**You have messages that are extremely time sensitive. No matter what happens, if they don’t make it to the broker within 30 seconds of send() returning, there’s no point. How can you enforce this?**

To enforce a **30-second delivery guarantee** for Kafka messages, you need to configure the **Kafka producer settings** appropriately.

Here’s how:

**Solution: Use delivery.timeout.ms to Set an Upper Bound**

The **delivery.timeout.ms** configuration defines the total time allowed for message delivery, including **retries and acknowledgments**.

**Key Configuration:**

Properties props = new Properties();

props.put(ProducerConfig.BOOTSTRAP\_SERVERS\_CONFIG, "broker:9092");

props.put(ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG, StringSerializer.class.getName());

props.put(ProducerConfig.VALUE\_SERIALIZER\_CLASS\_CONFIG, StringSerializer.class.getName());

props.put(ProducerConfig.ACKS\_CONFIG, "all"); // Ensures durability but may increase latency

props.put(ProducerConfig.RETRIES\_CONFIG, Integer.MAX\_VALUE); // Allow unlimited retries

props.put(ProducerConfig.DELIVERY\_TIMEOUT\_MS\_CONFIG, 30000); // Set timeout to 30 seconds

props.put(ProducerConfig.REQUEST\_TIMEOUT\_MS\_CONFIG, 29000); // Slightly lower than delivery timeout

props.put(ProducerConfig.LINGER\_MS\_CONFIG, 0); // Send messages immediately

**How It Works:**

1. **delivery.timeout.ms=30000** ensures that if a message isn't successfully acknowledged **within 30 seconds**, it **fails permanently**.
2. **request.timeout.ms=29000** prevents long waits for broker responses, keeping it slightly lower than delivery.timeout.ms.
3. **acks=all** ensures all replicas acknowledge the message, preventing silent failures.
4. **retries=Integer.MAX\_VALUE** allows retries **within the 30-second window**.
5. **linger.ms=0** minimizes batching delay and ensures immediate sending.

**Alternative: Use a Callback to Handle Timeouts**

You can also use a **callback function** to track if the message was delivered within 30 seconds:

ProducerRecord<String, String> record = new ProducerRecord<>("topic", "key", "value");

long startTime = System.currentTimeMillis();

producer.send(record, (metadata, exception) -> {

long elapsedTime = System.currentTimeMillis() - startTime;

if (exception != null) {

System.err.println("Message failed: " + exception.getMessage());

} else if (elapsedTime > 30000) {

System.err.println("Message took too long: " + elapsedTime + " ms");

} else {

System.out.println("Message sent successfully in " + elapsedTime + " ms");

}

});

**Final Answer:**

To ensure messages **either reach the broker within 30 seconds or fail**, set:

* **delivery.timeout.ms=30000**
* **request.timeout.ms=29000**
* **Monitor message delivery using a callback**

This approach guarantees that undelivered messages **fail after 30 seconds**, preventing useless deliveries.

Suppose you fixed the last problem correctly and have also implemented a callback, but in fact, some messages don’t fail until 90 seconds after the producer tries to send(). Where could this extra time be coming from?

If some messages **fail after 90 seconds** instead of **30 seconds**, despite correctly setting delivery.timeout.ms=30000, the extra delay could be caused by **socket timeout settings** or **network retries beyond the expected time window**. Here are possible reasons:

**1. TCP Socket Timeout (socket.connection.setup.timeout.ms)**

* The Kafka client might be waiting for a **TCP connection setup** before sending the message.
* The default timeout for establishing a socket connection might be **longer than expected**.

**Fix:** Set a strict socket timeout:

props.put(ProducerConfig.SOCKET\_CONNECTION\_SETUP\_TIMEOUT\_MS\_CONFIG, 30000);

This ensures that socket connection issues **don’t block the producer beyond 30 seconds**.

**2. retries and retry.backoff.ms Extending the Failure Time**

* If retries is enabled (Integer.MAX\_VALUE in previous config), Kafka will **keep retrying** within delivery.timeout.ms.
* However, **if the broker is unresponsive**, the retry backoff (retry.backoff.ms) could be **delaying the final failure**.
* **Possible scenario:** The client keeps retrying but the broker is unreachable.

**Fix:** Reduce retry.backoff.ms to ensure retries fit within 30 seconds:

props.put(ProducerConfig.RETRY\_BACKOFF\_MS\_CONFIG, 100); // Reduce delay between retries

props.put(ProducerConfig.MAX\_BLOCK\_MS\_CONFIG, 30000);

**4. Network-Level Delays (DNS Resolution, Load Balancer Timeouts)**

* If your Kafka cluster is behind a **load balancer or DNS**, resolution delays could be **adding extra time**.
* Some cloud-based Kafka services (e.g., Confluent Cloud) introduce **connection retries** at the network layer.

**Fix:**

* Test DNS resolution time:

nslookup your-kafka-broker.com

* Use a **direct broker IP** instead of a hostname to avoid DNS delays.

**5. Broker-Level Throttling or Slow Response**

* If the Kafka **broker is overloaded**, it may delay message acknowledgments.
* Look at **broker logs** (kafkaServer.log) to check for **high latency** or **throttling**.

**Fix:**

* Tune broker configs like queued.max.requests to reduce backlog.
* Use **metrics (kafka-producer-metrics)** to monitor response times.

**Final Answer:**

Even though delivery.timeout.ms=30000 is set, **extra time could be coming from:**  
✅ **TCP socket timeouts** (socket.connection.setup.timeout.ms)  
✅ **Retries extending beyond 30 seconds** (retry.backoff.ms)  
✅ **Blocking before send()** (max.block.ms)  
✅ **Network/DNS resolution delays**

To fix this, **set strict timeouts** (socket.connection.setup.timeout.ms, max.block.ms), **reduce retry delays**, and **monitor broker performance**.