

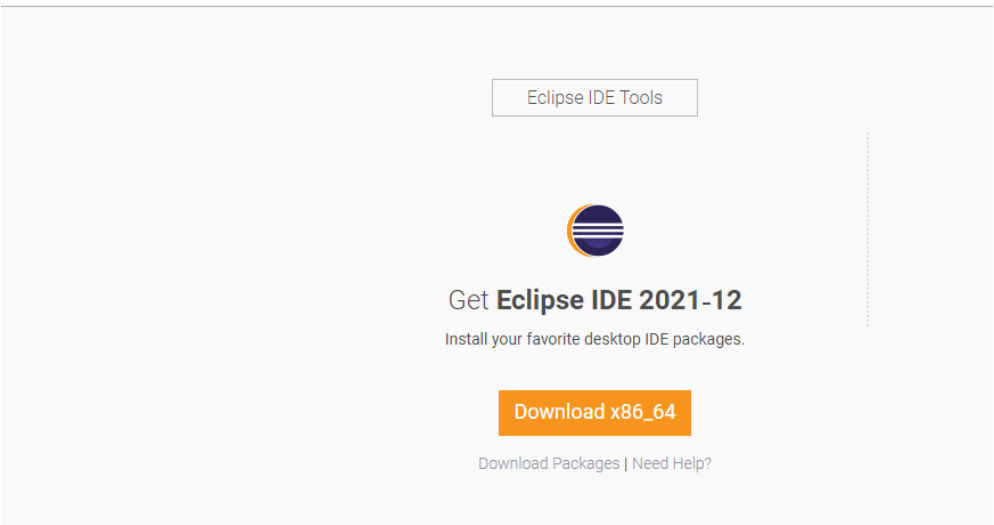
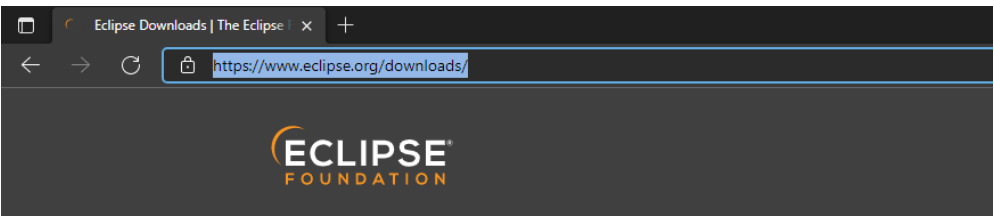
Reality Conforming Approach vs State of Art method for computing worst-case delays in AFDX.

Paper : [A Reality-Conforming Approach for QoS Performance Analysis of AFDX in Cyber-Physical Avionics Systems](#)

1. Download the Eclipse IDE from the Internet.

- Download latest Eclipse IDE from the given link.

Link : [Eclipse Downloads | The Eclipse Foundation](#)

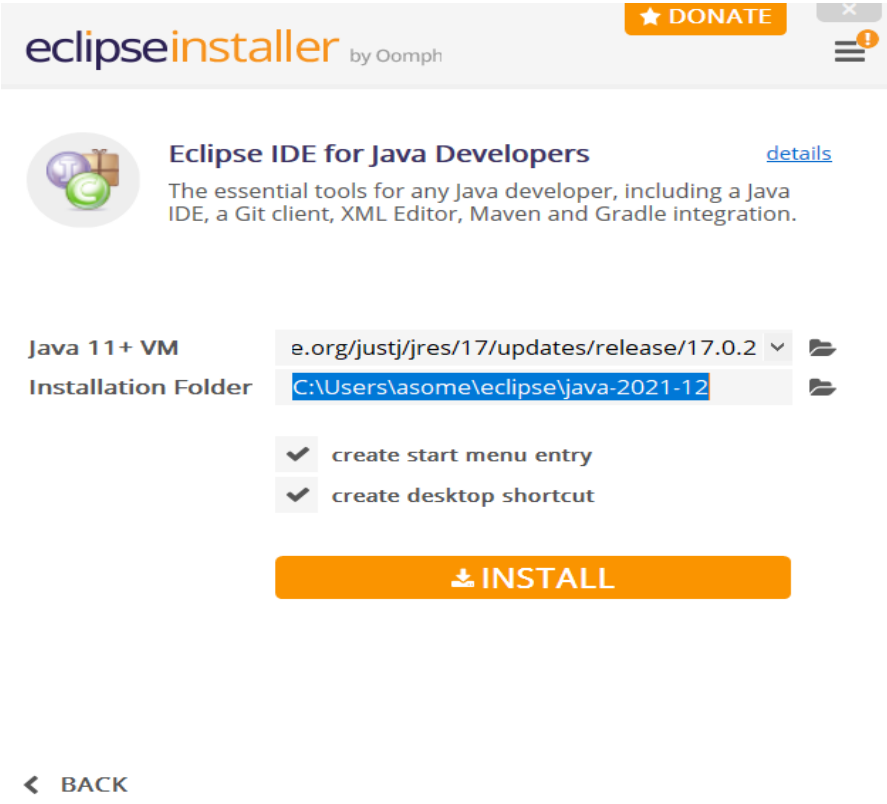


2. Install the Eclipse into the desired directory.

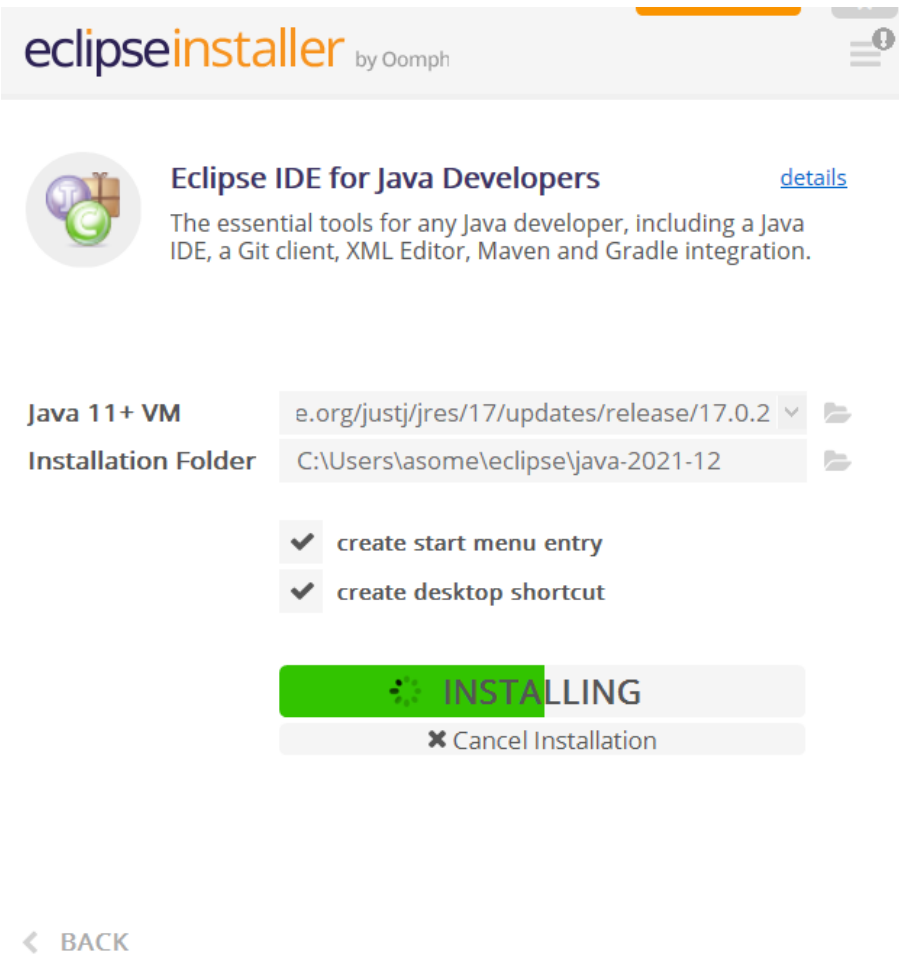
- Launch the Eclipse Installer.
- Select **Eclipse IDE for Java Developers**.



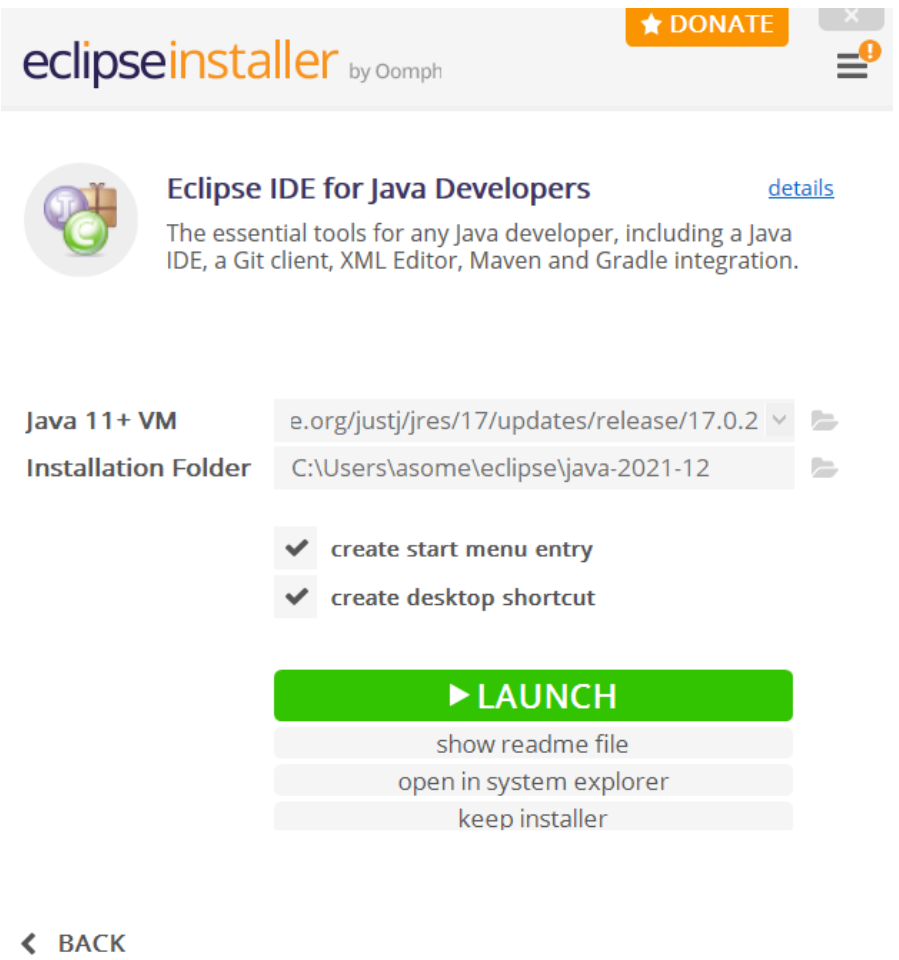
- Select the directory where you want to install Eclipse.



- Click the **INSTALL** option

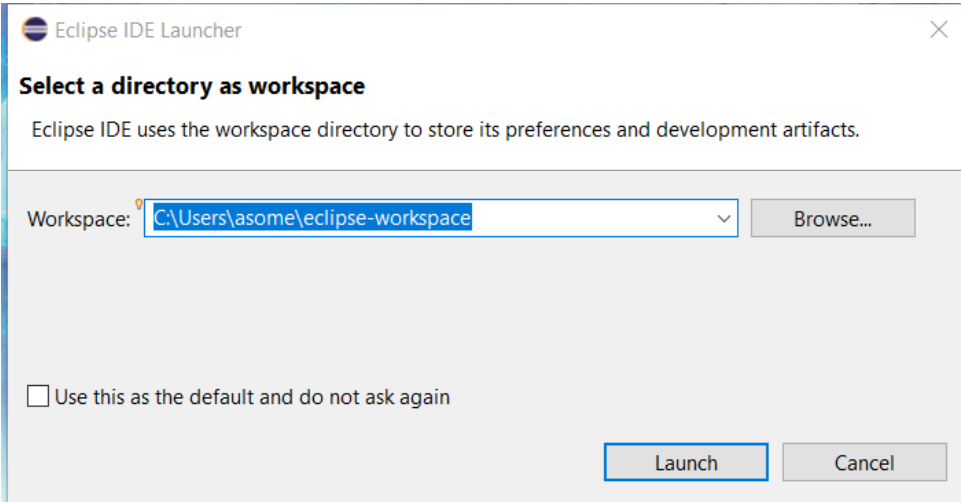


- After Installation is complete, Click on **LAUNCH** option.

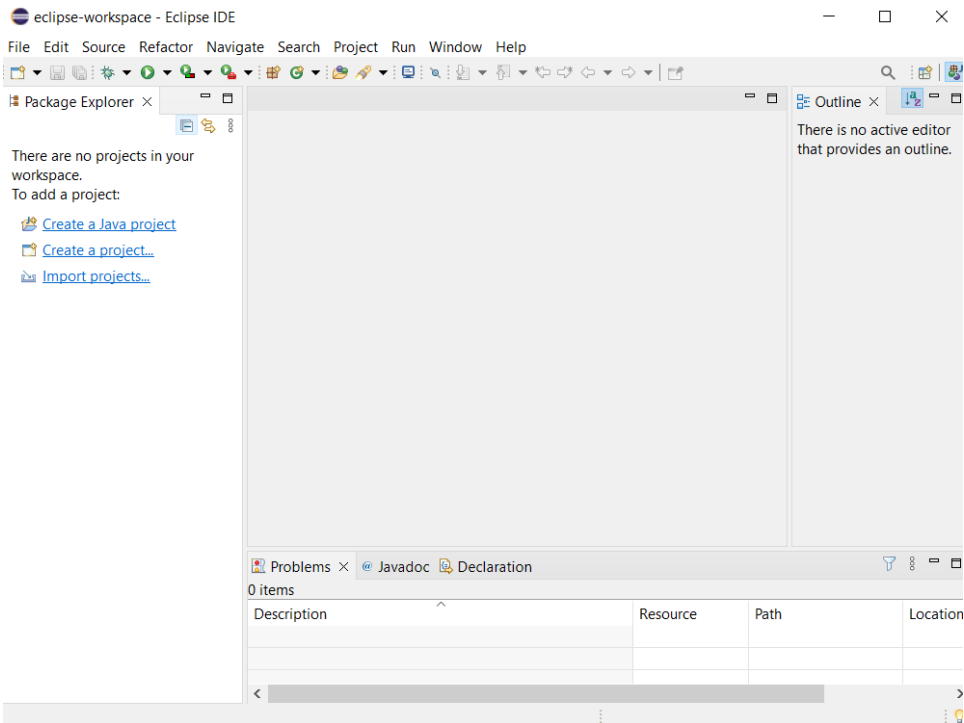


3. After Installation Launch the eclipse:

- Select a directory as a workspace.



- Click on **Launch** Button.
- It will launch Eclipse Workspace.

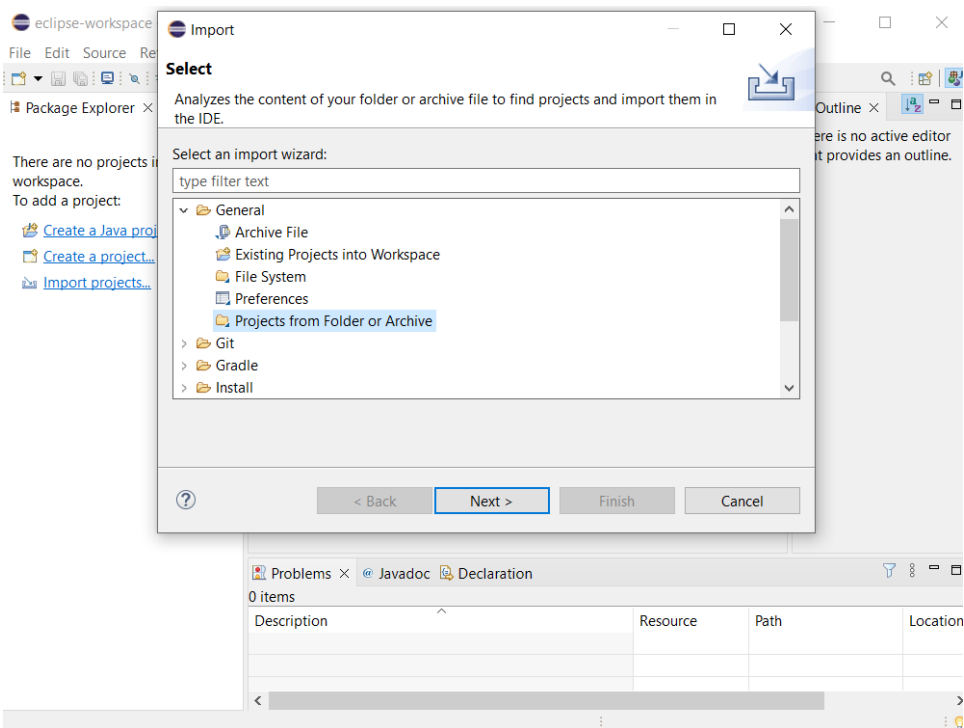


4. Download the AFDX Code

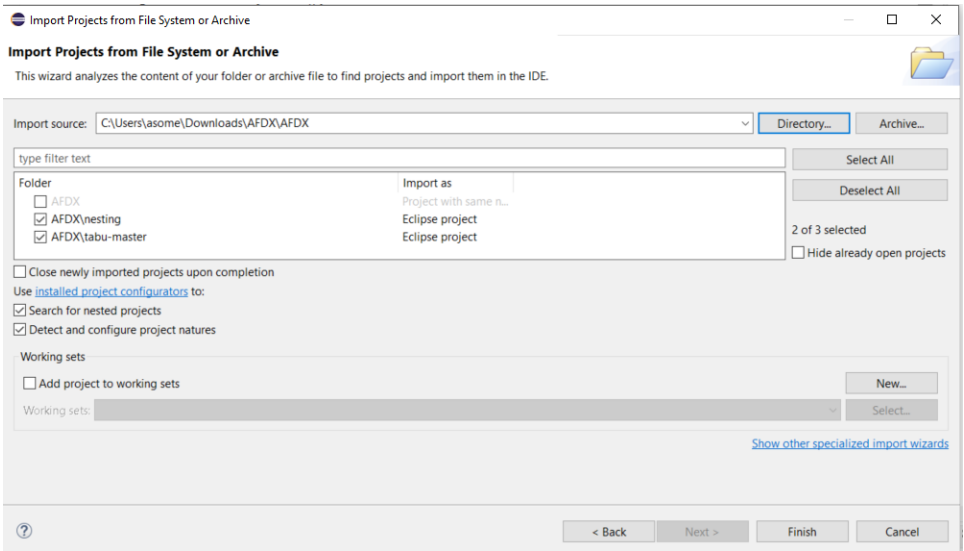
- Download AFDX from the portal link given below.
- Link:
- Move the AFDX project to the working directory of your project.

5. Import the AFDX folder into the Eclipse

- Inside Eclipse Workspace. Go to **File**, then select **Import**.
- Then go to **General** folder under **Select an import Wizard** and select **Projects from Folder or Archive** option.

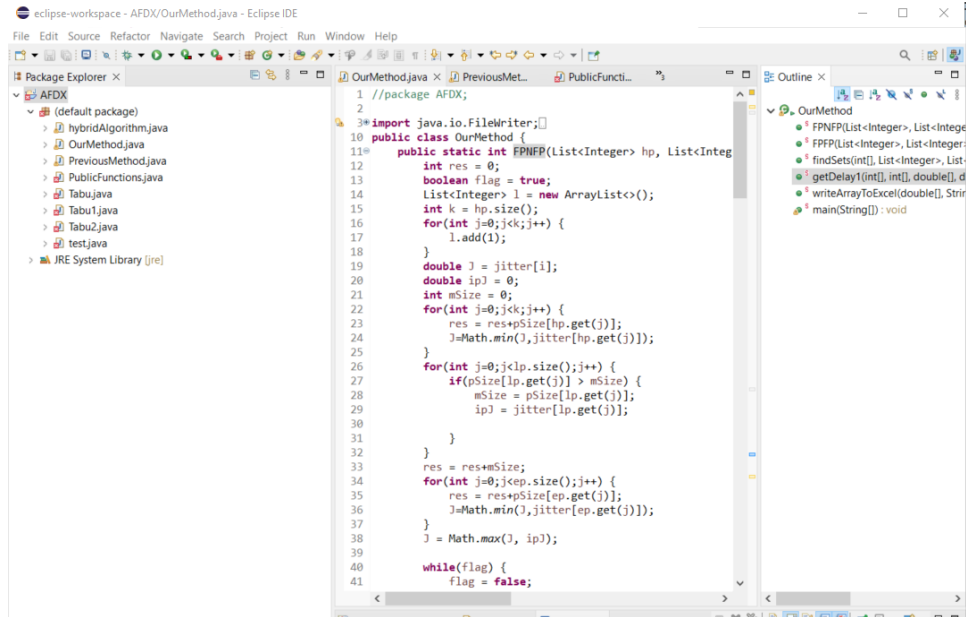


- Then choose the directory where AFDX folder is located and import it.



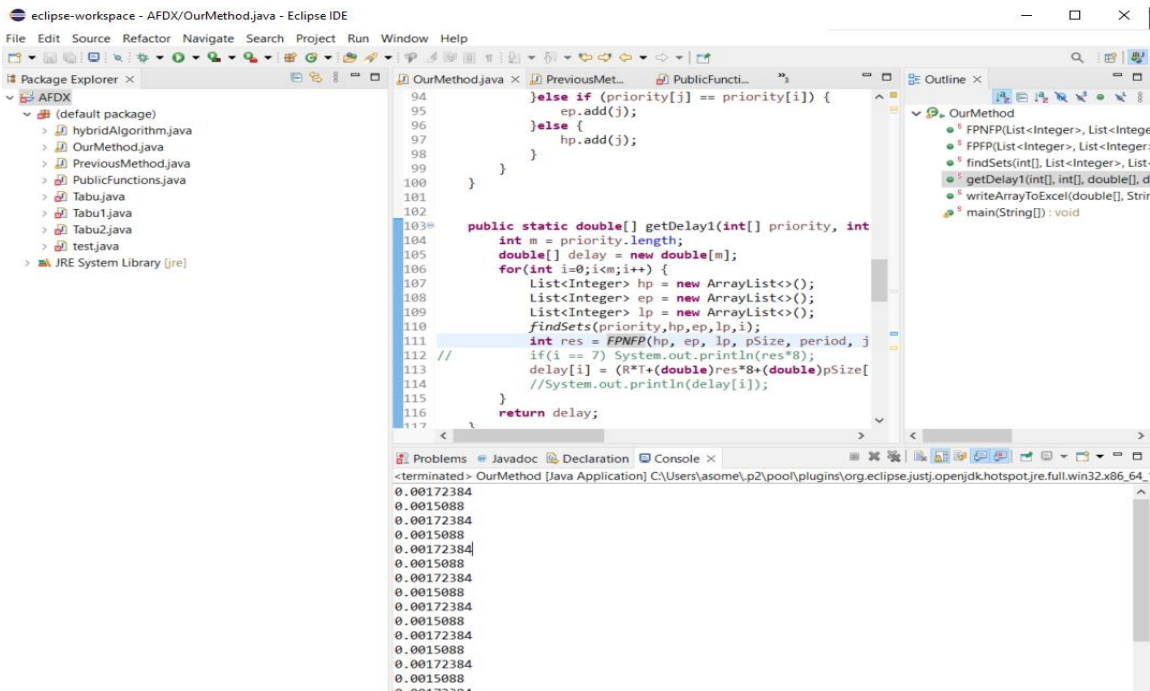
8. Running the Program.

- Inside AFDX, go to **OurMethod.java** and **State_of_Art.java**



```
OurMethod.java x State_of_Art.java x
34 double max = 0;
35 for(int j=0;j<rate.length;j++){
36     if(i == j) continue;
37     if(priority[i] >= priority[j]) {
38         burst = burst+(double)pSize[j];
39         loR = loR-rate[j];
40     }
41     burst = burst+max+(double)pSize[i];
42     delay[i] = burst/loR;
43     System.out.println("burst: "+burst);
44     // System.out.println("loR: "+loR);
45 }
46 return delay;
47 }
48 public static void main(String[] args) {
49     // TODO Auto-generated method stub
50     int numVL = 100;
51     int numOfPriority = 2;
52     int[] priority = new int[numVL];
53     double[] BAG = {2, 4, 8, 16, 32, 64, 128};
54     int[] size = {32, 64, 128, 256, 512, 1024, 1500};
55     for(int i=0;i<priority.length;i++) {
56         Random rand = new Random();
57         priority[i] = i%(numOfPriority);
58     }
59     double[] period = new double[numVL];
60     double[] jitter = new double[numVL];
61     for(int i=0;i<priority.length;i++) {
62         if(i<numVL/2) {
63             period[i] = BAG[2]/1000;
64         }else {
65             period[i] = BAG[3]/1000;
66         }
67     }
68 }
```

- Then run that file and record the result.



```
OurMethod.java
34 double max = 0;
35 for(int j=0;j<rate.length;j++){
36     if(i == j) continue;
37     if(priority[j] >= priority[i]) {
38         burst = burst+(double)pSize[j];
39         loR = loR-rate[j];
40     }
41     burst = burst+max+(double)pSize[i];
42     delay[i] = burst/loR;
43     System.out.println("burst: "+burst);
44     // System.out.println("loR: "+loR);
45 }
46 return delay;
47 }
48 public static void main(String[] args) {
49     // TODO Auto-generated method stub
50     int numVL = 100;
51     int numOfPriority = 2;
52     int[] priority = new int[numVL];
53     double[] BAG = {2, 4, 8, 16, 32, 64, 128};
54     int[] size = {32, 64, 128, 256, 512, 1024, 1500};
55     for(int i=0;i<priority.length;i++) {
56         Random rand = new Random();
57         priority[i] = i%(numOfPriority);
58     }
59     double[] period = new double[numVL];
60     double[] jitter = new double[numVL];
61     for(int i=0;i<priority.length;i++) {
62         if(i<numVL/2) {
63             period[i] = BAG[2]/1000;
64         }else {
65             period[i] = BAG[3]/1000;
66         }
67     }
68 }
```

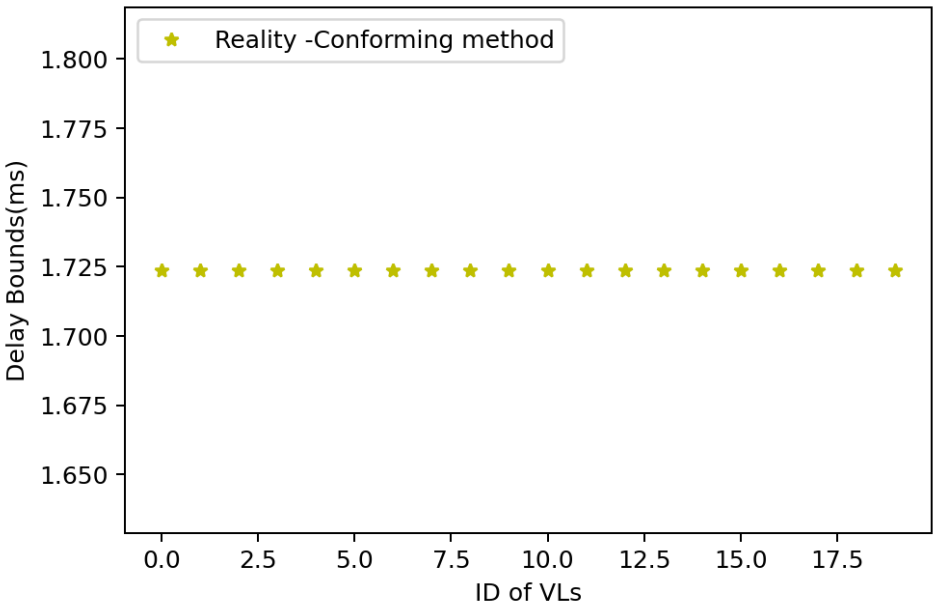
Console Output:

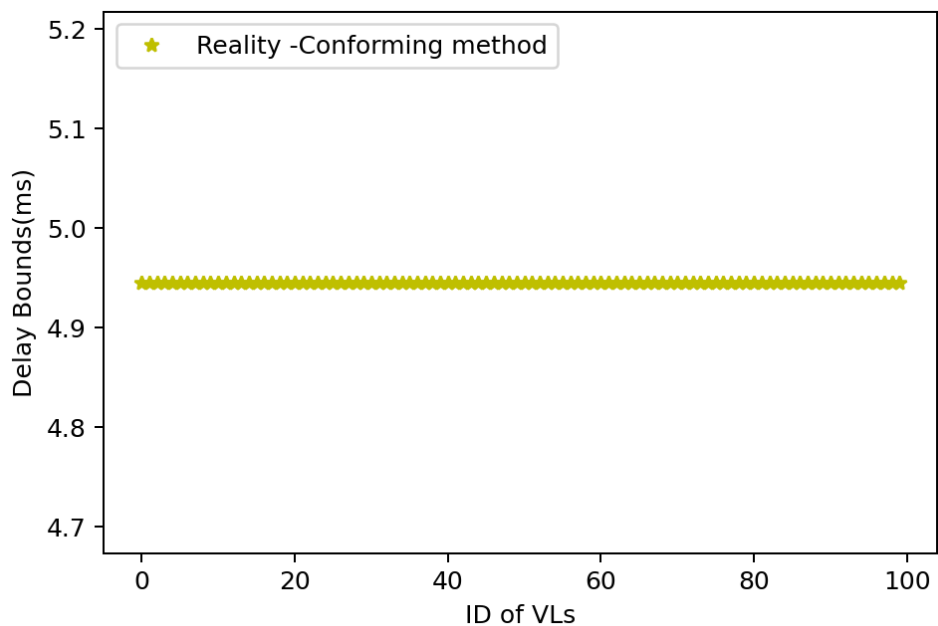
```
<terminated> State_of_Art [Java Application] C:\Users\asome\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\j
0.007832008361574763
0.0036288740299672997
0.007784668660475366
0.0036071315949060038
0.007845924421740648
0.003638819556599618
0.007839954345138074
0.003634550520604194
0.007816164617378387
0.003617574076777113
0.0077556141011165475
0.003639532037789417
0.007843933386201428
0.00363739543075192
```

Inside **OurMethod.java** and **State_of_Art.java**, change the required parameters to get the values of delays for several scenarios such as: When all virtual links have the same priority, different priorities with or without frame preemption.

A. Get Delays for Reality Conforming method when all VLS have the same priority.

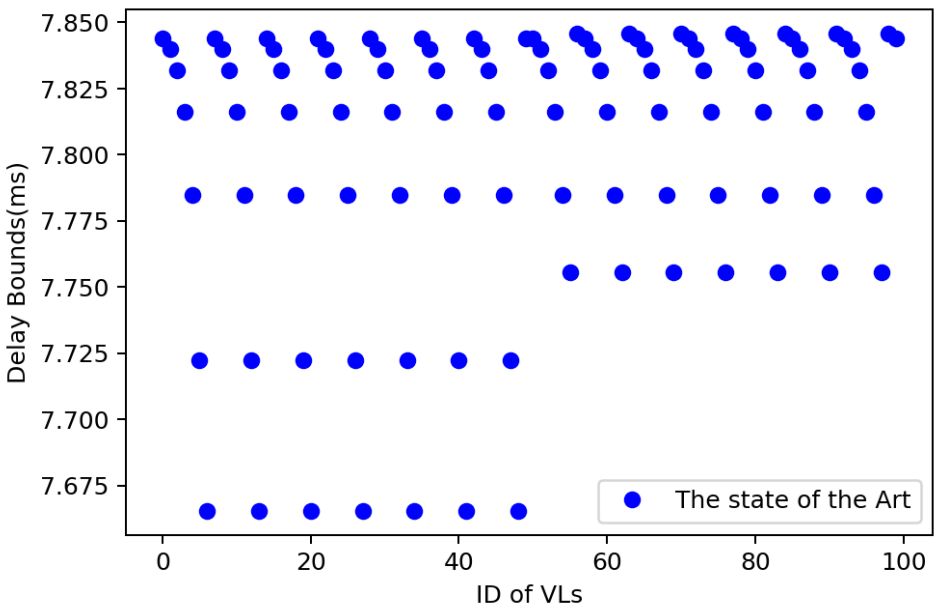
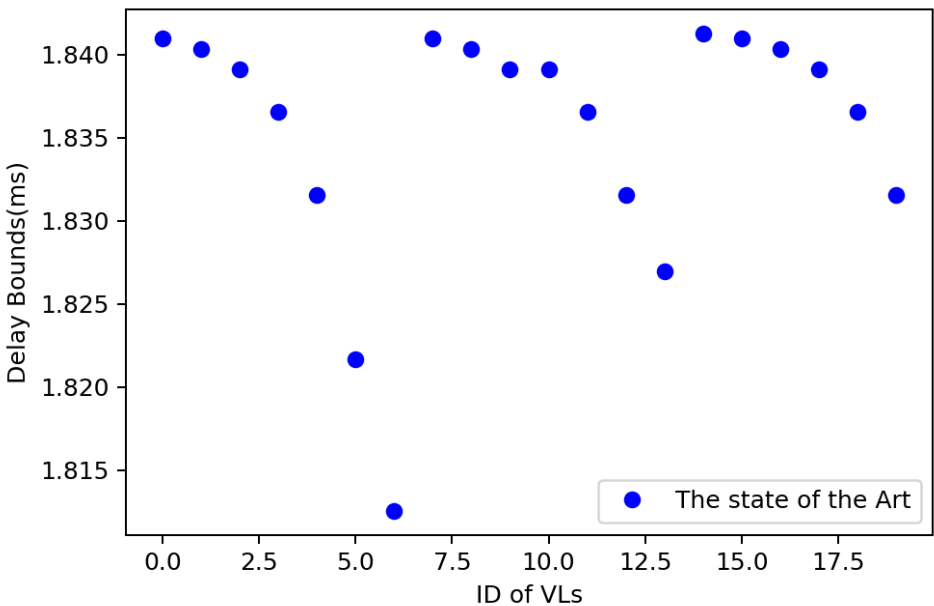
- Go to **OurMethod.java** file
- Inside **getDelay1** method, set **res = FPNFP** or **FPFP**
- In **main** method, set **numVL** (no of virtual links) to 20 or 100 depending upon no of VL's used.
- In **main** method, set **numP** (priority) to 1, indicate all VLs have same priority.
- Run the code and record the delay values to plot the graph.



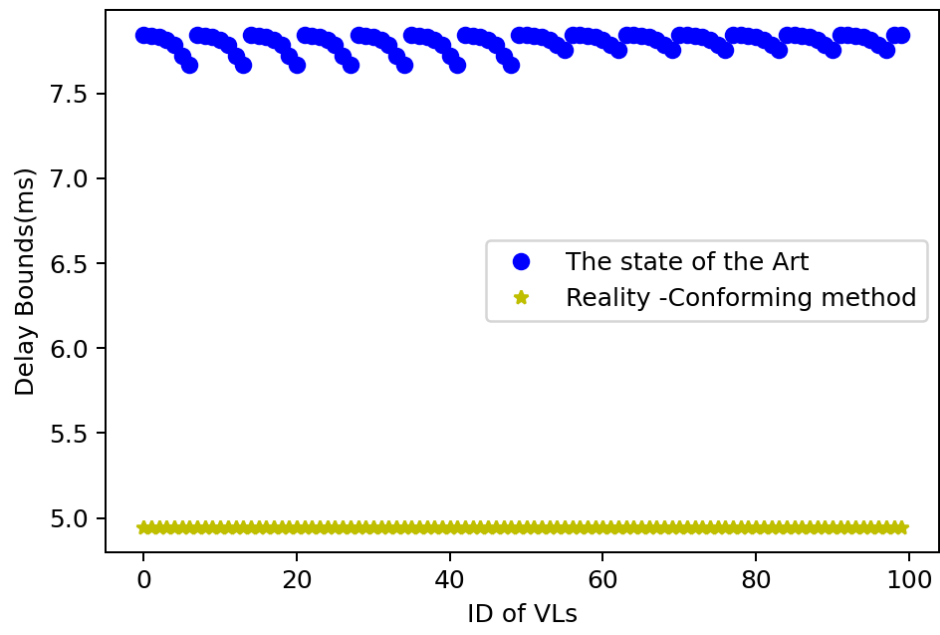
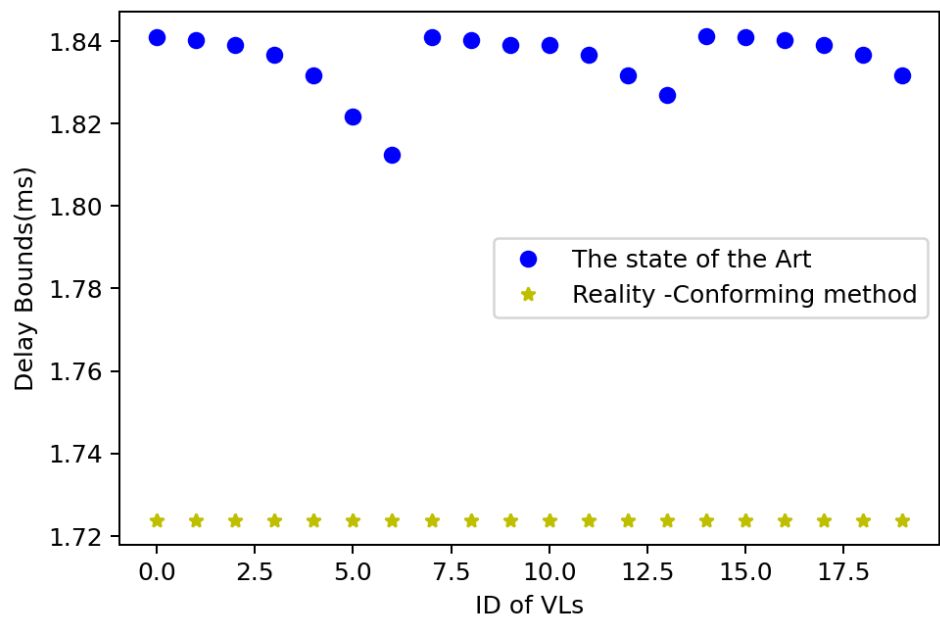


B. Get Delays for State of Art method when all VLS have the same priority.

- Go to *State_of_Art.java* file
- In *main* method, set *numVL* (no of virtual links) to 20 or 100 depending upon no of VL’s used.
- In *main* method, set *numOfPriority* (priority) to 1, indicate all VLs have same priority.
- In *main* method, at end of the method, set *delay* to *State_Of_ArtFP* or *State_Of_ArtNFP*
- Run the code and record the delay values to plot the graph.

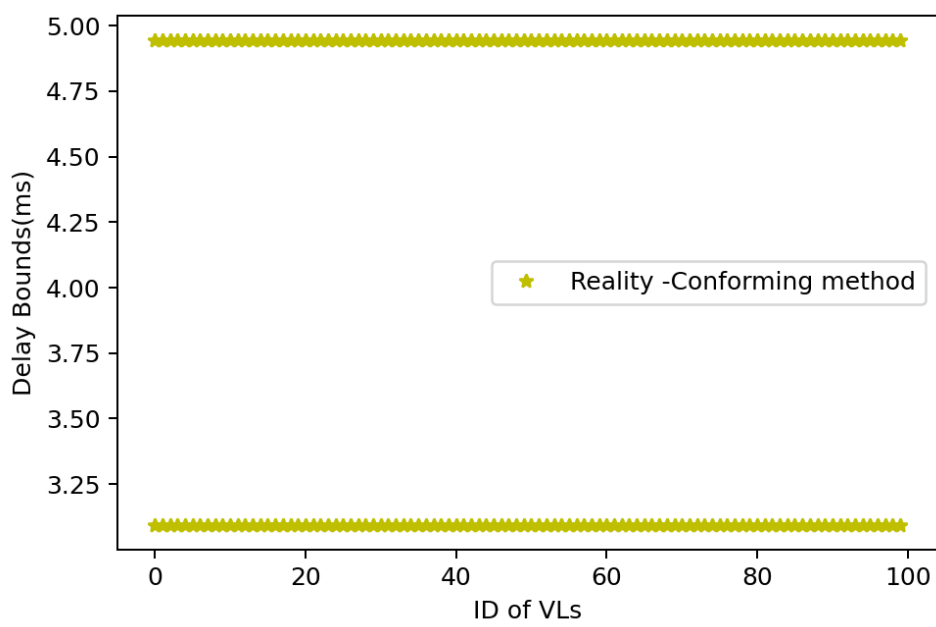
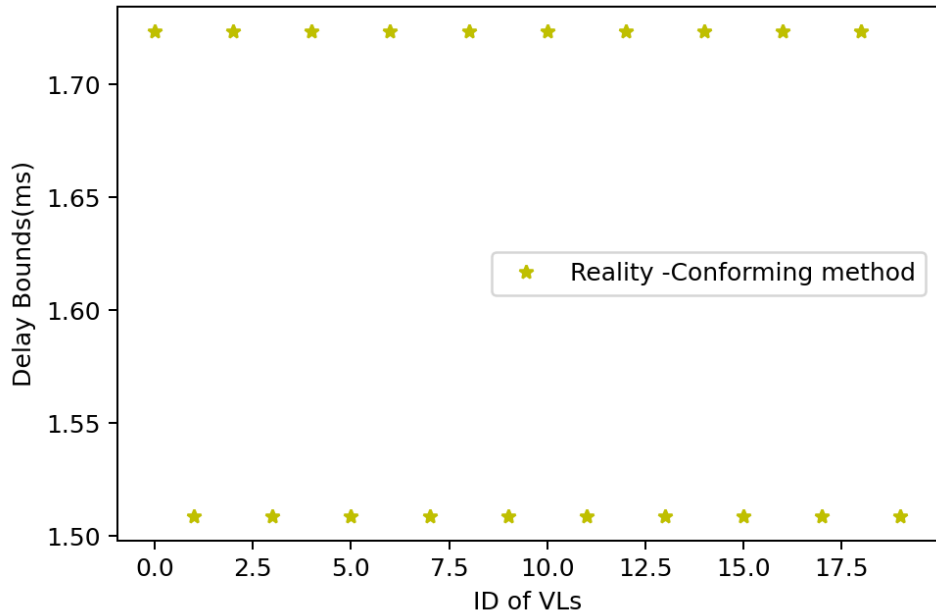


Plotting both State of the Art and Reality Conforming Method, we get following results:



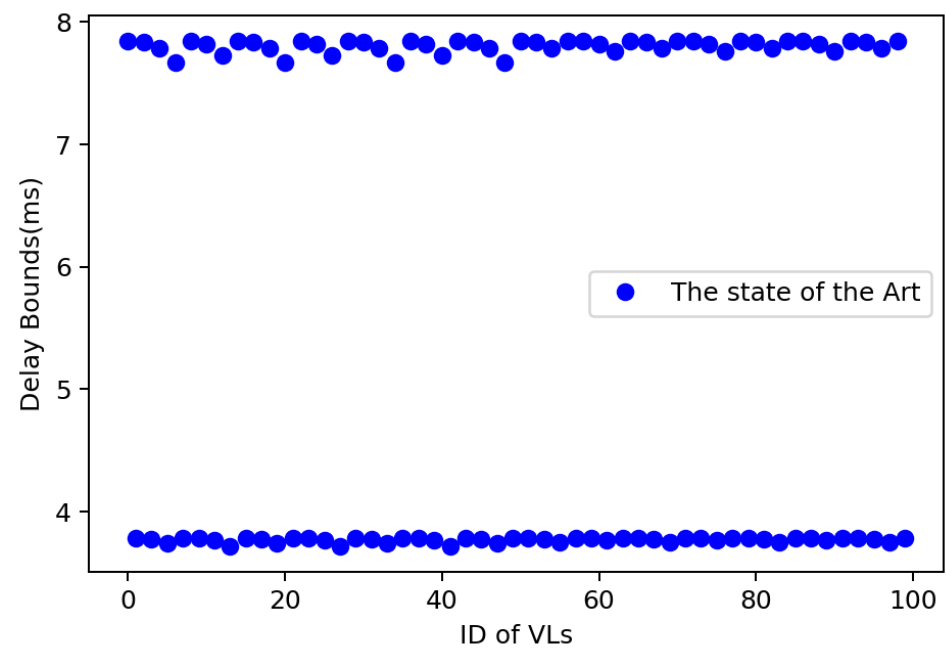
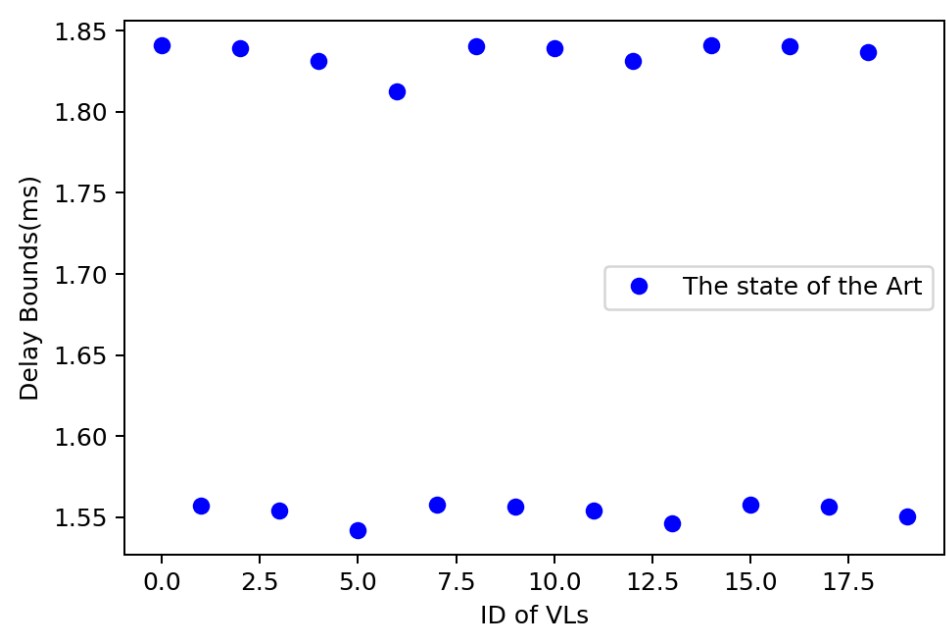
C. Get Delays for Reality Conforming method when there are two priorities without frame preemption.

- Go to **OurMethod.java** file
- Inside **getDelay1** method, set **res = FPNFP**
- In **main** method, set **numVL** (no of virtual links) to 20 or 100 depending upon no of VL's used.
- In **main** method, set **numP** (priority) to 2, indicate VLs have different priority
- Run the code and record the delay values to plot the graph.

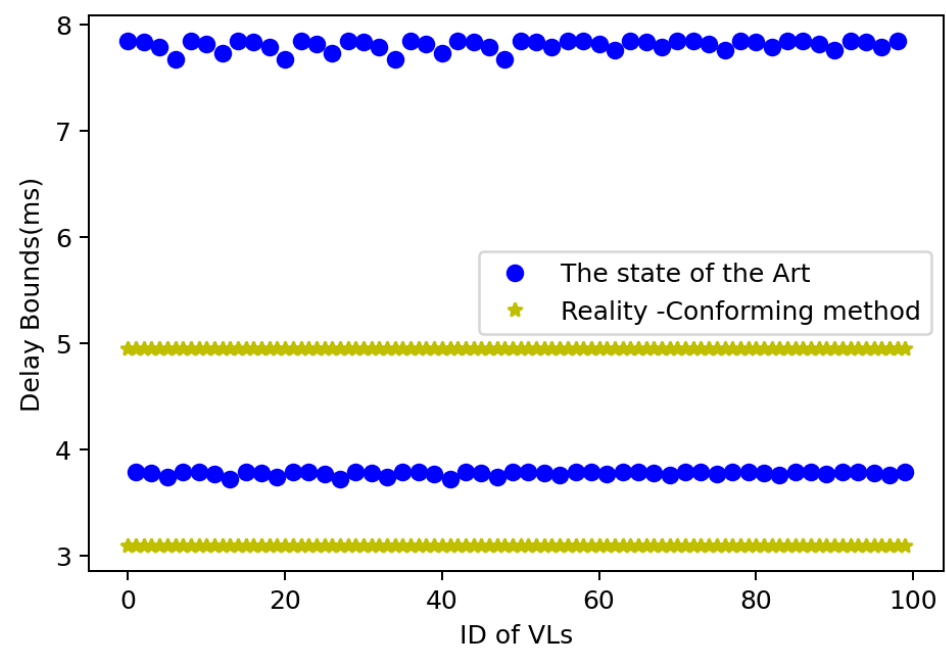
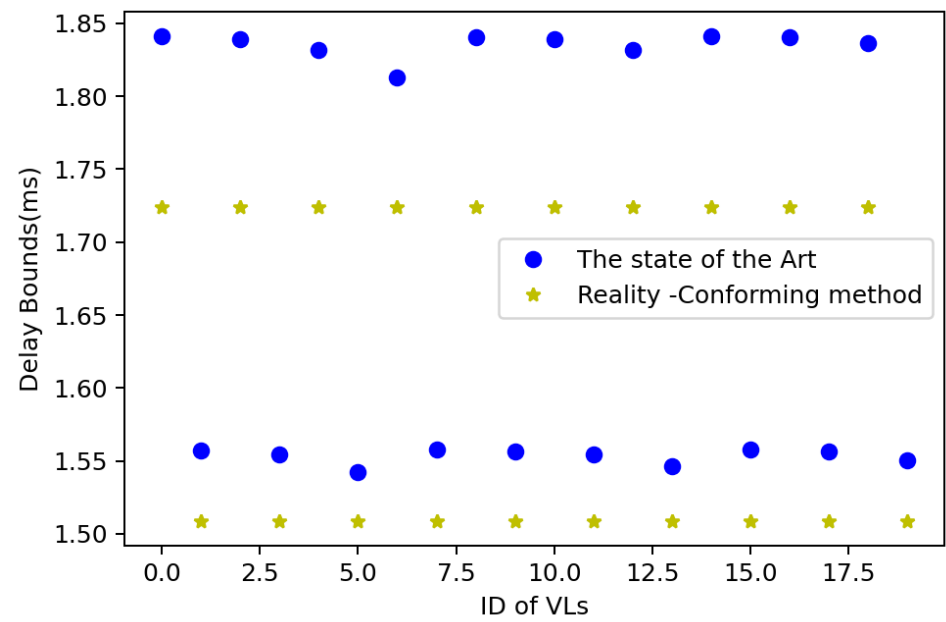


D. Get Delays for State of Art Method when there are two priorities without frame preemption.

- Go to *State_of_Art.java* file
- In *main* method, set *numVL* (no of virtual links) to 20 or 100 depending upon no of VL's used.
- In *main* method, set *numOfPriority* (priority) to 2, indicate VLs have different priority.
- In *main* method, at end of the method, set *delay* to *State_Of_ArtNFP*
- Run the code and record the delay values to plot the graph.

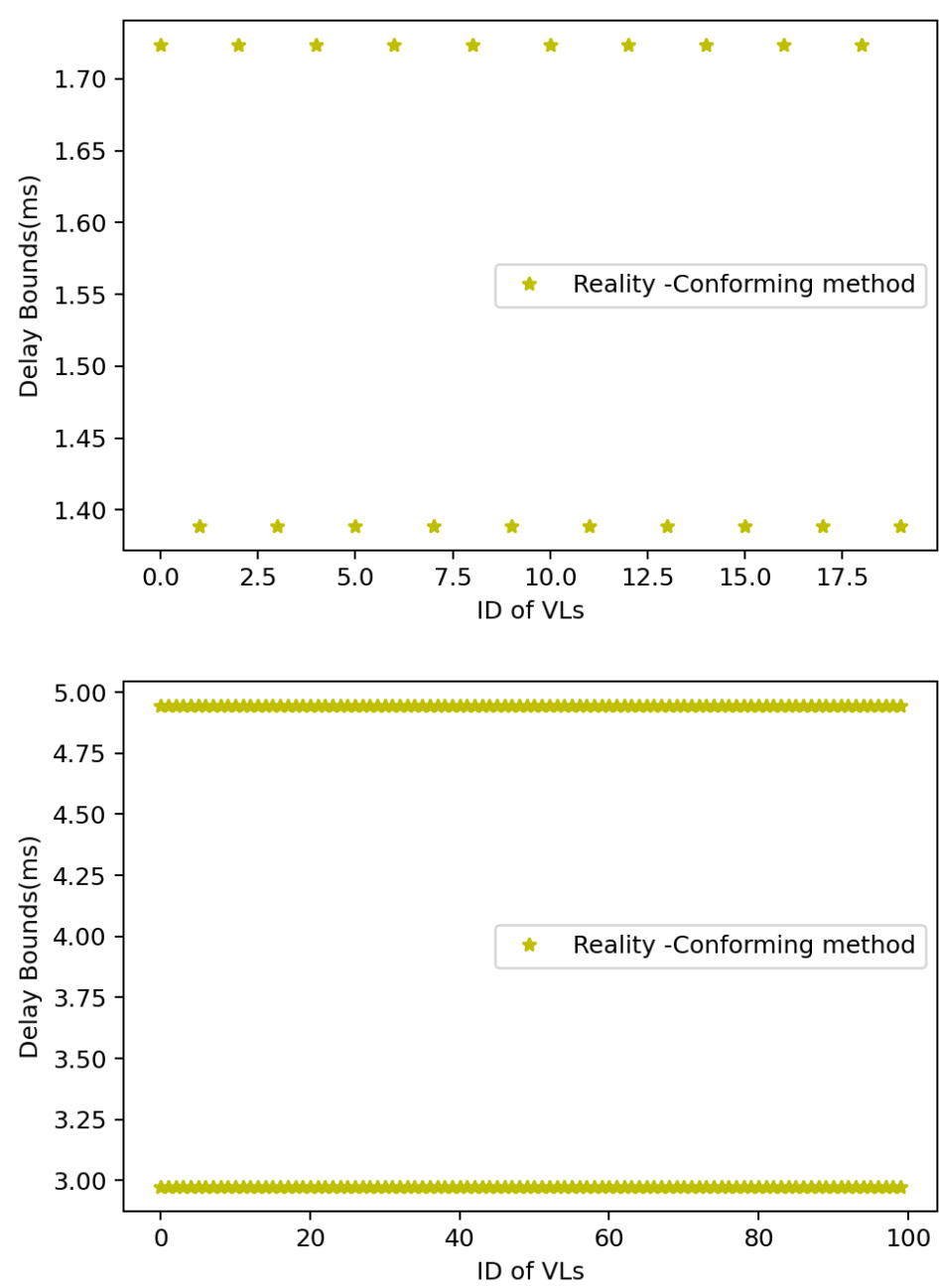


Plotting both State of the Art and Reality Conforming Method, we get following results:



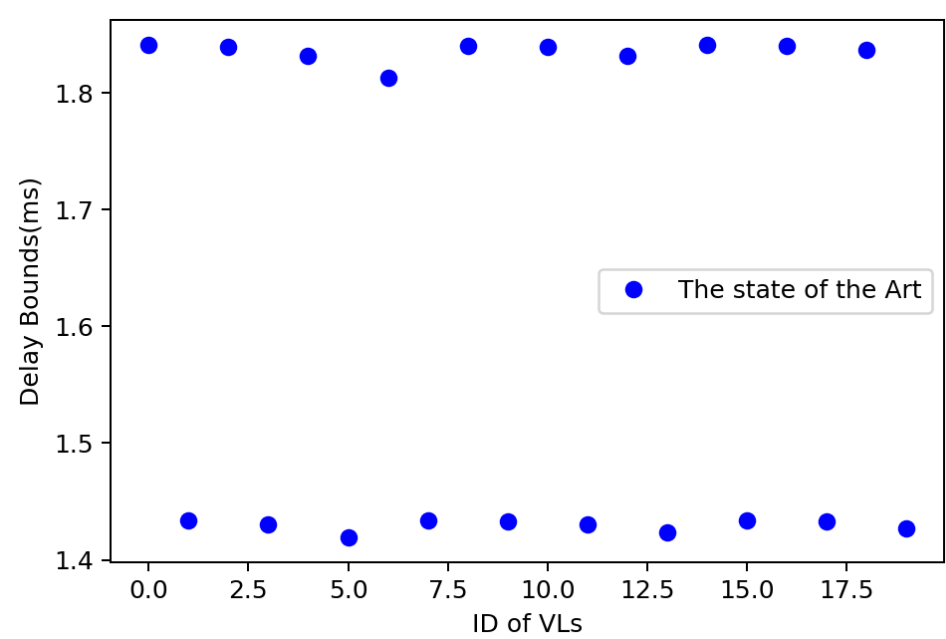
E. Get Delays for Reality Conforming method when there are two priorities with frame preemption.

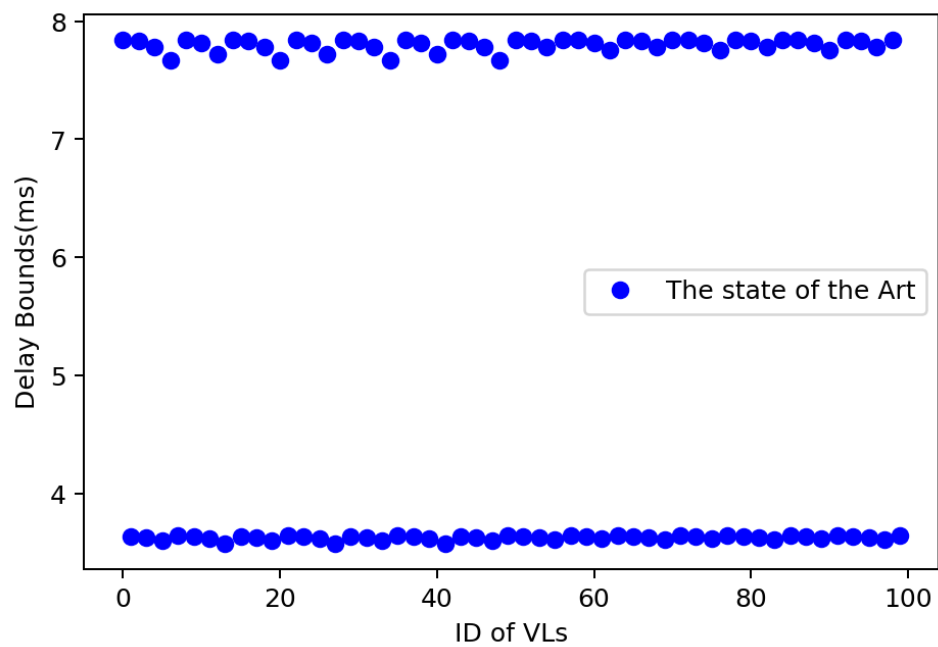
- Go to *OurMethod.java* file
- Inside *getDelay1* method, set *res* = *FPFP*
- In *main* method, set *numVL* (no of virtual links) to 20 or 100 depending upon no of VL's used.
- In *main* method, set *numP* (priority) to 2, indicate all VLs have same priority
- Run the code and record the delay values to plot the graph.



F. Get Delays for State of Art method when there are two priorities with frame preemption.

- Go to *State_of_Art.java* file
- In *main* method, set *numVL* (no of virtual links) to 20 or 100 depending upon no of VL's used.
- In *main* method, set *numOfPriority* (priority) to 2, indicate VLs have different priority.
- In *main* method, at end of the method, set *delay* to *State_Of_ArtFP*
- Run the code and record the delay values to plot the graph.





Plotting both State of the Art and Reality Conforming Method, we get following results:

