New Wire Format Proposal

This wiki contains some ideas on improving the Kafka wire format. This could be either a breaking change or introduced as new requests with the existing requests to be removed after one release. The goal would be to do some or all of the following:

Known Proposals

Description	API	New Field	Related JIRA	Discussion
Add correlation id to all requests	All requests	correlation_id: int32	KAFKA-49	A field to make it possible to multiplex requests over a single socket. This field would be set by the client and would be returned by the server with the response. This would allow a client to make multiple requests on a socket and receive responses asynchronously and know which response was for which request.
Reduce duplication in APIs	ProduceRequest MultiProducerRequest FetchRequest MultiFetchRequest	-	-	Currently we have both ProduceRequest and MultiProducerRequest and FetchRequest and MultiFetchRequest. The Multi*Request is just the single request version repeated N times. There are a few problems with this: (1) the ProduceRequest and FetchRequest are just special cases of the general Multi*Request format with no real benefit to them (the reason for their existence is largely historical), (2) having both means more API formats to maintain and evolve and test. We should get rid of the single topic/partition APIs and rename the existing Multi*Requests to ProduceRequest and FetchRequest to keep the naming clean.
Reduce repetition of topic name in Multi* APIs	MultiProducerRequest MultiFetchRequest	-	-	Currently the form of the APIs for the Multi* requests looks something like this: [(topic, partition, messages), (topic, partition, messages), ...]. (Here square brackets denote a variable length list and parenthesis denote a tuple or record). This format is driven by the fact that the Multi* requests are really just a bunch of repeated single topic/partition ProducerRequests. This is really inefficient, though, as a common case is that we are producing a bunch of messages for different partitions under the same topic (i.e. if we are doing the key-based partitioning). It would be better for the format to be [(topic, [(partition, messages), ...], topic, [(partition, messages), ...], ...]. This would mean that each topic name is only given once per request no matter how many partitions within that partition are being produced to.
Support "long poll" fields in fetch request	(Multi-)FetchRequest	max_wait:int32 min_size: int32	KAFKA-48	Add two fields to the fetch request which cause the request to not immediately response. Currently fetch requests always immediately return, potentially with no data for the consumer. It is hence up to the consumer to continually poll for updates. This is not desirable. A better approach would be for the consumer request to block until either (1) min_bytes are available in total amongst all the topics being requests or (2) max_wait time in milliseconds has gone by. This would greatly simplify implementing a high-throughput, high-efficiency, low-latency consumer.
Add producer acknowledgement count and timeout	(Multi-)ProduceRequest	required_acks: int8 replication_timeout: int32	KAFKA-49	Currently the produce requests are asynchronous with no acknowledgement from the broker. We should add an option to have the broker acknowledge. The orginal proposal was just to have a boolean "acknowledgement needed" but we also need a field to control the number of replicas to block on, so a generalization is to allow the required_acks to be an integer between 0 and the number of replicas. 0 yields the current async behavior whereas > 1 would mean that in addition to blocking on the master we also block on some number of replicas. The replication timeout is the time in ms after which the broker will respond back with an error even if the required number of acknowledgements have not been sent.
Add offset to produce response	ProduceResponse	message_set_offset: int64	KAFKA-49	As discussed in KAFKA-49 it would be useful for the acknowledgement from the broker to include the offset at which the message set is available on the broker.
Separate request id and version	All requests	version_id: int16		Currently we have a single int32 that identifies both the api and the version of the api. This is slightly more confusing then splitting out the request id and the version id into two 16 bit fields. This isn't a huge win but it does make it more clear the intention when bumping the version number versus adding a new request entirely.
Add a client id	All requests	client_id: string		Currently we can only correlate client applications to server requests via the tcp connection. This is a pain. It would be good to have a shared logical id for each application so that we can track metrics by client, log it with errors, etc.
Add replica id to fetch request	FetchRequest	replica_id: int32		This replica id allows the broker to count the fetch as an acknowledgement for all previous offsets on the given partition. This should be set to -1 for fetch requests from non-replicas outside the cluster.

Open Questions

1. Can we do a one-time incompatabile refactoring for this?
   a. Pros: no need to keep the old stuff working while adding the new stuff
   b. Con: hard to roll out. Requires updating all clients in other langs at the same time.
   c. One thought on this is that it is probably not too hard to make most of the above changes as new request types and map the old request types to the new. However if we are changing the request id and version id scheme then this will likely not be possible.
   d. If we want to do a 0.7.1 release we will need to figure out a sequencing and branching strategy so that no backwards-incompatable changes block this.
2. Any other fields need for replication or other use cases we know about?
3. Currently the multi-* responses give only a single error. I wonder if this is sufficient or do they potentially need more. For example if you send a produce request to the wrong partition we need to tell you the right partition, which would be different for each partition.

Request Details

This section gives the proposed format for the produce and fetch requests after all the above refactorings.

To aid understanding I will use the following notation:

- int8, int16, int32, and int64 will be integers of the given byte length
- string is a int16 giving the size N of the string followed by N bytes of UTF-8 characters.
- message_set denotes the existing message set format

- [] denote a variable length list prefixed by a int16
- {} denote the fields of a record. These aren't stored they are just used for grouping.
- // denote comments
- <x> denotes that x is a type that will be defined seperately

So as an example a list of records each of which has a name and id field would be denoted by:

```
[{id:int32, name:string},...]
```

Common fields

The following fields are common to all requests:

Request Fields

field	type	order	description
size	int32	1	The size of this request (not counting this 4 byte size). This is mandatory and required by the network layer. It must be the first field in the request.
request_type_id	int16	2	An id for the API being called (e.g. FetchRequest, ProduceRequest, etc.).
version_id	int16	3	A version number for the request format. This number starts at 0 and increases every time a protocol change is made for this API.
correlation_id	int32	4	An id that can be set by the client and will be returned untouched by the server in the response.
client_id	string	5	An user-defined identifier for the client which is used for logging and statistics purposes (e.g. to aggregate statistics across many client machines in a cluster).

Response Fields

field	type	order	description
size	int32	1	The size of this request (not counting this 4 byte size). This is mandatory and required by the network layer. It must be the first field in the request.
correlation_id	int32	2	An id that can be set by the client and will be returned untouched by the server in the response.
version_id	int16	3	A version number that indicates the format of the response message
error	int16	4	The id of the (request-level) error, if any occurred.

ProduceRequest

```
{
  size: int32 // the size of this request
  request_type_id: int16 // the request id
  version_id: int16 // the version of this request
  correlation_id: int32 // an id set by the client that will be returned untouched in the response
  client_id: string // an optional non-machine-specific identifier for this client
  required_acks: int8 // the number of acknowledgements required from the brokers before a response can be made
  ack_timeout: int32 // the time in ms to wait for acknowledgement from replicas
  data: [<topic_data_struct>] // the data for each of the topics, defined below
}

topic_data_struct =>
{
    topic: string // the topic name
    partition_data: [<partition_data_struct>] // the data for each partition in that topic, defined below
}

partition_data_struct =>
{
   partition: int32 // the partition id
   messages: message_set // the actual messages for that partition (same as existing)
}
```

ProduceResponse

```
{
  size: int32 // the size of this response
  correlation_id: int32 // an id set by the client returned untouched in the response
  version_id: int16 // the version of this response  error: int16 // the id of the error that occurred (if any)
  errors: [int16] // per-partition errors, one for each message set sent (or all -1 if none)
  offsets: [int64] // the offsets for each off the message sets supplied, in the order given in the request
}
```

The errors and offsets array MUST contain one entry for each message set given in the request an the order must match the order in the request. That is the Nth offset in the offset array corresponds to the Nth message set in the request.

FetchRequest

```
{
  size: int32 // the size of this request
  request_type_id: int16 // the request id
  correlation_id: int32 // an id set by the client returned untouched in the response
  version_id: int16 // the version of this request
  client_id: string // an optional non-machine-specific identifier for this client
  replica_id: int32 // the node id of the replica making the request or -1 if this client is not a replica
  max_wait: int32 // the maximum time to wait for a "full" response to accumulate on the server
  min_bytes: int32 // the minimum number of bytes accumulated to consider a response ready for sending
  topic_offsets: [<offset_data>]
}

offset_data =>
{
  topic: string
  partitions: [int32]
  offsets: [int32]
}
```

FetchResponse

```
{
  size: int32 // the size of this response
  correlation_id: int32 // an id set by the client returned untouched in the response
  version_id: int16 // the version of this response
  error: int16 // global error for this request (if any)
  data: [<topic_data_struct>] // the actual data requested (in the same format as defined for the produce
request)
}
```