

KIP-402: Improve fairness in SocketServer processors


- [Status](#)
- [Motivation](#)
- [Public Interfaces](#)
- [Proposed Changes](#)
- [Compatibility, Deprecation, and Migration Plan](#)
- [Rejected Alternatives](#)
 - [Add a configuration option for connection queue size](#)


Status

Current state: *Accepted*

Discussion thread: [here](#)

JIRA:

 Unable to render Jira issues macro, execution error.

 Unable to render Jira issues macro, execution error.

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

Motivation

`SocketServer` currently prioritizes processing of new connections over processing of existing connections. If there is a connection storm, all new connections are accepted and the new connections are processed by the associated `Processors` before processing any existing connections. This can cause delays in closing existing connections, resulting in excessive memory usage. It also prevents brokers from making any progress if delays in processing a connection storm results in request timeouts in clients, which then create even more connections. This can result in a lot of connections on the broker in `CLOSE_WAIT` state. For SSL connections, each connection requires 48k of intermediate buffers, which can trigger `OutOfMemory` errors when closing of existing connections is delayed due to a flood of new connections.

This KIP proposes to improve fairness in channel processing in `SocketServer` by limiting the number of new connections processed in an iteration by each `Processor` before processing existing connections. To avoid a huge backlog of accepted connections in the broker, a fixed size blocking queue will be used to limit the number of accepted new connections that have not yet been processed, applying backpressure and reducing resource usage on the broker. The total number of active connections on the broker will also be limited using a new configuration option to protect the broker from DDOS using a large number of connections from different IPs.

Public Interfaces

No new interfaces or will be added. The size of the blocking queue used for new connections will be set to 20 for each `Processor`. The backlog queue size for incoming connections on the server socket is currently the Java default of 50 and this limit is not configurable. With a default of `num.network, threads=3`, a per-processor queue size of 20 enables this backlog to be processed without blocking. Like polling interval in `SocketServer`, it is unlikely that users will require the queue size to be changed. During normal operation, a small limit allows progress to be made on new channels as well as existing channels. Since `Selector` is woken up when new connections arrive or existing connections are ready to be processed, this limit does not introduce any unnecessary delays in connection processing.

A new metric will be added to track the amount of time `Acceptor` is blocked from accepting connections due to backpressure. This will be a yammer `Meter`, consistent with other `SocketServer` metrics.

- `kafka.network:type=Acceptor,name=AcceptorBlockedPercent,listener={listenerName}`

A new broker configuration option will be added to limit the total number of connections that may be active on the broker at any time. This is in addition to the existing `max.connections.per.ip` config that will continue to limit the number of connections from each host ip address. When the limit is reached, new connections will not be accepted until one or more existing connections are closed. This will be a dynamic broker-wide config that can be updated without restarting the broker.

Config option: Name: `max.connections` Type: `Int` Default value: `Int.MaxValue`

The config may be prefixed with listener prefix to specify different listener-specific limits, enabling inter-broker connections to be created even if there are a large number of client connections on a different listener. Listener-specific limits will be applied in addition to the broker-wide limit. If a listener-specific limit is not specified, each listener can create up to the broker-wide limit as long as the total is within the limit. If a broker has multiple listeners, connections on the inter-broker listener will always succeed as long as they are within that listener's limit. In this case, the least-recently used connection on another listener will be closed to accommodate the inter-broker connection.

Proposed Changes

Acceptor accepts new connections and allocates them to `Processors` using round-robin allocation. In the current implementation, `Acceptor` accepts as fast as possible and adds new connections to unbounded queues associated with each `Processor`.

The connection queue for `Processors` will be changed to `ArrayBlockingQueue` with a fixed size of 20. `Acceptor` will use round-robin allocation to allocate each new connection to the next available `Processor` to which the connection can be added without blocking. If a `Processor`'s queue is full, the next `Processor` will be chosen. If the connection queue on all `Processors` are full, `Acceptor` blocks until the connection can be added to the selected `Processor`. No new connections will be accepted during this period. The amount of time `Acceptor` is blocked can be monitored using the new `AcceptorBlockedPercent` metric.

`Acceptor` will stop accepting new connections when the broker's `max.connections` limit is reached. New connections will be accepted as soon as a connection is closed by any of the `Processors`. `Acceptor` will also stop accepting new connections when its listener's `listener.name.{listener}.max.connections` limit is reached. New connections will be accepted as soon as a connection is closed by any of the `Processors` of that listener. Inter-broker connections will be protected in multi-listener brokers by closing client connections to accommodate inter-broker connections. Any time spent by `Acceptor` waiting for connections to close will also be included in the new `AcceptorBlockedPercent` metric. The existing `max.connections.per.ip` config will be applied without any changes. Connections dropped due to hitting the per-ip limit will not appear in the `AcceptorBlockedPercent` metric since these connections are accepted and then dropped.

Compatibility, Deprecation, and Migration Plan

- *What impact (if any) will there be on existing users?*

No externally visible interface changes are proposed in this KIP. During normal operations, this is unlikely to result in any impact. When a large number of connections are made to the broker at the same time, connections may be established slower than before and existing connections may be processed faster. As with the current implementation, this could result in request timeouts if the broker is overloaded. But resource usage on the broker will be reduced as a result of these changes.

Rejected Alternatives

Add a configuration option for connection queue size

In typical scenarios, Kafka uses long-lived connections, so a small queue size is sufficient to ensure that new connections are processed promptly and existing connections are not left behind. Queue size of 20 per-processor is proposed in this KIP to ensure that the server socket backlog queue size for which we use the Java default of 50 can be processed by 3 network threads without blocking. The goal of this KIP is to protect the broker in scenarios when a very large number of clients connect at the same time. This is likely to be true only for short bursts and hence the small queue size of 20 should be sufficient to ensure fairness in channel processing while protecting the broker from the surge. It is not clear that the number will need to be tweaked for different deployments since queue size is per-processor and the number of processors can be configured using `num.network.threads`