Preliminary Project Description - Group 18, Desk 20

Sjur Groven, sjurgr@stud.ntnu.no
Sondre Hauk Heier Holte, shholte@stud.ntnu.no
Last member withdrew from course.

TTK4145 - The Elevator Strikes Back

January 27, 2025

General strategy. We look to implement a primary-backup system, with a centralized view service. Each elevator interacts only with the primary and has no information about the other elevators in the network. The elevator will interface with the physical layer, except maybe for the hall buttons, which will interface with the primary directly. A class diagram of this setup can be seen in (3), while a basic activity diagram can be seen in (2). The messages sent between the primary and the elevator will be of the predefined types seen in (1). Heartbeats can detect disconnections, and they may include data packets.

Finite State Machine. Elevator level code. Each physical elevator will be assigned an instance of the "elevator" struct. This will contain information about the state of the elevator as well as the saved variables, namely: door timer, motor direction and current floor (-1 between floors). The state variable is a Boolean tuple that can take one of n values; see figure 1. Minimize the state space and make undesirable states "impossible" by explicitly stating which combinations of Booleans the state variable can take.

Divide functions into core functions (which make decisions) and shell functions (which perform actions based on a given decision). Pointer receivers will be used to implement functions that can alter the states and saved variables of each elevator. Ex: elevator_instance.stop() may set the motor direction to STOP.

Why Golang? Compiled language with efficient code execution. Scalable and effective concurrency model. Goroutines and channels are useful for synchronization and message passing.

Packet Loss. After weighing our options, we have chosen to implement UDP transmission to connect our nodes. Our initial approach will be to create our own handshake protocol to acknowledge that messages have been received.

Network disconnection. In the case where an elevator realizes it has disconnected, it will finish all current tasks, as well as accept and execute any new cab calls. However, it will not accept any new hall calls. After finishing, the elevator will enter an error state where it will stay idle until it is reconnected and synchronized with the master. The main argument for doing this is code testability and reducing logical complexity in the development phase.

If we succeed with this approach early enough, we will implement dynamic re-synchronization while serving current tasks. Ideally, this will reduce down-time but will introduce more complexity and possible bugs.

Node crash. In the event of a total system shutdown for an elevator, it will need to stop in place, reboot, receive its current tasks from the primary and continue execution.

Error handling by acceptance test. Throw an exception when received messages are not 'as expected'. Important with standard message types to detect exceptions. See table (4) for messages. Avoid complex control flow handling by restarting node.

Primary - Backup. The system will perform a backup takeover if connection is lost between primary and backup. Backup will then broadcast its address to the other nodes, and initiate its own backup. An oversimplified takeover protocol can be seen in figure (5).

Resources

The primary backup approach; `https://www.cs.cornell.edu/fbs/publications/DSbook.c8.pdf`

Figures and Tables

Msg from Elevator to Primary	Msg from Primary to Elevator	Msg from Primary to Backup
Active Cab Calls Assigned Hall Order Completed Elevator state	New Hall Order Heartbeat (when connection failure)	Heartbeat State update

Table 1: Messages from A to B

at floor	stop btn	moving	door open	
0	0	0	0	Standstill (after emergency)
0	0	1	0	Moving (between floors)
0	1	0	0	Emergency in shaft
1	0	0	0	Idle
1	0	0	1	Door open
1	0	1	0	Moving (passing a floor)
1	1	0	1	Emergency at a floor

Figure 1: Elevator Finite State Machine. We have chosen to implement stop button logic, for the sake of testing

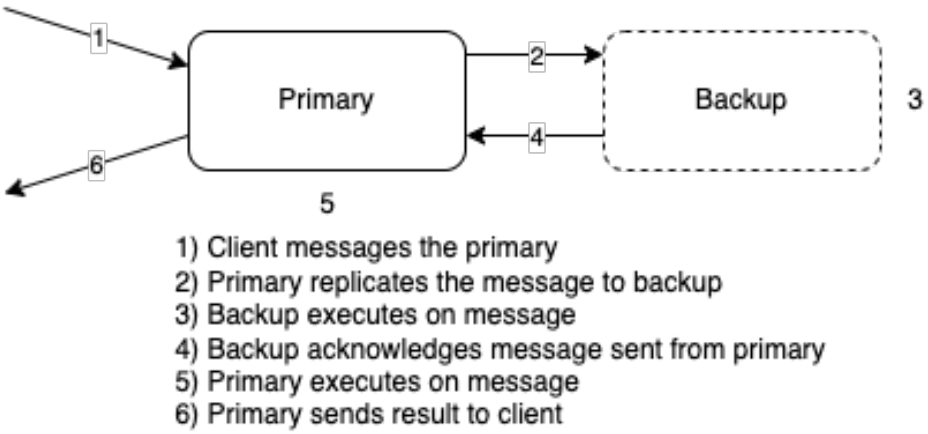

Figure 2: Preliminary Primary Backup System Protocol

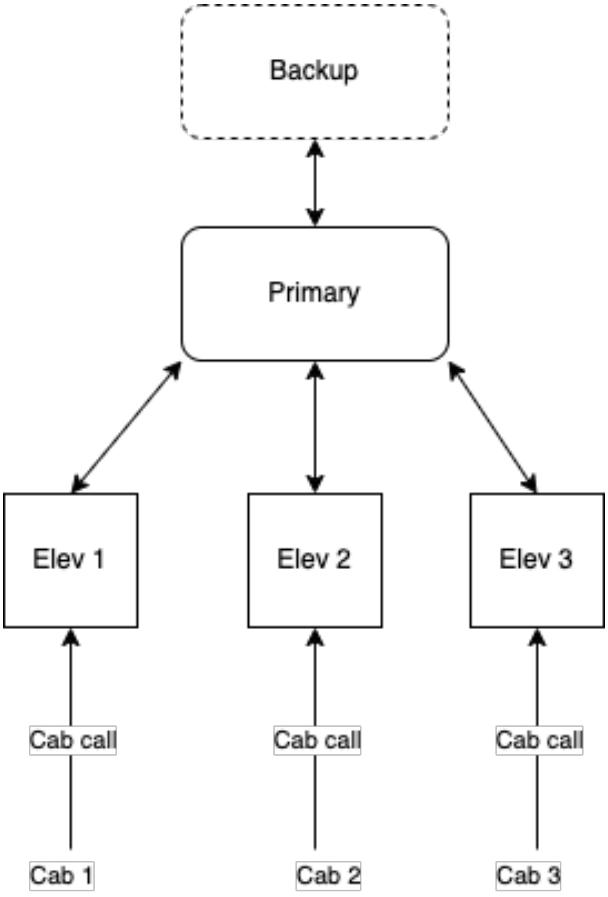

Figure 3: Primary Backup System

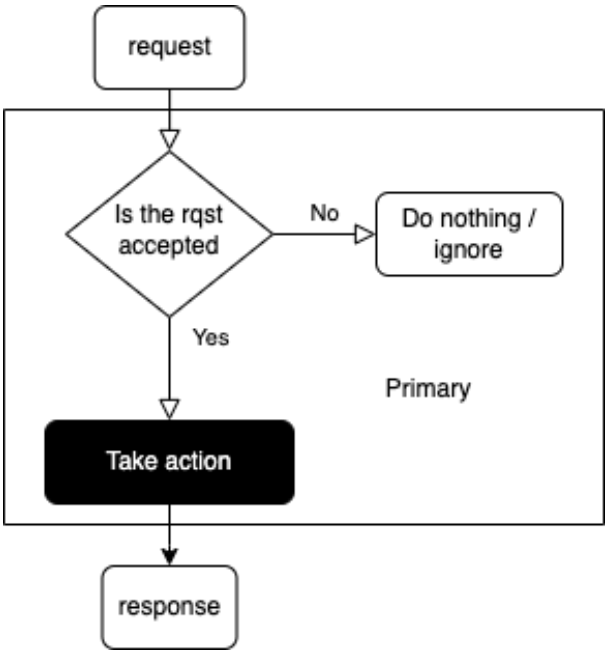

Figure 4: Error handling by acceptance test

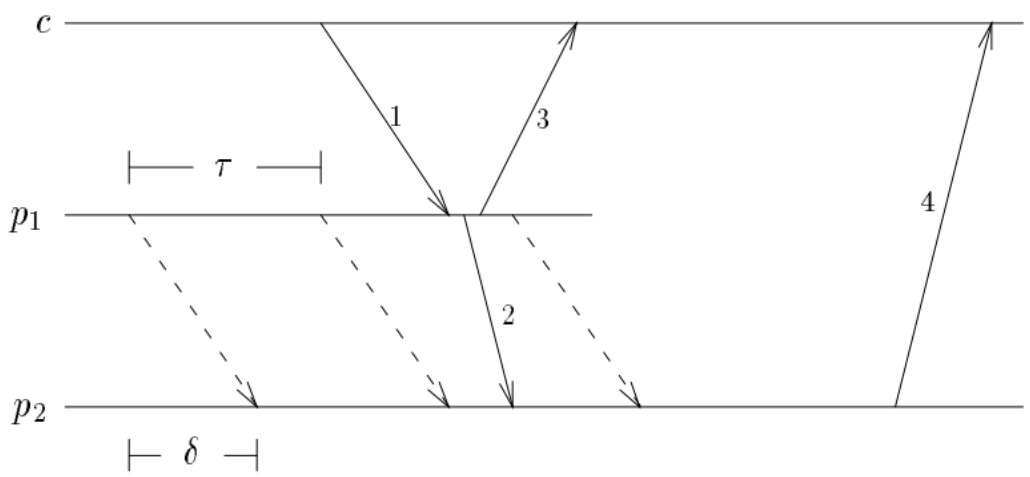

Figure 5: Simple takeover protocol. Client c, Primary p_1, Backup p_2. 1) Request from client to primary. 2) Request from primary to backup. 3) Result from primary to client. 4) Backup detects primary failure, and initiaties takeover.