Fault detection, Isolation and Modelling - Miniproject

NDS 820

June 11, 2019

This miniproject was made for the course, Fault detection, Isolation and Modelling, by group members in NDS 820.

NDS 820:

- Andrej
- Bandar
- Mikki
- Pierre
- Eleftheria
- Sebastian
- Tarik

Chapter 1

Fault detection, isolation and modelling

When working with a bigger system such as the one done for the project, faults and errors can occur in many areas. These areas could be in the implementation of the different components or in areas that can't be controlled such as the connection to the internet. In this chapter a look at a Failure Modes and Effect Analysis (FMEA), a reliability/availability analysis of the system with fault detection trough hidden markov models and fault recovery through forwards and backwards based recovery.
For the FMEA, different types of failure modes within the system are identified and analysed. For the reliability/availability analysis, different states/components in regards to security are identified and the probability for failures can be computed through the use of Markov Chains. The analysis will be based on the system from the project which looks as follows on fig. 1.1. The main components that will be looked at are the "Jump Host", "Webserver" and "LDAP".

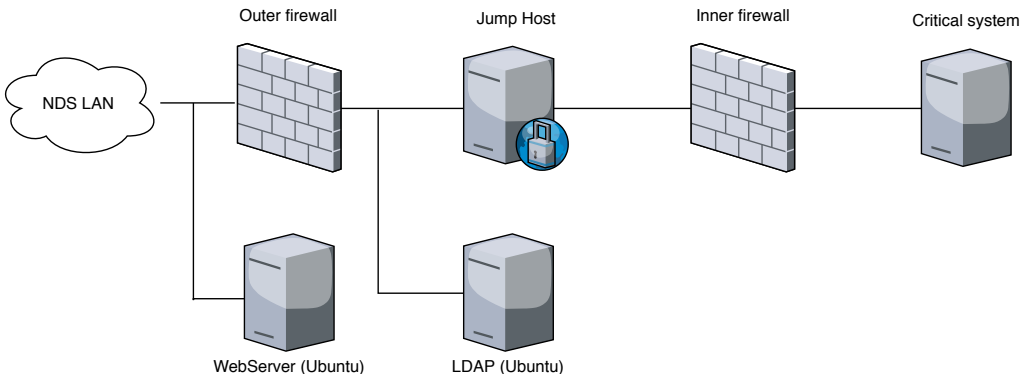

Figure 1.1: Overview of the system

1.1 Failure Mode and Effect Analysis (FMEA)

When doing an FMEA, determining potential failures, effects of those failures and their causes are required. This is done by looking at the behaviour of components within the system. The FMEA covers the following areas:

1. Determine Failure Modes
2. Determine Severity of Failures
3. Determine Probability of Occurrences
4. Severity-Occurrence Index Schema
5. Determine Probability of Detection
6. Compute Risk Priority Number
7. FMEA schema

The system consists of both hardware and software components where both will be taken into consideration for the analysis. The three main components that will be looked at will be: `LDAP server`, `Jump Host` and `Webserver` which are critical for the systems functionality.

1. Failure Modes

When doing an FMEA, the first step is to determine failure modes for the system. Failure modes describe what type of errors can occur within a system during operation. When analyzing the failure modes, each failure mode has both a cause and and an overall effect. This can be seen as a reason and consequence for what failure occurred. For the analysis, the main focus is the network related failures since the system that was used for the analysis mainly operates via communication over networks such as ethernet where failure modes both appear in hardware and software.

Table 1.1: Item, Failure Modes, Failure Causes and Effects.

Item	**Failure Mode**	**Failure Cause**	**Effect**
1. LDAP server	1.1 Failure to connect	1.1.1 System is Offline 1.1.2 Wrong Credentials Provided	1.1 System Failure
	1.2 No Response	1.2 Invalid/Malicious Query	1.2.2 System Failure
	1.3 Cannot Read Data	1.3 Corrupted Data	1.3 Loss of Data
	1.4 Cannot Add Data	1.4 Corrupted Data	1.4 Loss of Data
	1.5 Added Duplicate Data	1.5 Missing Exception Handling	1.5 Duplicate Data
	1.6 Overwriting Wrong Data	1.6.1 Wrong Index Location 1.6.2 Missing Exception Handling	1.6 Loss of Data
	1.7 Wrong Data Added	1.7 Information wasn't written correctly	1.7 Loss of Data
2. Jump Host	2.1 Failure to connect to LDAP	2.1.1 LDAP is Offline 2.1.2 Wrong Credentials Provided	2.1.1 System Failure 2.1.2 Delayed Data
	2.2 Failure to connect to Webserver	2.2.1 Webserver is Offline 2.2.2 Mismatch in User Agent 2.2.3 DNS Errors	2.2.1 System Failure 2.2.2 Delayed Data
	2.3 Failure to show 2FA phrase	2.3.1 Connection/Channel Problems 2.3.2 Webserver didn't send anything	2.3 System Failure
	2.4 Data Buffer Overflow	2.4 Too much information was sent to the jump host	2.4.1 System Failure 2.4.2 Data Loss
	2.5 Receiving Wrong 2FA Information	2.5 Webserver sent old or wrong data	2.5.1 System Failure 2.5.2 Delayed Data
3. Webserver	3.1 Failure to connect to LDAP	3.1.1 LDAP is Offline 3.1.2 Wrong Credentials Provided	3.1.1 System Failure 3.1.2 Delayed Data
	3.2 Data Loss During Transmission	3.2 Connection/Channel Problems	3.2.1 Loss of Data 3.2.2 Delayed Data

With all the Failure Modes listed above with their respective Failure Cause and Effect, it is clear to see what part of the system and items are more prone to failure. Obviously the items responsible for more critical operations like the LDAP data manipulation is more important since it can lead to data loss, than having a bad channel which would lead to delayed data.

2. Severity of Failures

With the Failure Modes listed in table 1.1, the effects of said failures can be looked at in terms of how severe they are and how they rank according to each other.

Table 1.2: Looking at each failure effect and its severity, ranking and comparing costumer and process effect

Effect	**Severity of Effect on Product (Costumer Effect)**	**Severity of Effect on Process (Process Effect)**	**Rank**
System Failure	Adds delays on operation for workers for overall system. Confidence in product is reduced per System Failure	Stops worker from accessing the system	10
Loss of Data	System is missing critical information for its operation resulting in more transmissions or manual work	Data and transmission would have to be repeated	9
			8
			7
			6
			5
Delayed Data	Delayed work and annoyance for workers	More network load	4
			3
Duplicate Data	Internal problems can occur for the system which adds unnecessary data	More bloated database	2
			1

The overall system works in a way that would make it easy for a worker to access and work within the system. The process is handled via network communication between several applications and systems. If a lot of system failures are occurring, the worker cannot do their job which they are paid to do. This reduces the confidence in the product and add delays to what the worker was supposed to do within the system. The overall result of this is that the worker can't do anything since the problem is out of their hands. This is also the reason why the System Failures are ranking the highest since it is the only one where work can potentially fully stop.
Problems related too Loss of Data would cause problems for the worker in the sense that they would have to repeat work if the system isn't capable of recovering what was lost. This would add delays and annoyance for the worker. The lesser problems which could cause delayed and duplicated data are less of a problem since they won't effect the overall work being done by a worker. They would only notice slight delays if there are problems with the network.

3. Probability of Occurrence

The next step is to rank the probability or likelihood of the effects occurring. This compliments the previous ranking and gives an idea of how often different failure effects happen within the system. Since in depth testing of failure occurrence wasn't performed, the ranking is put according to experience with the failures during design, implementation and testing of the system. For a better determination for the probability of occurrence you would have to do further tests for production ready like scenarios.

Table 1.3: Failure Effect Occurrence Ranking

Likelihood	**Failure Effect**	**Rank**
Very High		10
		9
High	Loss of Data	8
		7
Moderate	System Failure	6
		5
	Delayed Data	4
Low		3
	Duplicate Data	2
Very Low		1

The system that was made didn't take system load into account on the basis that the system isn't supposed to be used en masse by a lot of people. The system would therefore have less likelihood of having problems such as collisions because of traffic endured during operation. One of the key areas where problems could occur is when one actually happens. Since there is not a lot of handling of errors or checking of data, if something goes wrong it could lead to a series of failures which could have been prevented.

4. SO-Schema

To get a better overview of the FMEA, a SO schema is made which showcases the prioritization of failure effect that happen within the system. The SO schema take both severity and occurrence/likelihood probability into consideration and presents them both for each failure effect. The priority is assigned and determined based on the SO ranking. The three priority areas are as follows:

- 1. Critical Action Required
- 2. Significant Action Required
- 3. Annoyance Zone

Depending on the SO ranking, a failure effect can be completely omitted from the priority list. In this case the failure effect is so minor that it can be ignored since its severity and occurrence are low. On table 1.4, the different items, their failure effects, SO ranking and priority can be seen.

Table 1.4: SO-Schema Index with priority

Item	**Failure Effect**	**S**	**O**	**SO**	**Priority**
LDAP server jump host	System Failure	10	6	60	1
	Loss of Data	9	8	72	1
Webserver	Delayed Data	4	4	16	3
	Duplicate Data	2	2	4	-

From table 1.4 it can be seen how the failure effect are distributed by SO ranking and priority. Failure effect with the critical priority, 1, are the ones that are most important to take off from a design standpoint. The less important failure effects can be see for the Delayed Data effect which got the priority 3, which is in the annoyance zone. For this priority, an immediate action to fix the failure effect isn't needed since it doesn't appear high in the priority list. The failure effect should still be treated and to do so, different methods can be introduced to mitigate lower priority failure modes.

5. Probability of Detection

When looking at the things that can go wrong within a system such as the faults and effects that can occur, it is also important to address them in the sense of easily they would be to detect and determine if they can even occur within the system with a good detection system. In a good detection system, a failure cause or failure mode cannot occur and is fully prevented by the design of the system. An example is having measures put in place to handle them without compromising the rest of the system. How well the detection performs is also determined by how much effort is put into the system. Having measures in place for every failure mode in a system can be expensive and because of that, some trade-offs have to be made in order to handle the most severe and likely to occur failure modes rather than spending resources on handling failure modes which are located in the Annoyance Zone of the SO index. On the other hand, if the Annoyance Zone failure modes are easily detectable and prevented, resources can be put into that.
As such, the probability of detection for the different failure modes has the following layout:

Table 1.5: Probability of Detection table showing eahc failure mode and its detection.

Opportunity For Detection	**Failure Mode**	**Rank**	**Likelihood of Detectoin**
No detection opportunity	-	10	Almost Impossible
Not Likely to detect at any stage	-	9	Very Remote
Post Design Freeze and prior to launch	-	8	Remote
	-	7	Very Low
	1.6 Overwriting Wrong Data 2.5 Receiving Wrong 2FA Information	6	Low
Prior to Design Freeze	1.7 Wrong Data Added 2.4 Data Buffer Overflow	5	Moderate
	-	4	Moderately High
	1.3 Cannot Read Data 1.4 Cannot Add Data 2.3 Failure to show 2FA phrase 3.2 Data Loss During Transmission	3	High
Virtual Analysis-Correlated	1.1 Failure to connect 1.2 No response 1.5 Added Duplicate Data 2.1 Failure to connect to LDAP 2.2 Failure to connect to Webserver 3.1 Failure to connect to LDAP	2	Very High
Detection not applicable; Failure Prevention	-	1	Almost Certain

6. Risk Priority Number

Now that the Severity, Occurrence and Detection numbers have been found, a Risk Priority Number (RPN) can be computed for each fault/failure in the system. A RPN determines the combined severity, occurrence and detection for effects of the faults found within a system. Effects of faults with a high RPN are deemed as effects where action needs to be taken to minimize the effect. It gives similar information that the SO index does but expands it with detectability to show how easily the effect can be prevented. Similar to the SO index number, the RPN is computed as follows:

$$RPN = Severity \cdot Occurrence \cdot Detection \tag{1.1}$$

Where the S, O and D are ranked as previously where rank 1 is considered low for severity and occurrence but high for detection and rank 10 is considered high for severity and occurrence but low for detection.
Looking at the RPN for an individual effect doesn't tell us if it is high or low compared to the rest of the system. Therefore, each RPN should be compared to the rest of the RPN's of the system to get an idea of what RPN can be considered high.

Table 1.6: RPN table showing each failure modes RPN.

Failure Mode	**S**	**O**	**D**	**RPN**
1.1 Failure to connect	7	2	2	**28**
1.2 No response	9	3	2	**54**
1.3 Cannot Read Data	6	2	3	**36**
1.4 Cannot Add Data	6	2	3	**36**
1.5 Added Duplicate Data	5	2	2	**20**
1.6 Overwriting Wrong Data	7	2	6	**84**
1.7 Wrong Data Added	7	2	5	**70**
2.1 Failure to connect to LDAP	7	2	2	**28**
2.2 Failure to connect to Webserver	7	2	2	**28**
2.3 Failure to show 2FA phrase	8	3	3	**72**
2.4 Data Buffer Overflow	9	1	5	**45**
2.5 Receiving Wrong 2FA Information	8	2	6	**96**
3.1 Failure to connect to LDAP	7	1	2	**14**
3.2 Data Loss During Transmission	9	2	3	**54**

To see the RPN for the different faults within the system, it is easier to look at the FMEA schema which contains everything that has been done so far and will give a better overview of how each RPN for an effect ranks compared to other RPN's.

7. FMEA schema

To get an overview of everything that has been done so far in the FMEA, a table is made where everything is added. This includes the different failure modes, effects and causes as well as the SOD+RPN numbers for each failure. The FMEA schema also adds another column to the table which is the "Action Recommended" which gives an example as to what should be done if the failure has occurred. As such, the FMEA schema can be seen on table 1.7.

Table 1.7: FMEA Schema including, items, failures, effects, severity, cause, occurrence, detection, RPN and actions

Item	Failure Mode	Failure Effect	S	Failure Cause	O	D	RPN	Actions Recommended
1. LDAP server	1.1 Failure to connect	1.1 System Failure	7	1.1.1 System is Offline 1.1.2 Wrong Credentials Provided	2	2	28	Documented Procedure and implementation. Make sure data references and transfers correspond correctly to the API and documentation used.
	1.2 No Response	1.2.2 System Failure	9	1.2 Invalid/Malicious Query	3	2	54	Check system status and physical connections.
	1.3 Cannot Read Data	1.3 Loss of Data	6	1.3 Corrupted Data	2	3	36	Check database and software settings and errors.
	1.4 Cannot Add Data	1.4 Loss of Data	6	1.4 Corrupted Data	2	3	36	Ref. 1.3
	1.5 Added Duplicate Data	1.5 Duplicate Data	5	1.5 Missing Exception Handling	2	2	20	Ref. 1.1
	1.6 Overwriting Wrong Data	1.6 Loss of Data	7	1.6.1 Wrong Index Location 1.6.2 Missing Exception Handling	2	6	84	Ref. 1.1
	1.7 Wrong Data Added	1.7 Loss of Data	7	1.7 Information wasn't written correctly	2	5	70	Ref. 1.1
2. Jump Host	2.1 Failure to connect to LDAP	2.1.1 System Failure 2.1.2 Delayed Data	7	2.1.1 LDAP is Offline 2.1.2 Wrong Credentials Provided	2	2	28	Check system status and physical connections. Make sure systems point to the correct location.
	2.2 Failure to connect to Webserver	2.2.1 System Failure 2.2.2 Delayed Data	7	2.2.1 Webserver is Offline 2.2.2 Mismatch in User Agent 2.2.3 DNS Errors	2	2	28	Ref. 2.1
	2.3 Failure to show 2FA phrase	2.3 System Failure	8	2.3.1 Connection/Channel Problems 2.3.2 Webserver didn't send anything	3	3	72	Communication could be unstable or a bad implementation of data transfers may cause data to be lost.
	2.4 Data Buffer Overflow	2.4.1 System Failure 2.4.2 Data Loss	9	2.4 Too much information was sent to the Jumpserver	1	5	45	Implementation problems. Data should be restricted as to not allow overflows.
	2.5 Receiving Wrong 2FA Information	2.5.1 System Failure 2.5.2 Delayed Data	8	2.5 Webserver sent old or wrong data	2	6	96	Data should be checked before and after being sent to ensure integrity.
3. Webserver	3.1 Failure to connect to LDAP	3.1.1 System Failure 3.1.2 Delayed Data	7	3.1.1 LDAP is Offline 3.1.2 Wrong Credentials Provided	1	2	14	Ref. 2.1
	3.2 Data Loss During Transmission	3.2.1 Loss of Data 3.2.2 Delayed Data	9	3.2 Connection/Channel Problems	2	3	54	Ref. 2.3

8. Conclusion

The idea behind an FMEA schema just like in table 1.7 is to have an overview of all the failures that can occur within a system during the entire course of the development stage. Every time something changes, be it design changes, operating conditions, new regulations or when a customer gives feedback on a problem they found, the FMEA and the FMEA schema should be updated accordingly. From table 1.7 it can be seen that there are some failures with high RPN values which, as far as development goes, should be looked at first, before looking at the lower value RPN failures. In this example, the failure 2.5 would be looked at. When failure 2.5 has been rectified, the FMEA schema would be updated.
The FMEA isn't all advantages though. Though FMEA can help improve the product you are making such as reducing failures now and in the future as well as improving many key areas such as customer satisfaction, FMEA also has some limitations. The first limitation is what FMEA actually shows. In most cases the FMEA only gives a top-down view of the problem that you are working with. The exact implementation of the failure and what caused it can't be displayed in detail. This would require additional documentation separate from the FMEA which describes each failure in detail, the effect and causes in detail as well as how to rectify it properly. Another limitation is the computation of the Risk Priority Numbers (RPNs). When multiplying the Severity, Occurrence and Detection ranks, the resulting RPN could result in a value that actually identifies a less serious failure as a serious failure because of the combination of the SOD ranks. An example is a failure with low Severity, high occurrence and low detection. This would result in a high RPN value even though the failure is not a serious failure. As mentioned earlier, the RPN only tells us if the ranking is better or worse, but not by how much. Sure, the numbers indicate by how much but not properly. A RPN of 100 is not twice as bad as a RPN of 50 since it all depends on the SOD rankings used for the calculation. As such, each RPN should be used as an indicator, but an analysis of the fault compared to the number is required to see if it is necessary to handle that specific failure before other failures.

1.2 Detection and Isolation of Faults and Failures

Knowing what kind of faults and failures can appear within a system is good to know so you know what to expect when they happen. But knowing that they can happens isn't good enough. Knowing how and when they happen is important in the context of trying to detect and stop faults and failures from potentially damaging critical system components. The first type of detection happens during operation of the system. In this case, making sure that specific operations don't go wrong should be detected. The second is during initialisation of the system. Since the system should operate in a semi-autonomous state, making sure that every possible operation that can be carried out should be tested before the system enters an operational state.

When it is detected the system is in a faulty state, it is important to know in which faulty state it is in, so that causes can be identified and corrected. Failure mode should be isolated using various tools, such as Hidden Markov Models. In the following subsection, we design an HMM which isolates the faulty states the system was in using a sequence of effects observed.

1.2.1 HMM for fault isolation:

We are gonna use a HMM for fault isolation in the Jumphost.

In our model, the transition states are the error states of the Jumphost. The system emits the different faulty effects during transitions from state to state, where the symbol emitted only depends on the arriving state.

The Jumphost is modelled as a component that can be in 7 different faulty states, with 3 types of effects (which will be the emitted symbols of the HMM), as said in the FMEA.

The purpose of this will be to detect in what error modes the system is, by looking at an observed sequences of effects thanks to the Viterbi Algorithm, which computes the most likely sequence of error modes that could emit those consequences states.

Being able to isolate the states we are in makes it easier to recover from faults afterward.

It should be kept in mind this example is only theorical. We were not able to run tests to see how the system behaves in a real situation, and for us to observe different error effects and the system to be in different faulty states, the Jumphost would have to be really broken at this point.

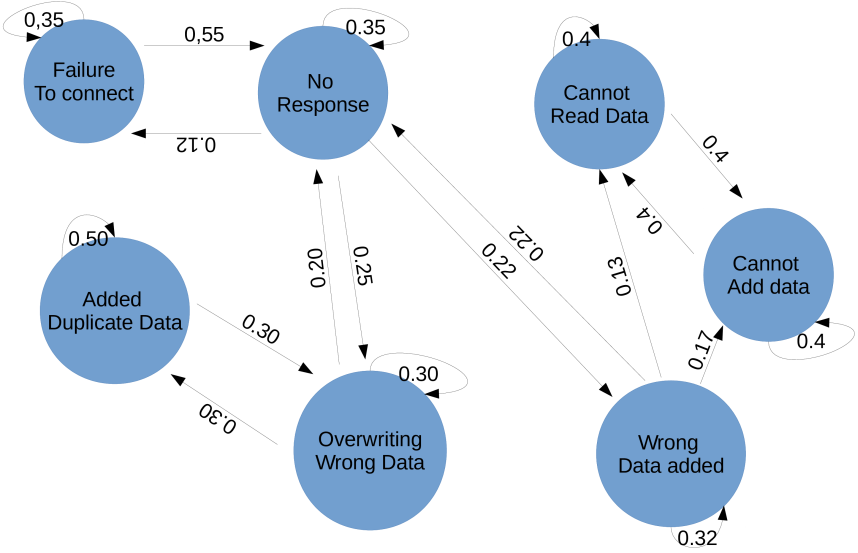

Figure 1.2: Reliability over time

This figure shows the different transition states, with the probability of transitioning from one state to another (we consider than the probability of being in a certain state depends on the previous states, notably because the failure mode causes can be the same). It is not represented of the figure but we consider there is always a

	System Failure	Duplicate Data	Loss of Data
Failure to connect	0.94	0.03	0.03
No Response	0.90	0.05	0.05
Wrong Data Added	0.08	0.08	0.84
Cannot Read Data	0.05	0.05	0.90
Cannot Add Data	0.12	0.09	0.79
Overwriting Wrong Data	0.04	0.04	0.92
Added Duplicate Data	0.09	0.82	0.09

probability of going to every failure mode, even when the states are not linked: the probability of going from a state to another not linked state is the same for every of those state, so that the sum of the probability of going from the start state to another is equal to one.

Here is the table we use for our computation. Keep in mind the symbol emitted only depends on the arrival state, and that the emission probabilities are arbitrary, based only on our FMEA analysis.

To use the Viterbi algorithm in a real situation, a sequence of observed effects have to be input. The algorithm will then return the most likely sequence of faulty states that led to this situation. This can then be used for repair purposes.

As an example, with the parameters we have chosen, if we observe the effects System Failure -¿ Loss of Data -¿ System Failure -¿ Duplicate Data, according to our algorithm we have most probably experienced the following sequence of faulty states : No Response -¿ Overwriting Wrong Data -¿ Duplicate Data added -¿ Duplicate Data added

1.3 Recovery of Faults and Failures

After a fault or failure has been detected, it is important to treat it by doing some recovering of what happened and making sure that the damage done by the fault or failure doesn't effect the rest of the system and that it doesn't happen again. In cases where a channel is bad or unreliable, data could be lost during transmission which would result in data loss. From section 1.1 it was seen that there are multiple places where data could be lost during transmissions. To help combat this problem, fault recovery can be used. By isolating transmissions, it can be analyzed and seen if an fault happens during operation of the transmission. When the fault happens, fault recovery is applied by using a fault detector. The detector will have the ability to check if a transmission succeeded or failed. In this section different fault recovery methods will be analyzed. The first area is fault handling where the prevention of faults being activated again is done and the second area is error handling where the elimination of errors from the system state is looked at.
In this case the following methods are looked at which belong to the error handling area:

- Forward/Backward Recovery
- Replace versus Restart versus Check-Pointing

1.3.1 Forward Based Recovery

The first type of recovery is the forward based recovery. This type of recovery, as seen on fig. 1.3, is based on detecting an error present in the task currently happening, aborting the task and start a subsequent task.

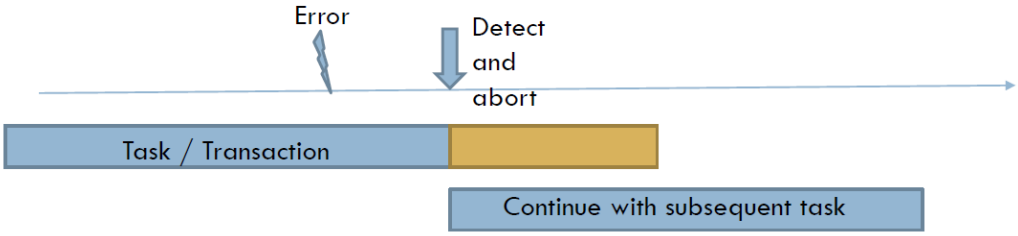

Figure 1.3: Forward based Recovery visualised

When doing this type of recovery, the task where the error happened is stopped which means that the resources used during the task are lost. This can be seen as a bad thing since time and resources are wasted but on

the other hand, it ensures that the subsequent task starts. When looked at in context of the report, forward based recovery cannot be used for task such as transmissions where data is lost. If data is lost, the task cannot be completed since the data is necessary for the task to be completed. If forward based recovery was used, the system would enter a system failure state since the purpose of the system couldn't be achieved. A place where forward based recovery could be used is with mass UDP traffic where a failed transmission is okay to be discarded since future tasks in the system don't rely on every transmission being successful.

1.3.2 Backward Based Recovery with Restart

The second type of recovery is backward based recovery. Using this recovery type, as seen on fig. 1.4, utilises the ability to restart a task that has previously failed. Compared to the forward based recovery, here the failed task is done again instead of discarding it.

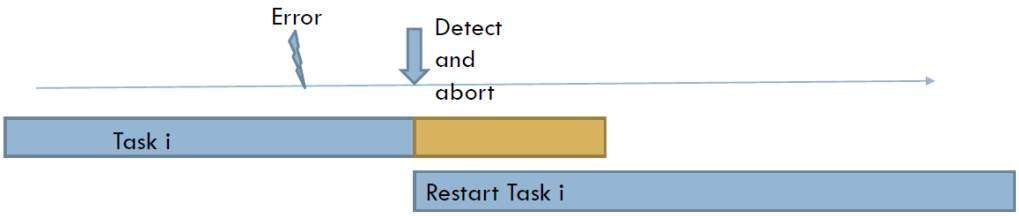

Figure 1.4: Backward based Recovery with Restart

Using backward based recovery is interesting to look at compared to forward based recovery in regards to the system presented from the project. The reason backwards based recovery is more relevant is because of the structure of the tasks that have to be performed. The presented system relies on the task being successful which means that having a failed task restart is a good thing since failed tasks are a bad thing for the system. In this case the tasks a network transmissions where important data is being sent. If the tasks fail, a system failure will occur which isn't an option for the system. In this case using backwards based recovery, no tasks that fail will be aborted and skipped and ensures that all of the important data that needs to be sent arrives at its destination without a failure.
The obvious advantage of using backward based recovery is that no task is lost, but if implemented in an bad manner, backward based recovery could lead to more problems than it solves. In the case of the bad implementation, if a task continues to fail and restart, the system enters a softlock state where it would never start the next task since the first task keeps failing. This results in an even bigger increase of execution time which results in decreased throughput of the system. To combat this problem, an expected number of restarts are implemented which ensures a set number of restarts that can happen. The expected number of restarts can be estimated by the probability of the task failing. By having a set number of restarts that can happen, softlocks can be avoided and would result in a smaller increase in execution time. After the set number of restarts has been reached, a fallback method can be used such as switching to forward based recovery where the task is deemed lost since the problem could have been the actual task and not how the task was executed. An example is a network transmission where the data sent is corrupted but the actual network and channel are fine.

1.3.3 Backward Based Recovery with Check-pointing and roll-back

The last type of recovery is the backwards based recovery but with added check-pointing and roll-back capabilities. Instead of restarting the entire task like with the backwards based recovery with restart, this recovery type has the ability to create "check points" during a task and if the task fails, the entire task wouldn't have to be restarted but instead can be restarted from the saved check point. As seen on fig. 1.5, the total execution time is only increased by the execution time lost which results in a lower overall execution time compared to the backwards recovery with restart.

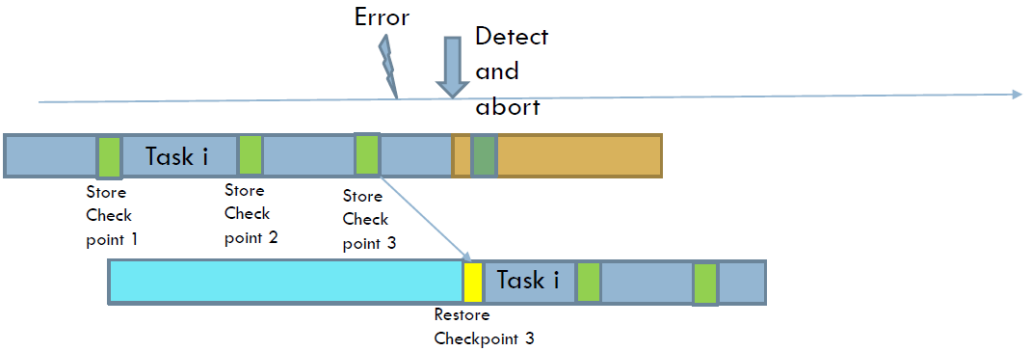

Figure 1.5: Backward based Recovery with Check-pointing

Implementing a check-point with roll-back system is a harder task to implement and wouldn't be applicable to every kind of task. Tasks which execute really fast wouldn't gain anything from using check-pointing since the overhead would probably add more execution time than the entire task. The benefit comes from tasks with a big execution time where check-pointing can cut down the execution time for restarts. The check-pointing also depends on where the failure appeared. If the latest check-point was made after the failure, then using that check-point would result in a repeated failure since the failure actually occurred before the check-point. This can, however, be alleviated by using a fault detector which can pinpoint when the failure happened and as such can choose a check-point before the failure happened.
When compared to the system shown, backwards based recovery with check-pointing could be implemented, but with slight changes as to what is considered a task within the system. As of now, a task was considered as a single network transaction such as sending some packets from a client to a server. If check-pointing was to be implemented, a rework would have t be done such as encapsulating several network transactions under a single task in order to have enough execution time for check-pointing to work.

In relation to the system where communication is a big factor, the check-pointing could be done between each transaction before important data is sent such as user information. If the user information is lost during transaction, the task could be repeated again from the point before the user information was sent. Detecting if the user information didn't reach its destination could be achieved by making sure that acknowledgement packets are sent and received properly. If not, a timeout would kick in and indicate that something went wrong.

To calculate the check-pointing, the following can be assumed and done:

- Task Duration: T
- Duration per checkpoint: C
- Exponential time to error
- Rate: µ

Duration of task slice and the check-pointing effort can be calculated as follows:

$$\frac{T}{(N+1)+C} \tag{1.2}$$

The probability that the task slice and checkpoints are successful can be calculated as follows:

$$p_s = e^{-\mu\cdot(\frac{T}{(N+1)+C})} \tag{1.3}$$

The expected duration for the slice and checkpoint is therefore:

$$\frac{\frac{T}{(N+1)+C}}{p_s} \tag{1.4}$$

To see how a task behaves over different amounts of checkpoints, the expected duration is multiplied by the number of checkpoints that for a task. This can be applied to an example for the system in Matlab as a short script where the probability and expected duration are calculated at different amounts of checkpoints. As such, the following Matlab script shows an example of a system with backward based recovery with check-pointing. The task performed here is a transmission task where a file is sent from a mobile device through multiple servers on an unstable network. For this example, the task duration is ~ 5 seconds with each checkpoint having a duration of 1/5 second and with an error rate of 0.1.

```
T = 5;              % Task Duration
C = 0.2;            % Duration per checkpoint
mu = 0.1;           % Error rate
N = 1:20;           % Number of checkpoints
mdur = zeros(1,length(N)); % Mean duration

for i=N                                     % Loop N times
    i = i-1;                                % Start at index 0
    ps = exp(-mu*(T/((i+1)+C)));            % Success probability
    edur = (i*(T/(i+1)+C)+T/(i+1))/ps;      % Expected duration
    mdur(i+1) = edur;                       % Add to array
end

plot(N,mdur);       % Plot durations
```

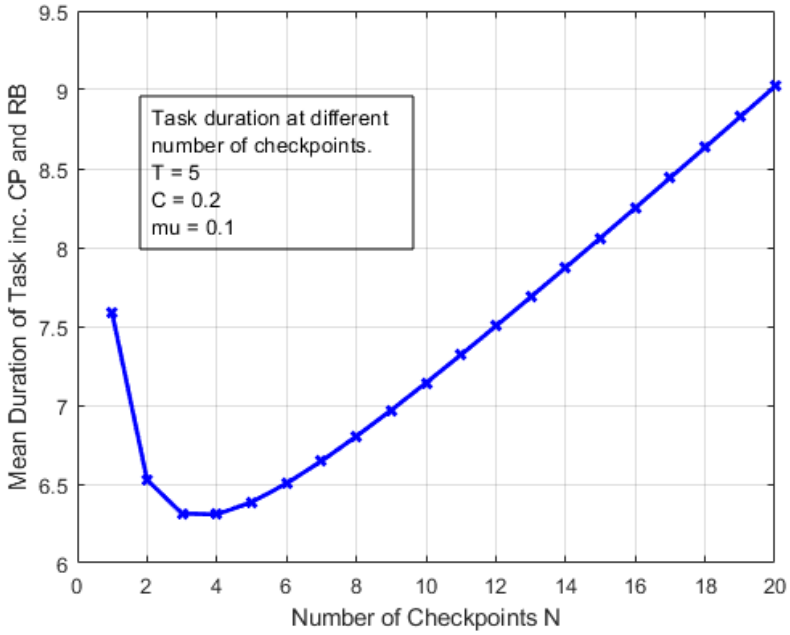

Figure 1.7: Example showing checkpoint recovery applied to a task with duration 5 and error rate 0.1

On fig. 1.7 it can be seen how the mean duration changes depending on how many checkpoints are used for the task. For the first couple of checkpoints it can be seen that a big reduction in mean duration is achieved. However, after a certain number of checkpoints the mean duration begins to shift in the opposite direction. In this case the turning point is 3 or 4 checkpoints which is the optimal amount for the parameters provided in the example. The reason for the turning point after 4 checkpoints is that more resources are used on checkpoints than what is saved by having them which in turn increases the mean duration of a task. It can also be seen from the equations that with an increase of task duration, which could happen with the network connection were to be very slow, the amount of checkpoints would increase with it. The mean duration would also see a big drop after the first couple of checkpoints at higher durations.

1.3.4 Conclusion

In the last couple of sections, different types of recovery methods were looked at in regards to the system used for the semester project. The different recovery methods included forward based recovery and backward based recovery with different additions as to how recovery was handled. From the different methods it was seen in regards to the system that not every method could be used since the system was very communication focused on smaller tasks that would be carried out. Because of this, each task had to be seen as a bigger communication task such as the process of connecting to a system and sending it some data. By grouping several smaller tasks into a bigger one, backward based recovery with restart and check-pointing could be implemented for several tasks. Recovery with restart could be used for task that failed completely why recovery with check-pointing

could be used for tasks where smaller tasks within failed and could be restarted from a check-point before the failure.

It was also seen that recovery for the actual software running within the system could also benefit from different recovery methods if the software were to be unstable and prone to failure during operation. This could have been things such as adding/reading data to/from a database. In this case backward based recovery could be used where data was backed up in case a failure occurred, which would give the ability to restart and check-point a task.

1.4 Availability/Reliability Analysis

The reliability of a system measures how long it can perform without suffering problems. It can be characterized by a mathematical function R, where R(t) is the probability of the system being up at time t.

Computing the reliability of the system is useful to see whether the system is robust enough for Telenor's purposes, If the average time it is on is too small, then we will now we have to change paramaters to make the system tougher, and if is is very large compared to the time we need, maybe we can make some changes to make the system faster in certain areas, for example.

In order to calculate the reliability, we model the system as such: *It is composed of 4 components (Jumphost, LPAD Server, VPN, Critical System) *All components can be in two states: they can be either working normally or broken *There are dependencies between the states of the components (for example, if the LPAD is down with the error Failure to connect, we cannot connect to the Jumphost which makes it broken as well)

To represent this model as a Markov Chain, we would then have to use 16 states which are representatives of the different system states, (we then have to use 2 power 4 states).

We can compute 2 types of reliability: one where as soon as a component is down we consider the system is down as well, or one where we consider the system down when its most important part, the critical system, is down. We will call that the critical reliability.

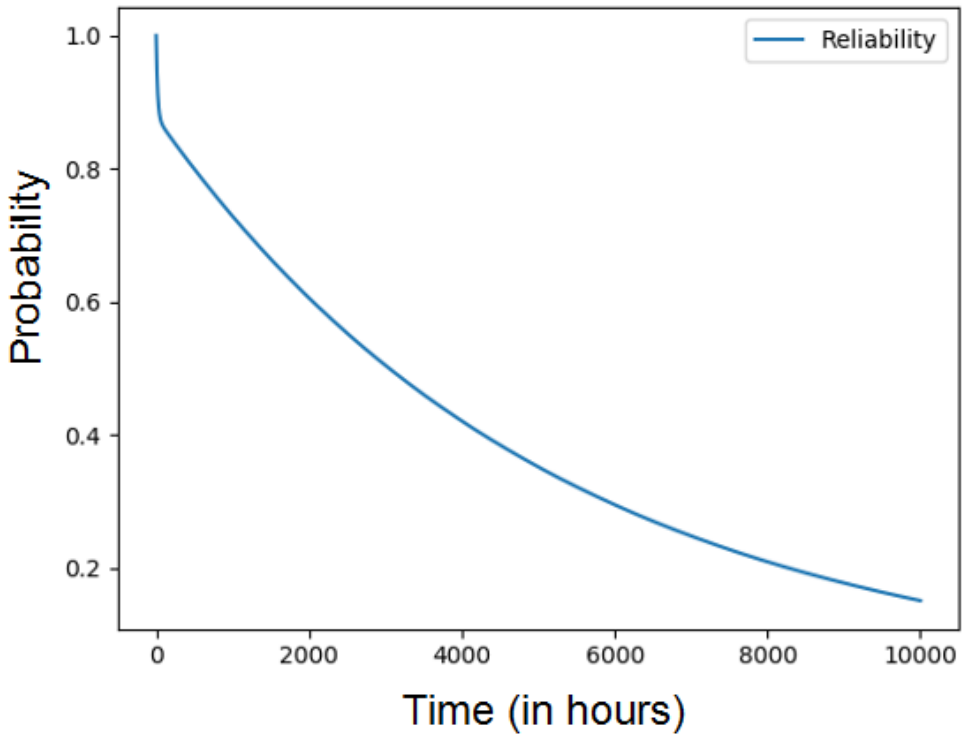

Figure 1.8: Reliability over time

This figure shows that the reliability starts by decreasing a lot linearly during the first 120 hours, and after that it decreases in a logarithm-like way.

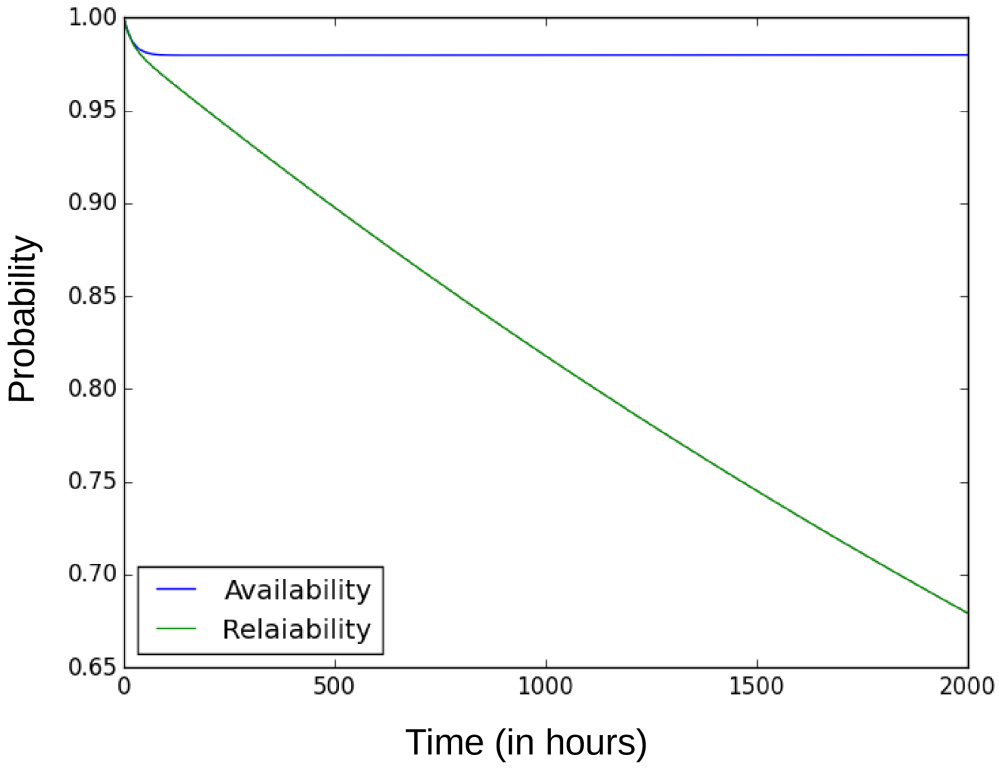

Figure 1.9: Critical Availability and Reliability over time

In the second figure, we also include critical availability. The availability, which is less relevant than reliability in our case, can be used to characterize the system ability to be up over time. It is described by a mathematical function A, where A(t) is the probability of the system being up at time t.

We can see that critical reliability seems to be decreasing in a logarithm-like way over time. Critical availability stabilizes at 0.98 after 100 hours.