This page explains how to use truncated exponential backoff (https://en.wikipedia.org/wiki/Exponential_backoff) to ensure that your devices do not generate excessive load.

When devices retry calls without waiting, they can produce a heavy load on the Cloud IoT Core servers. Cloud IoT Core automatically limits projects that generate excessive load. Even a small fraction of overactive devices can trigger limits that affect all devices in the same Google Cloud project.

To avoid triggering these limits, you are strongly encouraged to implement truncated exponential backoff with introduced jitter. If you have questions or would like to discuss the specifics of your algorithm, complete this form (https://docs.google.com/forms/d/e/1FAIpQLSfKIAKOXSbIgoKzr4FfAteMxC3kP6skJxFqWSBxVnQxJUmOSQ/viewform)
.

Truncated exponential backoff is a standard error-handling strategy for network applications. In this approach, a client periodically retries a failed request with increasing delays between requests. Clients should use truncated exponential backoff for all requests to Cloud IoT Core that return HTTP `5xx` and `429` response codes, a well as for disconnections from the MQTT server.

An exponential backoff algorithm retries requests exponentially, increasing the waiting time between retries up to a maximum backoff time. For example:

1. Make a request to Cloud IoT Core.
2. If the request fails, wait 1 + `random_number_milliseconds` seconds and retry the request.
3. If the request fails, wait 2 + `random_number_milliseconds` seconds and retry the request.
4. If the request fails, wait 4 + `random_number_milliseconds` seconds and retry the request.
5. And so on, up to a `maximum_backoff` time.

6. Continue waiting and retrying up to some maximum number of retries, but do not increase the wait period between retries.

where:

- The wait time is `min(((2^n)+random_number_milliseconds), maximum_backoff)`, with `n` incremented by 1 for each iteration (request).
- `random_number_milliseconds` is a random number of milliseconds less than or equal to 1000. This helps to avoid cases in which many clients are synchronized by some situation and all retry at once, sending requests in synchronized waves. The value of `random_number_milliseconds` is recalculated after each retry request.
- `maximum_backoff` is typically 32 or 64 seconds. The appropriate value depends on the use case.

The client can continue retrying after it has reached the `maximum_backoff` time. Retries after this point do not need to continue increasing backoff time. For example, suppose a client uses a `maximum_backoff` time of 64 seconds. After reaching this value, the client can retry every 64 seconds. At some point, clients should be prevented from retrying indefinitely.

The wait time between retries and the number of retries depend on your use case and network conditions.

The following samples show how to incorporate exponential backoff when using the MQTT bridge:

- C++ (https://github.com/GoogleCloudPlatform/cpp-docs/blob/master/iot/mqtt-ciotc/mqtt_ciotc.c)
- Java (https://github.com/GoogleCloudPlatform/java-docs-samples/blob/master/iot/api-client/manager/src/main/java/com/example/cloud/iot/examples/MqttExample.java)
- Node.js (https://github.com/GoogleCloudPlatform/nodejs-docs-samples/blob/master/iot/mqtt_example/cloudiot_mqtt_example_nodejs.js)

- Python (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/iot/api-client/mqtt_example/cloudiot_mqtt_example.py)

The following samples show how to incorporate exponential backoff when using the HTTP bridge:

- Java (https://github.com/GoogleCloudPlatform/java-docs-samples/blob/master/iot/api-client/manager/src/main/java/com/example/cloud/iot/examples/HttpExample.java)
- Node.js (https://github.com/GoogleCloudPlatform/nodejs-docs-samples/blob/master/iot/http_example/cloudiot_http_example.js)
- Python (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/iot/api-client/http_example/cloudiot_http_example.py)