

Analysis of the Space Propulsion System Problem Using RAVEN

Diego Mandelli*, C. Smith, A. Alfonsi, C. Rabiti

Idaho National Laboratory, Idaho Falls (ID), USA

Abstract: This paper presents a solution of the space propulsion problem using a PRA code currently under development at Idaho National Laboratory (INL). RAVEN (Risk Analysis and Virtual control ENvironment) is a multi-purpose Probabilistic Risk Assessment (PRA) software framework that allows dispatching different functionalities. It is designed to derive and actuate the control logic required to simulate the plant control system and operator actions (guided procedures) and to perform both Monte-Carlo sampling of randomly distributed events and Event Tree based analysis. In order to facilitate the input/output handling, a Graphical User Interface and a post-processing data-mining module are available. RAVEN also allows to interface with several numerical codes such as RELAP-7, RELAP5-3D and ad-hoc system simulators. For the space propulsion system problem, an ad-hoc simulator has been developed in Python and then interfaced to RAVEN. Such a simulator fully models both the deterministic (e.g., system dynamics and interactions between system components) and the stochastic behaviors (e.g., failures of components/systems such as distribution lines and thrusters). Stochastic analysis is performed using random sampling based methodologies (i.e., Monte-Carlo). Such analysis is accomplished in order to determine both the reliability of the space propulsion system and to propagate the uncertainties associated with a specific set of parameters. As also indicated in the scope of the benchmark problem, the results generated by the stochastic analysis are used to generate risk-informed insights such as conditions under which different strategies can be followed.

Keywords: propulsion system, dynamic PRA, Monte-Carlo

* Corresponding author: Diego Mandelli, diego.mandelli@inl.gov

1 INTRODUCTION

For the PSAM12 conference, a benchmark problem was issued with the intent of gathering people actively researching in the PRA field and have them work on the solution of a common problem. The problem statement (see Section 2) was structured in such a way that its essential features would challenge the capability of classical event-tree (ET) and fault-tree (FT) algorithms. In this respect, this paper presents a solution to such problem. Complete system dynamics and full system features have been implemented as an ad-hoc simulator code written in Python. Implementation details are provided in Section 4. We used the code RAVEN [1,2] to perform the statistical analysis using its Monte-Carlo sampling algorithms. RAVEN code description is shown in Section 3. Once the coupling between the system simulator and RAVEN has been established, a set of Monte-Carlo runs [3] has been performed in order to evaluate the temporal profile of system failure probability. Results are shown in Section 5.

2 BENCHMARK DESCRIPTION

An ion propulsion system is needed for a science mission in order to reach the orbit of a distant planet. The system is pictured in Fig. 1 (left) and consists of the following components:

- 1 Tank
- 1 Set of Distribution Lines
- 5 Propulsion thrusters (Fig. 1 right), each composed of:
 - 1 Propulsion Power Unit (PPU)
 - 2 Ion Engines: A (main engine) and B (backup engine)
 - 2 Valves

When a thruster is operating, the PPU provides power to just one ion engine. The other engine is in a standby mode, unless failed. When the number of engines required is greater than the actual number of engines available then the mission fails.

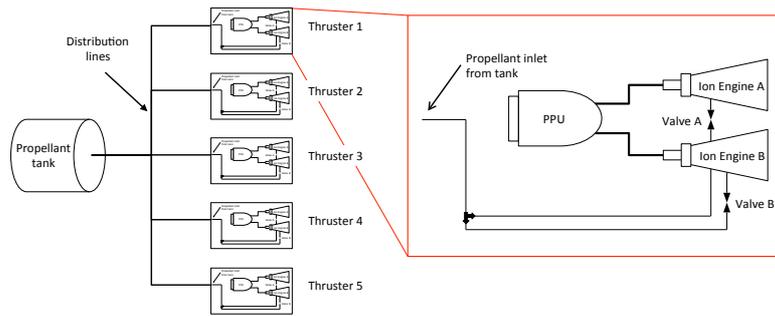


Figure 1: Schematics of the propulsion system (left) and of a thruster (right): PPU, ion Engines and valves

The standby thrusters remain in standby until they are needed to replace a failed thruster, and they are actuated in series. The strategy for thruster operation is to begin with:

- Power from the PPU going to Ion Engine A
- Open valve A (to provide propellant to the Ion Engine)
- Start Engine

Ion Engine A continues to be the operating engine of the thruster until the ion engine A fails. If engine A fails, the strategy is to:

- Close valve of engine A
- Shut down the PPU
- Switch the PPU to Ion Engine B
- Open valve of engine B
- Re-energize the PPU and operate with Ion Engine B.

Strategy to shut down the thruster is to:

- Shut-down engine
- Close valve of the ion engine
- Shut down the PPU

Failure mode and effect analysis for all the components considered are shown in Table 1 along with reliability data for PPU and Ion Engine.

Table 1: FEMA and failure mode probability values

Component	Failure Mode	Outcome	Value
PPU	Fails to start on demand	Thruster failure	1×10^{-4} (per demand)
	Failure to operate		1×10^{-6} (per hour)
	Failure to shut down on demand		1×10^{-5} (per demand)
	Fails to switch to Ion Engine B		2×10^{-6} (per demand)
Ion Engine	Fails to start on demand	Loss of redundancy or thruster failure	3×10^{-5} (per demand)
	Fails to operate		2×10^{-5} (per hour)
	Fails to shut down on demand		3×10^{-6} (per demand)
Propellant Valve	Failure to open on demand	Loss of Engine Mission failure	See Section 2.2
	Failure to close on demand		

2.1 Mission Profile

The phase mission is described schematically in Figure 2 that shows the time length of the mission (78,000 hours) and the strategy regarding the number of thrusters needed in specific time intervals. From Fig. 2 note:

- Support station: this station can be used to refuel and recover any damage and failure. After stopping, all components are as good as new. Decision to stop at the support station can be made anytime before 18,000 hours
- Gray region: this region represents an area that contributes to increase leakage failure rates of the distribution lines (see Section 2.3) only when propulsion system is active.

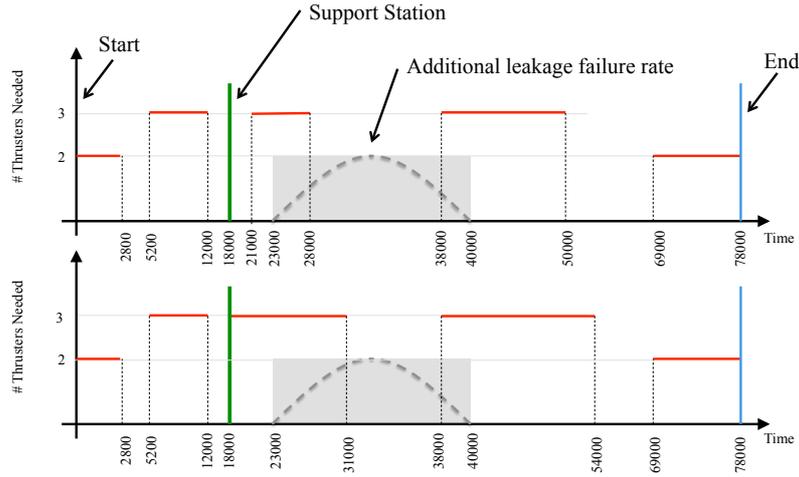


Figure 2: Mission scenarios – Scenario 1: without stop at the support scenario (top) and Scenario 2: with stop at the support scenario (bottom)

2.2 System Dynamics Interaction

The valve is subject to pressure shocks when the thruster is activated and deactivated. Following a change of the status of the valve there is a pressure oscillation ΔP around the nominal value with the following expression (time t in seconds):

$$\Delta P = \begin{cases} \sin(t\pi/4)\sin(t\pi/p) & t \in [0,4s] \\ 0 & elsewhere \end{cases} \quad (1)$$

where the parameter p is uncertain and characterized by a symmetric triangle distribution $[0.4,0.8]$. Such oscillation causes a damage accumulation d at each valve activation (either opening or closing) that can be modelled as:

$$d = \int_0^4 dt |\Delta P| \quad (2)$$

The probability of failure on demand p_{valve} of the valve as a function of the damage d is as follows:

$$p_{valve} = (e^d - 5) \cdot 10^{-4} \quad (3)$$

Remember that if the valve fails to open, only the associated ion engine is lost; if the valve fails to close, mission fails.

2.3 Leakages

Both distribution lines but and the fuel tank are subject to leakages and the failure rate for such events are as follows:

- Distribution lines: the damage accumulation $D(t)$ of the distribution lines can be modeled as a Gaussian random walk (Brownian motion) [4] having mean value (drift) equal to 1 and sigma equal to 0.4. In the gray region (see Figure 2) and only when propulsion system is active (i.e., thrusters are running), damage accumulation for the distribution lines is still considered a random walk but having mean value equal to 1.3 and sigma equal to 0.6. Distribution lines fail when $D(t) = 80,000$; when this happens the mission is lost. Note that, for this benchmark extension, the leakage model for the distribution lines replaces the reliability value reported in Table 2.
- Fuel tank: leakage failure rate = 10^{-5} hour⁻¹. When a leakage event occurs in the fuel tank, the rate r of fuel leaking out (in unit of fuel per hour) has been experimentally observed and modeled as a Gaussian distribution characterized by:

- Mean = 500
- Standard deviation = 100

Automatic repair system is available only for the fuel tank. Repair time is exponentially distributed with a value of lambda $\lambda = 24$ hour.

Thus, the amount of fuel leaked is equal to the time needed for the repair multiplied by the rate of fuel leaking out. Note that the size of the leakage and the time required to fix such leakage determines how much fuel has been lost and its contribution to reaching mission failure.

2.4 Initial Conditions Uncertainties

The minimum amount of fuel required for this mission is 110,000 units but an additional amount of fuel is loaded. The additional amount is uncertain and can be modelled with a uniform distribution between 5,500 and 16,500 units of fuel. Each working ion engine requires 1 unit of fuel each hour.

2.5 CCF Data

Common cause failures (CCF) should be assessed using the conditional probability values listed in Table 1 by the CCF model of choice. No specific CCF model is endorsed, but any simplification or approximation of CCF probabilities must be based on calculations using the values below.

Table 2: Common Cause Failure Modeling Values

Group Size	Conditional Failure probability [%]
2	8.0
3	4.0
4	2.0

3 RAVEN CODE

RAVEN is a generic software driver to perform parametric and probabilistic analysis of code simulating complex systems. Initially developed to provide dynamic risk analysis capabilities to the RELAP-7 code, it is currently being generalized with the addition of Application Programming Interfaces (APIs). These interfaces are used to extend RAVEN capabilities to any software as long as all the parameters that need to be perturbed are accessible by inputs files or directly via python interfaces. RAVEN is capable to investigate the system response probing the input space using Monte Carlo, grid strategies, or Latin Hyper Cube schemes.

RAVEN has been developed in a highly modular and pluggable way in order to enable easy integration of different programming languages (i.e., C++, Python) and coupling with other applications including the ones based on MOOSE. The code consists of three main modules: Code interface, External Python Manager and Graphical User Interface (GUI).

The interface is the container of all the tools needed to interact with codes such as RELAP-7/MOOSE, RELAP5 and user defined codes (i.e., Python scripts). It has been designed to be general and pluggable with different solvers simultaneously in order to allow an easier and faster development of the control logic/PRA capabilities for multi physics applications.

The core of PRA analysis is contained in an external driver/manager. It consists of a Python framework that contains the capabilities and interfaces to drive a PRA analysis.

The GUI is compatible with all the capabilities actually available in RAVEN. Its development is performed using QtPy, which is a Python interface for a C++ based library (C++) for GUI implementation. The GUI is based on a software named Peacock, which is a GUI interface for MOOSE based applications and it is able to assist the user in the creation of the input.

3.1 Software Infrastructure

The external driver manager has the control of the different modules that support the DET calculation:

- General Calculation Driver and Job Handler

- Visualization and Input supporting GUI
- Dynamic Event Tree (DET) and Probability Engine
- Database manager
- Post-processing and Data Mining module
- Distribute Computing Environment Interface

The General Calculation Driver is in charge of communicating information among the different modules. It is the only software branch that is aware of which modules are participating in the calculation.

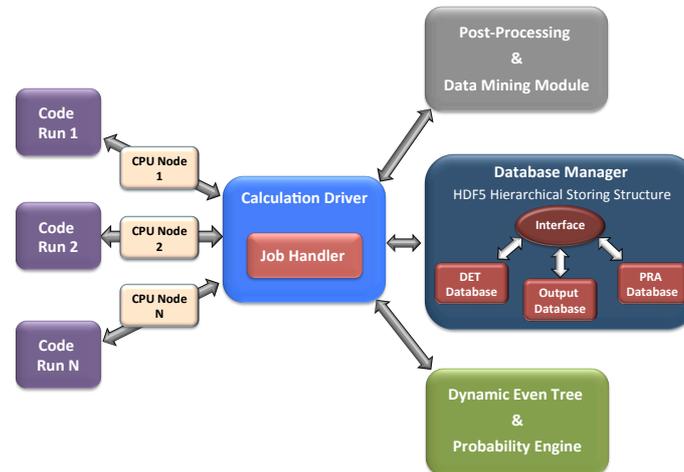


Figure 3: RAVEN code software structure

The Calculation Driver, through the Job Handler module, runs the simulator until a stopping condition is reached. The probability engine is in charge of computing the likelihood of branching generated by trigger signals (e.g. probability threshold on Clad Failure distribution overpassed).

The resulting data generated by the stochastic analysis, simulation probabilities, trigger information, and simulation results are sent to the Database manager, that, based on the kind of information received, distributes the data among the sub-structures (DET, PRA, Output databases). The database manager is able to store data in different formats (e.g., HDF5, CSV, etc.).

The user can decide to perform post-processing operations and/or data mining manipulation on the fly, through the Post-processing and data mining module. The whole calculation may be visualized through the RAVEN GUI that is able to exploit the communication capabilities present in the external calculation driver for following the simulation evolution (e.g., monitored variables or probability evolution through different branches, etc.).

The whole calculation infrastructure is designed to be agnostic regarding the machine in which it is running with extremely good performance either in PCs/workstations and HPC clusters.

4 RELIABILITY ANALYSIS

Scope of this benchmark is to 1) determine the time-dependent reliability of the propulsion system over the planned mission and 2) identify and rank the most risk significant components. From a safety point of view, mission failure occurs when one of the following conditions is satisfied before $t = 78,000$:

1. Number of thruster available is less than the number of thruster required
2. Fuel tank empty
3. Distribution lines accumulated damage reach threshold value $D(t) = 80,000$

In addition, it is needed to quantify the impact of support station on overall system reliability; a stop at the support station:

- Allows to fully repair the propulsion system
- Increase distribution lines damage accumulation (see Fig. 2 bottom)

5 SYSTEM MODELING

In this section, we show how the most relevant features of the benchmark problems described in Section 2 have been modeled and coded.

5.1 System Control Logic

The complete propulsion control logic has been implemented. For each of the following actions (that have been coded in Python) a pseudo code is provided:

1. Start single thruster

Pseudo code 1: StartSingleThruster()

```
do
  thrusterToStart = FindFirst() # get a thruster on stand-by
  if valveFailsToOpen then
    LoseEngine(thrusterToStart)
    StartSingleThruster()
  else
    if PPUFailsToStart then
      LoseThruster(thrusterToStart)
    else
      if engineFailToStart then
        LoseEngine(thrusterToStart)
      else
        Success = True
      endif
    endif
  endif
endif
while success
```

2. Lose engine

Pseudo code 3: LoseEngine(thruster=i)

```
if valveFailsToClose then
  missionFailure = True
else
  if availableEngines = 0
    LoseThruster(i)
  else
    if PPUFailsToSwitch
      LoseThruster(i)
    else
      switchToEngineB(i)
    endif
  endif
endif
```

3. Shut down all thrusters

Pseudo code 2: ShutDownAllThrusters()

```
for all available thrusters
  if valveFailsToClose then
    missionFailure = True
  else
    if engineFailsToShutdown then
      LoseEngine(i)
    else
      if PPUFailsToShutDown then
        LoseThruster(i)
      else
        setThrusterOnStandBy(i)
      endif
    endif
  endif
endif
endfor
```

4. Operate thrusters (given time step dt)

Pseudo code 5: OperateThrusters(dt)

```
if numberRequiredThrusters > numberOperatingThrusters
  numberThrusterToStart = numberRequiredThrusters - numberOperatingThrusters
  for i = 1 to numberThrusterToStart
```

```

StartSingleThruster()
  endfor
endif

if numberRequiredThrusters = 0 then
  ShutDownAllThrusters()
endif

for i = 1 to numberOfOperatingThrusters
  if PPUfailsToOperate then
    LoseThruster(i)
    StartSingleThruster()
  endif
  if EngineFailsToOperate then
    LoseEngine(i)
    StartSingleThruster()
  endif
endif
endfor

```

5.2 Distribution Lines Damage Accumulation

As indicated in Section 2, distribution lines damage accumulation can be described as a classical Brownian motion (also known as Wiener process or random walk) characterized by a mean value μ (i.e., drift) equal to 1 and a standard deviation σ (i.e., diffusion) equal to 0.4. Under this assumption, temporal evolution of accumulated damage $D = D(t)$ can be described by Eq. 4 [4]:

$$dD = \mu dt + \sigma dW_t \quad (4)$$

where W_t is a one-dimensional continuous process such that for any two times $t > s$ it has independent increments $W_t - W_s$ distributed normally with mean 0 and variance $t - s$. Such process can be easily coded using the following approach:

$$D(t + dt) = D(t) + \mu dt + N(0, \sigma)\sqrt{dt} \quad (5)$$

Equation 5 has been coded as a separate module in Python as shown in Code 1. Note that in the benchmark case, μ and σ change with time: $\mu = \mu(t)$ and $\sigma = \sigma(t)$. An example of damage accumulation profile for five independent simulation runs are shown in Fig. 4.

Code 1: Damage accumulation

```

def brownian(x0, n, dt, mu, sigma):
  x = np.zeros(n, np.dtype(float))
  W = np.zeros(n, np.dtype(float))
  x[0]=x0
  for i in range(1, len(W)):
    x[i] = x[i-1] + dt
    W[i] = W[i-1] + mu*dt + np.random.normal(0, sigma) * np.sqrt(dt)
  return W[len(W)]

```

5.3 CCF Failures

CCF failures affect only thruster operations (i.e., start, operation and shut-down). Common cause failures (CCF) have been modeled by taking the group conditional failure probabilities supplied in the benchmark problem specification (see Table 2) as β -factor values, i.e. the total failure probability or rate p_t^i of component i (PPU, valve, engine) is determined as:

$$p_t^i = (1 + \beta)p^i \quad (5)$$

where p^i are given in Table 1 and the β -factors are given in Table 2.

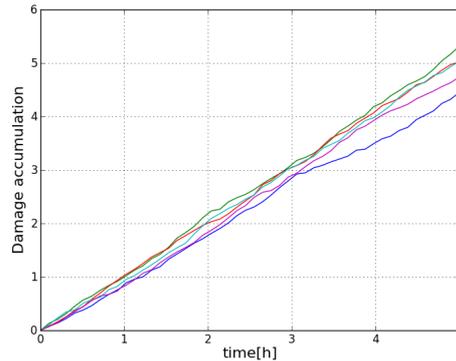


Figure 4: Examples of distribution lines damage accumulation in the [0,5] h interval

5.4 Valve failure probabilities

The probability of failure for the valve has to be considered as a probability per demand. Such probability value depends on the uncertain parameter p which is characteristic of each valve. Such value remains constant throughout the mission time but differs from valve to valve. In our model we initialize the values of valve failure probabilities (p_valve) at the beginning of each simulation run using the python script presented below. Figure 5 shows the temporal profile of the pressure oscillation for the two values of p ($p = 0.4$ and $p = 0.8$). Given the uncertainty associated with the parameter p , we investigated the probability distribution of p_valve that is shown in red line of Fig. 5 (right).

Code 2: p_valve determination

```
def pressureOscil(t,p):
    value=0
    if t<0 and t>4:
        value = 0
    else:
        value = np.sin(t*np.pi/4)*np.sin(t*np.pi/p)
    return abs(value)

def p_valve(p):
    integral = quad(pressureOscil,0,4, args=(p), limit=100000)
    value = (np.exp(integral)-5)*0.0001
    return value
```

5.5 Tank Leaks

At each time step dt the simulator queries the possibility of having a tank leak. As shown in Pseudo code 6, note that simulator account he possibility that more that one leaks can be present.

Pseudo code 6 (given time step dt): **updateTankLeaks** (dt)

```
if newLeakOccurs then
    leakRate = sampleRateValue()
    repairTime = sampleRepairTime()
    leakSet.push(leakRate, repairTime)
endif

for i = 1 to all leaks
    repairTime[i] = repairTime[i] - dt
endfor
```

5.6 Mission Simulation

Mission profile for each scenario:

- Scenario 1: with stop at the support station
- Scenario 2: without stop at the support station

have been divided into phases as shown in Fig. 6: phase numbers are shown in italics.

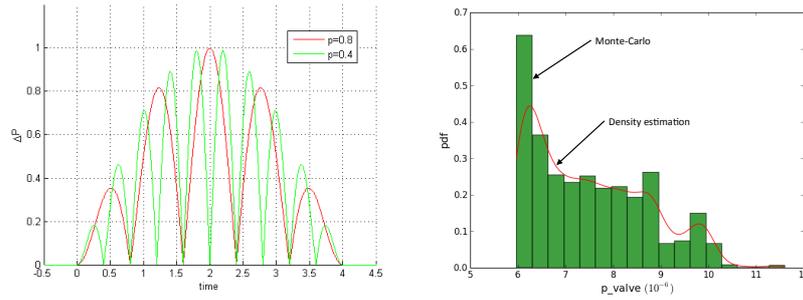


Figure 5: Temporal profile of pressure oscillation during valve activation for $p = 0.4$ and $p = 0.8$ (left) and probability distribution function of p_valve (right)

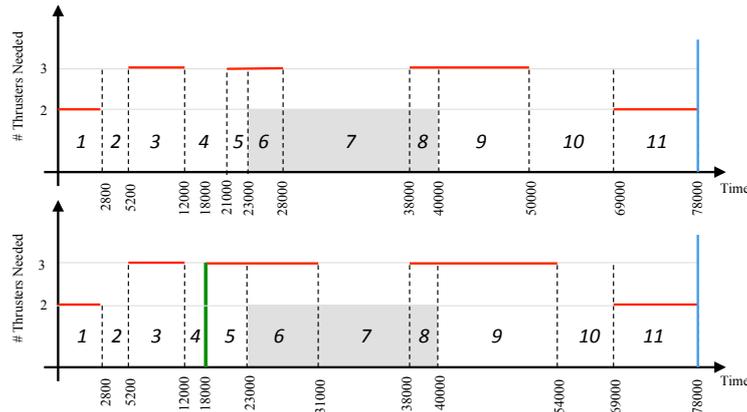


Figure 6: Mission profiles phase discretization for Scenario 1 (top) and Scenario 2 (bottom)

Mission temporal evolution has been calculated using a value of $dt = 1$. Pseudo code that perform a single simulation run is provided below:

Pseudo code 7: Simulation ()

```

for i = 1 to numberOfPhases
  for t = 1 to lengthOfPhase[i]
    OperateThrusters(dt)
    updateDistributionLinesDamage(dt)
    updateTankLeakages(dt)
    updateFuelAmount(dt)
    if missionFailure = True
      stopSimulation()
    endif
  endfor
endfor

```

where the pseudo code for updateFuelAmount (dt) is as follows:

Pseudo code 8: updateFuelAmount (dt)

```

for all operating thrusters
  fuelAmount = fuelAmount - 1 * dt
endfor
for i = 1 to all leaks
  fuelAmount = fuelAmount - leakRate[i] * dt
endfor

```

6 RESULTS

In order to determine the reliability of the overall system, we employed the RAVEN code to perform a Monte-Carlo (MC) analysis. The MC simulation of the system mission was performed with $3 \cdot 10^5$ histories for each scenario. Final reliability values are:

- Scenario 1 (without stop at support station): 16.17 %
- Scenario 2 (with stop at support station): 82.09 %.

Temporal profiles of the system reliability for both scenarios are shown in Figure 7.

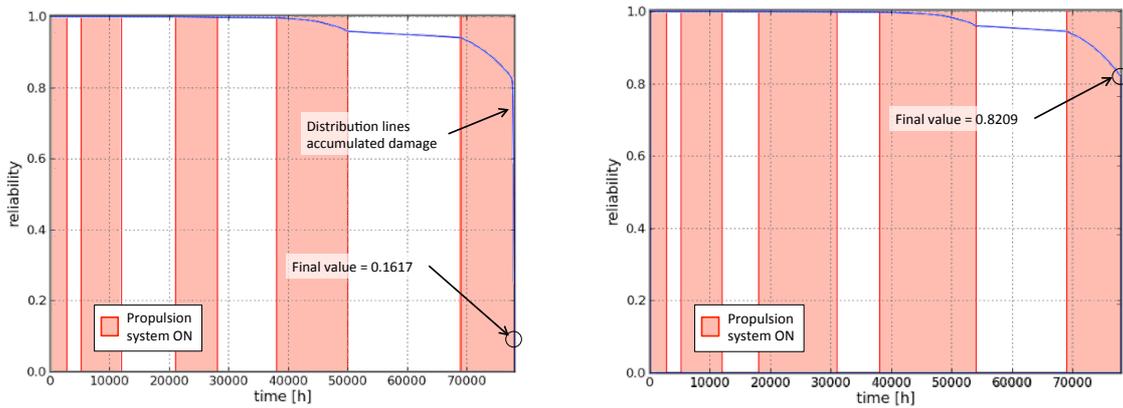


Figure 7: Temporal profiles of the system reliability for scenario 1 (left) and scenario 2 (right)

We also performed a series of MC experiments using the RAVEN code. In each experiment we isolated specific failure modes in order to evaluate the impact of these failure modes on the overall system reliability. In particular, we focused our attention on identifying the impact of thruster reliability, distribution lines damage accumulation, CCF and initial fuel uncertainties.

6.1 Impact of Thruster System Reliability

We evaluated the impact of the thruster dynamics only into the reliability function. In essence, at each dt (see Pseudo code 7), we considered only `OperateThrusters(dt)`. The temporal profiles of the reliability function for both scenarios are shown in Fig. 8. Final reliability values are (CCF considered):

- Scenario 1 (without stop at support station): 99.22 %
- Scenario 2 (with stop at support station): 99.05 %.

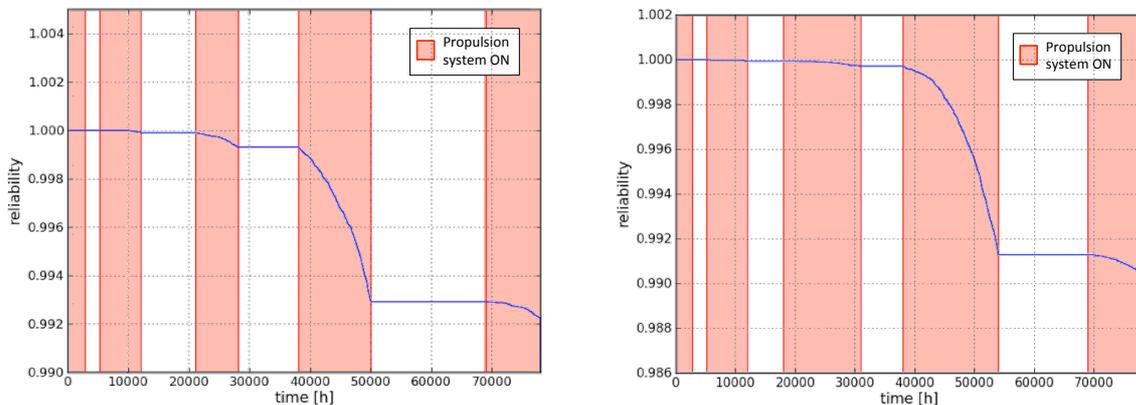


Figure 8: Scenario 1 (left) and Scenario 2 (right) temporal profile of reliability function if only thruster system is considered (with CCF)

6.2 Impact of CCF failures

For both scenarios we were able to determine that CCF failures are responsible for decreasing the overall system reliability by 2%. The temporal profiles of the reliability function for both scenarios are shown in Fig. 9.

6.3 Impact of Damage Accumulation

We evaluated the impact of distribution lines damage accumulation only into the reliability function: at each dt (see Pseudo code 7), we considered only `updateDistributionLinesDamage(dt)`.

We were able to notice that damage accumulation plays a major role in the overall system reliability. For Scenario 1, the distribution of accumulated damage at $t = 78,000$ is given in Fig. 10, i.e., there is about 49% probability that the accumulated damage reaches its threshold (80,000). In fact, such accumulated damaged is almost normally distributed with mean value of 80,070 and sigma equal to 117.

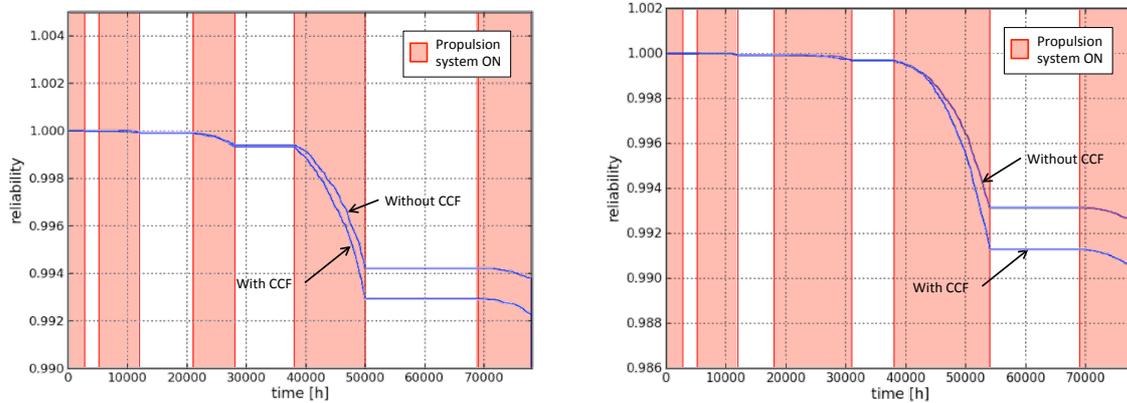


Figure 9: Scenario 1 (left) and Scenario 2 (right) temporal profile of reliability function if only thruster system is considered (without CCF)

On the other hand, for Scenario 2 (see Fig. 11), the accumulated damaged at $t = 78,000$ is normally distributed but with mean value of 62,938 and sigma equal to 109.6. Final reliability values are:

- Scenario 1 (without stop at support station): 19.6%
- Scenario 2 (with stop at support station): negligible

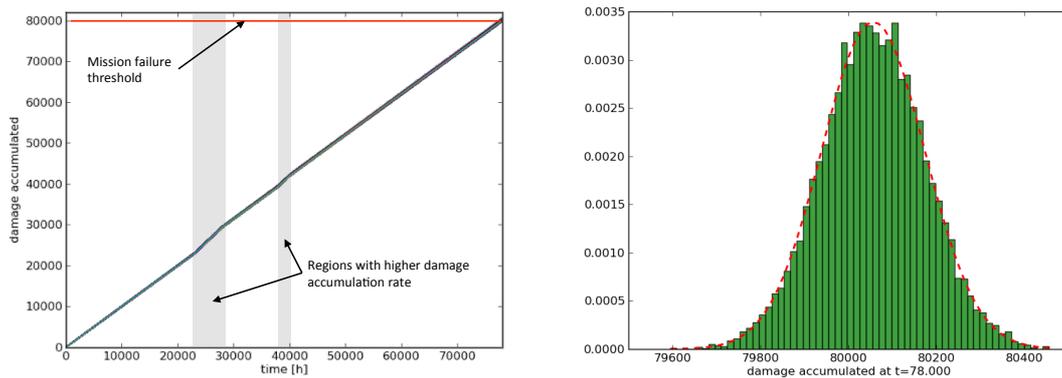


Figure 10: Distribution lines damage accumulation for Scenario 1

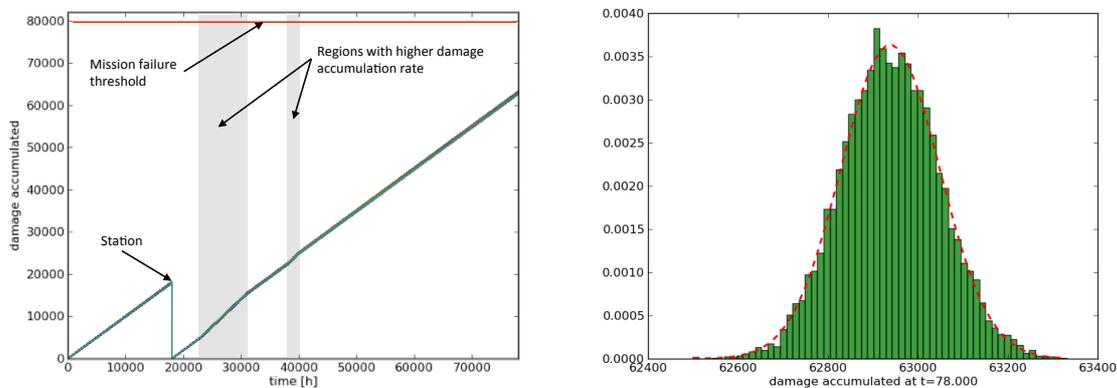


Figure 11: Distribution lines damage accumulation for Scenario 2

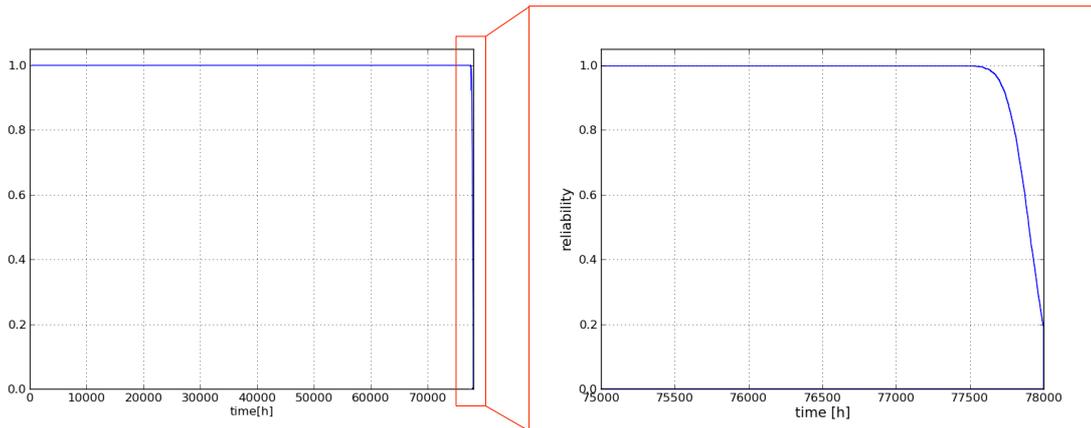


Figure 12: Temporal profile of system reliability due to accumulated damage for Scenario 1

6.4 Impact of Tank Leakages + initial fuel uncertainties

At the end, we evaluated the impact of tank leaks and initial fuel uncertainty into the reliability function by considering `updateTankLeakages(dt)` and `updateFuelAmount(dt)` at each dt (see Pseudo code 7).

Regarding overall tank system, four stochastic variables are considered:

- Initial value of fuel amount
- Probability of leaks occurrence
- Leak rate value
- Leak repair time

In particular, the last two variables determine, for each leak, the amount of fuel that is lost. Table 3 shows the amount of fuel loaded and needed for both scenarios. Note that the amount of fuel needed is less for scenario 2 since a refuelling is being performed at the support station. Note also that the amount of fuel surplus (loaded-needed) is obviously greater for scenario 2. Given that mission failure occurs when fuel is depleted before $t = 78,000$, we can see that this occurs when the amount of fuel lost due to tank leaks exceed the surplus.

Table 3: Fuel loaded, needed and surplus for both scenarios

Scenario	Fuel loaded		Fuel needed	Surplus
	Nominal	Added		
1: without stop	110,000	[5,500 16,500]	101,000	[14,500 25,500]
2: with stop	110,000	[5,500 16,500]	105,000 ¹	[10,500 21,500]

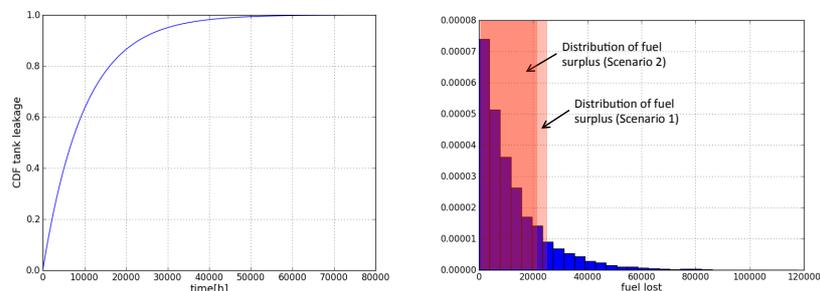


Figure 13: Tank leak cumulative distribution function (CDF) (left) and probability distribution function of fuel lost given a leak has occurred (right)

¹ For Scenario 2 (with stop at support station), the overall amount of fuel needed for the mission is measured from $t = 18,000$ h since: 1) ship is completely refuelled during the stop and 2) the likelihood of mission failure due to tank leakages before $t = 18,000$ h is negligible.

Figure 13 (left), shows that the likelihood of a tank leak is very high; in addition, it also shows the distribution of fuel lost due to tank leaks given that a leak has occurred (Fig. 12, right). This distribution is compared to the distribution of fuel surplus for both scenarios. Hence, from Fig. 12 (right), we expect that system reliability due to tank leaks and initial fuel uncertainties is considerably higher for scenario 2. In this respect, Fig. 13 shows, for scenario 1 (left) and 2 (right), the temporal profile of the reliability function.

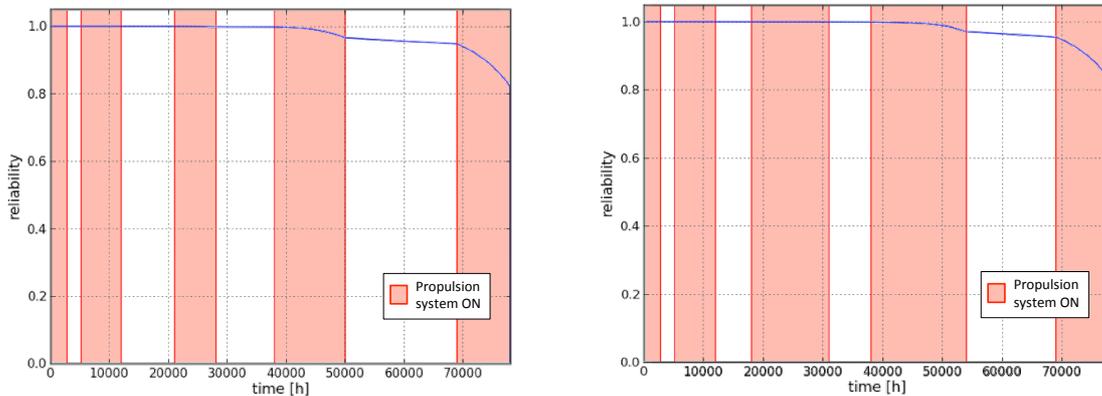


Figure 14: Temporal profile of reliability function for scenario 1 (left) and 2 (right) when only tank leaks and initial fuel uncertainty are considered

6.5 Summary of reliability contributions

As a summary of the results shown in the previous sections, for each scenario we show in Figure 15 the probability of success (green) and the major contributors to failures: tank leaks (red), distribution lines (yellow) and propulsion system (blue).

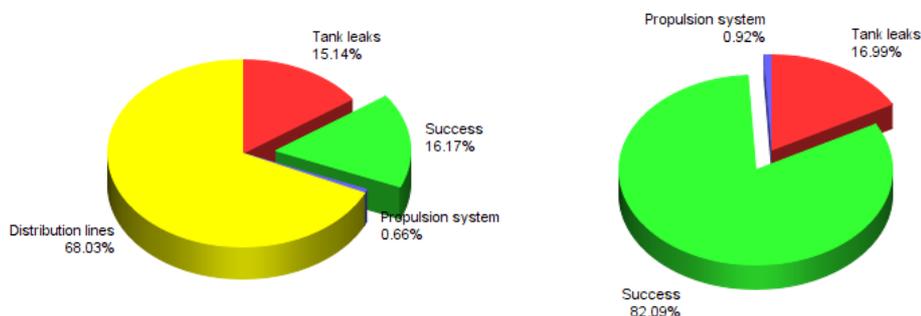


Figure 15: Probability of success (green) and the major contributors to failures: tank leaks (red), distribution lines (yellow) and propulsion system (blue) for Scenario 1 (left) and Scenario 2 (right)

7 CONCLUSIONS

This paper presented a solution of the PSAM12 space propulsion system benchmark. A stochastic analysis was performed in order to obtain the system reliability using the Monte-Carlo capabilities of the RAVEN code. An ad-hoc simulator has been developed and coded in python and then interfaced with RAVEN. We have obtained an overall system reliability of 16% for Scenario 2 and 82% for Scenario 1. This suggests that a stop at the support station is mandatory in order to maximize mission success. We also investigated the impact of: thruster reliability, distribution lines damage accumulation, CCF and initial fuel uncertainties.

References

- [1] A. Alfonsi, C. Rabiti, D. Mandelli, J. Cogliati, R. Kinoshita, “Raven as a tool for dynamic probabilistic risk assessment: Software overview,” in *Proceeding of M&C2013 International*

Topical Meeting on Mathematics and Computation, CD-ROM, American Nuclear Society, LaGrange Park, IL (2013).

- [2] C. Rabiti, D. Mandelli, A. Alfonsi, J. Cogliati, R. Kinoshita, “Mathematical framework for the analysis of dynamic stochastic systems with the RAVEN code,” in *Proceeding of M&C2013 International Topical Meeting on Mathematics and Computation*, CD-ROM, American Nuclear Society, LaGrange Park, IL (2013).
- [3] M. Marseguerra, E. Zio, J. Devooght, P.E. Labeau, “A concept paper on dynamic reliability via Monte Carlo simulation,” *Mathematics and Computers in Simulation*, **47**, pp. 371-382 (1998).
- [4] C. Gardiner, *Stochastic Methods for Physics, Chemistry, and the Natural Sciences* (4th edition), Springer Series in Synergetics, Springer-Verlag, Berlin, Third edition, (2004).

APPENDIX A: BASE BENCHMARK PROBLEM RESULTS

In this appendix we present the result obtained for the base benchmark problem. Reliability value obtained was 0.986 as shown in Fig. 16. Figure 16 shows the reliability drop caused by failure on demand

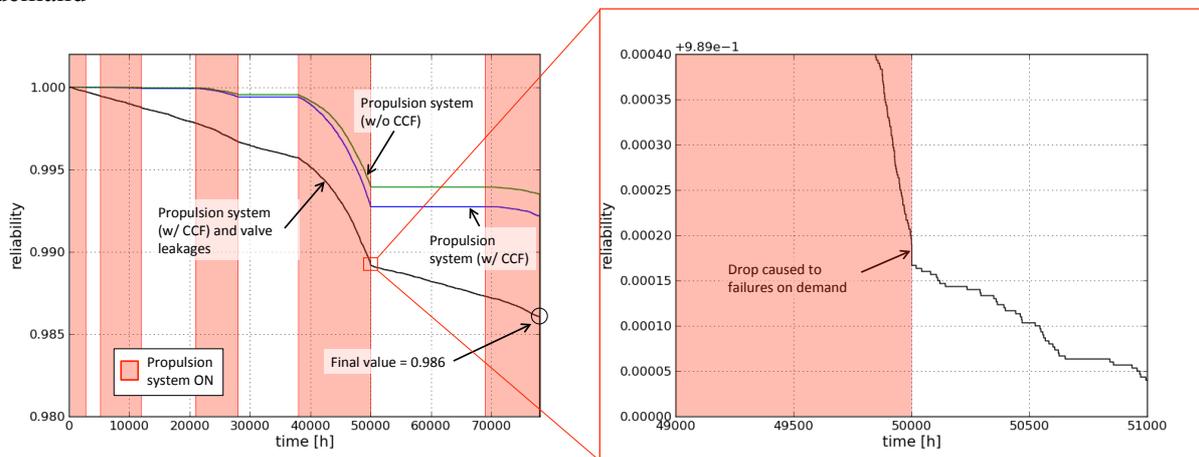


Figure 16: Temporal profile of reliability function for the base version of the benchmark problem

APPENDIX B: COMPARISON TABLE

- *Methodology Employed:* Monte-Carlo
- *Stochastic analysis tool:* RAVEN
- *Software parameters:* 300,000 runs
- *System simulator initially developed in Python and later in C++ for faster computational time*
- *Computational resources:* Intel core i7-3770 Quad Core (3.4 GHz), 16 GB SDRAM DDR3
- *Computational time:* ~ 5 hours

Component	Hypothesis/Approximations	Modelling
PPU	No approximation	Control logic implemented through scripts
Ion engine	No approximation	Control logic implemented through scripts
Valve	No approximation	p_valve values numerically determined for each simulation run
Distribution line	Stochastic process embedded into the simulator	Damage modelled as a Weiner process
Phase mission	Mission divided into 11 phases	Time step for the simulation set to 1 hour
Engine CCF	Beta factor	Failure probability p_t^i is determined as $p_t^i = (1 + \beta)p^i$