1. Protocol Buffer Definition (uber.proto):

The uber.proto file defines the service interface and message types for our ride-sharing system. Here's how the functionalities are defined:

a) RequestRide: Allows riders to request a ride.

b) SubscribeRides: Enables drivers to subscribe to ride requests.

c) GetRideStatus: Lets riders check the status of their ride.

d) SubscribeToRideUpdates: Allows riders to receive real-time updates about their ride.

e) AcceptRide: Enables drivers to accept a ride request.

f) RejectRide: Allows drivers to reject a ride request.

g) CompleteRide: The newly added functionality for drivers to mark a ride as complete.

Each of these RPCs has corresponding request and response message types defined in the proto file.

2. Server Implementation (server.py):

The server implements the RideSharingServicer class, which provides the logic for each RPC defined in the proto file:

a) RequestRide:

- Generates a unique ride ID
- Stores ride information
- Starts a timer for auto-rejection
- Notifies available drivers

b) SubscribeRides:

- Maintains a queue of ride requests for each driver
- Sends ride requests to available drivers

c) GetRideStatus:

- Returns the current status of a requested ride

d) SubscribeToRideUpdates:

- Maintains a queue of status updates for each rider
- Sends real-time updates to riders

e) AcceptRide:

- Updates ride status to 'Accepted'
- Assigns the driver to the ride
- Notifies the rider

f) RejectRide:

- Allows a driver to reject a ride request

g) CompleteRide (newly added):

- Updates ride status to 'Completed'
- Changes driver status back to 'AVAILABLE'
- Notifies the rider of ride completion

The server uses threading to handle multiple clients concurrently and a lock mechanism to ensure thread-safe operations on shared data.

3. Client Implementation (client.py):

The client provides separate modes for riders and drivers:

Rider Mode:

- Allows riders to request rides by providing their location and destination
- Enables riders to check ride status
- Implements a listener for real-time ride updates

Driver Mode:

- Allows drivers to subscribe to ride requests
- Enables drivers to accept or reject ride requests
- Now includes functionality to complete a ride

The new CompleteRide functionality is implemented in the driver_mode function:

- After a driver accepts a ride, they're prompted to press Enter when the ride is completed
- This triggers a CompleteRide RPC call to the server
- The server then updates the ride status and driver availability

4. Security:

Both the server and client implement mutual TLS (mTLS) for secure communication:

- The server loads its certificate, private key, and a trusted CA certificate
- The client loads role-specific (rider or driver) certificates and keys
- This ensures that only authenticated and authorized clients can connect to the server

5. Concurrency and Real-time Updates:

- The server uses threading to handle multiple clients simultaneously
- Queues are used to manage ride requests for drivers and status updates for riders
- The SubscribeRides and SubscribeToRideUpdates RPCs use server-side streaming to provide real-time updates

This implementation provides a comprehensive ride-sharing system with real-time communication between riders, drivers, and the central server, all while maintaining secure connections. The new CompleteRide functionality rounds out the ride lifecycle, allowing drivers to mark rides as complete and become available for new requests.