Reliable Payments Using Idempotency

Nimish Dighe[1], Yog Pendse[2]
[1, 2] ASM Institute of Management and Computer Studies, Thane

Abstract- *Digital technologies have made a significant impact on all aspects of life, enabling communication and easier ways to access and share information. Digital technologies have also revolutionized the banking and payments sector. People no longer rely only on traditional cash payments. With the advent of mobile phones and internet technologies, we can easily make online payments at the comfort of our fingertips. While online payments are quick and convenient, they also have some occasional problems like payment timeouts and duplicate charges for the same payment. These types of payment issues commonly arise due to technical issues or poor network connectivity. This paper explores the concept of idempotency and how we can leverage it to overcome the problem of payment timeouts and avoid multiple charges for the same payment.*

Keywords- Duplicate charges, Online payments, Payment timeout, Safe retries.

I. INTRODUCTION

Online payments are becoming an increasingly popular method to pay, because they are fast and convenient. However, they require a stable internet connection to ensure a complete transaction. We use our mobile phone to quickly make in-store payments or online payments, where the internet connection may not be reliable.

The issues commonly faced while making online payments in a poor internet connection are outlined in the steps below:

1. The user is presented with a payment page where he can see the transaction details and is asked to enter his credentials to authorize the transaction.

2. The user authorizes the transaction, and waits for a response. However, due to poor network conditions, the user may get a timeout error.

3. The user then retries the transaction, and the user may finally get a payment successful / failed response.

In the above steps, there are chances of double payments being processed. This is because the server may have processed the transaction, but the user did not receive the response due to poor network on the user's end. When the user retries the transaction, the server will process the request again, and this will result in double payments or multiple payments, depending on the number of retries performed by the user.

II. PROBLEM STATEMENT

1. Double payments / multiple payments for the same transaction is a problem commonly faced during online payments.

2. This user needs to perform additional steps to reverse the duplicate transactions, which may include contacting the issuer's bank, the recipient or the payment gateway.

3. The process of manually reversing the double payments can take u2pto 72 hours.

4. High failure rates of online payments can also cause online businesses to lose their valuable customers and orders.

5. It can also dissuade new users from making online payments, who have not tried it before.

While we already focus on encryption for secure transactions and passwords for authentication, it is also necessary to improve the reliability of online payments, to provide a better payment experience.

Also, there should be a mechanism to detect if the same request has been made multiple times to avoid the problem of duplicate transactions.

III. CONCEPT OF IDEMPOTENCY

Idempotency is a concept that a certain operation may be performed multiple times, while having the same result as the first application. Idempotency is the concept of 'At most once' execution, which means multiple identical requests should have the same effect as a single request.

In the concept of Idempotency there are request-level keys. The client generates a UUID (Universally Unique Identifier) for each request. This will serve as a unique

identifier for each request and it will also ensure that the requests that are already processed, are not processed again.

Idempotency helps to achieve consistency between systems, by ensuring that subsequent retries do not affect the state of the system.

VI. PROPOSED SOLUTION

As discussed above, we need to generate a UUID for each request. In this section, we will understand what a UUID is, how it is generated and how we will leverage it in our proposed solution.

What is a UUID?

A Universally Unique Identifier, commonly abbreviated as UUID, is a 128-bit number that is unique for practical purposes. The standard methods for generating a UUID is defined in RFC 4122. When the UUIDs are generated according to the standard methods, they are practically unique and provide certain uniqueness guarantees. The generation of UUID does not rely on a central registration authority or multiple different parties. Anyone can create a UUID and be almost certain that it isn't a duplicate of an existing UUID.

How a UUID is generated?

The procedure to generate a version 4 UUID is as follows:

1) Initially, generate 16 random bytes(=128 bits).

2) Adjust certain bits according to RFC 4122 section 4.4 as follows:
i) Set the four most significant bits of the 7th byte to 0100'B, so the high nibble is '4'.
ii) Set the two most significant bits of the 9th byte to 10'B, so the high nibble will be one of '8', '9', 'A', or 'B'.

iii) Encode the adjusted bytes as 32 hexadecimal digits.

iv) Add four hyphen "-" characters to obtain blocks of 8, 4, 4, 4 and 12 hex digits.

v) Output the resulting 36-character string.

UUID can be easily generated on many of the commonly used devices and programming languages. For example, Java provides a UUID class with a static method randomUUID() which will return a type-4 pseudo randomly generated UUID. We can then call the toString() method on the generated UUID to get a String representation of the generated UUID.

Proposed steps:

1. When the user makes a payment request, the client device generates an idempotency key for that request.

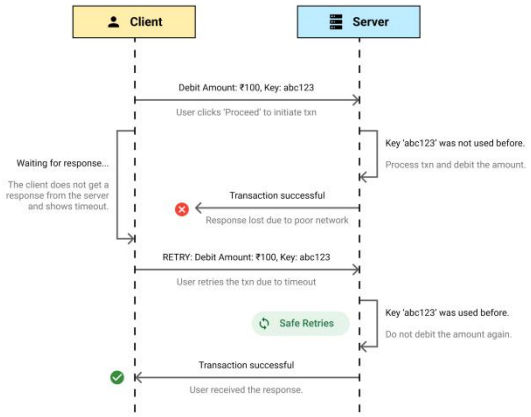
2. The request is then sent to the server. The server first checks whether a request with the same key has already been made before.

3. This will help the server to check if the same request has been initiated again. If a request with the same idempotency key has already been processed before, the server will determine that this is a duplicate request. Hence, it will not process the request again, and simply send the response back to the client.

4. If a request with the same idempotency key has not been processed before, then the server will determine that this is a new request. It will then process the request and store the corresponding idempotency key for future checks. After processing the request, the server will send the response back to the client.

The key will be unique for each request so that, whenever the timeout error occurs, the transaction will not be processed twice. This will also ensure atomicity of payments i.e. either the payment will be executed completely or not at all.

V. EXAMPLE



1. Here, in the above diagram as you can see there is a client and server.

User initiates a request to debit □100 with the auto-generated key ‘abc123’. User clicks on proceed to initiate the transaction.

2 The server checks whether the key ‘abc123’ was used before or not. As the server notices that it wasn’t used before, the transaction gets processed and the server debits the amount.

3 The server sends the response to the client about the transaction being successful, but due to poor network the response never reaches the client.

4 On the other side, the client is still waiting for the response. As the client does not get a response, the payment is timed-out. After seeing the time-out message, the user assumes that the transaction was not processed and clicks retry thereby repeating the same process again.

5 Now, the server checks again whether the key was used before or not. So, yes, it was already used before in a previous request. Thus, the server determines that it is a duplicate request.

6 Therefore, the server does not debit the amount again and just sends the response back to the client about the transaction being successful.

7 This is called the concept of Safe Retries. It means that, no matter how many times the user retries the same request, the amount will not be debited more than once.

VI. CONCLUSION

Payment timeout is a problem commonly faced by the users during an online transaction. The payment timeout problem generally occurs due to poor network and other technical limitations. This can result in a situation where the amount is debited from the sender, but it is not received by the beneficiary. Another problem that can occur after a payment timeout is multiple payments for the same transaction, if the user retries the transaction after seeing a timeout message. To overcome these problems, we can leverage the concept of idempotency to ensure that the payment is processed ‘at most once’.

As seen in the example, the client device generates a UUID or a key for each transaction, which is then passed to the server along with the request. This key helps the server to detect if a duplicate request has been made by the user, and avoid processing them. This ensures the concept of ‘Safe Retries’, which means that the transaction will get processed only once, even after multiple retries.

REFERENCES

[1] G. Ramalingam and KapilVaswaniMicrosoft Research, India. Fault Tolerance vs Idempotence.

[2] Google Standard Payments.
Real-time Payments Systems and Third-party access.

[3] Jon Chew (Airbnb), Avoiding Double Payments in a Distributed Payments Systems.

[4] D.Radha, P.Meenakshi, B.UtthirachSelvi
"A Study on Idempotent Commutative Gamma Semigroup", International Journal of Emerging Technologies and Innovative Research (www.jstor.org), ISSN:2349-5162, Vol.6, Issue 2, page no.1059-1063, February-2019, Available :http://www.jetir.org/papers/JETIRAB06198.pdf

[5] G.L. Litvinov, V.P. Maslov& G.B. Shpiz.
Idempotent Functional Analysis.