Payment Gateway System Report

Introduction

This report outlines the design and implementation of a miniature Stripe-like payment gateway system. The system consists of three main components: **Bank Servers**, **Clients**, and a **Payment Gateway**. The system is designed to handle secure transactions between clients and banks using gRPC for communication, SSL/TLS for secure authentication, and a Two-Phase Commit (2PC) protocol for transaction integrity. The system also incorporates robust logging, fault tolerance, and scalability features.

System Design

1. **Bank Servers**

Each bank server represents an individual bank and is responsible for managing accounts and transactions. The bank servers are designed to:

- **Prepare, Commit, and Abort Transactions**: These operations are part of the 2PC protocol.
- **Fetch Bank Balances**: Clients can query account balances through the gateway.
- **Handle Timeouts**: Transactions are aborted if they exceed a configurable timeout.

Design Choices:

- **Bank-Specific Data**: Each bank server loads its account data from a JSON file (`{bank_address}_server_accounts.json`). This ensures that each bank operates independently and only manages its own accounts.
- **2PC Participation**: Banks act as participants in the 2PC protocol, voting on whether to approve or reject transactions based on account balances and availability.

---

2. **Clients**

Clients interact with the payment gateway to perform transactions and query balances. Key features include:

- **Authentication**: Clients must log in using a username and password.
- **Payment Execution**: Clients can initiate payments, which are queued if the gateway is offline.
- **Balance Fetching**: Clients can query their account balances.

Design Choices:

- **Offline Payment Handling**: If the gateway is offline, payments are queued and retried periodically until the gateway is back online.
- **Secure Communication**: Clients use SSL/TLS to communicate with the gateway, ensuring secure transmission of sensitive data.
- **Idempotency**: Payments are assigned a unique `payment_id` to ensure that retries do not result in duplicate transactions.

3. **Payment Gateway**

The payment gateway acts as the central coordinator between clients and bank servers. Its responsibilities include:

- **Authentication**: Validates client credentials and issues authentication tokens.
- **Transaction Coordination**: Implements the 2PC protocol to ensure transaction integrity.
- **Balance Fetching**: Aggregates balance information from multiple banks for a client.
- **Health Checks**: Monitors the availability of the gateway and bank servers.

Design Choices:

- **2PC Coordinator**: The gateway coordinates the 2PC protocol, ensuring that transactions are either committed or aborted across all participating banks.
- **Timeout Handling**: Transactions are aborted if they exceed a configurable timeout, preventing indefinite hangs.
- **Logging**: Detailed logs are captured for each request, response, and error, providing transparency and aiding in debugging.

Secure Authentication, Authorization, and Logging

1. **Authentication**

- **SSL/TLS Mutual Authentication**: All communication between clients, the gateway, and bank servers is secured using SSL/TLS. Certificates are used to verify the identity of each party.
- **Username/Password Authentication**: Clients authenticate with the gateway using a username and password, which are validated against preloaded user data (`user_details.json`).

2. **Authorization**

- **Role-Based Access Control**: Clients can only access their own account information and initiate transactions from their own accounts. This is enforced using gRPC interceptors.
- **gRPC Interceptors**: Interceptors are used to validate authentication tokens and ensure that clients have the necessary permissions to perform specific operations.

3. **Logging**

- **Verbose Logging**: A logging interceptor captures detailed information for each gRPC request, including:
  - Transaction amount.
  - Client identification (e.g., username, IP address).
  - Method name (e.g., `ExecutePayment`, `FetchBalance`).
  - Errors and exceptions, including error codes and messages.
- **Retry Logging**: The interceptor logs retry attempts for failed transactions, ensuring idempotency and aiding in debugging.

Two-Phase Commit (2PC) with Timeout

1. **2PC Protocol**

The 2PC protocol is used to ensure transaction integrity across multiple banks. The protocol consists of two phases:

1. **Prepare Phase**: The gateway sends a `PrepareTransaction` request to all participating banks. Each bank votes on whether to approve or reject the transaction based on account balances and availability.
2. **Commit/Abort Phase**: If all banks approve the transaction, the gateway sends a `CommitTransaction` request. If any bank rejects the transaction, the gateway sends an `AbortTransaction` request.

2. **Timeout Handling**

- **Configurable Timeout**: Transactions are aborted if they exceed a configurable timeout (default: 30 seconds). This prevents indefinite hangs and ensures system responsiveness.
- **Timeout Monitoring**: A separate thread monitors the transaction timeout and aborts the transaction if it is not completed within the specified time.

3. **Idempotency**

- **Unique Payment IDs**: Each payment is assigned a unique `payment_id`, ensuring that retries do not result in duplicate transactions.
- **Correctness Proof**: The use of unique `payment_id`s and the 2PC protocol ensures that transactions are idempotent. If a transaction is retried, the same `payment_id` is used, and the 2PC protocol ensures that the transaction is either committed or aborted consistently across all banks.

Failure Handling Mechanisms

1. **Offline Payments**

- **Queuing**: If the gateway is offline, payments are queued on the client side and retried periodically until the gateway is back online.
- **Retry Mechanism**: A separate thread on the client periodically retries queued payments, ensuring that no payments are lost due to temporary network failures.

2. **2PC Timeouts**

- **Timeout Abort**: If a transaction exceeds the configured timeout, it is aborted, and the client is notified of the failure.
- **State Recovery**: The gateway maintains a record of processed transactions (`processed_transactions`), ensuring that the system can recover from crashes and continue processing transactions consistently.

Conclusion

The payment gateway system is designed to be secure, fault-tolerant, and scalable. Key features include SSL/TLS mutual authentication, role-based authorization, detailed logging, and robust failure handling

mechanisms. The use of the 2PC protocol ensures transaction integrity, while configurable timeouts and idempotency mechanisms ensure system responsiveness and consistency. This system provides a solid foundation for building a production-grade payment gateway.

Future Work

- **Transaction History**: Implement a transaction history feature to allow clients to view past transactions.
- **Admin Interface**: Add an admin interface to manage users and accounts dynamically.
- **Scalability Testing**: Conduct scalability testing to evaluate system performance under high load.