536 B |
```

GridGain

Connection Reuse, Pooling, Multiplexing

```
using (var conn = new NpgsqlConnection("db-host"))
{
    conn.Open();
    ...
} // Close
```

```
ConnectionMultiplexer multiplexer = GetSingletonConnection();
IDatabase db = multiplexer.GetDatabase();

RedisValue[] results = await Task.WhenAll(
    db.StringGetAsync("1"), db.StringGetAsync("2"));
```

Connection Loss

Long-lived multiplexed connection – increased disconnect risk

- Singleton driver instance becomes broken – unacceptable
- Detect error and reconnect

Retry

Driver API call (connection.Query):

1. Failed due to connection loss
2. Connection restored
3. Retry on new connection
    a. When stateless (not a cursor or transaction)
    b. When safe (idempotent)
    c. When allowed by RetryPolicy (like Polly)

Keepalive (Heartbeats)

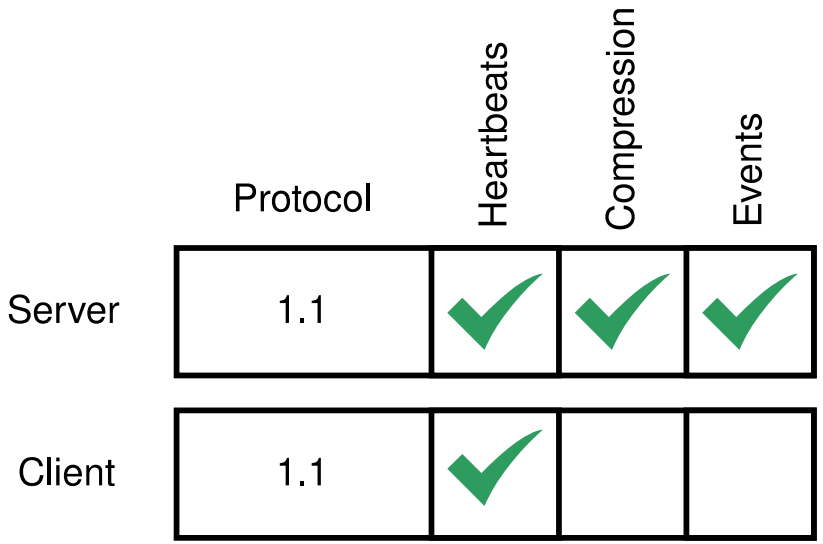
Idle connection may become broken “silently”

- Send periodic pings to detect connection loss
- Add IdleTimeout on server
- Don’t use TCP Keepalive
  - Difficult to configure, default 2 hours

Protocol Evolution and Compatibility

Include Feature Flags and Protocol Version in Handshake

- Feature Flags - add for incremental changes
  - Compression | Events | Heartbeats
- Protocol Version - increase for breaking changes
  - Pick highest common



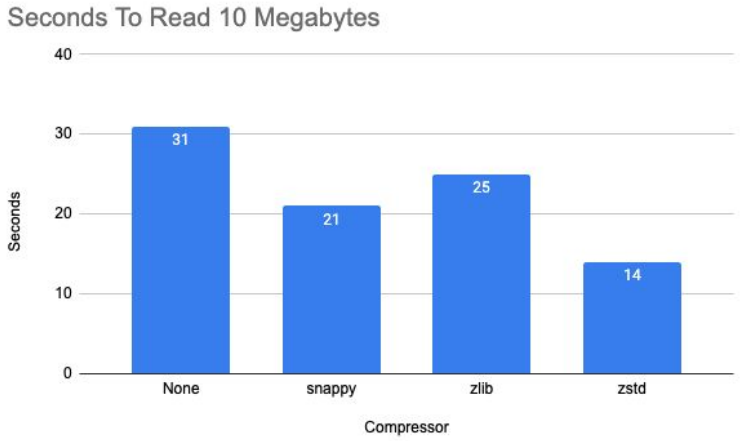
Compression & Encryption

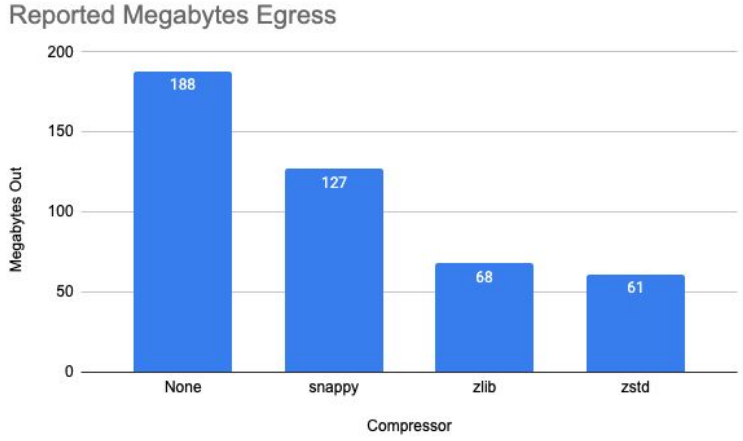
Encryption:

- TLS (SslStream in .NET)
- Can be used for client authentication (client certificates)

Compression

- Network, storage, or both
- PostgreSQL: Storage only (LZ4)
- Mongo, MySQL: Network & Storage





https://www.mongodb.com/developer/how-to/mongodb-network-compression/

Server Events

- PostgreSQL: LISTEN/NOTIFY
- Redis: pub/sub
- MongoDB: change streams
- Ignite: continuous queries

Protocol changes:

- MessageType header field (Response or Notification)
- Clients always read from socket (active Socket.Receive call)

```
while (true)
{
    var msg = await ReceiveNextAsync();
    if (msg.Type == MessageTypeResponse)
    {
        if (_requests.TryRemove(msg.Id, out var tcs))
            tcs.SetResult(msg.Payload);
    }
    else if (msg.Type == MessageTypeNotification)
    {
        if (_listeners.TryRemove(msg.Id, out var listener))
            listener.Invoke(msg.Payload);
    }
}
```

Summary

- Carefully design and document the protocol type system
- Avoid redundant data transfer with Schema Caching
- Consider Object Mapping to improve serialization performance
- Reduce resource usage with multiplexing
- Detect and repair broken connections automatically
- Make the protocol extensible with feature flags and versioning
- Don't reinvent encryption, use TLS
- Consider adding compression
- Add support for server-side messages

The End

- Raw TCP/IP socket communication - Stephen Cleary youtube.com/watch?v=cSIL4nWmZuQ
- CockroachDB: Why Postgres cockroachlabs.com/blog/why-postgres/
- Npgsql Performance npgsql.org/doc/performance.html
- Head-of-line Blocking en.wikipedia.org/wiki/Head-of-line_blocking
- QUIC en.wikipedia.org/wiki/QUIC
- TCP Half-Open en.wikipedia.org/wiki/TCP_half-open
- cwiki.apache.org/confluence/display/IGNITE/IEP-75+Thin+Client+MsgPack+Serialization
- ignite.apache.org
- Redis Protocol redis.io/docs/reference/protocol-spec/
- PostgreSQL Protocol postgresql.org/docs/current/protocol.html
- Tarantool Protocol tarantool.io/en/doc/latest/dev_guide/internals/box_protocol/

Pavel Tupitsyn, GridGain
ptupitsyn@apache.org
ptupitsyn.github.io