Client / Server Communication (VST 0)

HTTP

Use VelocyPack as body. Content-Type is “application/vpack”

Binary Protocol

This is not a request / response protocol. It is symmetric (in principle). Messages can be sent back and forth, pipelined, multiplexed, uni-directional or bi-directional.

It is possible that a message generates

- no response
- exactly one response
- multiple responses

The VelocyStream does **not** impose or specify or require one of the above behaviors. The application must define a behavior in general or on a per-request basis, see below. The server and client must then implement that behavior.

Message vs. Chunk

The consumer (client or server) will deal with messages. A message consists of one or more VelocyPacks. So typically with the drive a message will be defined as vector of VelocyPacks. How many VelocyPacks are part of a message is completely application dependent, see below for ArangoDB.

It is possible that the messages are very large. As messages can be multiplexed over one connection, large message need to be split into chunks. The sender/receiver class will accept a vector of VelocyPacks, split them into chunks, send these chunks over the wire, assemble these chunks, generates a vector of VelocyPacks and passes this to the consumer.

Chunks

In order to allow reassemble chunks, each package is prefixed by a small header. A chunk is always at least 20 bytes long. The byte order is ALWAYS little endian.

Case 1: just one chunk

length	uint32_t	total length in bytes of the current chunk, including this header
chunk isFirstChunk	uint32_t (lower 31bits) (highest bit)	= 1 = 1
messageId	uint64_t	a unique identifier, it is the responsibility of the sender to generate such an identifier (zero is reserved for not set ID)
messageLength	uint64_t	the total size of the message.
Binary data	binary data blob	size b1

Clarification: “chunk” and “isFirstChunk” are combined into a 32bit value. Therefore it will be encoded as

uint32_t chunkX

and extracted as

chunk = chunkX >> 1
isFirstChunk = chunkX & 0x1

The total size of the data package is (16 + b1) bytes. This number is stored in length field. If one needs to messages larger than UINT32_MAX, then these messages must be chunked. In general it is a good idea to restrict the maximal size to a few megabyte.

The total message size is “length” - 16.

Case 2: more than one chunk

In order to optimize memory allocation, the first chunk contains the number of chunks and the total message size.

First chunk

length	uint32_t	total length in bytes of the current chunk, including this header
chunk isFirstChunk	uint32_t (lower 31bits) (highest bit)	the total number of chunks, this number is **greater than** 1. Therefore it is possible to distinguish between case 1 and 2. = 1
messageId	uint64_t	a unique identifier, it is the responsibility of the sender to generate such an identifier
messageLength	uint64_t	the total size of the message.
partial binary data	binary data blob	

Additional chunks

The following chunks do **not** contain the message length

length	uint32_t	total length in bytes of the current chunk, including this header
chunk isFirstChunk	uint32_t (lower 31bits) (highest bit)	an increasing number without gaps. The second chunks contains a 1, the third a 2 and so on = 0
messageId	uint64_t	a unique identifier, it is the responsibility of the sender to generate such an identifier
messageLength	uint64_t	the total size of the message.
partial binary data	binary data blob	

Notes

When sending a (small) message, it is import to ensure that only one TCP packet is sent. For example, by using sendmmsg under Linux (https://blog.cloudflare.com/how-to-receive-a-million-packets/)

ArangoDB

Request / Response

For an ArangoDB client, the request is:

```
[
/* 0 - version: */     1,                    // [int]
/* 1 - type: */        1,                    // [int] 1=Req, 2=Res,..
/* 2 - database: */    "test",               // [string]
/* 3 - requestType: */ 1,                    // [int] 0=Delete, ...
/* 4 - request: */     "/_api/collection",   // [string]
/* 5 - parameter: */   { force: true },      // [[string]->[string]]
/* 6 - meta: */        { x-arangodb: true }  // [[string]->[string]]
]
Body (binary data)
```

If database is missing (entry is "null"), then "`_system`" is assumed.

Type:

- 1 = Request
- 2 = Response (final response for this message id)
- 3 = Response (but at least one more response will follow)
- 1000 = Authentication

requestType:

- 0 = DELETE
- 1 = GET
- 2 = POST
- 3 = PUT
- 4 = HEAD (not used in VPP)
- 5 = PATCH
- 6 = OPTIONS (not used in VPP)

For example:

The HTTP request

```
http://localhost:8529/_db/test/_admin/echo?a=1&b=2&c[]=1&c[]=3
X-ArangoDB-Async: true
```

is equivalent to

```
[
     version: 1,
     type: 1,
```

```
            database: “test”,
            requestType: 1,
            request: “/_admin/echo”,
            parameter: {
                  a: 1,
                  b: 2,
                  c: [ 1, 3 ]
            },
            meta: {
                  x-arangodb-async: true
            }
      ]
```

The request is a message beginning with one VelocyPacks. This VelocyPack always contains the header fields, parameters and request path. If the meta field does not contain a content type, then the default “application/vpack” is assumed and the body will be one or multiple VelocyPack object.

The response will be

```
[
      /* 0 - version: */          1,
      /* 1 - type: */             2 or 3,
      /* 2 - responseCode: */     400,
      /* 3 - meta: */             { etag: “1234” } // [[str]->[str]]
]
Body (binary data)
```

Request can be pipelined or mixed. The responses are mapped using the “messageId” in the header. It is responsibility of the **sender** to generate suitable “messageId” values.

The default content-type is “application/vpack”.

Authentication

A connection can be authenticated with the following message:

```
      [
            version: 1,
            type: 1000,
            encryption: “plain”,
            user: “admin”,
            password: “plaintext”
      ]

Or
```

```
[
      version: 1,
      type: 1000,
      encryption: "jwt",
      token: "abcd..."
]
```

The response is

```
...
```

Content-Type and Accept

In general the content-type will be VPP, that is the body is an object stored as VelocyPack.

Sometimes it is necessary to respond with unstructured data, like text, css or html. The body will be a VelocyPack object containing just a binary attribute and the content-type will be set accordingly.

The rules are as follows.

Http

Request: Content-Type

- "application/json": the body contains the JSON string representation
- "application/vpack": the body contains a velocy pack

There are some handler that allow lists of JSON (seperared by newline). In this case we also allow multiple velocy packs without any separator.

Request: Accept

- "application/json": send a JSON string representation in the body, if possible
- "application/vpack": send velocy pack in the body, if possible

If the request asked for "application/json" or "application/vpack" and the handler produces something else (i. e. "application/html"), then the accept is ignored.

If the request asked "application/json" and the handler produces "application/vpack", then the VPACK is converted into JSON.

If the request asked "application/vpack" and the handler produces "application/json", then the JSON is converted into VPACK.

VPP

Similar to HTTP with the exception: the "Accept" header is not supported and "application/json" will always be converted into "application/vpack". This means that the body

contains one or more velocy-packs. In general it will contain one - notable exception being the import.

If the handler produces something else (i. e. “application/html”), then The body will be a binary blob (instead of a velocy-pack) and the content-type will be set accordingly.

The first bytes sent after a connection (the “client” side - even if the program is bi-directional, there is a server listening to a port and a client connecting to a port) are

VST/1.0\r\n\r\n

(11 Bytes)

TODO

- encryption type for auth is not used in implementation (check - spec and implementation MUST match)
- compression: lz4? everything or large (strings | objects) only?