# KIP-219 - Improve quota communication



## Status

**Current state**: *Accepted*

**Discussion thread**: *here*

**JIRA**: *KAFKA-6028*

**Released:** 2.0.0

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

## Motivation

This KIP addresses a fundamental weakness in Kafka's quota design.

Kafka currently throttles clients that violate quotas by delaying the responses. In other words, a client cannot know that it has been throttled until it receives the response. While this works as long as the throttle time is low and the clients are cooperative, this may not always work when the delay is long. Consider the following case for producers (although the same defect can affect consumers):

1. A producer sends a ProduceRequest.
2. The broker decides the producer needs to be throttled for X seconds
3. The producer hits its request timeout before receiving the ProduceResponse
4. The producer disconnects and retries the ProduceRequest.
5. The second ProduceRequest gets throttled again (possibly for an even longer duration than the previous throttle).
6. The producer times out again, disconnects and retries.
7. The pattern continues indefinitely putting more and more load on the cluster.

This scenario is not far fetched. In fact we have seen this problem on multiple occasions when a MapReduce job tries to push data to a Kafka cluster.

The fundamental problem is that clients see only a timeout on violating quota. Therefore clients have no way to distinguish between timeout scenarios (such as network partitions) and quota violation scenarios.

The current quota mechanism has a few other caveats:

1. The throttling is delayed, i.e. applied on the NEXT request instead of the current request.
2. Not able to identify a client before reading the entire request from the wire.

Those two caveats are sort of independent of the lack of communication between brokers and clients. Solving them correctly needs some major changes in the network layer. Therefore this KIP is not trying to address them but only focus on improve the quota communication between brokers and clients.

## Public Interfaces

In order to let the clients know whether the received response is already throttled or not, we will need to bump up all the related request version.

## Proposed Changes

We propose the following changes:

1. After the broker processes a client request and decides that the client is to be throttled for time X, the broker will not hold the response as it does today but will return the response immediately. The response will fill its throttle time field with time X.
2. The broker will then mute the channel corresponding to this client for X seconds. (The broker can use a delayed queue to unmute the channel.)
3. When a client receives a response with throttle time X, it should refrain from sending any further requests for time X. (The usual idle timeout i.e., *connections.max.idle.ms* will still be honored during the throttle time X. This makes sure that the brokers will detect client connection closure in a bounded time.)
4. For throttled FetchRequests, the brokers will return an empty FetchResponse immediately with expected throttle time as if a non-empty fetch response is returned. This makes sure the throttle time of each fetch response increases differently so the consumers will not send the next

FetchRequest at the same time. The broker will also mute the channels for the throttle time based on the non-empty response. The consumer rebalance will not affect the throttle time on the consumers, i.e. the consumers will still refrain from sending requests to a broker after the rebalance if it is doing so before the rebalance.

To indicate that the broker has not throttled the response, we will bump up all request version (without wire format change) so that the clients knows whether it should hold back from sending the next request or not.

Although older client implementations (prior to knowledge of this KIP) will immediately send the next request after the broker responds without paying attention to the throttle time field, the broker is protected by virtue of muting the channel for time X. i.e., the next request will not be processed until the channel is unmuted. Although this does not prevent the the socket buffers from being utilized, the broker's request handlers are not recruited to handle the throttled client's request if it has not backed off sufficiently to honor the throttle time. Since this subsequent request is not actually handled until the broker unmutes the channel, the client can hit *request.timeout.ms* and reconnect. However, this is no worse than the current state.

# Compatibility, Deprecation, and Migration Plan

The change is fully backwards compatible.

# Test Plan

Set quota to a very low value to verify that the clients are neither timing out or sending more requests to the broker before throttle time has passed.

# Rejected Alternatives

One potential solution to the above problem is to set a very high *request.timeout.ms* on the client side. This is not ideal because the request timeout is not supposed to be set in response to a throttling condition.