Strife

1 Introduction

This section details the implementation of Strife, a miniature version of Stripe that interfaces with multiple bank servers to manage financial transactions. The system provides secure authentication, transaction consistency, idempotent payment processing, offline payment handling, and implements a two-phase commit protocol.

2 System Architecture

The payment system consists of four primary components that work together to process transactions:

- **Bank Servers**: Independent servers representing different banks (ICICI, SBI), each maintaining account data in CSV files and providing transaction processing services through gRPC endpoints.
- **Payment Gateway**: Central coordinator that authenticates clients, routes transaction requests to appropriate banks, and manages the two-phase commit protocol.
- **Clients**: End-user applications that initiate transactions, handle offline scenarios, and provide a simple command-line interface.
- **Transaction ID Server**: Generates unique transaction identifiers to ensure idempotency throughout the system.

3 Secure Authentication and Authorization

3.1 Multi-Level Authentication

The system implements a two-tier authentication approach:

- **Gateway Authentication**: Clients authenticate with username/password credentials, receiving a **token** for subsequent requests.
- **Bank Authentication**: During transactions, banks verify account details independently.

3.2 Security Implementation

Communication security is ensured through:

- **TLS** with mutual authentication between all components
- Certificate-based approach with a Certificate Authority (CA)
- **Interceptors** that verify token validity for protected operations
- Color-coded logging for monitoring security events and transaction flow

4 Idempotent Payment Processing

The system prevents duplicate transaction processing through several mechanisms:

- **Transaction ID Server** generates unique identifiers for all transactions
- **Transaction Logs** maintained by each bank record processed transactions
- **Composite Keys** (TransactionID + "-debit" or "-credit") track each operation's state independently
- Before processing, banks check if the transaction ID exists in their logs and avoid reprocessing

5 Offline Payment Handling

The client implements a robust mechanism for handling network failures:

- In-memory **queue** stores transactions that fail due to connectivity issues
- **Background thread** periodically attempts to process queued transactions (every 10 seconds)
- **Failure classification** distinguishes between permanent failures (insufficient funds) and temporary failures (network issues)
- Only temporary failures are queued for retry, preventing wasteful attempts for transactions that would always fail

6 Two-Phase Commit Implementation

To ensure transaction consistency across distributed banks, the system implements a two-phase commit protocol:

6.1 Phase 1: Preparation

- Gateway requests **PrepareDebit** from sender's bank
- If successful, requests **PrepareCredit** from receiver's bank
- Banks validate accounts and funds but don't modify balances
- Both banks must agree to proceed

6.2 Phase 2: Commit

- If both preparations succeed, gateway issues **CommitDebit** to sender's bank
- Then issues **CommitCredit** to receiver's bank
- If any step fails, appropriate **abort operations** are triggered
- Special handling for different failure types (bank down, authentication errors)

The implementation manages transaction state between phases using a **preparedTransactions** map in each bank, enabling proper coordination of the distributed operation.

7 Conclusion

The Strife payment gateway successfully implements all required components of a distributed financial system. Key strengths include:

- **Secure communication** with TLS and token-based authentication
- **Transaction consistency** through the two-phase commit protocol
- **Idempotency** ensuring safe transaction retries
- **Offline resilience** through queuing and automatic retry
- **Comprehensive logging** for monitoring and debugging

These features combine to create a robust payment gateway that maintains data consistency and security throughout the distributed system, even in challenging network conditions.