Preliminary design description

Members:

Rasmus Nummelin, rvnummel@stud.ntnu.no

Tom Inge Bjørkeng, tibjorke@stud.ntnu.no

Jens Munkerud, jvmunker@stud.ntnu.no

Lab table: 8, group number 79, Friday 08-12.

Project description and reflection

We have been tasked with controlling n elevators over m floors. We must assume that our system will have a fault, but ensure that a fault doesn't imply a failure. We have identified the following core problems:

- Handling high packet loss without entering a failure-state
- How to go from an error-state to a working state.
- Synchronization between elevators.

System design

Our plan is to create the system in Figure 1. We have three elevators working in a peer-to-peer configuration utilizing UDP as our network protocol. Each elevator will be modeled as a state machine according to Figure 2. By modelling the elevator as a state machine we can split the communication into two parts. The first is state-changes. When an elevators state changes, this change should be broadcast and acknowledged on the network. The second is a heartbeat containing the current state of the elevator and it's world view. To ensure all local calculations are correct, synchronized data is essential.

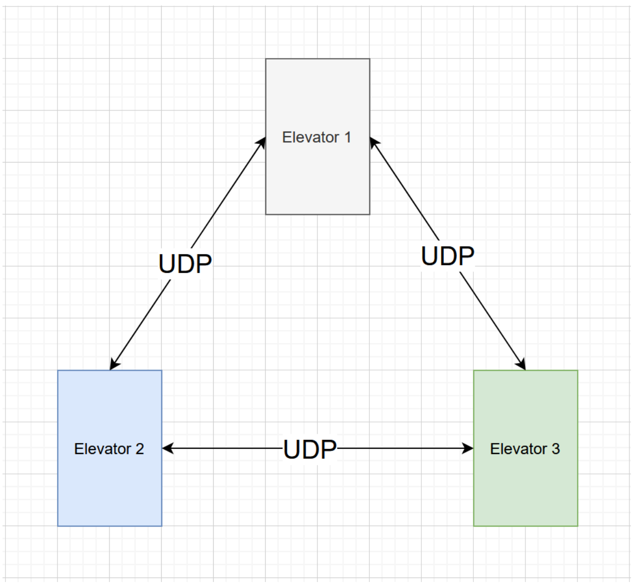


Figure 1

Packet loss

For state-changes acknowledgements will ensure that a message is actually sent, as these are essential for correct functioning. In addition a hash-function will ensure the data received is correct. Heartbeats don't require acknowledgement, as they are sent often anyway and are not essential for correct functioning.

Handling error-states

Network disconnects will be handled by a heartbeat function running on all elevators. When this times out on a given elevator, it will enter an independent state and finish it's remaining cab orders. After this it will try to reinitialize to connect to the network. It's hall orders will be served by the remaining working elevators. Both powerloss and any program error (not yet mentioned) will trigger the initializeElevator() function, acting as an elevator reinitialization (Utilizing merging of error states).

Synchronization between elevators

Due to the peer-to-peer topology, our synchronization problem will be one of data. We will ensure that data is synchronized by two main strategies. Ensuring the data is received and correct. Data received will be handled by acknowledging, prompting a resend if no acknowledgment is received. Data correction will be ensured by transmitting a hash of the data with the data. Comparing the received hash and a calculated hash, we can quickly identify corrupt data, which would lead to no acknowledgement being sent.

We will use unique order ID: [orderID, elevatorID] to ensure all orders are done in order. If two orders receive an identical orderID, the lowest elevatorID gets the orderID and the second order receives a new orderID.

General Design

We have chosen UDP to gain more control over the networking as opposed to TCP. Things like loss of connectivity with high packet loss, and having a long timeout window makes TCP hard to work with. UDP gives us more flexibility in creating a system for safely transferring data, without constraining us to TCPs rules.

A cab light can only be turned on once the order has been distributed on the network, all elevators have acknowledged it and the elevator in question has received these acks correctly.

We will use the 3 states “Idle, Moving, Door Open” since they will make it easier to implement the handed-out elevator algorithm cost function. The finite-state-machine diagram is shown below.

The external data includes everything noted in the JSON hall_request_assigner algorithm. This is because it completely defines the state of any elevator at all times.

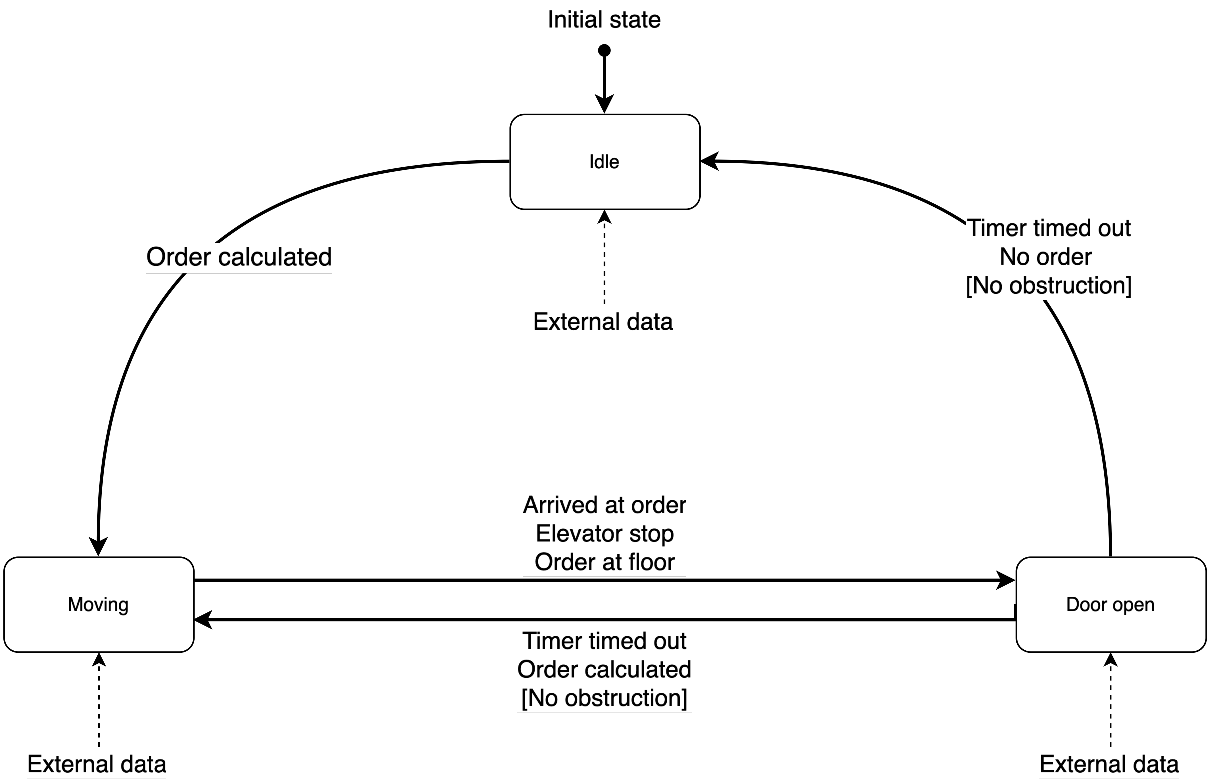
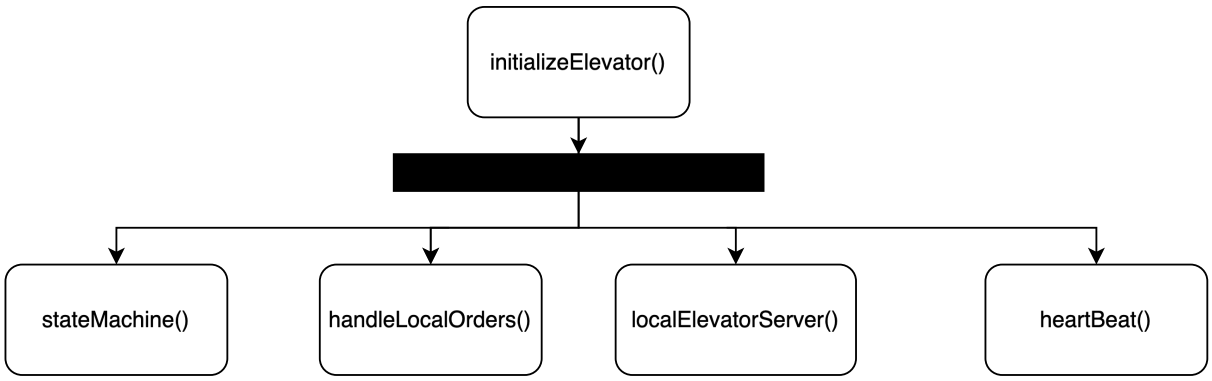


Figure 2



Overview of expected goroutines running on each elevator.