The user process commands implemented are open, new-cohort, delete-cohort, exit, deposit, withdrawal, transfer, lost-transfer, checkpoint, and rollback. The format for each command is as specified below: The format for each command is as specified below:

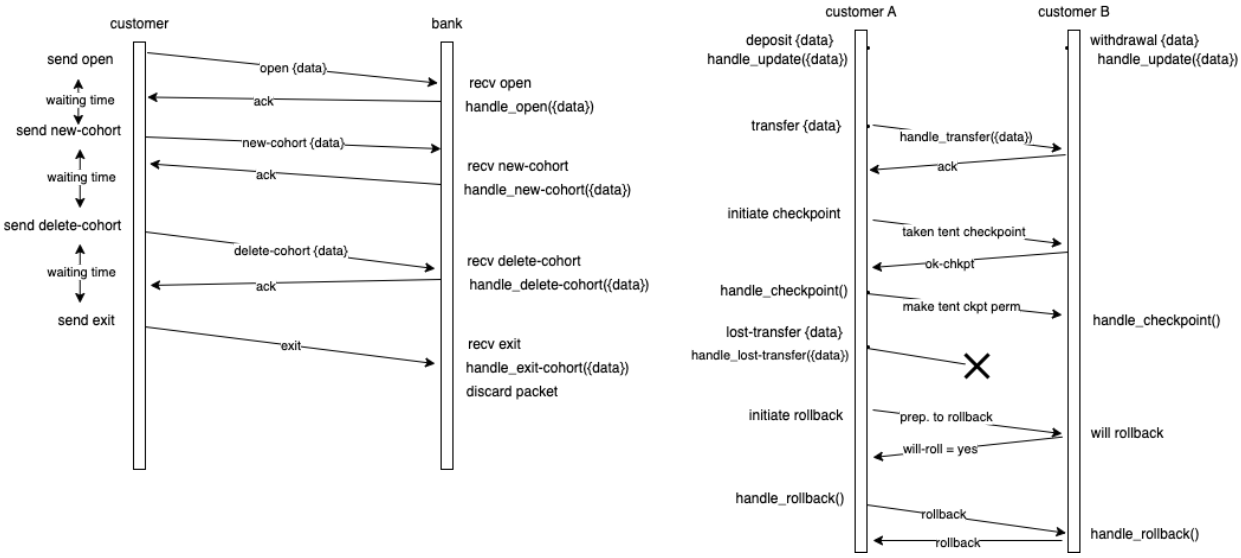
- `open <customer> <balance> <IP> <portb> <portp>`
  `<customer>` will be the customer name, `<balance>` will be their initial balance, `<IP>` is the IP address of the customer, `<portb>` and `<portp>` will be ports available for the customer
- `new-cohort <customer> <n>`
  `<customer>` will be the customer initiating the new cohort, and `<n>` is the number of total customers in the new cohort.
- `delete-cohort <customer>`
  `<customer>` is the name of the customer who is in an existing cohort
- `exit <customer>`
  `<customer>` is the name of a customer with an existing bank account
- `deposit <amount>`
  `<amount>` is the amount to add to the current balance of the customer account
- `withdrawal <amount>`
  `<amount>` is the amount to take out of the current balance of the customer account
- `transfer <amount> <q> <label>`
  `<amount>` is the amount that should be taken out of the current account balance, `<q>` is the customer who should receive the amount being transferred
- `lost-transfer <amount> <q>`
  `<amount>` is the amount that should be transferred to `<q>` who is the customer receiving the amount. Note that lost-transfer will is only a simulation.
- `checkpoint`
  `checkpoint` command consists of a series of messages that should be sent between customers in cohort. The first message that should be sent is take-tent-ckpt which will take a tentative checkpoint from the customer who initiates the checkpoint to all the customers in the cohort. After the customers receive this message, they will need to agree to a checkpoint by sending a message back to the initiator. Once all have agreed to a checkpoint, initiator

sends a new message make-tent-ckpt-perm which will conclude the checkpoint for everyone.

- `rollback`

Similarly to `checkpoint`, the rollback consists of a series of messages sent between customers in cohort. The first message to be sent is prepare-rollback from the customer who initiates rollback to the other customers in the cohort. From there, once the customers agree to a rollback then the initiator will call a rollback command which will run handle_rollback(), to each customer.

Design with timespace diagrams



Architecture

In order to store customer information, we defined a struct, `Customer`, which contains all the information sent over by the customer themselves. Another struct, `Bank`, was also created to store the list of customers as well as determine the total number of customers which will increase and decrease as customers open accounts and exit. In the Bank struct there is also some cohort data that is being stored there such as the number of cohorts and an array of cohorts themselves. The final struct, `Cohort`, was implemented as well to create cohorts and to store values such as the size, cohort id, and the customers in the cohort. The data structure of choice to store the customer information for this submission was an array of our defined struct, `Customer`. The `Customer` struct holds all the information sent by the customer and once a new

`Customer` is created, we add them to the banks customers array. We created two new structs called state_t and properties_t where we can track the labels last_received, last_sent, first_sent associated to a customer using the identifier customer_name in the properties_t struct and in the state_t struct we associate these properties_t as props[MAX_COHORT_SIZE] for each customer in the cohort along with the labels balance, ok_checkpoint, resume_execution and will_rollback to track the state of each customer in the cohort.