FOG AND EDGE COMPUTING

Practical File

Course Code : INITE23

Submitted By –

Name: Rohit Kumar

Roll no.: 2020UIN3322

Branch: ITNS

Semester: 6th

Academic Year: 2022-23

Submitted To:

Ms. Shipra Gautam

INDEX

SNO	PROGRAMS	DATE	SIGNATURE
1	To study and install the iFogSim simulator.	10/01/23	
2	Create a topology and simulate it using iFogSim.	17/01/23	
3	Perform scheduling and load balancing using iFogSim.	24/01/23	
4	Create a simulation setup for electroencephalography (EEG) Beam Tractor Game. Analyse the performance of the network in terms of CPU execution delay and energy consumed.	31/01/23	
5	Create a simulation setup for a VR Game. Analyse the performance of the network in terms of CPU execution delay and energy consumed.	07/02/23	
6	Create a simulation setup for an Intelligent Surveillance using Data Centre Networks (DCNs). Analyse the performance of the network in terms of CPU execution delay and energy consumed.	14/02/23	
7	Create a Fog Network with a broker, and Fog nodes and End devices ranging from 5 to 25. Analyse the performance of the network in terms of Throughput and Latency.	21/02/23	
8	Create a Fog Network with a broker, and Fog nodes and End devices ranging from 5 to 25. Analyse the performance of the network in terms of Packet Delivery Ratio (PDR).	28/02/23	
9	Create a Fog Network with a broker, and Fog nodes and 10 End devices use communication protocol MQTT. Analyse the performance of the network in terms of Packet Delivery Ratio (PDR).	14/03/23	
10	Create a fog Network with a broker and multiple fog nodes and implement scalable simulations using varying number of end devices for each setup of simulation.	28/03/23	
11	Create a Fog Network with a broker, and Fog nodes and 10 End devices use communication protocol UDP. Analyse the performance of the network in terms of Packet Delivery Ratio (PDR).	11/04/23	
12	Create a Fog Network with a broker, and Fog nodes and 10 End devices use communication protocol TCP. Analyse the performance of the network in terms of Packet Delivery Ratio (PDR).	18/04/23	

PRACTICAL-1

To study and install the iFogSim simulator.

The iFogSim simulator:

iFogSim is a Java based open-source simulation tool for simulating fog computing scenarios. iFogSim enables the modelling and simulation of fog computing to evaluate resource management and scheduling policies across edge and cloud resources under different scenarios.

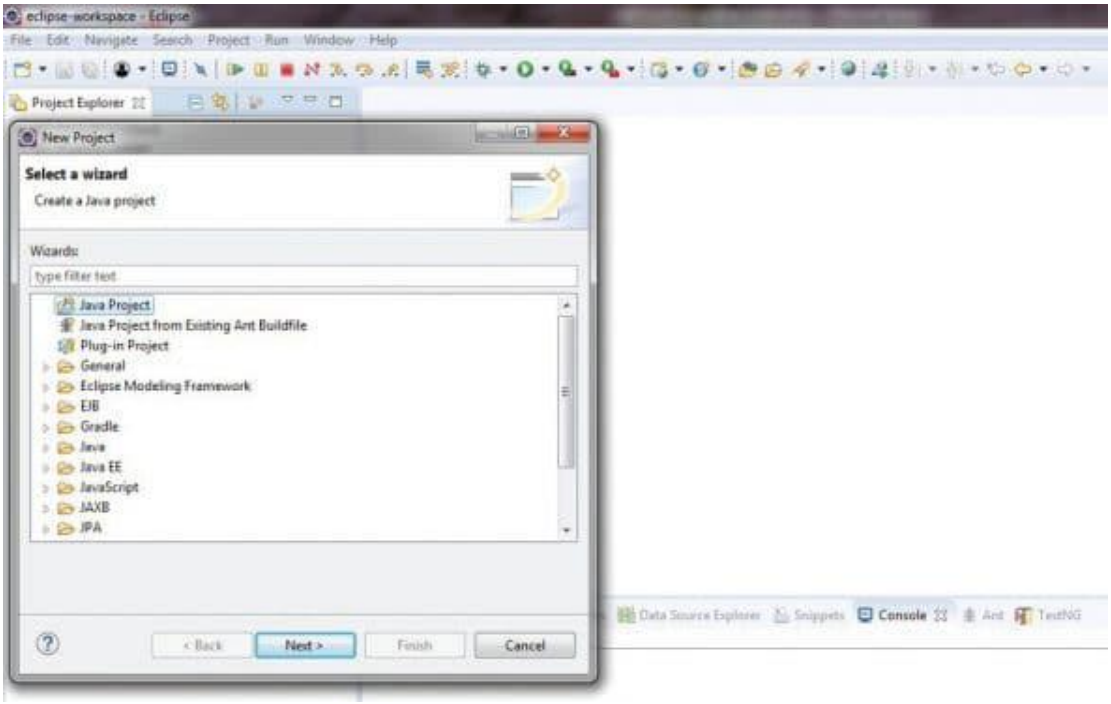
This is a high-performance open-source toolkit for fog computing, edge computing and IoT, which is used to model and simulate the networks of edge computing, the Internet of Things and fog computing. iFogSim integrates the resource management techniques that can be further customised as per the research area.

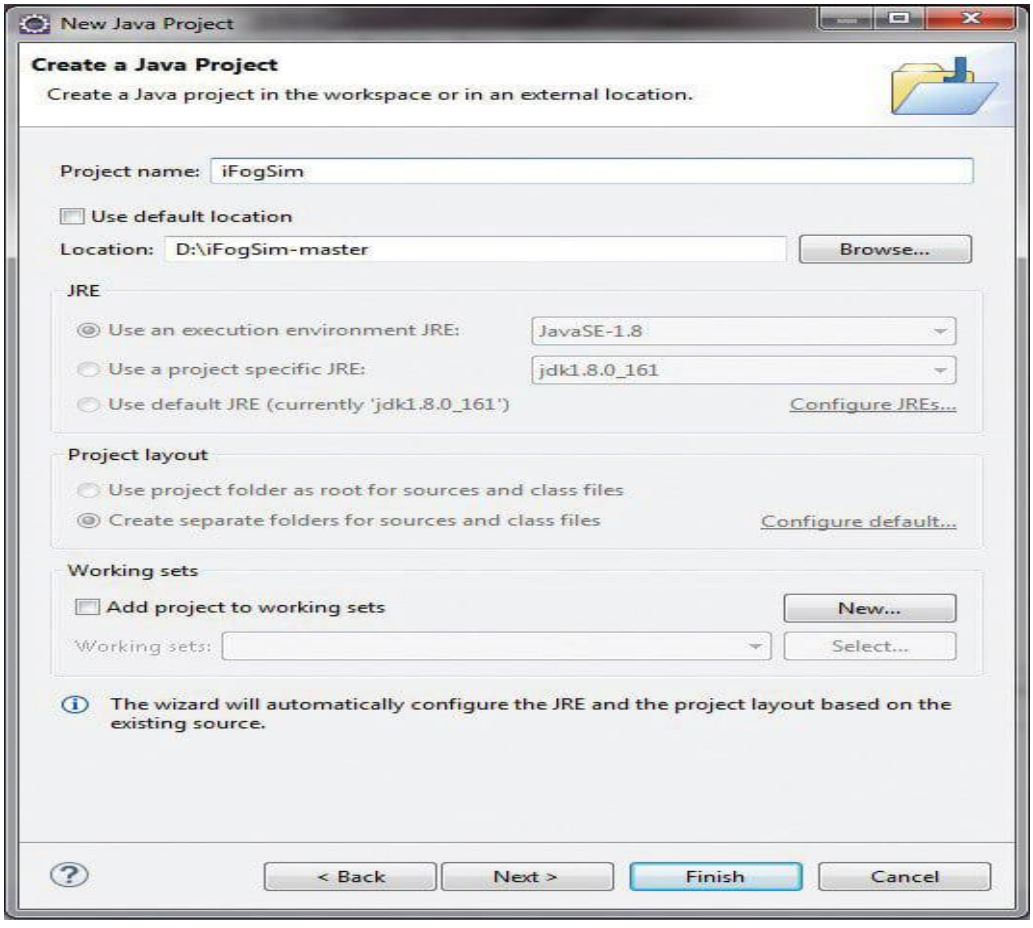
iFogSim works in association with CloudSim, a widely used library for the simulation of cloud-based environments and for resource management. The CloudSim layer handles the events between the components of fog computing using iFogSim.

The following are the classes of iFogSim that are required to simulate the fog network: Fog device, Sensor, Actuator, Tuple, Application, Monitoring edge, Resource management service.

Installing iFogSim

- The iFogSim library can be downloaded from the URL**:** *https://github.com/Cloudslab/iFogSim*.
- This library is written in Java, and therefore the Java Development Kit (JDK) will be required to customise and work with the toolkit.
- After downloading the compression toolkit in the Zip format, it is extracted, and a folder *iFogSim-master* is created. The iFogSim library can be executed on any Java based integrated development environment (IDE) like Eclipse, Netbeans, JCreator, JDeveloper, jGRASP, BlueJ, IntelliJ IDEA or Jbuilder.
- To integrate iFogSim on an Eclipse ID, we need to create a new project in the IDE.


eclipse-workspace - Eclipse
File Edit Navigate Search Project Run Window Help
Project Explorer
New Project
Select a wizard
Create a Java project
Wizards:
type filter text
Java Project
Java Project from Existing Ant Buildfile
Plug-in Project
General
Eclipse Modeling Framework
EJB
Gradle
Java
Java EE
JavaScript
JAXB
JPA
< Back
Next >
Finish
Cancel
Data Source Explorer
Snippets
Console
Ant
TestNG


New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.
Project name: iFogSim
Use default location
Location: D:\iFogSim-master
Browse...
JRE
Use an execution environment JRE: JavaSE-1.8
Use a project specific JRE: jdk1.8.0_161
Use default JRE (currently 'jdk1.8.0_161')
Configure JREs...
Project layout
Use project folder as root for sources and class files
Create separate folders for sources and class files
Configure default...
Working sets
Add project to working sets
New...
Working sets:
Select...
The wizard will automatically configure the JRE and the project layout based on the existing source.
< Back
Next >
Finish
Cancel

PRACTICAL-2

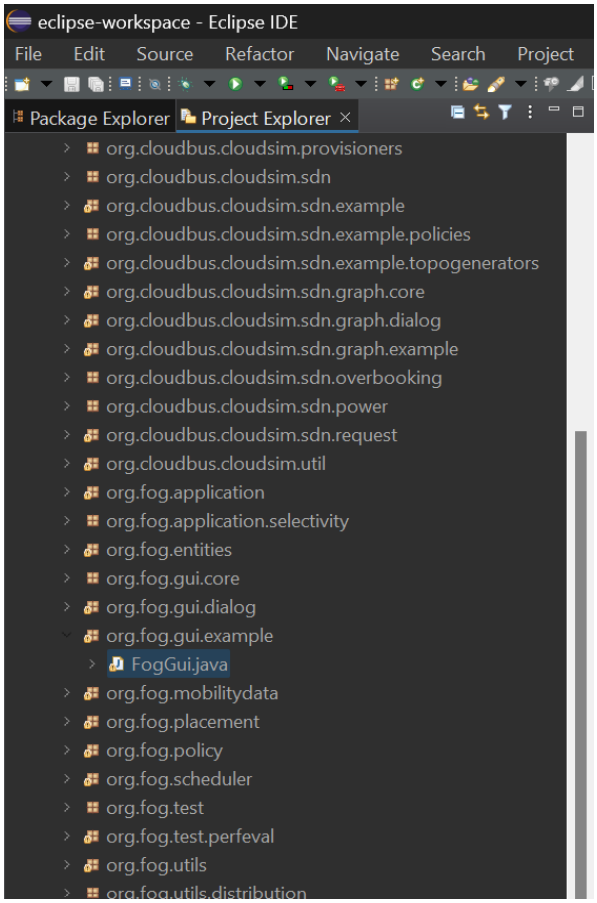
Create a topology and simulate it using iFogSim

Procedure:-

1.) Open Eclipse.

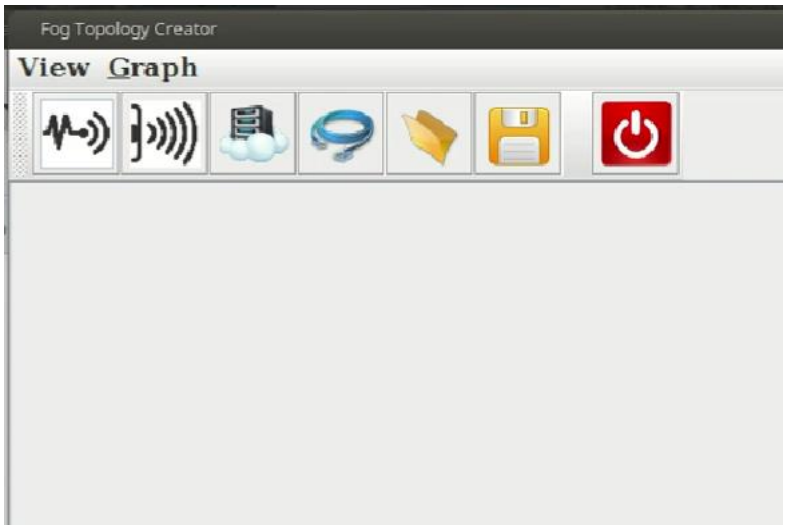
2.) Open the project in which you have imported ifogsim.

3.) Click on the package "org.fog.fui.example".



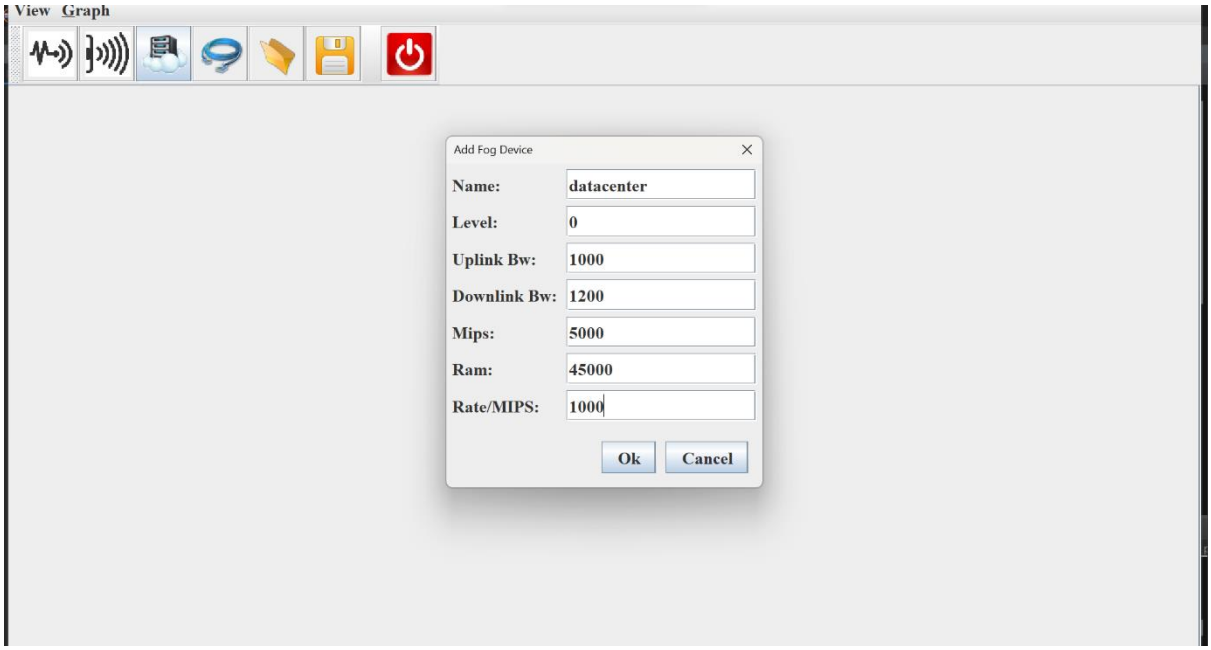
4.) Right Click on the file "FogGui.java".

5) Run as Java Application

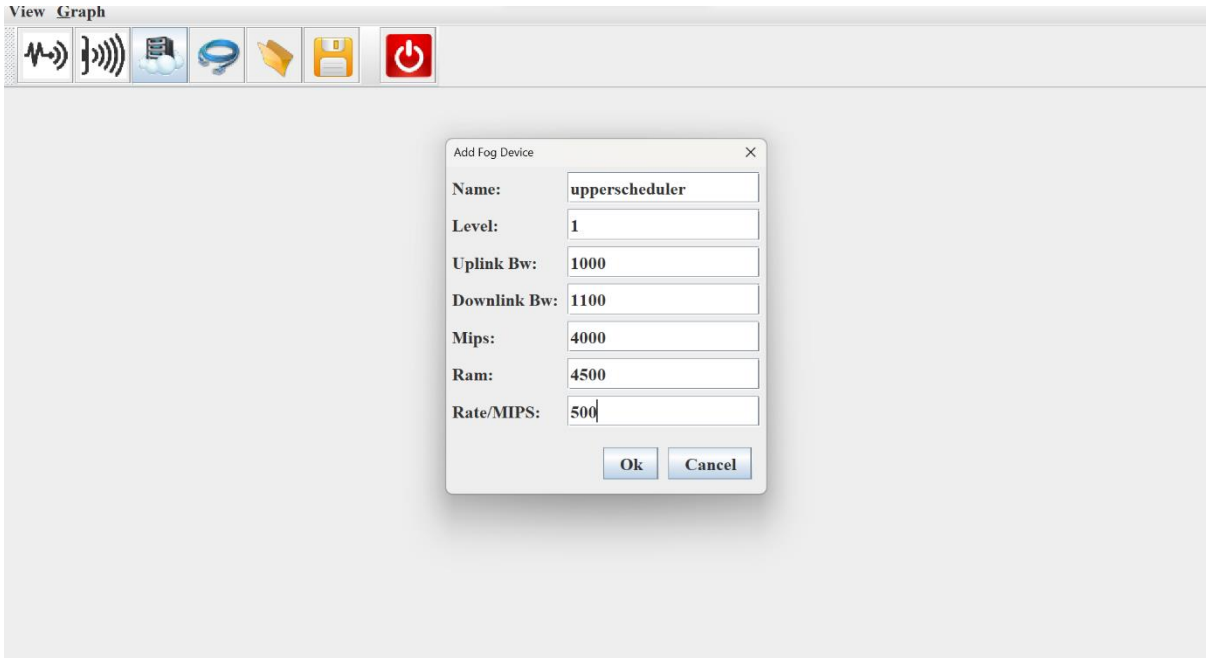


6) Now, Create a FogbasedScheduler. Add Fog devices.

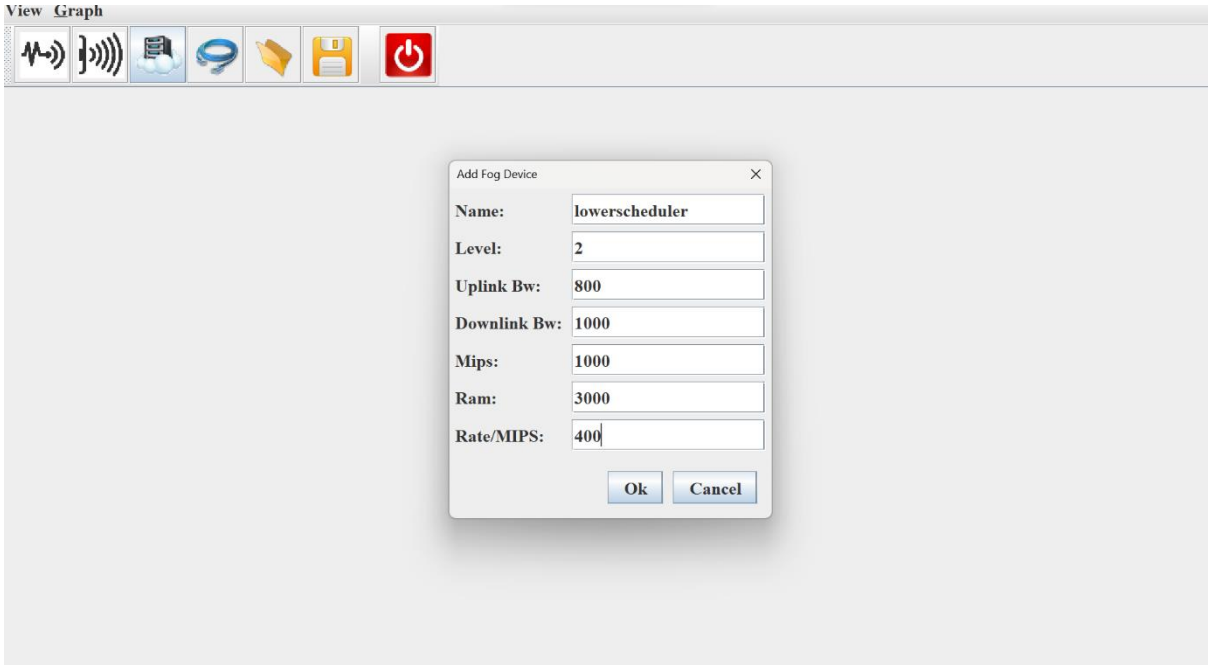
First, add the data center and configure the Parameters



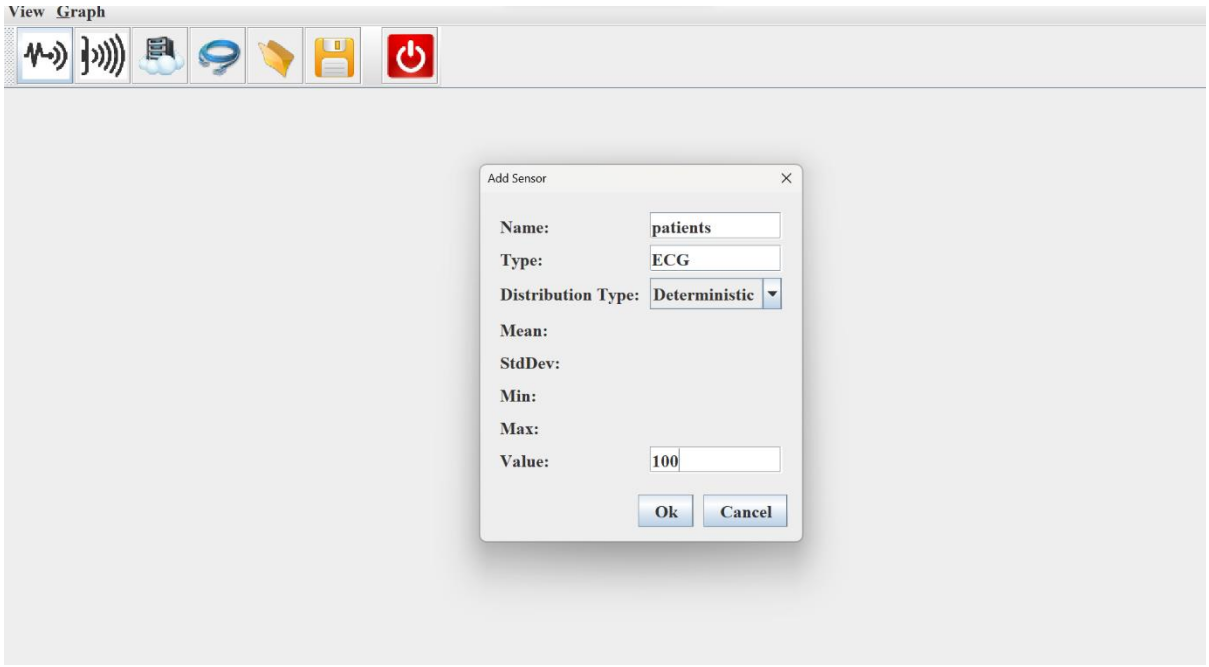
7)Now, add upperscheduler and configure the parameters.



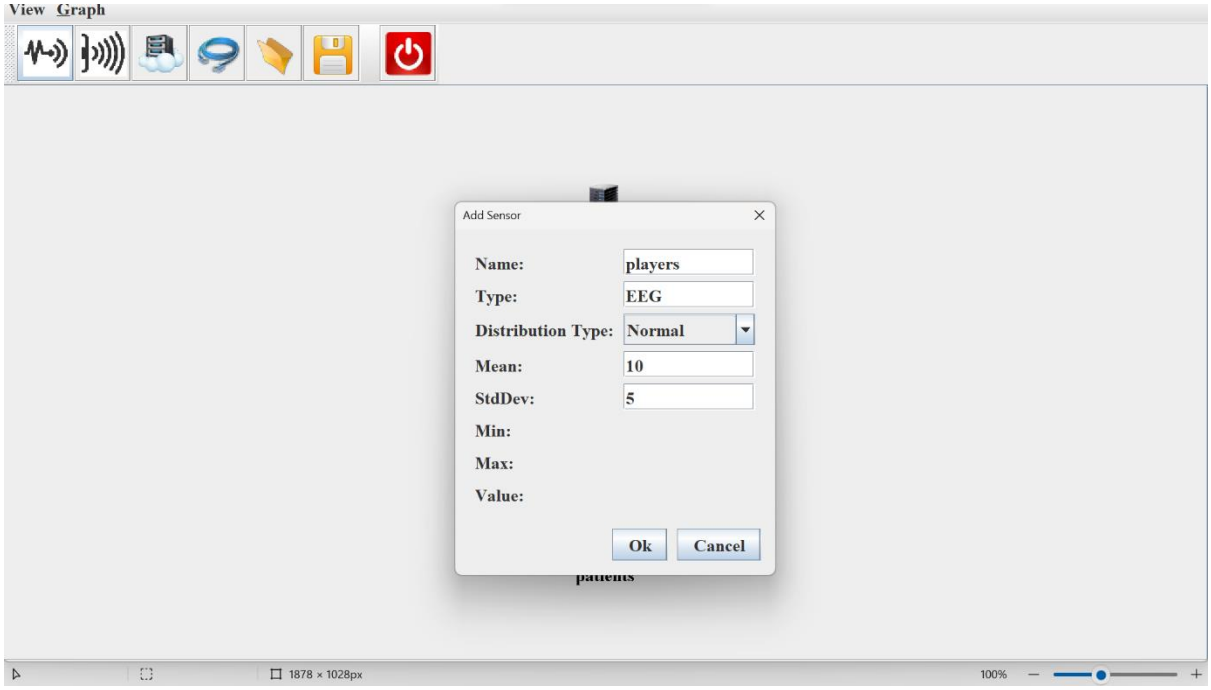
8) Add the lowerscheduler and configure it.



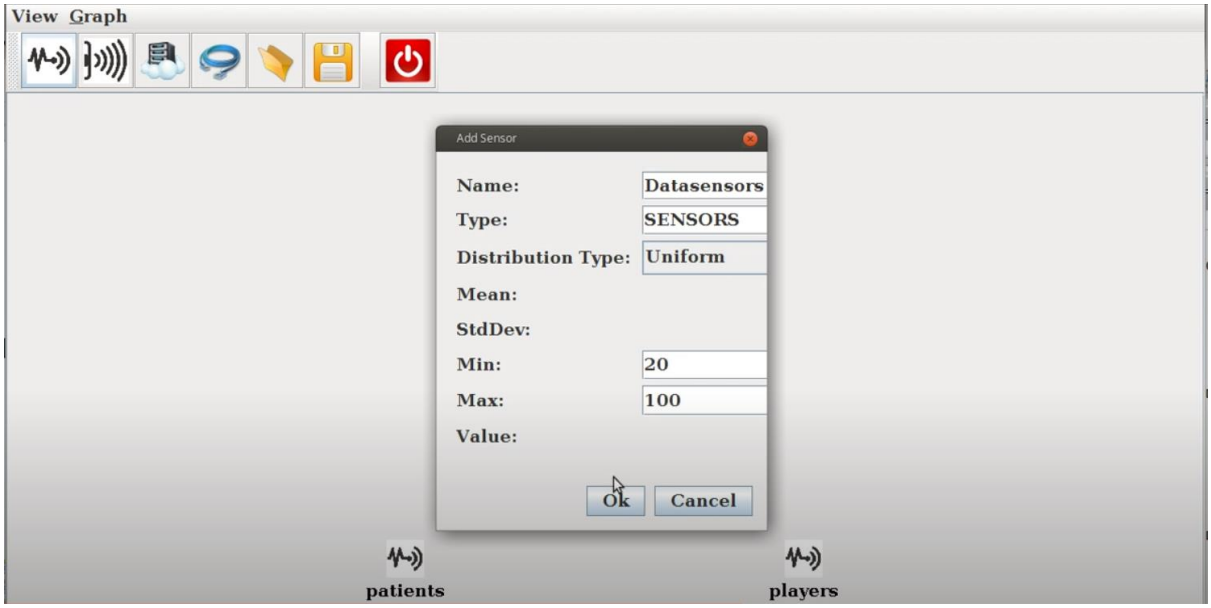
9)Now, add sensors (patients)



10) Add Players sensor

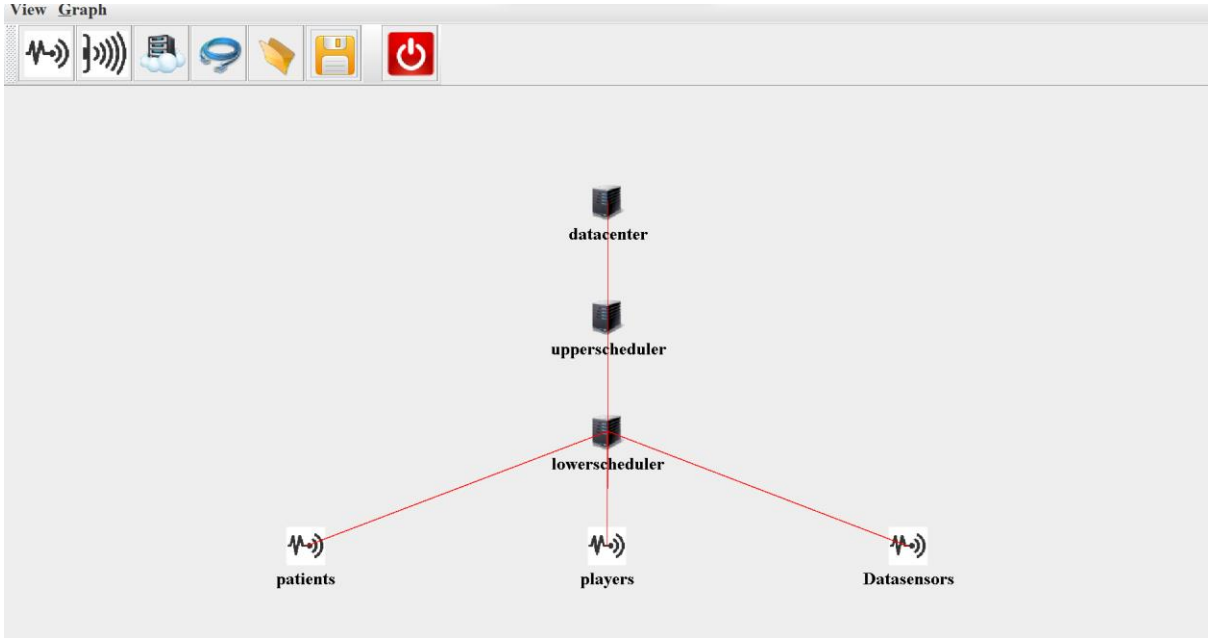


11)Add Datasensors

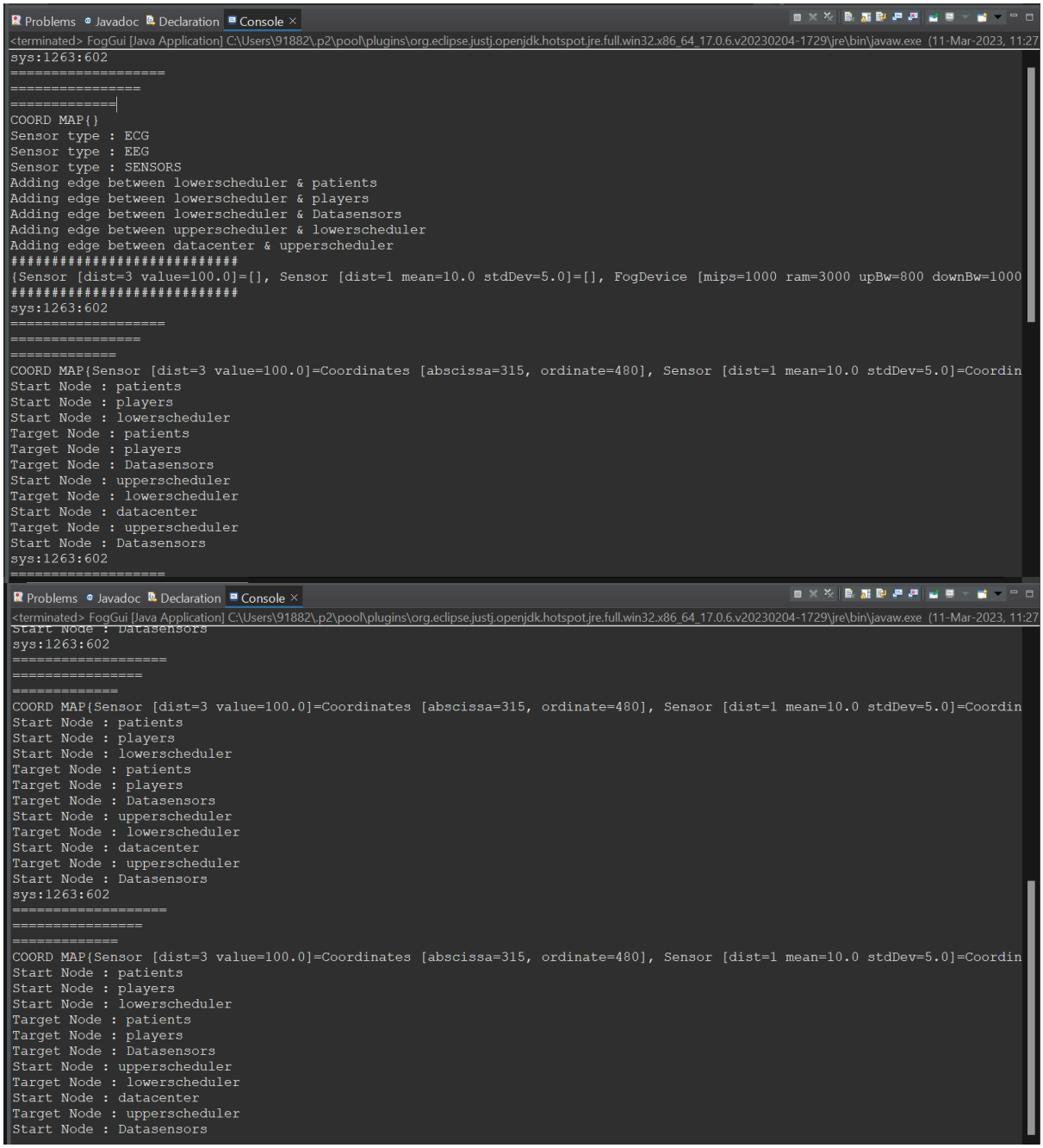


12) Add links between them and after adding links save the configuration by clicking on save button.

OUTPUT


View Graph
datacenter
upperscheduler
lowerscheduler
patients
players
Datasensors

OUTPUT IN CONSOLE :-



PRACTICAL-3

Perform scheduling and load balancing using iFogSim.

Procedure:-

1.) Open Eclipse .

2.) Open the project in which you have imported ifogsim.

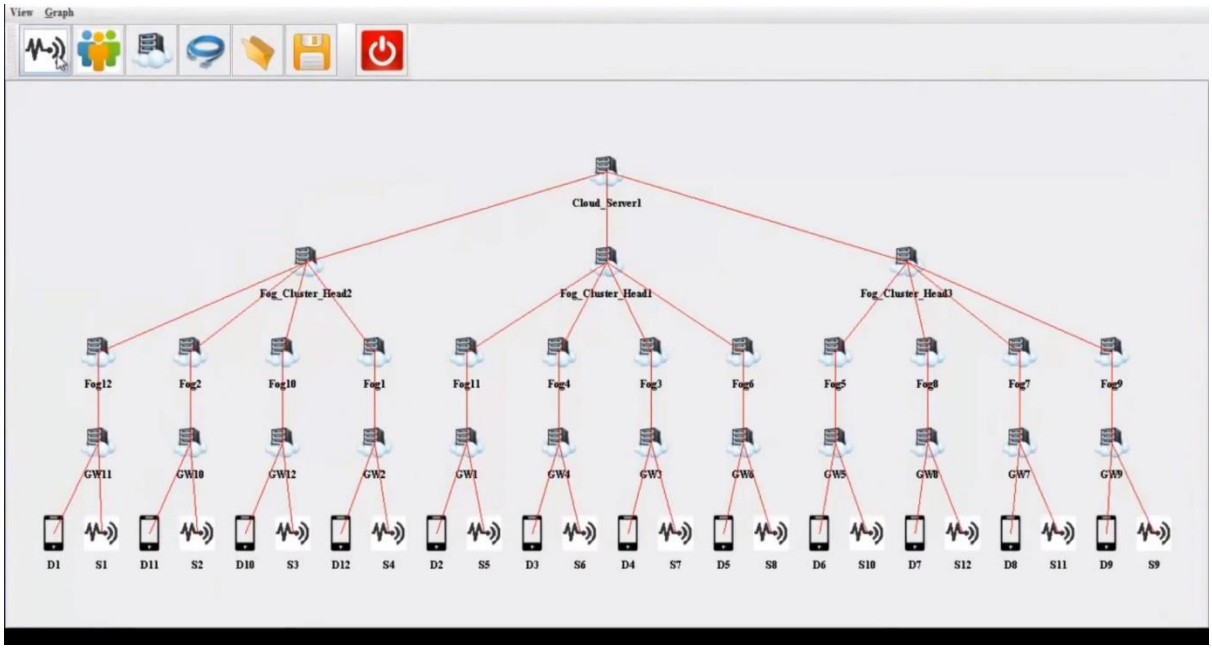
3.) Click on the package "org.fog.fui.example".

4.) Right Click on the file "FogGui.java".

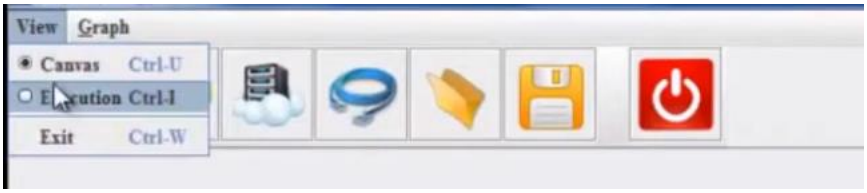
5) Run as Java Application

6.) Now Import a ‘Network Topology’ file

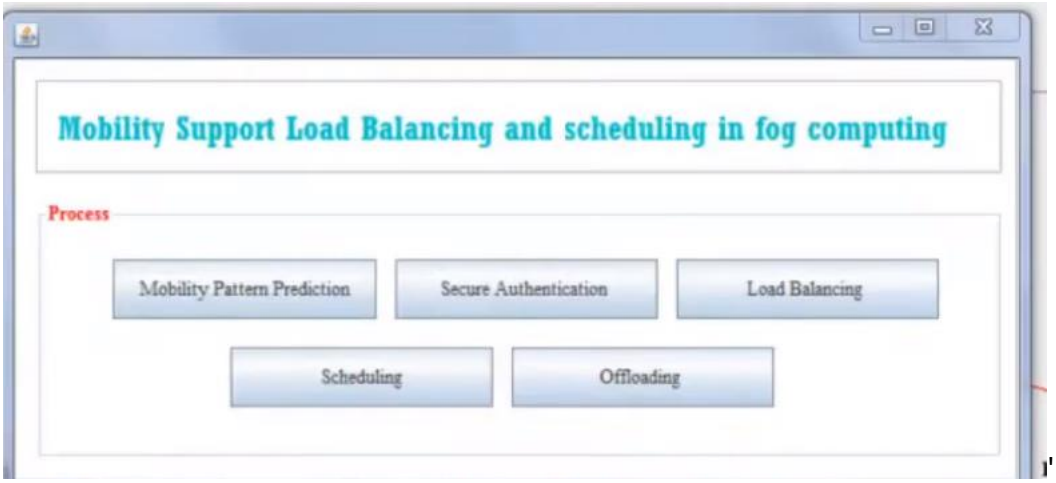
We have already created a topology network named as ‘NetwrokTopology’ used as to import our project.



7.) After successfully imported topology, then click Choose View-> Execution

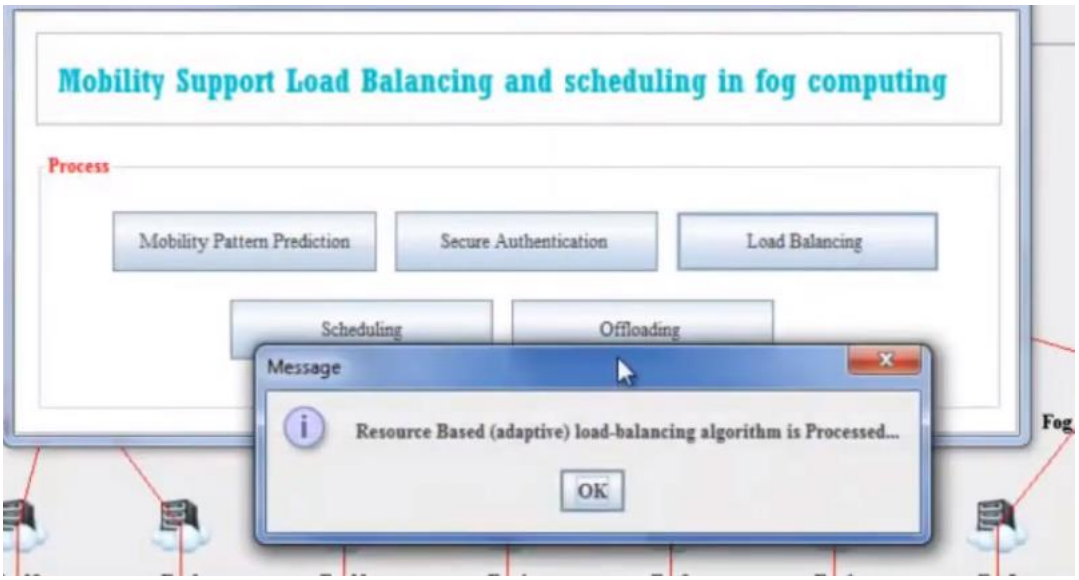


8.) Now mobility support load balancing and scheduling in fog computing dialog box will be popup.

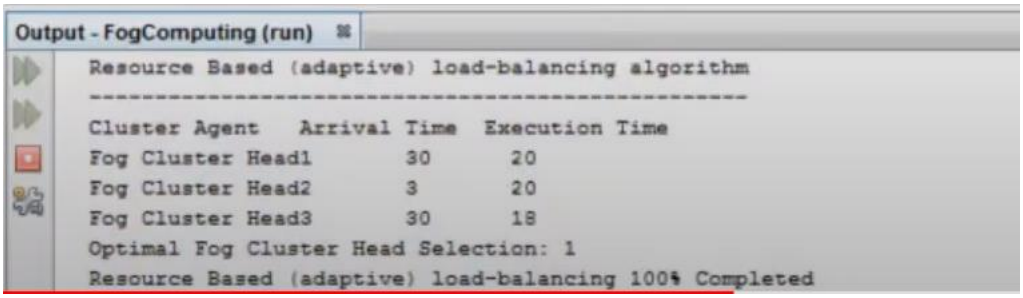


9)To perform load balancing ,click on load balancing option a message box will be pop up that is shown belowe :

Clicl on "ok" button.



OUTPUT FOR LOAD BALANCING

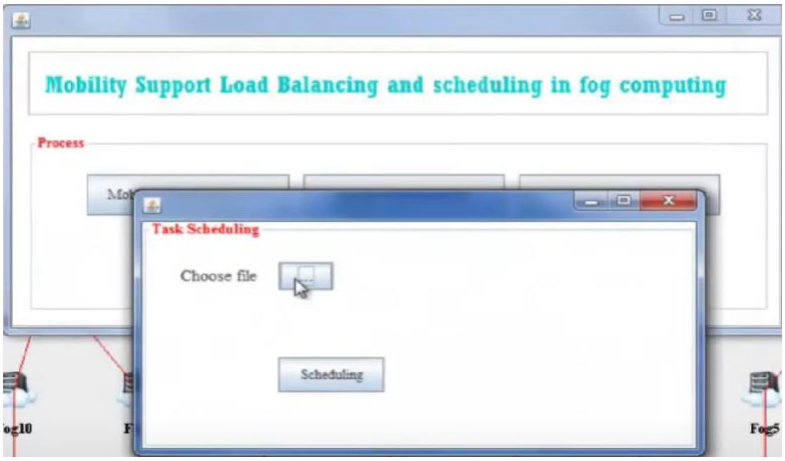


10) To schedule the task which is performed by using the following cases:

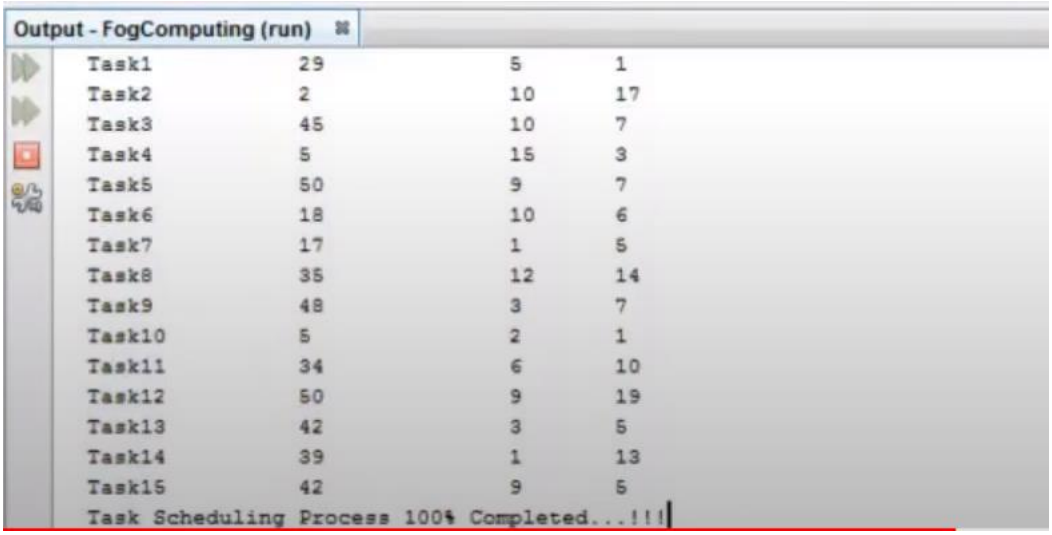
a. If the number process of is minimum Earliest Deadline First (EDF) is executed

b. If the process is overloaded Ant Colony Optimization algorithm is executed.

Click on scheduling option and choose the file then click on "scheduling" button.



OUTPUT FOR TASK SCHEDULING



PRACTICAL-4

Create a simulation setup for electroencephalography (EEG) Beam Tractor Game. Analyse the performance of the network in terms of CPU execution delay and energy consumed.

```
package org.fog.test.perfeval;


import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.power.PowerHost;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
import org.cloudbus.cloudsim.sdn.overbooking.BwProvisionerOverbooking;
import org.cloudbus.cloudsim.sdn.overbooking.PeProvisionerOverbooking;
import org.fog.application.AppEdge;
import org.fog.application.AppLoop;
import org.fog.application.Application;
import org.fog.application.selectivity.FractionalSelectivity;
import org.fog.entities.Actuator;
import org.fog.entities.FogBroker;
import org.fog.entities.FogDevice;
import org.fog.entities.FogDeviceCharacteristics;
import org.fog.entities.Sensor;
```

```
import org.fog.entities.Tuple;
import org.fog.placement.Controller;
import org.fog.placement.ModuleMapping;
import org.fog.placement.ModulePlacementMapping;
import org.fog.policy.AppModuleAllocationPolicy;
import org.fog.scheduler.StreamOperatorScheduler;
import org.fog.utils.FogLinearPowerModel;
import org.fog.utils.FogUtils;
import org.fog.utils.TimeKeeper;
import org.fog.utils.distribution.DeterministicDistribution;

/**
 * Simulation setup for case study 1 - EEG Beam Tractor Game
 * @author Harshit Gupta
 *
 */
public class TwoApps {
	static List<FogDevice> fogDevices = new ArrayList<FogDevice>();
	static List<FogDevice> mobiles = new ArrayList<FogDevice>();
	static List<Sensor> sensors = new ArrayList<Sensor>();
	static List<Actuator> actuators = new ArrayList<Actuator>();

	static int numOfDepts = 1;
	static int numOfMobilesPerDept = 4;
	static double EEG_TRANSMISSION_TIME = 5.1;
	//static double EEG_TRANSMISSION_TIME = 10;

	public static void main(String[] args) {

		Log.printLine("Starting TwoApps...");

		try {
			Log.disable();
```

```
int num_user = 1; // number of cloud users
Calendar calendar = Calendar.getInstance();
boolean trace_flag = false; // mean trace events

CloudSim.init(num_user, calendar, trace_flag);

String appId0 = "vr_game_0";
String appId1 = "vr_game_1";

FogBroker broker0 = new FogBroker("broker_0");
FogBroker broker1 = new FogBroker("broker_1");



Application application0 = createApplication0(appId0, broker0.getId());
Application application1 = createApplication1(appId1, broker1.getId());
application0.setUserId(broker0.getId());
application1.setUserId(broker1.getId());

createFogDevices();

createEdgeDevices0(broker0.getId(), appId0);
createEdgeDevices1(broker1.getId(), appId1);

ModuleMapping moduleMapping_0 = ModuleMapping.createModuleMapping(); // initializing a module mapping
ModuleMapping moduleMapping_1 = ModuleMapping.createModuleMapping(); // initializing a module mapping

moduleMapping_0.addModuleToDevice("connector", "cloud"); // fixing all instances of the Connector module to the Cloud
moduleMapping_0.addModuleToDevice("concentration_calculator", "cloud"); // fixing all instances of the Concentration Calculator module to the Cloud
moduleMapping_1.addModuleToDevice("connector_1", "cloud"); // fixing all instances of the Connector module to the Cloud
```

```
                moduleMapping_1.addModuleToDevice("concentration_calculator_1", "cloud"); // fixing all instances of the Concentration Calculator module to the Cloud

                for(FogDevice device : fogDevices){

                        if(device.getName().startsWith("m")){

                                moduleMapping_0.addModuleToDevice("client", device.getName());  // fixing all instances of the Client module to the Smartphones

                                moduleMapping_1.addModuleToDevice("client_1", device.getName());  // fixing all instances of the Client module to the Smartphones

                        }

                }


                Controller controller = new Controller("master-controller", fogDevices, sensors,

                                actuators);


                controller.submitApplication(application0, new ModulePlacementMapping(fogDevices, application0, moduleMapping_0));

                controller.submitApplication(application1, 1000, new ModulePlacementMapping(fogDevices, application1, moduleMapping_1));


        TimeKeeper.getInstance().setSimulationStartTime(Calendar.getInstance().getTimeInMillis());


                CloudSim.startSimulation();


                CloudSim.stopSimulation();


                Log.printLine("VRGame finished!");

            } catch (Exception e) {

                e.printStackTrace();

                Log.printLine("Unwanted errors happen");

            }

        }


        private static void createEdgeDevices0(int userId, String appId) {

            for(FogDevice mobile : mobiles){
```

```
                String id = mobile.getName();

                Sensor eegSensor = new Sensor("s-"+appId+"-"+id, "EEG", userId, appId, new
DeterministicDistribution(EEG_TRANSMISSION_TIME)); // inter-transmission time of EEG sensor follows a
deterministic distribution

                sensors.add(eegSensor);

                Actuator display = new Actuator("a-"+appId+"-"+id, userId, appId, "DISPLAY");

                actuators.add(display);

                eegSensor.setGatewayDeviceId(mobile.getId());

                eegSensor.setLatency(6.0);  // latency of connection between EEG sensors and the
parent Smartphone is 6 ms

                display.setGatewayDeviceId(mobile.getId());

                display.setLatency(1.0);  // latency of connection between Display actuator and the
parent Smartphone is 1 ms

            }

        }


        private static void createEdgeDevices1(int userId, String appId) {

            for(FogDevice mobile : mobiles){

                String id = mobile.getName();

                Sensor eegSensor = new Sensor("s-"+appId+"-"+id, "EEG_1", userId, appId, new
DeterministicDistribution(EEG_TRANSMISSION_TIME)); // inter-transmission time of EEG sensor follows a
deterministic distribution

                sensors.add(eegSensor);

                Actuator display = new Actuator("a-"+appId+"-"+id, userId, appId, "DISPLAY_1");

                actuators.add(display);

                eegSensor.setGatewayDeviceId(mobile.getId());

                eegSensor.setLatency(6.0);  // latency of connection between EEG sensors and the
parent Smartphone is 6 ms

                display.setGatewayDeviceId(mobile.getId());

                display.setLatency(1.0);  // latency of connection between Display actuator and the
parent Smartphone is 1 ms

            }

        }


        /**

         * Creates the fog devices in the physical topology of the simulation.
```

```
	 * @param userId
	 * @param appId
	 */
	private static void createFogDevices() {
		FogDevice cloud = createFogDevice("cloud", 44800, 40000, 100, 10000, 0, 0.01, 16*103, 16*83.25); // creates the fog device Cloud at the apex of the hierarchy with level=0
		cloud.setParentId(-1);
		FogDevice proxy = createFogDevice("proxy-server", 2800, 4000, 10000, 10000, 1, 0.0, 107.339, 83.4333); // creates the fog device Proxy Server (level=1)
		proxy.setParentId(cloud.getId()); // setting Cloud as parent of the Proxy Server
		proxy.setUplinkLatency(100); // latency of connection from Proxy Server to the Cloud is 100 ms

		fogDevices.add(cloud);
		fogDevices.add(proxy);

		for(int i=0;i<numOfDepts;i++){
			addGw(i+"", proxy.getId()); // adding a fog device for every Gateway in physical topology. The parent of each gateway is the Proxy Server
		}

	}

	private static FogDevice addGw(String id, int parentId){
		FogDevice dept = createFogDevice("d-"+id, 2800, 4000, 10000, 10000, 1, 0.0, 107.339, 83.4333);
		fogDevices.add(dept);
		dept.setParentId(parentId);
		dept.setUplinkLatency(4); // latency of connection between gateways and proxy server is 4 ms
		for(int i=0;i<numOfMobilesPerDept;i++){
			String mobileId = id+"-"+i;
			FogDevice mobile = addMobile(mobileId, dept.getId()); // adding mobiles to the physical topology. Smartphones have been modeled as fog devices as well.
```

```
                mobile.setUplinkLatency(2); // latency of connection between the smartphone and proxy server is 4 ms
                fogDevices.add(mobile);
            }
            return dept;
        }

        private static FogDevice addMobile(String id, int parentId){
            FogDevice mobile = createFogDevice("m-"+id, 1000, 1000, 10000, 270, 3, 0, 87.53, 82.44);
            mobile.setParentId(parentId);
            mobiles.add(mobile);
            /*Sensor eegSensor = new Sensor("s-"+id, "EEG", userId, appId, new DeterministicDistribution(EEG_TRANSMISSION_TIME)); // inter-transmission time of EEG sensor follows a deterministic distribution
            sensors.add(eegSensor);
            Actuator display = new Actuator("a-"+id, userId, appId, "DISPLAY");
            actuators.add(display);
            eegSensor.setGatewayDeviceId(mobile.getId());
            eegSensor.setLatency(6.0);  // latency of connection between EEG sensors and the parent Smartphone is 6 ms
            display.setGatewayDeviceId(mobile.getId());
            display.setLatency(1.0);  // latency of connection between Display actuator and the parent Smartphone is 1 ms
*/          return mobile;
        }

        /**
         * Creates a vanilla fog device
         * @param nodeName name of the device to be used in simulation
         * @param mips MIPS
         * @param ram RAM
         * @param upBw uplink bandwidth
         * @param downBw downlink bandwidth
         * @param level hierarchy level of the device
         * @param ratePerMips cost rate per MIPS used
```

```
	 * @param busyPower
	 * @param idlePower
	 * @return
	 */
	private static FogDevice createFogDevice(String nodeName, long mips,
			int ram, long upBw, long downBw, int level, double ratePerMips, double busyPower,
double idlePower) {

		List<Pe> peList = new ArrayList<Pe>();

		// 3. Create PEs and add these into a list.
		peList.add(new Pe(0, new PeProvisionerOverbooking(mips))); // need to store Pe id and
MIPS Rating

		int hostId = FogUtils.generateEntityId();
		long storage = 1000000; // host storage
		int bw = 10000;

		PowerHost host = new PowerHost(
				hostId,
				new RamProvisionerSimple(ram),
				new BwProvisionerOverbooking(bw),
				storage,
				peList,
				new StreamOperatorScheduler(peList),
				new FogLinearPowerModel(busyPower, idlePower)
			);

		List<Host> hostList = new ArrayList<Host>();
		hostList.add(host);

		String arch = "x86"; // system architecture
		String os = "Linux"; // operating system
		String vmm = "Xen";
```

```
		double time_zone = 10.0; // time zone this resource located
		double cost = 3.0; // the cost of using processing in this resource
		double costPerMem = 0.05; // the cost of using memory in this resource
		double costPerStorage = 0.001; // the cost of using storage in this
                                                                        // resource
		double costPerBw = 0.0; // the cost of using bw in this resource
		LinkedList<Storage> storageList = new LinkedList<Storage>(); // we are not adding SAN

	// devices by now

		FogDeviceCharacteristics characteristics = new FogDeviceCharacteristics(
				arch, os, vmm, host, time_zone, cost, costPerMem,
				costPerStorage, costPerBw);

		FogDevice fogdevice = null;
		try {
			fogdevice = new FogDevice(nodeName, characteristics,
					new AppModuleAllocationPolicy(hostList), storageList, 10, upBw,
downBw, 0, ratePerMips);
		} catch (Exception e) {
			e.printStackTrace();
		}

		fogdevice.setLevel(level);
		return fogdevice;
	}

	/**
	 * Function to create the EEG Tractor Beam game application in the DDF model.
	 * @param appId unique identifier of the application
	 * @param userId identifier of the user of the application
	 * @return
	 */
	@SuppressWarnings({"serial" })
```

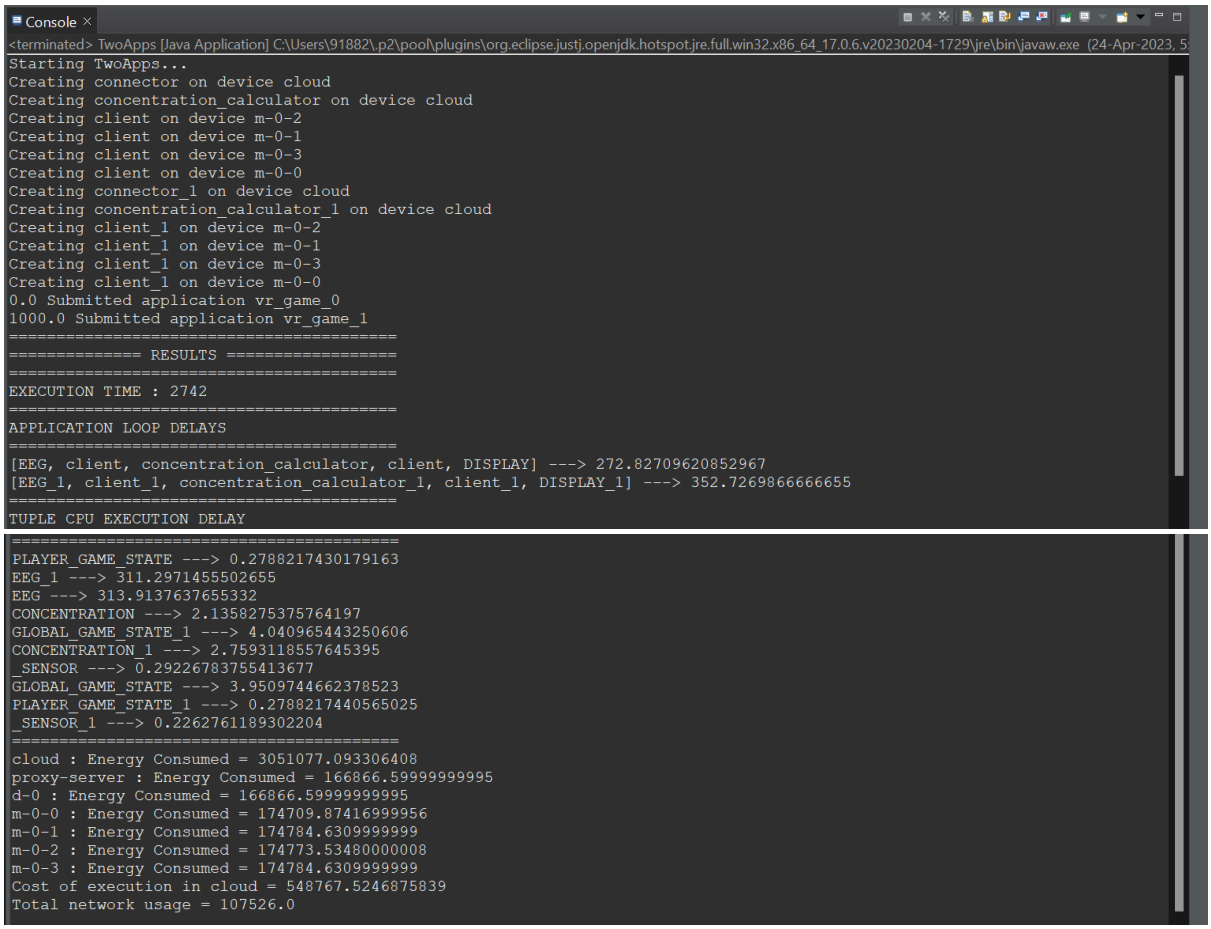
```
	private static Application createApplication0(String appId, int userId){


		Application application = Application.createApplication(appId, userId); // creates an empty application model (empty directed graph)


		/*
		 * Adding modules (vertices) to the application model (directed graph)
		 */
		application.addAppModule("client", 10); // adding module Client to the application model
		application.addAppModule("concentration_calculator", 10); // adding module Concentration Calculator to the application model
		application.addAppModule("connector", 10); // adding module Connector to the application model


		/*
		 * Connecting the application modules (vertices) in the application model (directed graph) with edges
		 */
		if(EEG_TRANSMISSION_TIME==10)
			application.addAppEdge("EEG", "client", 2000, 500, "EEG", Tuple.UP, AppEdge.SENSOR); // adding edge from EEG (sensor) to Client module carrying tuples of type EEG
		else
			application.addAppEdge("EEG", "client", 3000, 500, "EEG", Tuple.UP, AppEdge.SENSOR);
		application.addAppEdge("client", "concentration_calculator", 3500, 500, "_SENSOR", Tuple.UP, AppEdge.MODULE); // adding edge from Client to Concentration Calculator module carrying tuples of type _SENSOR
		application.addAppEdge("concentration_calculator", "connector", 100, 1000, 1000, "PLAYER_GAME_STATE", Tuple.UP, AppEdge.MODULE); // adding periodic edge (period=1000ms) from Concentration Calculator to Connector module carrying tuples of type PLAYER_GAME_STATE
		application.addAppEdge("concentration_calculator", "client", 14, 500, "CONCENTRATION", Tuple.DOWN, AppEdge.MODULE);  // adding edge from Concentration Calculator to Client module carrying tuples of type CONCENTRATION
		application.addAppEdge("connector", "client", 100, 28, 1000, "GLOBAL_GAME_STATE", Tuple.DOWN, AppEdge.MODULE); // adding periodic edge (period=1000ms) from Connector to Client module carrying tuples of type GLOBAL_GAME_STATE
		application.addAppEdge("client", "DISPLAY", 1000, 500, "SELF_STATE_UPDATE", Tuple.DOWN, AppEdge.ACTUATOR);  // adding edge from Client module to Display (actuator) carrying tuples of type SELF_STATE_UPDATE
```

```
		application.addAppEdge("client", "DISPLAY", 1000, 500, "GLOBAL_STATE_UPDATE", Tuple.DOWN, AppEdge.ACTUATOR);  // adding edge from Client module to Display (actuator) carrying tuples of type GLOBAL_STATE_UPDATE


		/*
		 * Defining the input-output relationships (represented by selectivity) of the application modules.
		 */
		application.addTupleMapping("client", "EEG", "_SENSOR", new FractionalSelectivity(0.9)); // 0.9 tuples of type _SENSOR are emitted by Client module per incoming tuple of type EEG
		application.addTupleMapping("client", "CONCENTRATION", "SELF_STATE_UPDATE", new FractionalSelectivity(1.0)); // 1.0 tuples of type SELF_STATE_UPDATE are emitted by Client module per incoming tuple of type CONCENTRATION
		application.addTupleMapping("concentration_calculator", "_SENSOR", "CONCENTRATION", new FractionalSelectivity(1.0)); // 1.0 tuples of type CONCENTRATION are emitted by Concentration Calculator module per incoming tuple of type _SENSOR
		application.addTupleMapping("client", "GLOBAL_GAME_STATE", "GLOBAL_STATE_UPDATE", new FractionalSelectivity(1.0)); // 1.0 tuples of type GLOBAL_STATE_UPDATE are emitted by Client module per incoming tuple of type GLOBAL_GAME_STATE


		/*
		 * Defining application loops to monitor the latency of.
		 * Here, we add only one loop for monitoring : EEG(sensor) -> Client -> Concentration Calculator -> Client -> DISPLAY (actuator)
		 */
		final AppLoop loop1 = new AppLoop(new ArrayList<String>(){{add("EEG");add("client");add("concentration_calculator");add("client");add("DISPLAY");}});
		List<AppLoop> loops = new ArrayList<AppLoop>(){{add(loop1);}};
		application.setLoops(loops);


		return application;
	}


	@SuppressWarnings({"serial" })
	private static Application createApplication1(String appId, int userId){


		Application application = Application.createApplication(appId, userId); // creates an empty application model (empty directed graph)
```

```
		/*
		 * Adding modules (vertices) to the application model (directed graph)
		 */
		application.addAppModule("client_1", 10); // adding module Client to the application model
		application.addAppModule("concentration_calculator_1", 10); // adding module Concentration Calculator to the application model
		application.addAppModule("connector_1", 10); // adding module Connector to the application model


		/*
		 * Connecting the application modules (vertices) in the application model (directed graph) with edges
		 */
		if(EEG_TRANSMISSION_TIME==10)
			application.addAppEdge("EEG_1", "client_1", 2000, 500, "EEG_1", Tuple.UP, AppEdge.SENSOR); // adding edge from EEG (sensor) to Client module carrying tuples of type EEG
		else
			application.addAppEdge("EEG_1", "client_1", 3000, 500, "EEG_1", Tuple.UP, AppEdge.SENSOR);
		application.addAppEdge("client_1", "concentration_calculator_1", 3500, 500, "_SENSOR_1", Tuple.UP, AppEdge.MODULE); // adding edge from Client to Concentration Calculator module carrying tuples of type _SENSOR
		application.addAppEdge("concentration_calculator_1", "connector_1", 100, 1000, 1000, "PLAYER_GAME_STATE_1", Tuple.UP, AppEdge.MODULE); // adding periodic edge (period=1000ms) from Concentration Calculator to Connector module carrying tuples of type PLAYER_GAME_STATE
		application.addAppEdge("concentration_calculator_1", "client_1", 14, 500, "CONCENTRATION_1", Tuple.DOWN, AppEdge.MODULE);  // adding edge from Concentration Calculator to Client module carrying tuples of type CONCENTRATION
		application.addAppEdge("connector_1", "client_1", 100, 28, 1000, "GLOBAL_GAME_STATE_1", Tuple.DOWN, AppEdge.MODULE); // adding periodic edge (period=1000ms) from Connector to Client module carrying tuples of type GLOBAL_GAME_STATE
		application.addAppEdge("client_1", "DISPLAY_1", 1000, 500, "SELF_STATE_UPDATE_1", Tuple.DOWN, AppEdge.ACTUATOR);  // adding edge from Client module to Display (actuator) carrying tuples of type SELF_STATE_UPDATE
		application.addAppEdge("client_1", "DISPLAY_1", 1000, 500, "GLOBAL_STATE_UPDATE_1", Tuple.DOWN, AppEdge.ACTUATOR);  // adding edge from Client module to Display (actuator) carrying tuples of type GLOBAL_STATE_UPDATE
```

```
                /*
                 * Defining the input-output relationships (represented by selectivity) of the application modules.
                 */
                application.addTupleMapping("client_1", "EEG_1", "_SENSOR_1", new FractionalSelectivity(0.9)); // 0.9 tuples of type _SENSOR are emitted by Client module per incoming tuple of type EEG
                application.addTupleMapping("client_1", "CONCENTRATION_1", "SELF_STATE_UPDATE_1", new FractionalSelectivity(1.0)); // 1.0 tuples of type SELF_STATE_UPDATE are emitted by Client module per incoming tuple of type CONCENTRATION
                application.addTupleMapping("concentration_calculator_1", "_SENSOR_1", "CONCENTRATION_1", new FractionalSelectivity(1.0)); // 1.0 tuples of type CONCENTRATION are emitted by Concentration Calculator module per incoming tuple of type _SENSOR
                application.addTupleMapping("client_1", "GLOBAL_GAME_STATE_1", "GLOBAL_STATE_UPDATE_1", new FractionalSelectivity(1.0)); // 1.0 tuples of type GLOBAL_STATE_UPDATE are emitted by Client module per incoming tuple of type GLOBAL_GAME_STATE


                /*
                 * Defining application loops to monitor the latency of.
                 * Here, we add only one loop for monitoring : EEG(sensor) -> Client -> Concentration Calculator -> Client -> DISPLAY (actuator)
                 */
                final AppLoop loop1 = new AppLoop(new ArrayList<String>(){{add("EEG_1");add("client_1");add("concentration_calculator_1");add("client_1");add("DISPLAY_1");}});
                List<AppLoop> loops = new ArrayList<AppLoop>(){{add(loop1);}};
                application.setLoops(loops);


                return application;
        }
}
```

OUTPUT



```
Console ×
<terminated> TwoApps [Java Application] C:\Users\91882\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1729\jre\bin\javaw.exe  (24-Apr-2023, 5
Starting TwoApps...
Creating connector on device cloud
Creating concentration_calculator on device cloud
Creating client on device m-0-2
Creating client on device m-0-1
Creating client on device m-0-3
Creating client on device m-0-0
Creating connector_1 on device cloud
Creating concentration_calculator_1 on device cloud
Creating client_1 on device m-0-2
Creating client_1 on device m-0-1
Creating client_1 on device m-0-3
Creating client_1 on device m-0-0
0.0 Submitted application vr_game_0
1000.0 Submitted application vr_game_1
=======================================
============= RESULTS =================
=======================================
EXECUTION TIME : 2742
=======================================
APPLICATION LOOP DELAYS
=======================================
[EEG, client, concentration_calculator, client, DISPLAY] ---> 272.82709620852967
[EEG_1, client_1, concentration_calculator_1, client_1, DISPLAY_1] ---> 352.7269866666655
=======================================
TUPLE CPU EXECUTION DELAY
```



```
=======================================
PLAYER_GAME_STATE ---> 0.2788217430179163
EEG_1 ---> 311.2971455502655
EEG ---> 313.9137637655332
CONCENTRATION ---> 2.1358275375764197
GLOBAL_GAME_STATE_1 ---> 4.040965443250606
CONCENTRATION_1 ---> 2.7593118557645395
_SENSOR ---> 0.29226783755413677
GLOBAL_GAME_STATE ---> 3.9509744662378523
PLAYER_GAME_STATE_1 ---> 0.2788217440565025
_SENSOR_1 ---> 0.2262761189302204
=======================================
cloud : Energy Consumed = 3051077.093306408
proxy-server : Energy Consumed = 166866.59999999995
d-0 : Energy Consumed = 166866.59999999995
m-0-0 : Energy Consumed = 174709.87416999956
m-0-1 : Energy Consumed = 174784.6309999999
m-0-2 : Energy Consumed = 174773.53480000008
m-0-3 : Energy Consumed = 174784.6309999999
Cost of execution in cloud = 548767.5246875839
Total network usage = 107526.0
```

PRACTICAL-5

Create a simulation setup for a VR Game. Analyse the performance of the network in terms of CPU execution delay and energy consumed.

```
package org.fog.test.perfeval;


import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.power.PowerHost;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
import org.cloudbus.cloudsim.sdn.overbooking.BwProvisionerOverbooking;
import org.cloudbus.cloudsim.sdn.overbooking.PeProvisionerOverbooking;
import org.fog.application.AppEdge;
import org.fog.application.AppLoop;
import org.fog.application.Application;
import org.fog.application.selectivity.FractionalSelectivity;
import org.fog.entities.Actuator;
import org.fog.entities.FogBroker;
import org.fog.entities.FogDevice;
import org.fog.entities.FogDeviceCharacteristics;
import org.fog.entities.Sensor;
```

```
import org.fog.entities.Tuple;
import org.fog.placement.Controller;
import org.fog.placement.ModuleMapping;
import org.fog.placement.ModulePlacementEdgewards;
import org.fog.placement.ModulePlacementMapping;
import org.fog.policy.AppModuleAllocationPolicy;
import org.fog.scheduler.StreamOperatorScheduler;
import org.fog.utils.FogLinearPowerModel;
import org.fog.utils.FogUtils;
import org.fog.utils.TimeKeeper;
import org.fog.utils.distribution.DeterministicDistribution;

/**
 * Simulation setup for case study 1 - EEG Beam Tractor Game
 * @author Harshit Gupta
 *
 */
public class VRGameFog {
	static List<FogDevice> fogDevices = new ArrayList<FogDevice>();
	static List<Sensor> sensors = new ArrayList<Sensor>();
	static List<Actuator> actuators = new ArrayList<Actuator>();

	static boolean CLOUD = false;

	static int numOfDepts = 2;
	static int numOfMobilesPerDept = 5;
	static double EEG_TRANSMISSION_TIME = 5;

	public static void main(String[] args) {

		Log.printLine("Starting VRGame...");

		try {
```

```
Log.disable();
int num_user = 1; // number of cloud users
Calendar calendar = Calendar.getInstance();
boolean trace_flag = false; // mean trace events

CloudSim.init(num_user, calendar, trace_flag);

String appId = "vr_game"; // identifier of the application

FogBroker broker = new FogBroker("broker");

Application application = createApplication(appId, broker.getId());
application.setUserId(broker.getId());

createFogDevices(broker.getId(), appId);

ModuleMapping moduleMapping = ModuleMapping.createModuleMapping(); // initializing a module mapping

if(CLOUD){
    // if the mode of deployment is cloud-based
    /*moduleMapping.addModuleToDevice("connector", "cloud", numOfDepts*numOfMobilesPerDept); // fixing all instances of the Connector module to the Cloud
    moduleMapping.addModuleToDevice("concentration_calculator", "cloud", numOfDepts*numOfMobilesPerDept); // fixing all instances of the Concentration Calculator module to the Cloud
*/  moduleMapping.addModuleToDevice("connector", "cloud"); // fixing all instances of the Connector module to the Cloud
    moduleMapping.addModuleToDevice("concentration_calculator", "cloud"); // fixing all instances of the Concentration Calculator module to the Cloud
    for(FogDevice device : fogDevices){
        if(device.getName().startsWith("m")){
            //moduleMapping.addModuleToDevice("client", device.getName(), 1);  // fixing all instances of the Client module to the Smartphones
            moduleMapping.addModuleToDevice("client", device.getName());  // fixing all instances of the Client module to the Smartphones
```

```
                                        }

                                }

                        }else{

                                // if the mode of deployment is cloud-based

                                //moduleMapping.addModuleToDevice("connector", "cloud",
numOfDepts*numOfMobilesPerDept); // fixing all instances of the Connector module to the Cloud

                                moduleMapping.addModuleToDevice("connector", "cloud"); // fixing all
instances of the Connector module to the Cloud

                                // rest of the modules will be placed by the Edge-ward placement policy

                        }



                        Controller controller = new Controller("master-controller", fogDevices, sensors,

                                        actuators);


                        controller.submitApplication(application, 0,

                                        (CLOUD)?(new ModulePlacementMapping(fogDevices,
application, moduleMapping))

                                                        :(new ModulePlacementEdgewards(fogDevices,
sensors, actuators, application, moduleMapping)));



        TimeKeeper.getInstance().setSimulationStartTime(Calendar.getInstance().getTimeInMillis());


                        CloudSim.startSimulation();


                        CloudSim.stopSimulation();


                        Log.printLine("VRGame finished!");

                } catch (Exception e) {

                        e.printStackTrace();

                        Log.printLine("Unwanted errors happen");

                }

        }
```

```
	/**
	 * Creates the fog devices in the physical topology of the simulation.
	 * @param userId
	 * @param appId
	 */
	private static void createFogDevices(int userId, String appId) {
		FogDevice cloud = createFogDevice("cloud", 44800, 40000, 100, 10000, 0, 0.01, 16*103, 16*83.25); // creates the fog device Cloud at the apex of the hierarchy with level=0
		cloud.setParentId(-1);
		FogDevice proxy = createFogDevice("proxy-server", 2800, 4000, 10000, 10000, 1, 0.0, 107.339, 83.4333); // creates the fog device Proxy Server (level=1)
		proxy.setParentId(cloud.getId()); // setting Cloud as parent of the Proxy Server
		proxy.setUplinkLatency(100); // latency of connection from Proxy Server to the Cloud is 100 ms

		fogDevices.add(cloud);
		fogDevices.add(proxy);

		for(int i=0;i<numOfDepts;i++){
			addGw(i+"", userId, appId, proxy.getId()); // adding a fog device for every Gateway in physical topology. The parent of each gateway is the Proxy Server
		}

	}

	private static FogDevice addGw(String id, int userId, String appId, int parentId){
		FogDevice dept = createFogDevice("d-"+id, 2800, 4000, 10000, 10000, 1, 0.0, 107.339, 83.4333);
		fogDevices.add(dept);
		dept.setParentId(parentId);
		dept.setUplinkLatency(4); // latency of connection between gateways and proxy server is 4 ms
		for(int i=0;i<numOfMobilesPerDept;i++){
			String mobileId = id+"-"+i;
			FogDevice mobile = addMobile(mobileId, userId, appId, dept.getId()); // adding mobiles to the physical topology. Smartphones have been modeled as fog devices as well.
```

```
                mobile.setUplinkLatency(2); // latency of connection between the smartphone and proxy server is 4 ms
                fogDevices.add(mobile);
            }
            return dept;
        }

        private static FogDevice addMobile(String id, int userId, String appId, int parentId){
            FogDevice mobile = createFogDevice("m-"+id, 1000, 1000, 10000, 270, 3, 0, 87.53, 82.44);
            mobile.setParentId(parentId);
            Sensor eegSensor = new Sensor("s-"+id, "EEG", userId, appId, new DeterministicDistribution(EEG_TRANSMISSION_TIME)); // inter-transmission time of EEG sensor follows a deterministic distribution
            sensors.add(eegSensor);
            Actuator display = new Actuator("a-"+id, userId, appId, "DISPLAY");
            actuators.add(display);
            eegSensor.setGatewayDeviceId(mobile.getId());
            eegSensor.setLatency(6.0);  // latency of connection between EEG sensors and the parent Smartphone is 6 ms
            display.setGatewayDeviceId(mobile.getId());
            display.setLatency(1.0);  // latency of connection between Display actuator and the parent Smartphone is 1 ms
            return mobile;
        }

        /**
         * Creates a vanilla fog device
         * @param nodeName name of the device to be used in simulation
         * @param mips MIPS
         * @param ram RAM
         * @param upBw uplink bandwidth
         * @param downBw downlink bandwidth
         * @param level hierarchy level of the device
         * @param ratePerMips cost rate per MIPS used
         * @param busyPower
```

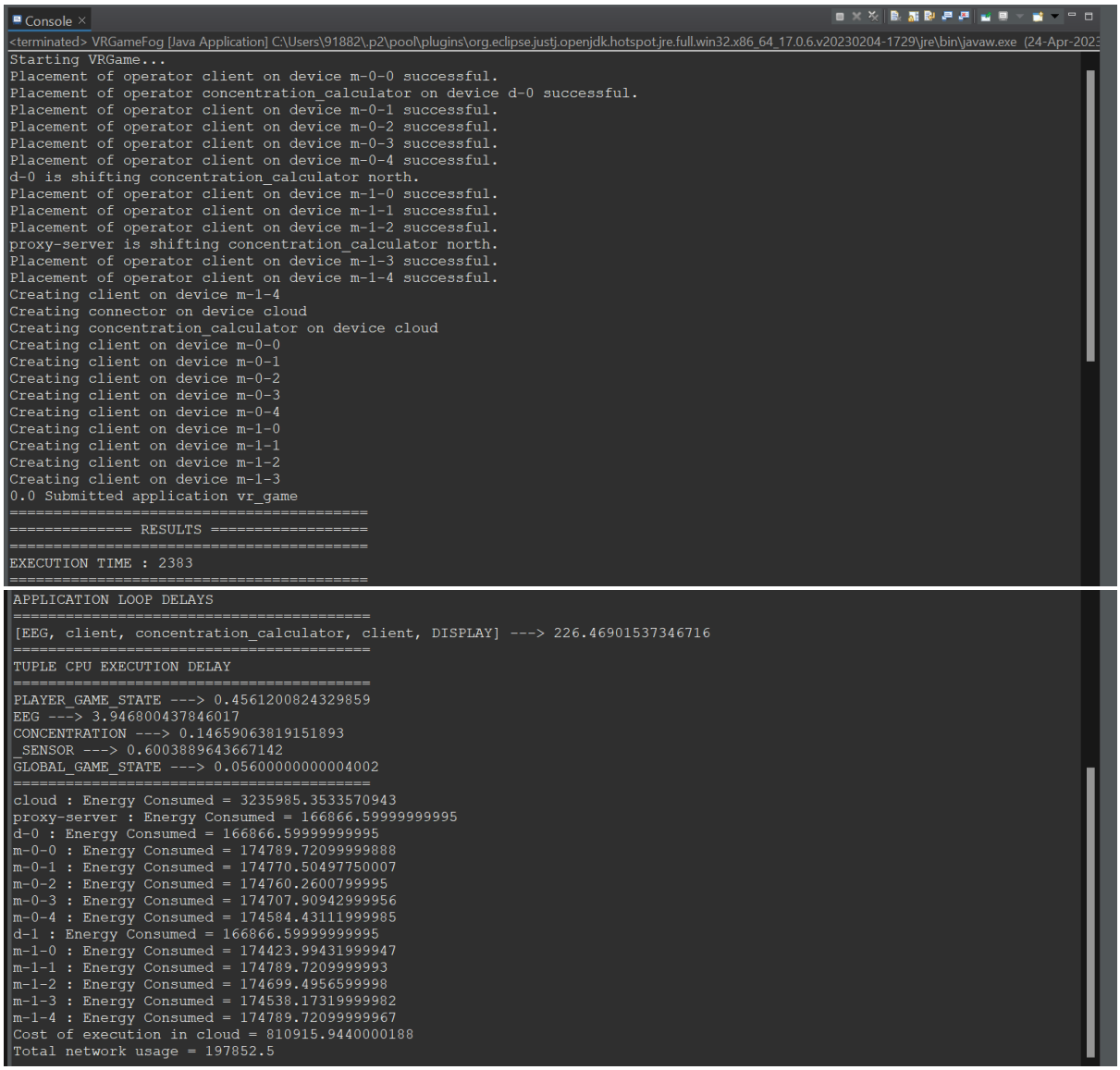
```
	 * @param idlePower
	 * @return
	 */
	private static FogDevice createFogDevice(String nodeName, long mips,
			int ram, long upBw, long downBw, int level, double ratePerMips, double busyPower,
double idlePower) {

		List<Pe> peList = new ArrayList<Pe>();

		// 3. Create PEs and add these into a list.
		peList.add(new Pe(0, new PeProvisionerOverbooking(mips))); // need to store Pe id and
MIPS Rating

		int hostId = FogUtils.generateEntityId();
		long storage = 1000000; // host storage
		int bw = 10000;

		PowerHost host = new PowerHost(
				hostId,
				new RamProvisionerSimple(ram),
				new BwProvisionerOverbooking(bw),
				storage,
				peList,
				new StreamOperatorScheduler(peList),
				new FogLinearPowerModel(busyPower, idlePower)
			);

		List<Host> hostList = new ArrayList<Host>();
		hostList.add(host);

		String arch = "x86"; // system architecture
		String os = "Linux"; // operating system
		String vmm = "Xen";
		double time_zone = 10.0; // time zone this resource located
```

```
		double cost = 3.0; // the cost of using processing in this resource
		double costPerMem = 0.05; // the cost of using memory in this resource
		double costPerStorage = 0.001; // the cost of using storage in this
										// resource
		double costPerBw = 0.0; // the cost of using bw in this resource
		LinkedList<Storage> storageList = new LinkedList<Storage>(); // we are not adding SAN
	// devices by now

		FogDeviceCharacteristics characteristics = new FogDeviceCharacteristics(
				arch, os, vmm, host, time_zone, cost, costPerMem,
				costPerStorage, costPerBw);

		FogDevice fogdevice = null;
		try {
			fogdevice = new FogDevice(nodeName, characteristics,
					new AppModuleAllocationPolicy(hostList), storageList, 10, upBw,
downBw, 0, ratePerMips);
		} catch (Exception e) {
			e.printStackTrace();
		}

		fogdevice.setLevel(level);
		return fogdevice;
	}

	/**
	 * Function to create the EEG Tractor Beam game application in the DDF model.
	 * @param appId unique identifier of the application
	 * @param userId identifier of the user of the application
	 * @return
	 */
	@SuppressWarnings({"serial" })
	private static Application createApplication(String appId, int userId){
```

```
		Application application = Application.createApplication(appId, userId); // creates an empty application model (empty directed graph)


		/*
		 * Adding modules (vertices) to the application model (directed graph)
		 */
		application.addAppModule("client", 10); // adding module Client to the application model
		application.addAppModule("concentration_calculator", 10); // adding module Concentration Calculator to the application model
		application.addAppModule("connector", 10); // adding module Connector to the application model


		/*
		 * Connecting the application modules (vertices) in the application model (directed graph) with edges
		 */
		if(EEG_TRANSMISSION_TIME==10)
			application.addAppEdge("EEG", "client", 2000, 500, "EEG", Tuple.UP, AppEdge.SENSOR); // adding edge from EEG (sensor) to Client module carrying tuples of type EEG
		else
			application.addAppEdge("EEG", "client", 3000, 500, "EEG", Tuple.UP, AppEdge.SENSOR);
		application.addAppEdge("client", "concentration_calculator", 3500, 500, "_SENSOR", Tuple.UP, AppEdge.MODULE); // adding edge from Client to Concentration Calculator module carrying tuples of type _SENSOR
		application.addAppEdge("concentration_calculator", "connector", 100, 1000, 1000, "PLAYER_GAME_STATE", Tuple.UP, AppEdge.MODULE); // adding periodic edge (period=1000ms) from Concentration Calculator to Connector module carrying tuples of type PLAYER_GAME_STATE
		application.addAppEdge("concentration_calculator", "client", 14, 500, "CONCENTRATION", Tuple.DOWN, AppEdge.MODULE);  // adding edge from Concentration Calculator to Client module carrying tuples of type CONCENTRATION
		application.addAppEdge("connector", "client", 100, 28, 1000, "GLOBAL_GAME_STATE", Tuple.DOWN, AppEdge.MODULE); // adding periodic edge (period=1000ms) from Connector to Client module carrying tuples of type GLOBAL_GAME_STATE
		application.addAppEdge("client", "DISPLAY", 1000, 500, "SELF_STATE_UPDATE", Tuple.DOWN, AppEdge.ACTUATOR);  // adding edge from Client module to Display (actuator) carrying tuples of type SELF_STATE_UPDATE
```

```
		application.addAppEdge("client", "DISPLAY", 1000, 500, "GLOBAL_STATE_UPDATE", Tuple.DOWN, AppEdge.ACTUATOR);  // adding edge from Client module to Display (actuator) carrying tuples of type GLOBAL_STATE_UPDATE


		/*
		 * Defining the input-output relationships (represented by selectivity) of the application modules.
		 */
		application.addTupleMapping("client", "EEG", "_SENSOR", new FractionalSelectivity(0.9)); // 0.9 tuples of type _SENSOR are emitted by Client module per incoming tuple of type EEG
		application.addTupleMapping("client", "CONCENTRATION", "SELF_STATE_UPDATE", new FractionalSelectivity(1.0)); // 1.0 tuples of type SELF_STATE_UPDATE are emitted by Client module per incoming tuple of type CONCENTRATION
		application.addTupleMapping("concentration_calculator", "_SENSOR", "CONCENTRATION", new FractionalSelectivity(1.0)); // 1.0 tuples of type CONCENTRATION are emitted by Concentration Calculator module per incoming tuple of type _SENSOR
		application.addTupleMapping("client", "GLOBAL_GAME_STATE", "GLOBAL_STATE_UPDATE", new FractionalSelectivity(1.0)); // 1.0 tuples of type GLOBAL_STATE_UPDATE are emitted by Client module per incoming tuple of type GLOBAL_GAME_STATE


		/*
		 * Defining application loops to monitor the latency of.
		 * Here, we add only one loop for monitoring : EEG(sensor) -> Client -> Concentration Calculator -> Client -> DISPLAY (actuator)
		 */
		final AppLoop loop1 = new AppLoop(new ArrayList<String>(){{add("EEG");add("client");add("concentration_calculator");add("client");add("DISPLAY");}});
		List<AppLoop> loops = new ArrayList<AppLoop>(){{add(loop1);}};
		application.setLoops(loops);


		return application;
	}
}
```

OUTPUT



```
Console ×
<terminated> VRGameFog [Java Application] C:\Users\91882\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1729\jre\bin\javaw.exe  (24-Apr-2023
Starting VRGame...
Placement of operator client on device m-0-0 successful.
Placement of operator concentration_calculator on device d-0 successful.
Placement of operator client on device m-0-1 successful.
Placement of operator client on device m-0-2 successful.
Placement of operator client on device m-0-3 successful.
Placement of operator client on device m-0-4 successful.
d-0 is shifting concentration_calculator north.
Placement of operator client on device m-1-0 successful.
Placement of operator client on device m-1-1 successful.
Placement of operator client on device m-1-2 successful.
proxy-server is shifting concentration_calculator north.
Placement of operator client on device m-1-3 successful.
Placement of operator client on device m-1-4 successful.
Creating client on device m-1-4
Creating connector on device cloud
Creating concentration_calculator on device cloud
Creating client on device m-0-0
Creating client on device m-0-1
Creating client on device m-0-2
Creating client on device m-0-3
Creating client on device m-0-4
Creating client on device m-1-0
Creating client on device m-1-1
Creating client on device m-1-2
Creating client on device m-1-3
0.0 Submitted application vr_game
==========================================
============== RESULTS ==================
==========================================
EXECUTION TIME : 2383
==========================================
APPLICATION LOOP DELAYS
==========================================
[EEG, client, concentration_calculator, client, DISPLAY] ---> 226.46901537346716
==========================================
TUPLE CPU EXECUTION DELAY
==========================================
PLAYER_GAME_STATE ---> 0.4561200824329859
EEG ---> 3.946800437846017
CONCENTRATION ---> 0.14659063819151893
_SENSOR ---> 0.6003889643667142
GLOBAL_GAME_STATE ---> 0.05600000000004002
==========================================
cloud : Energy Consumed = 3235985.3533570943
proxy-server : Energy Consumed = 166866.59999999995
d-0 : Energy Consumed = 166866.59999999995
m-0-0 : Energy Consumed = 174789.72099999888
m-0-1 : Energy Consumed = 174770.50497750007
m-0-2 : Energy Consumed = 174760.2600799995
m-0-3 : Energy Consumed = 174707.90942999956
m-0-4 : Energy Consumed = 174584.43111999985
d-1 : Energy Consumed = 166866.59999999995
m-1-0 : Energy Consumed = 174423.99431999947
m-1-1 : Energy Consumed = 174789.7209999993
m-1-2 : Energy Consumed = 174699.4956599998
m-1-3 : Energy Consumed = 174538.17319999982
m-1-4 : Energy Consumed = 174789.72099999967
Cost of execution in cloud = 810915.9440000188
Total network usage = 197852.5
```

PRACTICAL-6

Create a simulation setup for an Intelligent Surveillance using Data Centre Networks (DCNs). Analyse the performance of the network in terms of CPU execution delay and energy consumed.

```
package org.fog.test.perfeval;


import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.power.PowerHost;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
import org.cloudbus.cloudsim.sdn.overbooking.BwProvisionerOverbooking;
import org.cloudbus.cloudsim.sdn.overbooking.PeProvisionerOverbooking;
import org.fog.application.AppEdge;
import org.fog.application.AppLoop;
import org.fog.application.Application;
import org.fog.application.selectivity.FractionalSelectivity;
import org.fog.entities.Actuator;
import org.fog.entities.FogBroker;
import org.fog.entities.FogDevice;
import org.fog.entities.FogDeviceCharacteristics;
```

```
import org.fog.entities.Sensor;
import org.fog.entities.Tuple;
import org.fog.placement.Controller;
import org.fog.placement.ModuleMapping;
import org.fog.placement.ModulePlacementEdgewards;
import org.fog.placement.ModulePlacementMapping;
import org.fog.policy.AppModuleAllocationPolicy;
import org.fog.scheduler.StreamOperatorScheduler;
import org.fog.utils.FogLinearPowerModel;
import org.fog.utils.FogUtils;
import org.fog.utils.TimeKeeper;
import org.fog.utils.distribution.DeterministicDistribution;

/**
 * Simulation setup for case study 2 - Intelligent Surveillance
 * @author Harshit Gupta
 *
 */
public class DCNSFog {
	static List<FogDevice> fogDevices = new ArrayList<FogDevice>();
	static List<Sensor> sensors = new ArrayList<Sensor>();
	static List<Actuator> actuators = new ArrayList<Actuator>();
	static int numOfAreas = 1;
	static int numOfCamerasPerArea = 4;

	private static boolean CLOUD = false;

	public static void main(String[] args) {

		Log.printLine("Starting DCNS...");

		try {
			Log.disable();
```

```
int num_user = 1; // number of cloud users
Calendar calendar = Calendar.getInstance();
boolean trace_flag = false; // mean trace events

CloudSim.init(num_user, calendar, trace_flag);

String appId = "dcns"; // identifier of the application

FogBroker broker = new FogBroker("broker");

Application application = createApplication(appId, broker.getId());
application.setUserId(broker.getId());

createFogDevices(broker.getId(), appId);

Controller controller = null;

ModuleMapping moduleMapping = ModuleMapping.createModuleMapping(); // initializing a module mapping
for(FogDevice device : fogDevices){
    if(device.getName().startsWith("m")){ // names of all Smart Cameras start with 'm'
        moduleMapping.addModuleToDevice("motion_detector", device.getName());  // fixing 1 instance of the Motion Detector module to each Smart Camera
    }
}
moduleMapping.addModuleToDevice("user_interface", "cloud"); // fixing instances of User Interface module in the Cloud
if(CLOUD){
    // if the mode of deployment is cloud-based
    moduleMapping.addModuleToDevice("object_detector", "cloud"); // placing all instances of Object Detector module in the Cloud
    moduleMapping.addModuleToDevice("object_tracker", "cloud"); // placing all instances of Object Tracker module in the Cloud
}
```

```
                controller = new Controller("master-controller", fogDevices, sensors,
                                actuators);

                controller.submitApplication(application,
                                (CLOUD)?(new ModulePlacementMapping(fogDevices,
application, moduleMapping))
                                                :(new ModulePlacementEdgewards(fogDevices,
sensors, actuators, application, moduleMapping)));


        TimeKeeper.getInstance().setSimulationStartTime(Calendar.getInstance().getTimeInMillis());

                CloudSim.startSimulation();

                CloudSim.stopSimulation();

                Log.printLine("VRGame finished!");
            } catch (Exception e) {
                e.printStackTrace();
                Log.printLine("Unwanted errors happen");
            }
        }

        /**
         * Creates the fog devices in the physical topology of the simulation.
         * @param userId
         * @param appId
         */
        private static void createFogDevices(int userId, String appId) {
            FogDevice cloud = createFogDevice("cloud", 44800, 40000, 100, 10000, 0, 0.01, 16*103,
16*83.25);
            cloud.setParentId(-1);
            fogDevices.add(cloud);
```

```
		FogDevice proxy = createFogDevice("proxy-server", 2800, 4000, 10000, 10000, 1, 0.0, 107.339, 83.4333);
		proxy.setParentId(cloud.getId());
		proxy.setUplinkLatency(100); // latency of connection between proxy server and cloud is 100 ms
		fogDevices.add(proxy);
		for(int i=0;i<numOfAreas;i++){
			addArea(i+"", userId, appId, proxy.getId());
		}
	}

	private static FogDevice addArea(String id, int userId, String appId, int parentId){
		FogDevice router = createFogDevice("d-"+id, 2800, 4000, 10000, 10000, 1, 0.0, 107.339, 83.4333);
		fogDevices.add(router);
		router.setUplinkLatency(2); // latency of connection between router and proxy server is 2 ms
		for(int i=0;i<numOfCamerasPerArea;i++){
			String mobileId = id+"-"+i;
			FogDevice camera = addCamera(mobileId, userId, appId, router.getId()); // adding a smart camera to the physical topology. Smart cameras have been modeled as fog devices as well.
			camera.setUplinkLatency(2); // latency of connection between camera and router is 2 ms
			fogDevices.add(camera);
		}
		router.setParentId(parentId);
		return router;
	}

	private static FogDevice addCamera(String id, int userId, String appId, int parentId){
		FogDevice camera = createFogDevice("m-"+id, 500, 1000, 10000, 10000, 3, 0, 87.53, 82.44);
		camera.setParentId(parentId);
		Sensor sensor = new Sensor("s-"+id, "CAMERA", userId, appId, new DeterministicDistribution(5)); // inter-transmission time of camera (sensor) follows a deterministic distribution
		sensors.add(sensor);
		Actuator ptz = new Actuator("ptz-"+id, userId, appId, "PTZ_CONTROL");
```

```
		actuators.add(ptz);
		sensor.setGatewayDeviceId(camera.getId());
		sensor.setLatency(1.0);  // latency of connection between camera (sensor) and the parent Smart Camera is 1 ms
		ptz.setGatewayDeviceId(camera.getId());
		ptz.setLatency(1.0);  // latency of connection between PTZ Control and the parent Smart Camera is 1 ms
		return camera;
	}

	/**
	 * Creates a vanilla fog device
	 * @param nodeName name of the device to be used in simulation
	 * @param mips MIPS
	 * @param ram RAM
	 * @param upBw uplink bandwidth
	 * @param downBw downlink bandwidth
	 * @param level hierarchy level of the device
	 * @param ratePerMips cost rate per MIPS used
	 * @param busyPower
	 * @param idlePower
	 * @return
	 */
	private static FogDevice createFogDevice(String nodeName, long mips,
			int ram, long upBw, long downBw, int level, double ratePerMips, double busyPower, double idlePower) {

		List<Pe> peList = new ArrayList<Pe>();

		// 3. Create PEs and add these into a list.
		peList.add(new Pe(0, new PeProvisionerOverbooking(mips))); // need to store Pe id and MIPS Rating

		int hostId = FogUtils.generateEntityId();
```

```
        long storage = 1000000; // host storage
        int bw = 10000;

        PowerHost host = new PowerHost(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerOverbooking(bw),
                        storage,
                        peList,
                        new StreamOperatorScheduler(peList),
                        new FogLinearPowerModel(busyPower, idlePower)
                );

        List<Host> hostList = new ArrayList<Host>();
        hostList.add(host);

        String arch = "x86"; // system architecture
        String os = "Linux"; // operating system
        String vmm = "Xen";
        double time_zone = 10.0; // time zone this resource located
        double cost = 3.0; // the cost of using processing in this resource
        double costPerMem = 0.05; // the cost of using memory in this resource
        double costPerStorage = 0.001; // the cost of using storage in this
                                                                        // resource
        double costPerBw = 0.0; // the cost of using bw in this resource
        LinkedList<Storage> storageList = new LinkedList<Storage>(); // we are not adding SAN
// devices by now

        FogDeviceCharacteristics characteristics = new FogDeviceCharacteristics(
                        arch, os, vmm, host, time_zone, cost, costPerMem,
                        costPerStorage, costPerBw);
```

```
		FogDevice fogdevice = null;
		try {
			fogdevice = new FogDevice(nodeName, characteristics,
					new AppModuleAllocationPolicy(hostList), storageList, 10, upBw,
downBw, 0, ratePerMips);
		} catch (Exception e) {
			e.printStackTrace();
		}

		fogdevice.setLevel(level);
		return fogdevice;
	}
	/**
	 * Function to create the Intelligent Surveillance application in the DDF model.
	 * @param appId unique identifier of the application
	 * @param userId identifier of the user of the application
	 * @return
	 */
	@SuppressWarnings({"serial" })
	private static Application createApplication(String appId, int userId){

		Application application = Application.createApplication(appId, userId);
		/*
		 * Adding modules (vertices) to the application model (directed graph)
		 */
		application.addAppModule("object_detector", 10);
		application.addAppModule("motion_detector", 10);
		application.addAppModule("object_tracker", 10);
		application.addAppModule("user_interface", 10);

		/*
		 * Connecting the application modules (vertices) in the application model (directed graph)
with edges
		 */
```

application.addAppEdge("CAMERA", "motion_detector", 1000, 20000, "CAMERA", Tuple.UP, AppEdge.SENSOR); // adding edge from CAMERA (sensor) to Motion Detector module carrying tuples of type CAMERA

application.addAppEdge("motion_detector", "object_detector", 2000, 2000, "MOTION_VIDEO_STREAM", Tuple.UP, AppEdge.MODULE); // adding edge from Motion Detector to Object Detector module carrying tuples of type MOTION_VIDEO_STREAM

application.addAppEdge("object_detector", "user_interface", 500, 2000, "DETECTED_OBJECT", Tuple.UP, AppEdge.MODULE); // adding edge from Object Detector to User Interface module carrying tuples of type DETECTED_OBJECT

application.addAppEdge("object_detector", "object_tracker", 1000, 100, "OBJECT_LOCATION", Tuple.UP, AppEdge.MODULE); // adding edge from Object Detector to Object Tracker module carrying tuples of type OBJECT_LOCATION

application.addAppEdge("object_tracker", "PTZ_CONTROL", 100, 28, 100, "PTZ_PARAMS", Tuple.DOWN, AppEdge.ACTUATOR); // adding edge from Object Tracker to PTZ CONTROL (actuator) carrying tuples of type PTZ_PARAMS

/*

* Defining the input-output relationships (represented by selectivity) of the application modules.

*/

application.addTupleMapping("motion_detector", "CAMERA", "MOTION_VIDEO_STREAM", new FractionalSelectivity(1.0)); // 1.0 tuples of type MOTION_VIDEO_STREAM are emitted by Motion Detector module per incoming tuple of type CAMERA

application.addTupleMapping("object_detector", "MOTION_VIDEO_STREAM", "OBJECT_LOCATION", new FractionalSelectivity(1.0)); // 1.0 tuples of type OBJECT_LOCATION are emitted by Object Detector module per incoming tuple of type MOTION_VIDEO_STREAM

application.addTupleMapping("object_detector", "MOTION_VIDEO_STREAM", "DETECTED_OBJECT", new FractionalSelectivity(0.05)); // 0.05 tuples of type MOTION_VIDEO_STREAM are emitted by Object Detector module per incoming tuple of type MOTION_VIDEO_STREAM

/*

* Defining application loops (maybe incomplete loops) to monitor the latency of.

* Here, we add two loops for monitoring : Motion Detector -> Object Detector -> Object Tracker and Object Tracker -> PTZ Control

*/

final AppLoop loop1 = new AppLoop(new ArrayList<String>(){{add("motion_detector");add("object_detector");add("object_tracker");}});

final AppLoop loop2 = new AppLoop(new ArrayList<String>(){{add("object_tracker");add("PTZ_CONTROL");}});

List<AppLoop> loops = new ArrayList<AppLoop>(){{add(loop1);add(loop2);}};

```
        application.setLoops(loops);

        return application;

    }

}
```

OUTPUT