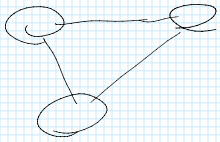
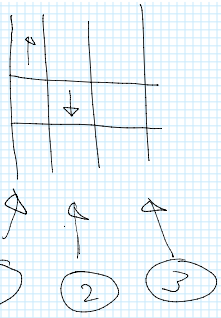
Elevatordesign

10 February, 2020 15:36

Topology

- Peer 2 peer
- UDP





Cycle of one elevator

- Init (see loss of power)
- choose new order → cost function
- Broadcast accepted order
- Moving (to the called floor and then complete the order)
- complete order → Broadcast order completed
- Idle

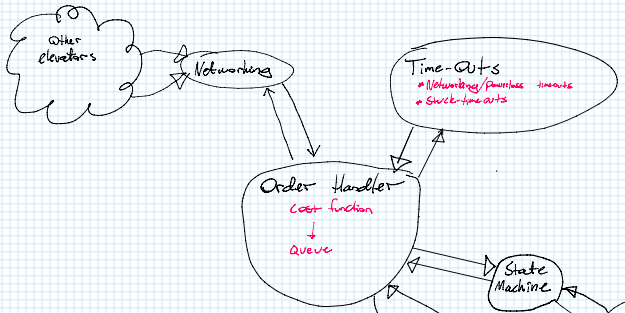
order handling

reception: . broadcast
. save it in the txt_file

from broadcast: . save it in the txt file
a new order or an update

Modulation

Modules within one elevator



Order_handler

structures Queue, Order

Update_Queue(*order, *Queue), if an order does not exist, enqueue it

Create_order

Choose_new_order(*Queue)
→ Cost_fnc

Check_timeout_flags()

Timeouts

When a new order is processed
↳ time it

Start_order_timer(order)
→ Start thread
on timeout → raise flag

State Machine

Network

broadcast_msg(*message)
↳ Default message - elevators state
→ Request our own state + orders
Inherit their world view.

recieve_msg()
↳ Unpack message → order

Initialize_network()
↳ Sveres network
↳ Set target IP/ports

One Node loses network

Detection

* Message reception → Resets timeout on that one elevator
↳ All the elevators are keeping tabs/timeouts on each other

Error handling

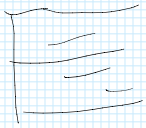
* Assume the elevator (e) which has lost connection is DEAD
↳ update txt_file

Loss of Power (Brief)

Detection

From the others point of view
* Timeouts (same as network loss)

Error handling:



↳ At startup/initiation
1. Check state
2. Ask other elevators for current order queue
3. Resume

Unforseen event / Never arrives at destination

Detection

* Every elevator has a local timeout.
Update with alive message

Error handling

* Timeout fires ⟶ Broadcast "Stuck"
↳ Others reschedule order.

Guaranteeing an order

* Every elevator is equipped with a cost function
↳ Atleast one elevator will propably take the order
↳ Don't care if there is a double schedule

* The elevators normally only share changes.
↳ Exception routines are described above.

* A msg is validated as shared if we receive at least one ack or timeout from the other

Tecnical Implementation

non-blocking with few threads

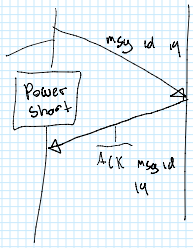
Reliabilities, module / higher level
↳ We try :)

Datastructure of an order

1 struct, 8 member variables. (numbers)

ID	floor	direction	time of reception	order taken (by who?)	time order taken	destination floor

Saved in a txt - file



questions:

Should we be able to manage negative floors?