1. When finding the overall balances in a group, why didn't we use a "group by" clause to sum all the user balances from the expense table?

The number of users in a group is variable. We could try to:

a) Denormalize the expenses table into two tables of balances and expense_info.

balances:

expense_id	user_id	balance	paid	owes

expense_info:

id	title	desc	imageUrl	group_id

b) On a 'getGroupExpenses' request for group id=123, fire the database query:

'select user_id, sum(balance) from balances where expense_id in (select id from expense where group_id = '123') group by user_id'

This requires a nested query, which could perform poorly.
However, this style of querying will perform very well for finding an individual user's expenses.

Depending on what you are trying to optimise, you may not may not denormalize the tables.

2. When a user sends a userID for 'getGroupPaymentGraph' authorisation, what stops them from sending a different person's userID?

The userId is authenticated with the corresponding secret like a password or token. This requires an authentication service.
Have a look at the Gmail architecture videos for a better understanding.

3. The expense service exposes the payment graph API. Will this be called by the user service for individual expenses?
The client can send the expenseId of a particular expense. The getPaymentGraph method will return the required resolution transactions as a graph.

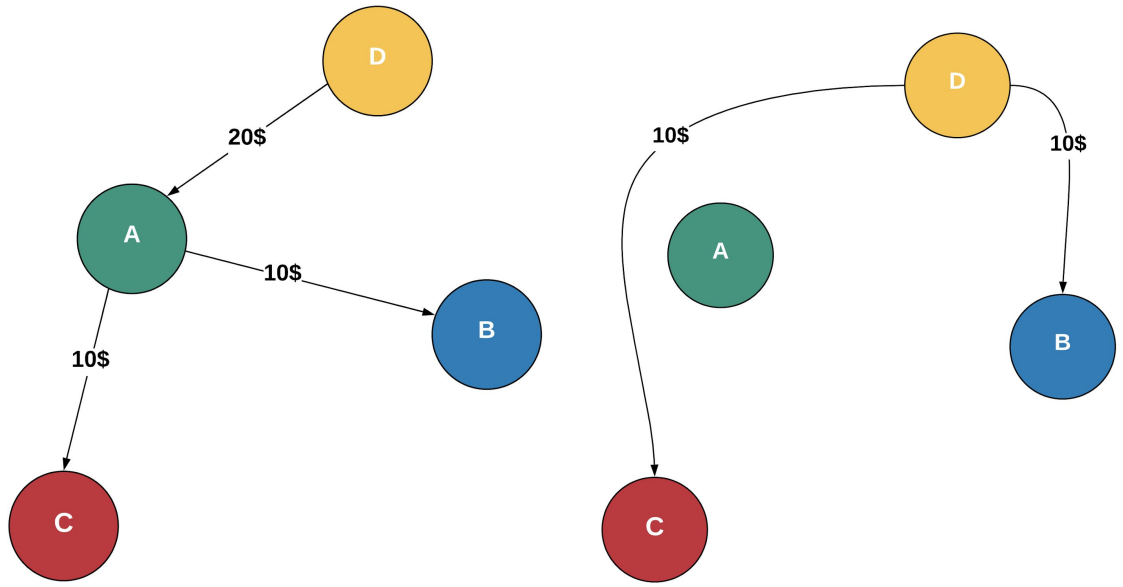
If we need all expenses pertaining to a user, the expense service can be directly queried on a separate API of getExpensesForUser, which will take a user_id as parameter.

4. Won't this form of settlement potentially cause two unknown users interacting with each other?

Yes. Let's take an example.

A, B and C are all friends.
A and D also friends. But D knows neither B nor C.
B owes A → 10 dollars
C owes A → 10 dollars
A owes D → 20 dollars

If we simplify debts, B and C will each have to pay 10 dollars to D.



Original *Simplified*

In real life, this might create an awkward situation where D needs to send payment reminders to unknown parties (B and C).

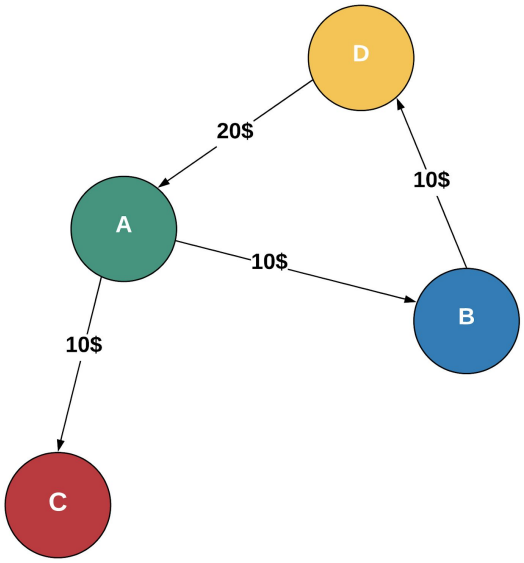
Hence, we should have "simplify payments" as an optional feature for groups. They could opt for a transitive settlement instead.

4. What is a transitive group settlement for a group?

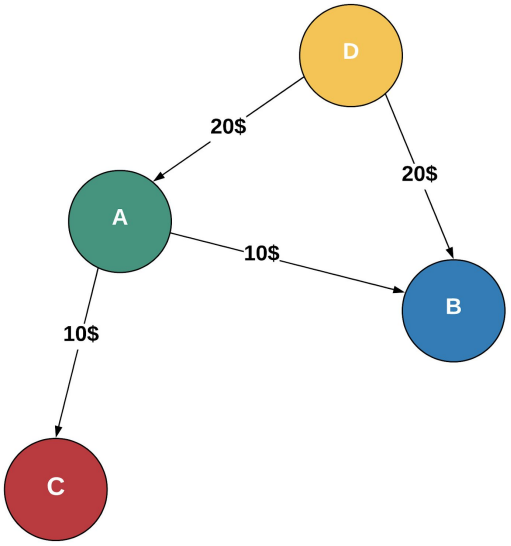
Take all the expenses in a group. Create payment graphs for each expense. You now have a set of payment graphs.
Sum all the "user pair" balances. Taking the above example:
B owes A → 10 dollars => pair (A,B) = 10
C owes A → 10 dollars => pair (A,C) = 10
A owes D → 20 dollars => pair (D,A) = 20

We now have 3 transactions instead of the 'simplified' case, where we would have 2.
If D owes B → 10, the group will have 4 transactions (Shown in *Transitive(1)*):
pair (A,B) = 10
pair (A,C) = 10

pair (D,A) = 20
pair (B,D) = 10



Transitive(1)



Transitive(2)

In the simplified case, the only transaction would be C owes D → 10.

You might have another expense with B owing D → 30. In that case, the pair of (B,D) will be summed to 10 - 30 = -20. (Shown in *Transitive(2)*).

Hence, instead of summing on users, we sum on user pairs.

5. We store IDs instead of users in the cache. To similarly save on memory, why don't we query a list of balanceMaps from the expense service, instead of a list of expenses?
The getGroupExpenses API should be designed with extensibility and the original intent in mind.

Users may want to view all of their expenses for a group, with the relevant metadata of expense title, date, etc...
If the group service is overloading the expense service API, we can try caching the results. If that doesn't work, we can design a custom API for the group service, which just returns the balance maps relevant to a group.