# RDMSim Exemplar: User Guide

January 2021

## 1 Introduction

RDMSim exemplar represents a simulating environment for a Remote Data Mirroring (RDM) network [1, 2]. It has been designed to help researchers working in the area of self-adaptive systems (SASs) to validate their approaches. The simulator has been designed keeping in view the operational model of the RDM network presented in [2, 3].

In this document, we provide the details of using RDMSim exemplar to perform experiments with self-adaptive decision-making techniques. The document is organized as follows: The second section contains the details about the RDMSim package and its download details. In third section, we provide the details about the architecture of the RDMSim exemplar. In fourth section, we discuss the details of creating custom adaptation logic for RDMSim network with the help of a simple adaptation example.

## 2 RDMSim Package

The RDMSim package is available in the form of a zip file containing the source files of the simulator. It can be downloaded from the following git hub repository:

`https://gitlab.com/humasamin/rdmsimexemplar`

The RDMSim package contains two sub directories as follows:

### 1. Source

The Source directory contains the source code of the RDMSim simulator. The source code can be executed with the help of Eclipse software[1]. The Source directory contains the following projects:

#### a) RDMNetwork

RDMNetwork project represents the simulator software for the RDM network. It helps in execution of experiments by running simulations for the RDM network.

#### b) TestRDM

TestRDM project contains a custom adaptation example that uses RDMSim exemplar to perform adaptation experiments.

The Source directory also contains a directory *config_log_files*. The *config_log_files* directory contains the configuration and log files for the RDMSim exemplar. The configuration file can be used for the configuration parameter settings of the simulator during the execution of the experiments. The details of the configuration parameters are provided in 3. The *log* file is used to store the results log of the experiments.

[1] www.eclipse.org

## 2. Jar Files

The *Jar Files* directory contains ***RDMSim.jar*** a Java Archive File for the RDMSim. It helps in usage of the RDMSim exemplar as a library by other java programs. It also contains the *json-simple.jar* file that is used by the simulator to deal with the configuration files for the RDMSim Exemplar.

# 3 RDMSim Architecture

The *RDMSim* exemplar has been developed to facilitate the implementation of a two-layered architecture for a self-adaptive RDM, as shown in Fig 1. The architecture structures a Managing System on top of the Managed System (the *RDMSim*). Let's study each layer.

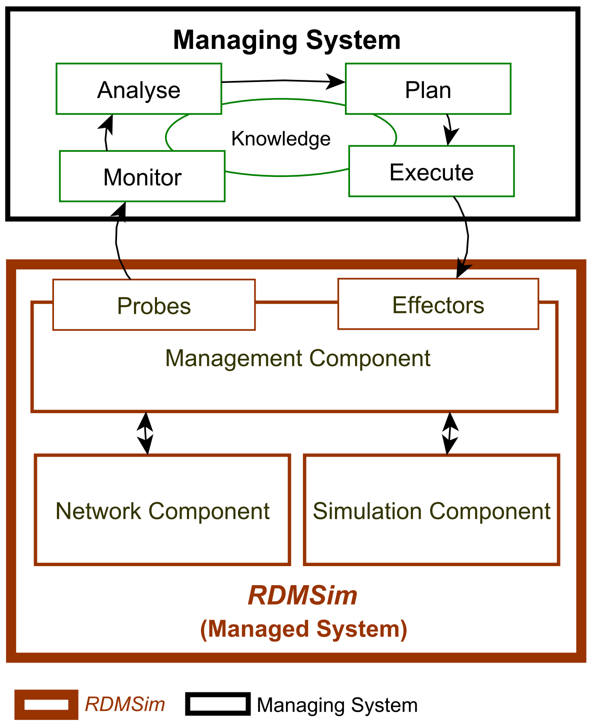



Figure 1: RDMSim Architecture

## 3.1 Managing System

The Managing System, at the upper layer, is responsible for providing the self-adaptation logic. It realizes a feedback loop that monitors the environment and the managed system, and adapts the latter when necessary. The feedback loop consists of Monitor-Analyse-Plan-Execute over a Knowledge base K (MAPE-K) [4]. MAPE-K loop is considered an architectural blueprint for self-adaptive systems (SASs) used to perform adaptation decisions on the Managed System (i.e. *RDMSim* in our case). When using the *RDMSim* exemplar, researchers will provide their own decision-making techniques to serve as a Managing System for it. The Managing System can be based on based on different techniques such as Multi-Criteria Decision-Making[5], Reinforcement Learning[6] and Evolutionary Computation[7, 8] etc.

Table 1: Probe Functions

| Function | Description |
|---|---|
| Topology getCurrentTopology() | Returns the current topology for the network. |
| int getBandwidthConsumption() | Returns the bandwidth consumption of the network. |
| int getActiveLinks() | Returns the number of active links. |
| int getTimeToWrite() | Returns the time to write data for the network. |
| Monitorables getMonitorables() | Returns the values for all the monitorable metrics. |

Table 2: Effector Functions

| Function | Description |
|---|---|
| void setNetworkTopology(int timestep,Topology selectedtopology) | To set the network topology at a particular timestep. |
| void setActiveLinks(int active_links) | to set the number of active links for the network. |
| void setTimeToWrite(double time_to_write) | To set the time to write data for the network. |
| void setBandwidthConsumption(double bandwidth_consumption) | To set bandwidth consumption for the network. |
| void setCurrentTopology(Topology current_topology) | To set topology for the network. |

## 3.2 Managed System

The *RDMSim* represents the Managed System. The *RDMSim* provides probes and effectors that can be used by the Managing System to interact with the simulator. Probes are used to monitor information (M in MAPE) whereas the effectors are used to execute the adaptation decisions (E in MAPE) on the Managed System.

Next, we present the architecture of the Managed System implemented as Java Packages for the *RDMSim* software The components in the architecture for *RDMSim*, presented in Fig. 1, are as follows:

### 3.2.1 Management Component,

which acts as a bridge between the Managing System and other internal components of the *RDMSim*, by providing the implementation of probes and effectors to be used by the Managing System on top. The functions provided by the probes and effectors are used to both monitor the status of the RDM (i.e. cost, reliability and performance), and also change the network topology and different network parameters according to the decision made, which are described in Table 1 and 2 respectively.

### 3.2.2 Network Component,

which provides an implementation of the main physical elements elements of the RDM. These elements include the number of mirrors (i.e. servers) and the network links that represent a fully connected network of mirrors. As an example, for 25 mirrors, it will create a network of 300 links. The users of *RDMSim* can change the number of mirrors to create a custom RDM network for their experiments. The Network Component also provides an implementation of the monitorables and topologies for the network. Specifically, in the *RDMSim*, we provide an implementation of three monitorables:

*Mon1:* ***Active Network Links:*** provides the current active network links to measure the reliability of the RDM. The higher the number of active links, the more reliable will be the RDM.

*Mon2:* ***Bandwidth Consumption:*** provides the current bandwidth consumption to measure the operational cost for the RDM in terms of inter-site network traffic. The higher the bandwidth consumption more will be the operational cost for the RDM. Bandwidth Consumption is measured in GigaBytes per second.

*Mon3:* ***Time to Write Data to mirrors:*** provides the time taken to write data. It measures the performance of the network in terms of writing time to maintain multiple copies of data on each remote site. A big writing time leads to reduction of performance of the RDM. Time to Write Data is measured in milliseconds.

For the communication between the mirrors, we consider synchronous mirroring [3]. During synchronous mirroring, sequential writing is performed to prevent data loss [9]. In sequential writing, the primary mirror (i.e. the sender) waits for an acknowledgement (known as a *handshake*) regarding the receipt and writing of data from the secondary mirror (i.e. the receiver). This process is performed for each active link on the communication path between the mirrors. Therefore, the time to write data is computed as *Total Writing Time= ($\alpha$* number of active links) * Time to Write Data Unit*[2]. Here, $\alpha$ represents a fraction of active links to constitute the communication path between two mirrors. $\alpha$ can have a value of greater than zero and less than and equal to one. For our experiments, we have set $\alpha = 1$.

Similarly, the bandwidth consumption is also dependent on the number of active links. More active links imply more data transmission, which leads to a higher bandwidth consumption [9]. Hence, we compute the Bandwidth Consumption as *Total Bandwidth Consumed=number of active links * Bandwidth per link*[3].

### 3.2.3 Simulation Component,

which includes the implementation of the uncertainty scenarios that represent the different dynamic environmental conditions that the RDM can face, and which will be simulated. It allows the setting of the simulation properties, such as the number of simulation runs and the chosen uncertainty scenario(s) to be executed by the *RDMSim*.

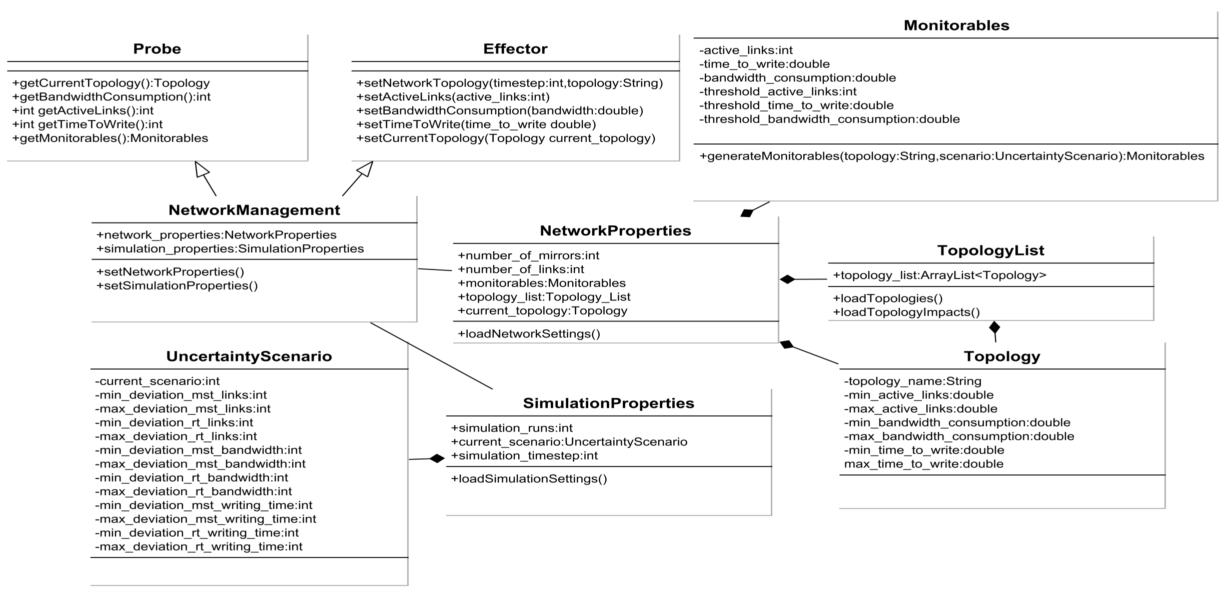



Figure 2: RDMSim Class Diagram

A partial class diagram representing the elements of the Management Component, Network Component and Simulation Component is shown in Fig 2. The *NetworkManagement* class along with the *Probe* and *Ef-*

[2]To implement realistic impacts, we vary the time between 10 to 20 milliseconds
[3]To implement realistic impacts we vary the Bandwidth per link between 20 to 30 GBps

*fector* interfaces provides an implementation of the Management Component. The classes *NetworkProperties*, *Monitorables*, *Topology* and *TopologyList* are part of the Network Component and provide an implementation of the corresponding features of the RDM. The *SimulationProperties* and *UncertaintyScenario* classes are part of the Simulation Component, and are used to implement the functionalities related to the simulations to be executed.

# 4 Custom Adaptation Example

In order to develop a custom adaptation logic, the RDMSIM simulator provides the interfaces of Probe and Effector. The Probe and Effector functions can be used to implement the MAPE-K feedback loop to support self-adaptation. The Probe helps in monitoring of the data about the number of active links, bandwidth consumption and performance of the network in the form of time to write data. The Effector helps in setting the network topology and tuning of the network settings such as changing number of active links etc.

Next, a step by step example of writing a custom adaptation logic is presented.

## 4.1 Example

We provide a simple adaptation example that performs adaptations by switching between the topologies of Minimum Spanning Tree (MST) and Redundant Topology (RT) using the effector. The data about the monitorable metrics such as number of active links, bandwidth consumption and time to write is data gathered by using the probing functions.

**Step: 1 Create a Java Project**

First of all, create a new Java Project in Eclipse IDE using the following steps:

```
Click on File -> New -> Project -> Java Project
```

Name the project as *TestRDM* and Click Finish as shown in Fig 3.

**Step: 2 Adding the RDMSim to buildpath of the project**

Right click on the "TestRDM" project in the Project Explorer pane – Select *Build Path* – Select *Configure Build Path* as shown in Fig. 4.

A properties dialog box will be displayed as shown in Fig 5.

Go to *Libraries* tab and click on *Add External JARs* and add *RDMSim.jar* to the project from the Jar Files directory of the RDMSim package. Click on *Apply and Close*[4].

**Step: 3 Adding Configuration file to Project**

Go to the Source folder of the RDMSim package and copy the *config-log-files* folder. Right click on the root of the *TestRDM* project in the Project Explorer pane and paste.

Now, the *RDMSim* exemplar is ready to be used as part of the *TestRDM* project. Next, we describe the step by step usage of the RDMSim exemplar by writing our own adaptation logic.

**Step: 4 Loading Configuration Settings and Instantiation of Probe and Effector**

The first step in implementing the custom adaptation logic is to load the configuration settings for the experiment from the *configuration.json* file and instantiation of the Probe and Effector components. The Probe and Effector components will enable the communication between our *TestRDM* program and *RDMSim*. This can be done by using the NetworkManagement class in your program as follows:

[4] Please also add the json-simple.jar file as an External Jar

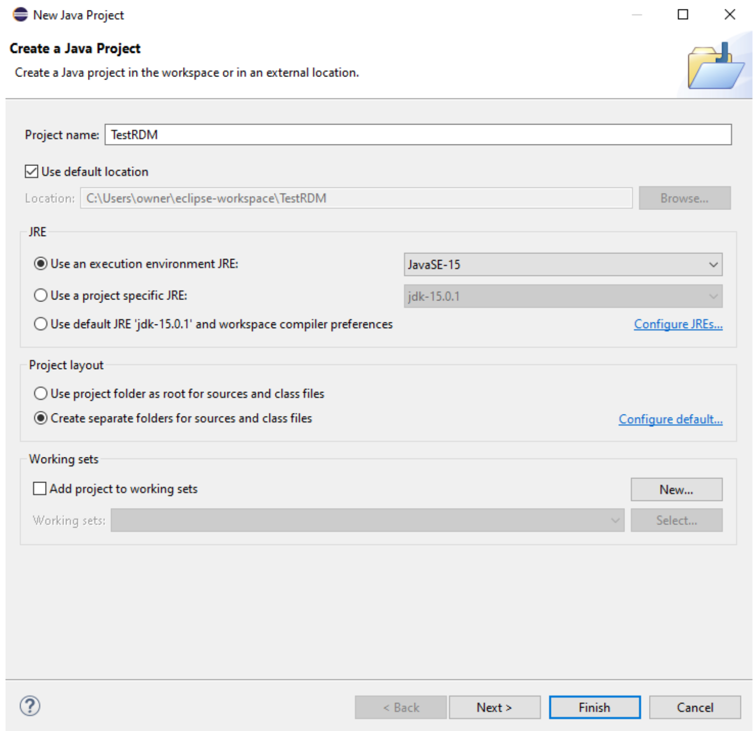



Figure 3: Create New Project

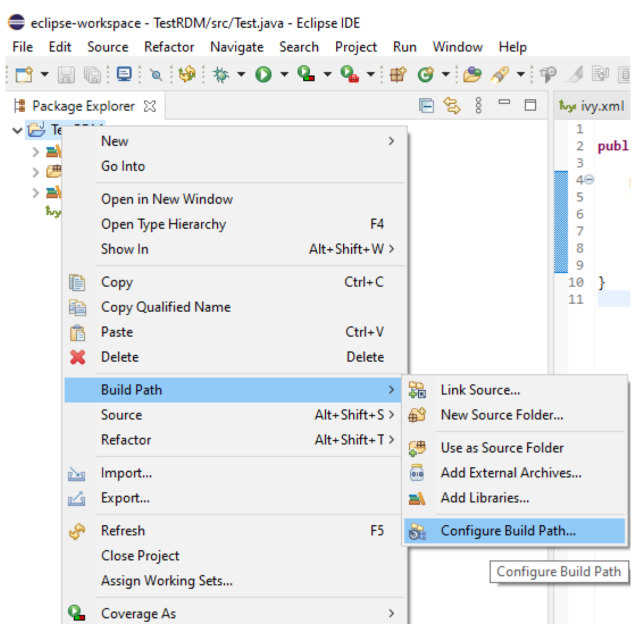



Figure 4: Add RDMSim to the project

```
NetworkManagment network_management=new NetworkManagment();
Probe probe=network_management.getProbe();
```

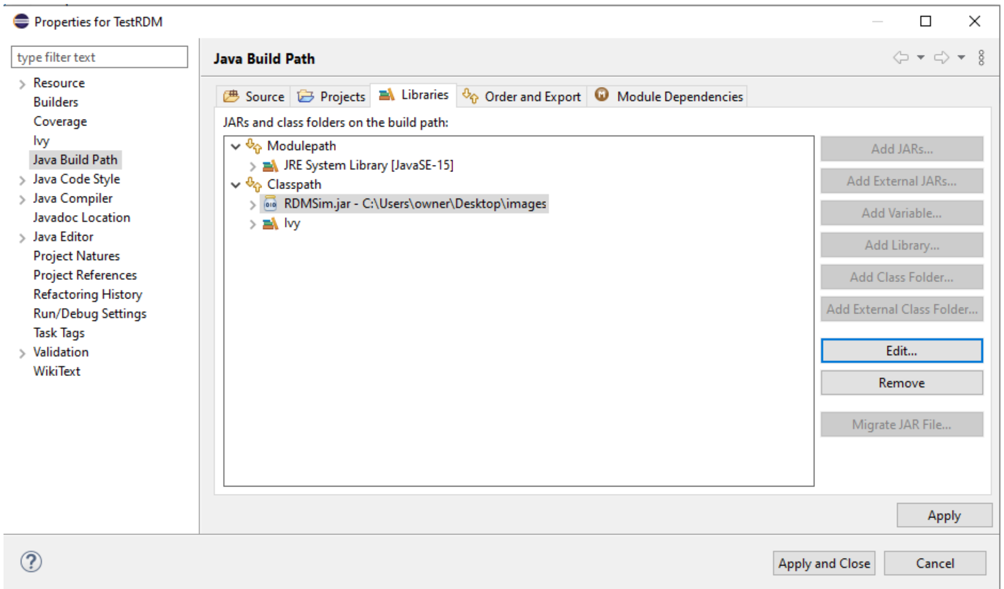



Figure 5: Properties Dialog

```
Effector effector=network_management.getEffector();
```

The NetworkManagement class is responsible for loading the configuration parameters and instantiating the Probe and Effector instances. The configuration settings include the parameters like number of simulation time steps, the number of mirrors for the RDM network, number of active links and uncertainty scenario to be considered for the experiments. The details of the configuration parameters is provided in Table 3.

For this example, we consider an RDM network of 25 mirrors having 300 links in total to create a fully-connected network. We have set the other configuration parameters considering this infrastructure. The values provided in the configuration file can be considered as the default values that are set considering the expert knowledge provided in [3]. These values can be changed according to the experiments' requirements.

Once the Probe and Effector are instantiated, the Probe and Effector functions can be used to monitor the RDMSim network and apply the adaptations using the functions provided in Table 1 and 2 respectively.

### Step: 5 Monitoring of the RDMSim network using Probe functions

In order to monitor the RDMSim network, we can use the probe functions provided in Table 1. For example, to get the values of all the monitorable metrics for a particular simulation time step we can use the getMonitorables() function as follows:

```
Monitorables m=probe.getMonitorables();
```

### Step: 6 Performing Adaptations on the RDMSim network using Effector functions

In order to perform adaptations on the network, we can use the Effector functions provided in Table 2. For example, to change the network topology at a particular timestep, we can use the setNetworkTopology() function as follows:

```
effector.setNetworkTopology(10,"mst");
```

It will set the Minimum Spanning Tree (MST) topology for the network at the simulation timestep 10.

Next, we present a step by step implementation of a simple MAPE-K feeback loop using the code provided in Steps 4 to 6.

Table 3: Configuration Settings

| **Configuration Parameter** | Value Type | **Value Range** | **Description** |
|---|---|---|---|
| **time_steps** | Integer | 0 to Valid Integer Range | It represents the total number of simulation time steps (simulation runs). |
| **mirror_number** | Integer | 5 to 30000 | It represents the number of mirrors in the RDM Network. |
| **link_threshold** | Double | 0 to 100 | Represents the percentage satisfaction threshold for active links. |
| **bandwidth_threshold** | Double | 0 to 100 | Represents the percentage satisfaction threshold for bandwidth consumption. |
| **writing_times_threshold** | Double | 0 to 100 | Represents the percentage satisfaction threshold for time to write data. |
| **topologies** | List of String values | [mst,rt] | Represents the list of topology names. We have defined two topologies Minimum Spanning Tree (mst) and Redundant Topology (rt). |
| **topology_active_links** | List representing the range: [min, max] | min: 0 to 100 percent of total number of links max: 0 to 100 percent of total number of links | Represents the range (min, max) for the active links for the specific topology. For example: mst_active_links:[min,max] |
| **topology_bandwidth_consumption** | List representing the range: [min, max] | min: 0 to 100 percent of total bandwidth max: 0 to 100 percent of total bandwidth | Represents the range (min, max) for the bandwidth consumption for the specific topology. |
| **topology_writing_time** | List representing the range: [min, max] | min: 0 to 100 percent of total time to write max: 0 to 100 percent of total time to write | Represents the range (min, max) for the time to write data for the specific topology. For example: mst_writing_time:[min,max] |
| **current_scenario** | Integer | 0 to 6 | Represents the uncertainty scenario to be executed by the RDM Simulator. 0 represents the stable scenario. 1 to 6 represents the detrimental scenarios 1 to 6. |
| **deviation_scenario_scenario_topology_monitorable** | List representing the deviation range: [min, max] | 0 to 100 | Represents the percentage deviation range for the monitorable metric for the current scenario under a specific topology. For example: deviation_scenario_0_mst_links:[min,max] |

## MAPE-K loop implementation

Create a Java Class in the TestRDM project and name it MAPE_KLoop using the following steps:

```
Right Click on TestRDM project--> New -->Class
```

The MAPE_KLoop class will be used to implement the phases (Monitor,Analyse,Plan and Execute) of the MAPE-K feedback loop. Copy and Paste the following code in the MAPE_KLoop class.

```
import rdm.management.Probe;
import rdm.management.Effector;
import rdm.management.NetworkManagment;
import rdm.network.Monitorables;
import rdm.network.Topology;

public class MAPE_KLoop {

Probe probe;
Effector effector;

public MAPE_KLoop(Probe probe, Effector effector)
{
this.probe=probe;
this.effector=effector;
}
//Monitor the network using probe functions
public void monitor(int simulation_timestep)
{
Monitorables m=probe.getMonitorables();
analysisAndPlanning(simulation_timestep, m);
}

//Analysis and planning for the adaptation
public void analysisAndPlanning(int simulation_timestep,Monitorables m)
{
String selected_topology;

if (probe.getBandwidthConsumption()>m.getThresholdBandwidthConsumption()
    ||probe.getTimeToWrite()>m.getThresholdTimeToWrite())
{
selected_topology="mst";
execute(simulation_timestep,selected_topology);
}
else if(probe.getActiveLinks()>m.getThresholdActiveLinks())
{
selected_topology="rt";
execute(simulation_timestep,selected_topology);
}
else
{
selected_topology="rt";
```

```
execute(simulation_timestep,selected_topology);
}
}

//Execute the adapatation using functions of the effector
public void execute(int simulation_timestep,String selected_topology)
{
effector.setNetworkTopology(simulation_timestep,selected_topology);

}

public void run(int simulation_timestep)
{
monitor(simulation_timestep);
}
}
```

The phases of MAPE-K loop are implemented as follows:

**1. Monitor Phase**

We have implemented the monitor() function in the class to implement the monitor phase of MAPE-K loop. The monitor() function calls the getMonitorables() function to get the values of all the monitorable metrics at a particular simulation time step.

**2. Analyse and Plan Phases**

We have provided the analysisAndPlanning() function to implement the analyse and plan phase of the MAPE-K feedback loop. In our simple example, we are selecting the topology randomly at each time step. This can be implemented using more intelligent decision-making algorithms such as Reinforcement Learning [10] and Evolutionary Computation techniques [3, 7].

**3. Execute Phase**

Once the topology is selected, the adaptations are performed using the Effector functions provided in Table 2. We use the setNetworkTopology() function to set the topology at a particular time step as shown in the execute() function of the MAPE_KLoop class

## Run the MAPE-K loop at each Simulation Time Step

In order to run the MAPE-K loop at each simulation timestep, create an object of the MAPE_K class and execute the feedback loop by calling the run() function of the MAPE_K class at each simulation time step.

For this purpose, create a new class named *Test*. Copy and Paste the following code to it.

```
import rdm.management.Effector;
import rdm.management.NetworkManagment;
import rdm.management.Probe;
import rdm.management.RDMSimulator;

public class Test {
public static void main(String[] args) {
  //Step: 1 Load the configuration settings
```

```
NetworkManagment nm=new NetworkManagment();
//Step: 2 Instantiate the probe and effector
Probe probe=nm.getProbe();
Effector effector=nm.getEffector();

//Step 3: Instantiate the mape-K feedback loop
MAPE_KLoop loop=new MAPE_KLoop(probe,effector);

//Run simulation for the number of simulation runs defined to execute the feedback loop
for(int timestep=0;timestep<NetworkManagment.simulation_properties.getSimulationRuns();
timestep++) {
//start the feedback loop
loop.run(timestep);
}
RDMSimulator.displayResults(args);

}

}
```

## 4.2 Executing the program

**Step: 1**

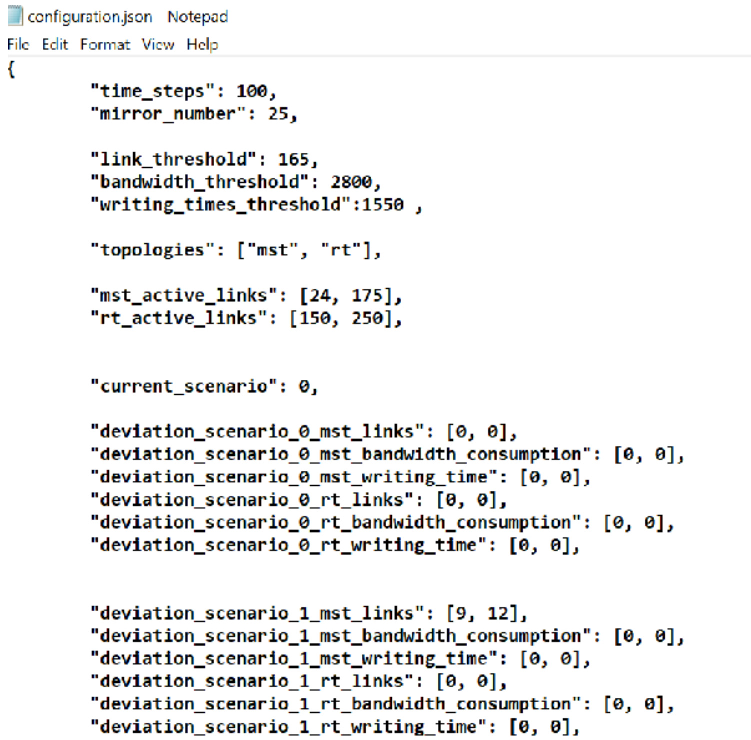
In order to execute the program, first of all set the configuration parameter values in the "configuration.json" file as shown in Fig 6. You can open the file by doubling clicking on the file in the Project Explorer pane.

**Step: 2**

Once the configuration parameters are set, run the program by clicking on the run button[5]. It will display the graphs showing the satisfaction levels of the monitorable metrics and the results log per time step as shown in Fig 7.

A complete coded example for *TestRDM* is provided as part of the RDMSim package.

[5] Make sure the VM arguments are set



configuration.json Notepad
File Edit Format View Help


```
{
        "time_steps": 100,
        "mirror_number": 25,

        "link_threshold": 165,
        "bandwidth_threshold": 2800,
        "writing_times_threshold":1550 ,

        "topologies": ["mst", "rt"],

        "mst_active_links": [24, 175],
        "rt_active_links": [150, 250],


        "current_scenario": 0,

        "deviation_scenario_0_mst_links": [0, 0],
        "deviation_scenario_0_mst_bandwidth_consumption": [0, 0],
        "deviation_scenario_0_mst_writing_time": [0, 0],
        "deviation_scenario_0_rt_links": [0, 0],
        "deviation_scenario_0_rt_bandwidth_consumption": [0, 0],
        "deviation_scenario_0_rt_writing_time": [0, 0],


        "deviation_scenario_1_mst_links": [9, 12],
        "deviation_scenario_1_mst_bandwidth_consumption": [0, 0],
        "deviation_scenario_1_mst_writing_time": [0, 0],
        "deviation_scenario_1_rt_links": [0, 0],
        "deviation_scenario_1_rt_bandwidth_consumption": [0, 0],
        "deviation_scenario_1_rt_writing_time": [0, 0],
```


Figure 6: Configuration File

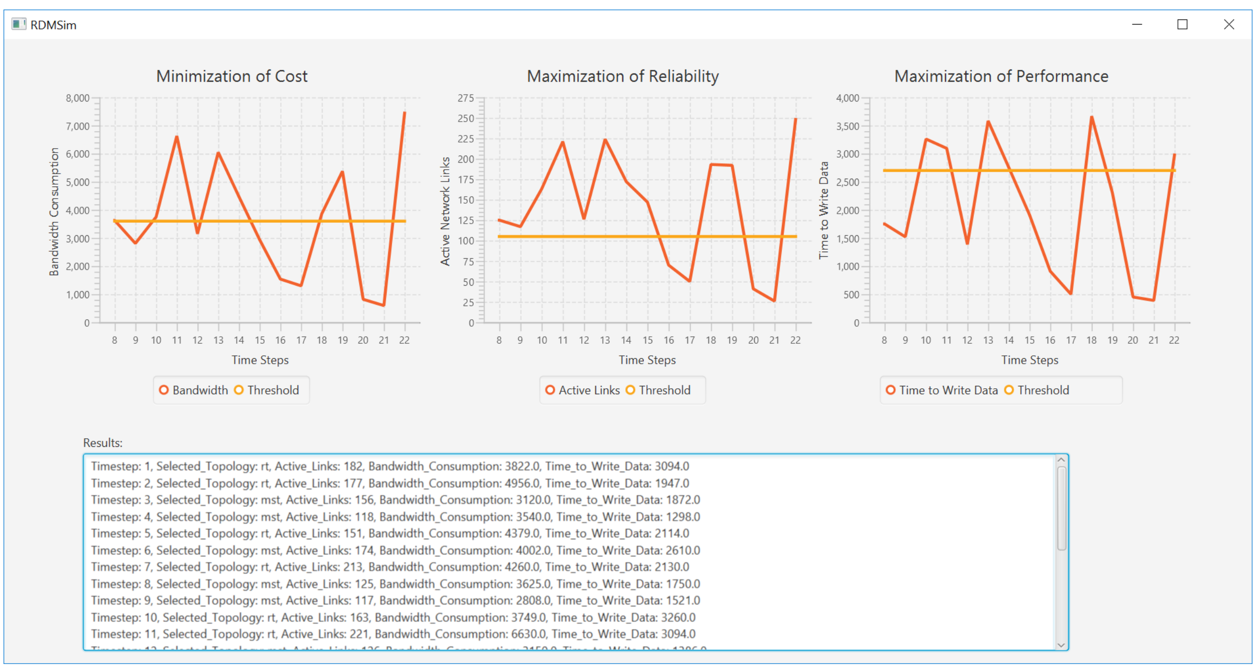



Figure 7: Graphical User Interface

# References


[1] M. Ji, A. C. Veitch, J. Wilkes *et al.*, "Seneca: remote mirroring done write." in *USENIX Annual Technical Conference, General Track*, 2003, pp. 253–268.

[2] K. Keeton, C. Santos, D. Beyer, and J. Chase J.and Wilkes, "Designing for disasters," *USENIX Conference on File and Storage Technologies, Berkeley*, 2004.

[3] E. M. Fredericks, "Mitigating uncertainty at design time and run time to addressassurance for dynamically adaptive systems," *Michigan S.University. PhD Thesis.*, 2015.

[4] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[5] E. Triantaphyllou, "Multi-criteria decision making methods," in *Multi-criteria decision making methods: A comparative study*. Springer, 2000, pp. 5–21.

[6] H. Samin, "Priority-awareness of non-functional requirements under uncertainty," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*. IEEE, 2020, pp. 416–421.

[7] A. Ramirez, B. Cheng, N. Bencomo, and P. Sawyer, "Relaxing claims: Coping with uncertainty while evaluating assumptions at run time," *MODELS*, 2012.

[8] K. M. Bowers, E. M. Fredericks, and B. H. C. Cheng, "Automated optimization of weighted non-functional objectives in self-adaptive systems," T. E. Colanzi and P. McMinn, Eds. Cham: Springer International Publishing, 2018.

[9] "Chapter Two - Database Management Systems," in *RDF Database Systems*, O. Curé and G. Blin, Eds. Boston: Morgan Kaufmann, Jan. 2015, pp. 9–40.

[10] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley, "A Survey of Multi-Objective Sequential Decision-Making," *Journal of AI Research*, vol. 48, 2013.