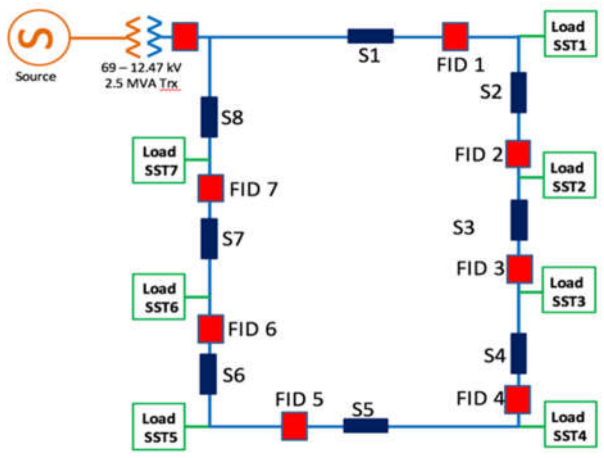
CoDES Algorithm DGI User Manual

Contents

1. Simulation Setup

The current DGI code works in a 7-node system that FSU is using in their HIL testbed, as shown in the figure below.

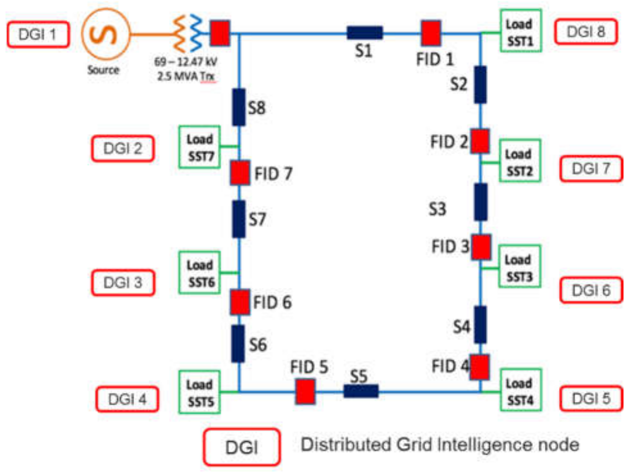


1.1. Algorithm Inputs and Outputs

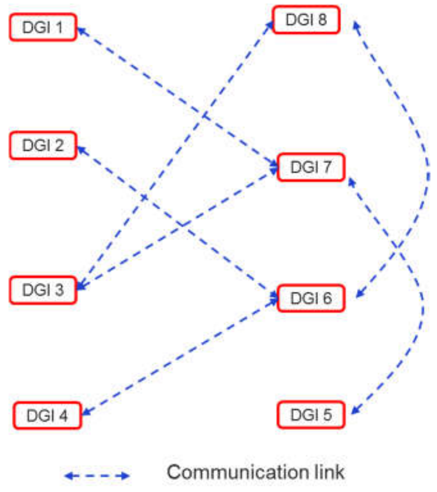
- Algorithm inputs:
  - Day-ahead Renewable forecast (24 hours, 1 hour resolution)
  - Load forecast (24 hours, 1 hour resolution)
  - DESD specifications
- Algorithm outputs:
  - DESD charging/discharging commands (24 hours, 1 hour resolution)
  - Power trade with grid (24 hours, 1 hour resolution)

1.2. Communication network

- 8 DGI nodes, located on 7 SSTs and the grid, as shown below:



- Communication network could be changed as long as the graph is connected. The current setup is as follows:



1.3. Devices Specifications

- Renewable generations (kW)

Hour	SST 1	SST 2	SST 3	SST 4	SST 5	SST 6	SST 7
1 am	0	0	0	0	0	0	0
2 am	0	0	0	0	0	0	0
3 am	0	0	0	0	0	0	0
4 am	0	0	0	0	0	0	0
5 am	0	0	0	0	0	0	0
6 am	0	0	0	0	0	0	0
7 am	0	0	0	0	0	0	0
8 am	1.15	0.15	0.10	0.06	0.23	0.48	0
9 am	0.45	0.43	0.30	0.38	0.67	1.34	0.03
10 am	0.97	0.95	0.73	0.80	1.12	2.31	0.07
11 am	1.89	1.84	1.37	1.67	2.24	3.54	0.19
12 pm	1.69	1.72	1.49	1.54	1.90	3.52	0.26
1 pm	3.16	3.17	2.52	2.89	3.70	4.42	0.45
2 pm	3.52	3.60	2.96	3.17	3.79	4.66	0.11
3 pm	3.57	3.72	3.18	3.30	3.73	2.77	0.15
4 pm	3.30	3.46	2.91	3.02	3.42	1.00	0.26
5 pm	1.71	1.85	1.63	1.54	1.77	0.59	0.16
6 pm	0.77	0.84	0.75	0.59	0.79	1.08	0.11
7 pm	1.00	1.31	1.17	0.71	0.97	0.48	0.11
8 pm	0.12	0.15	0.13	0	0.14	0.16	0.02
9 pm	0	0	0	0	0	0	0
10 pm	0	0	0	0	0	0	0
11 pm	0	0	0	0	0	0	0
12 am	0	0	0	0	0	0	0

- Load (kW)

Hour	SST 1	SST 2	SST 3	SST 4	SST 5	SST 6	SST 7
1 am	1.53	2.62	0.95	1.01	3.55	1.13	2.70
2 am	0.83	2.06	0.90	0.74	2.67	2.31	2.53
3 am	1.17	1.41	0.80	0.53	2.59	0.66	2.10
4 am	1.15	1.43	0.80	0.73	1.69	0.54	2.50
5 am	0.81	1.33	0.73	0.45	1.63	0.64	2.03
6 am	0.77	1.20	1.65	0.73	1.15	0.53	1.93
7 am	1.18	1.08	0.65	0.54	1.04	0.64	2.18
8 am	0.76	0.96	0.30	0.77	0.76	0.92	2.38
9 am	1.19	0.64	0.60	0.52	0.35	0.77	1.09
10 am	1.43	0.62	0.40	0.81	0.31	0.99	1.09
11 am	1.28	0.58	2.29	0.77	0.34	1.28	0.79
12 pm	1.38	0.77	3.88	0.85	0.33	1.95	3.37
1 pm	1.48	0.75	1.65	1.22	0.31	2.27	1.55
2 pm	1.57	0.98	0.72	1.44	0.33	0.85	2.07
3 pm	2.77	1.39	1.10	1.65	0.33	2.64	1.78
4 pm	2.76	2.06	1.04	1.91	0.92	3.00	1.43
5 pm	2.28	2.45	0.39	1.81	1.83	2.88	1.51
6 pm	2.08	2.42	0.28	2.47	1.85	2.66	1.39
7 pm	3.13	1.85	0.23	2.74	2.92	2.32	0.98
8 pm	2.26	2.71	0.21	2.56	4.40	0.64	2.45
9 pm	1.92	1.89	0.43	3.36	2.41	2.55	4.37
10 pm	1.28	2.99	2.22	2.40	2.67	1.44	4.78
11 pm	0.82	2.84	1.50	1.83	2.68	0.71	3.75
12 am	1.31	2.17	1.07	1.54	3.65	0.58	3.33

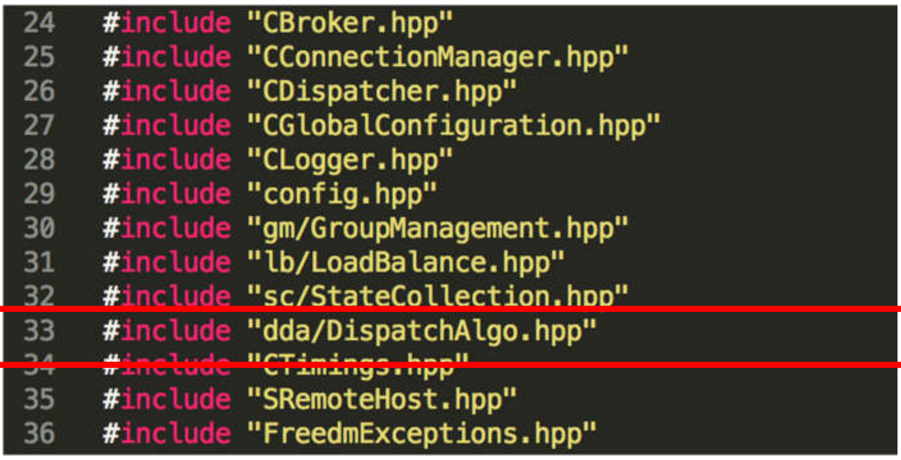
- DESD specifications

Capacity (kWh)	Maximum charging/discharging rate (kW)	Initial SOC (kWh)	Charging/discharging efficiency
7	3.3	0.25x7	0.92

- Grid specifications
    - Maximum power rate: 100 kW

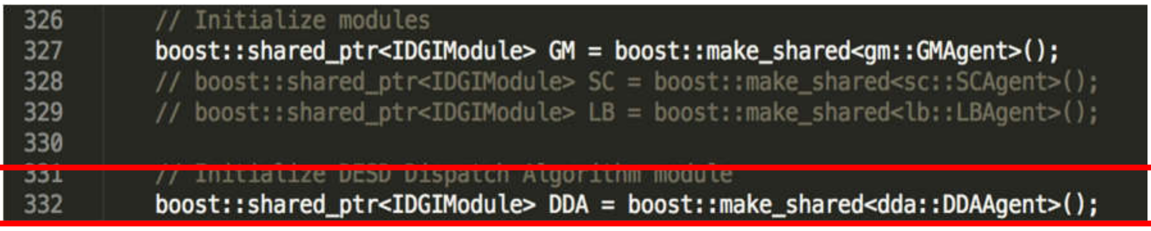
2. Schedule Module in DGI Prosix Main (already finished in the uploaded code)

Before compiling the code. Make sure the module has been registered in the DGI main function, *PosixMain.cpp*. And "dda/DispatchAlgo.hpp" is included in the *PosixMain.cpp*.

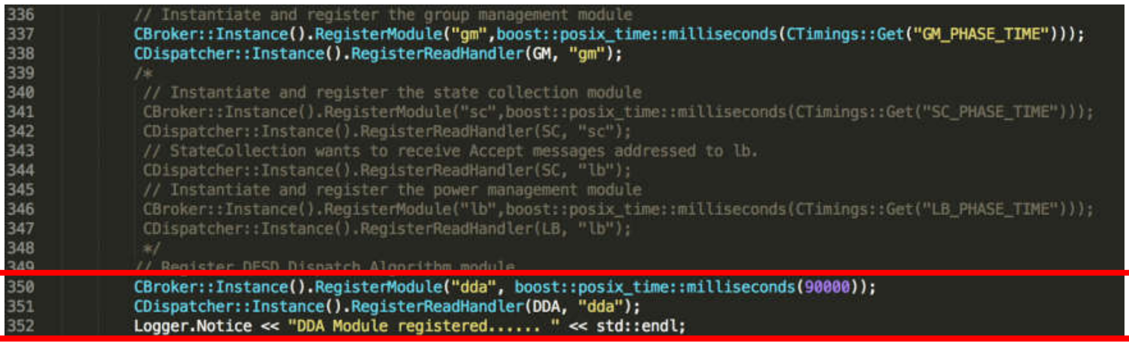


```
24  #include "CBroker.hpp"
25  #include "CConnectionManager.hpp"
26  #include "CDispatcher.hpp"
27  #include "CGlobalConfiguration.hpp"
28  #include "CLogger.hpp"
29  #include "config.hpp"
30  #include "gm/GroupManagement.hpp"
31  #include "lb/LoadBalance.hpp"
32  #include "sc/StateCollection.hpp"
33  #include "dda/DispatchAlgo.hpp"
34  #include "CTimings.hpp"
35  #include "SRemoteHost.hpp"
36  #include "FreedmExceptions.hpp"
```

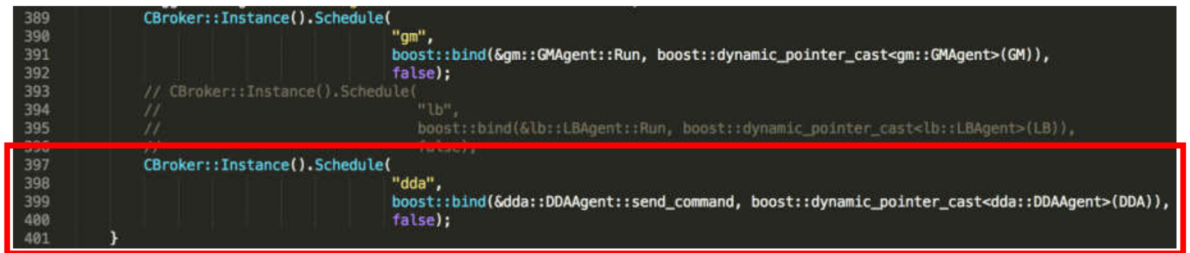
In the code, the dda module has already been registered with 90 seconds phase time. Detailed code are shown in the source code between line 326 to line 407.



```
326      // Initialize modules
327      boost::shared_ptr<IDGIModule> GM = boost::make_shared<gm::GMAgent>();
328      // boost::shared_ptr<IDGIModule> SC = boost::make_shared<sc::SCAgent>();
329      // boost::shared_ptr<IDGIModule> LB = boost::make_shared<lb::LBAgent>();
330
331      // Initialize DESD Dispatch Algorithm module
332      boost::shared_ptr<IDGIModule> DDA = boost::make_shared<dda::DDAAgent>();
```



```
336          // Instantiate and register the group management module
337          CBroker::Instance().RegisterModule("gm",boost::posix_time::milliseconds(CTimings::Get("GM_PHASE_TIME")));
338          CDispatcher::Instance().RegisterReadHandler(GM, "gm");
339          /*
340           // Instantiate and register the state collection module
341           CBroker::Instance().RegisterModule("sc",boost::posix_time::milliseconds(CTimings::Get("SC_PHASE_TIME")));
342           CDispatcher::Instance().RegisterReadHandler(SC, "sc");
343           // StateCollection wants to receive Accept messages addressed to lb.
344           CDispatcher::Instance().RegisterReadHandler(SC, "lb");
345           // Instantiate and register the power management module
346           CBroker::Instance().RegisterModule("lb",boost::posix_time::milliseconds(CTimings::Get("LB_PHASE_TIME")));
347           CDispatcher::Instance().RegisterReadHandler(LB, "lb");
348           */
349          // Register DESD Dispatch Algorithm module
350          CBroker::Instance().RegisterModule("dda", boost::posix_time::milliseconds(90000));
351          CDispatcher::Instance().RegisterReadHandler(DDA, "dda");
352          Logger.Notice << "DDA Module registered...... " << std::endl;
```

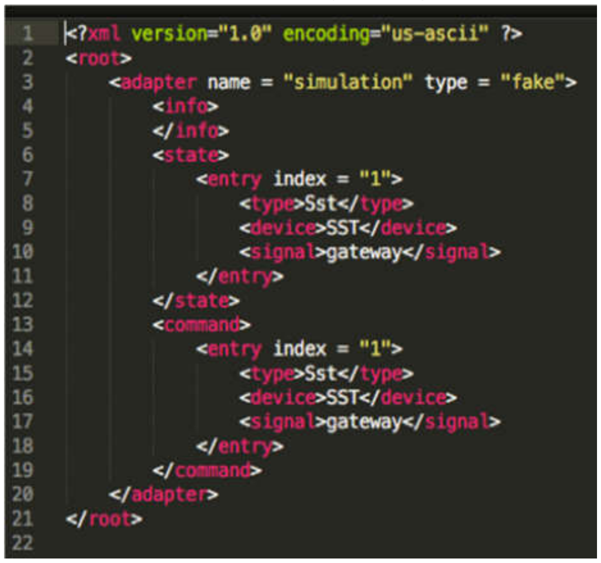


```
389          CBroker::Instance().Schedule(
390                                      "gm",
391                                      boost::bind(&gm::GMAgent::Run, boost::dynamic_pointer_cast<gm::GMAgent>(GM)),
392                                      false);
393          // CBroker::Instance().Schedule(
394          //                              "lb",
395          //                              boost::bind(&lb::LBAgent::Run, boost::dynamic_pointer_cast<lb::LBAgent>(LB)),
396          //                              false);
397          CBroker::Instance().Schedule(
398                                      "dda",
399                                      boost::bind(&dda::DDAAgent::send_command, boost::dynamic_pointer_cast<dda::DDAAgent>(DDA)),
400                                      false);
401      }
```

3. Configure the DGI (already finished in the uploaded code)

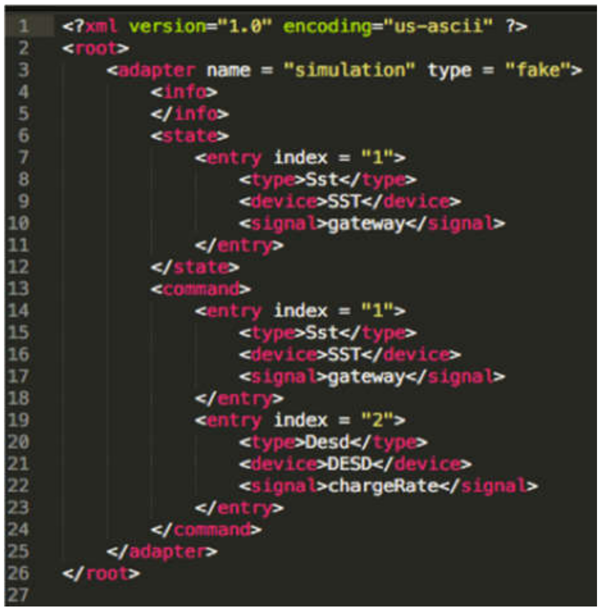
This step describes how to configure the DGI program by add/modifying files in the */config* directory for each DGI instance (total eight including a grid node). The current implementation requires the following files to be included in the directory:

1. Adapter.xml
    a. This file is used for DGI to interface with RTDS, PSCAD or physical devices such that states (with <state> header) can be read from and command (with <command> header) can be sent to plants. Currently the controller is not connected to the DGI program, so all the adapters are set to be fake.
    b. For the grid node, the file is currently configured as:



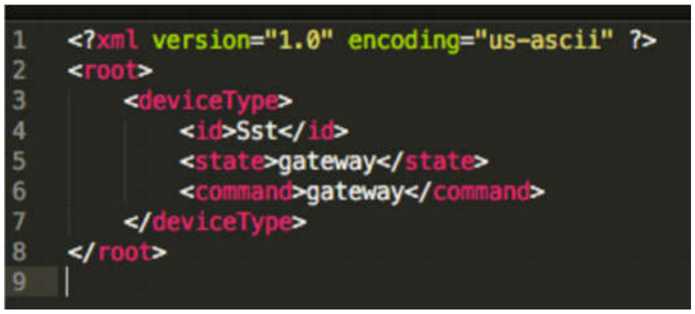
```
<?xml version="1.0" encoding="us-ascii" ?>
<root>
    <adapter name = "simulation" type = "fake">
        <info>
        </info>
        <state>
            <entry index = "1">
                <type>Sst</type>
                <device>SST</device>
                <signal>gateway</signal>
            </entry>
        </state>
        <command>
            <entry index = "1">
                <type>Sst</type>
                <device>SST</device>
                <signal>gateway</signal>
            </entry>
        </command>
    </adapter>
</root>
```

    c. For the other nodes, the file is currently configured as:



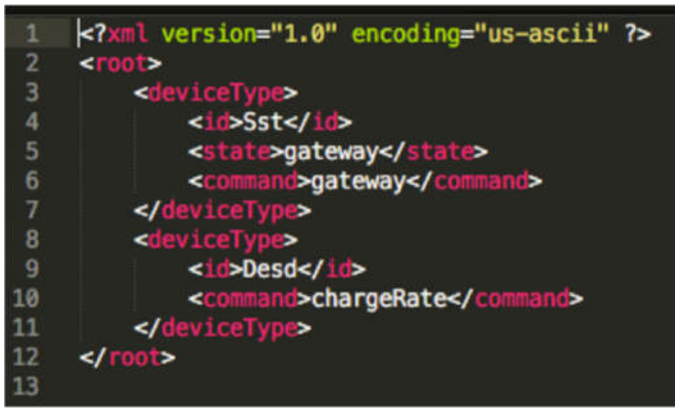
```
<?xml version="1.0" encoding="us-ascii" ?>
<root>
    <adapter name = "simulation" type = "fake">
        <info>
        </info>
        <state>
            <entry index = "1">
                <type>Sst</type>
                <device>SST</device>
                <signal>gateway</signal>
            </entry>
        </state>
        <command>
            <entry index = "1">
                <type>Sst</type>
                <device>SST</device>
                <signal>gateway</signal>
            </entry>
            <entry index = "2">
                <type>Desd</type>
                <device>DESD</device>
                <signal>chargeRate</signal>
            </entry>
        </command>
    </adapter>
</root>
```

2. Device.xml
    a. This file is used for DGI to know what devices are connected to the controller with <deviceType> header.
    b. For the grid node, the file is currently configured as:



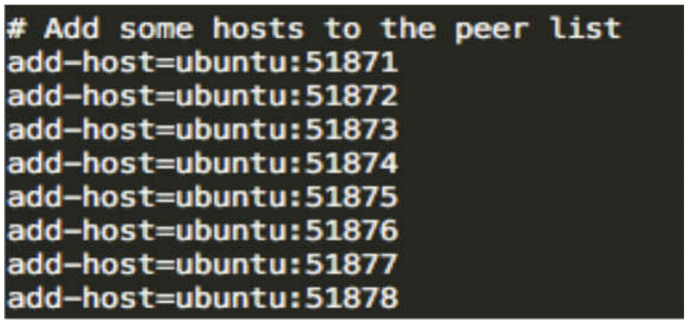
```
<?xml version="1.0" encoding="us-ascii" ?>
<root>
    <deviceType>
        <id>Sst</id>
        <state>gateway</state>
        <command>gateway</command>
    </deviceType>
</root>
```

    c. For the other nodes, the file is currently configured as:



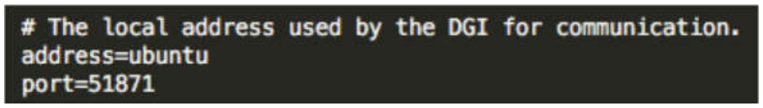
```
<?xml version="1.0" encoding="us-ascii" ?>
<root>
    <deviceType>
        <id>Sst</id>
        <state>gateway</state>
        <command>gateway</command>
    </deviceType>
    <deviceType>
        <id>Desd</id>
        <command>chargeRate</command>
    </deviceType>
</root>
```

3. Freedm.cfg
    a. This is the most important configuration file for DGI, there are several things need to be configured before executing the DGI program.
    b. Under the line "*# Add some hosts to the peer list*". You will need to add the address of all the DGI node into this host list. The current configuration is as follows as I am using a single machine to run multiple DGI instances and my localhost address is ubuntu (you will be replacing this with your own localhost address). Essentially each DGI instance will have the same peer list.



```
# Add some hosts to the peer list
add-host=ubuntu:51871
add-host=ubuntu:51872
add-host=ubuntu:51873
add-host=ubuntu:51874
add-host=ubuntu:51875
add-host=ubuntu:51876
add-host=ubuntu:51877
add-host=ubuntu:51878
```

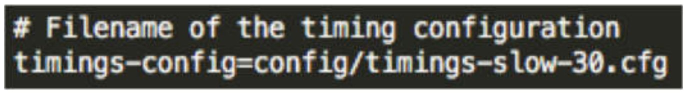
c. Under the line "# *The local address used by the DGI for communication*". You will need to configure the IP address and port of the current DGI instance. One example is:



```
# The local address used by the DGI for communication.
address=ubuntu
port=51871
```

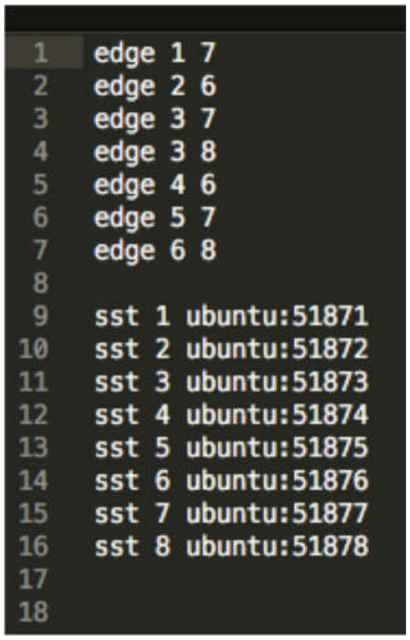
d. If you are using multiple machines to run the DGI program distributed, make sure all the IP addresses are resolvable by each DGI instance.

e. Under the line "# *Filename of the timing configuration*". You will need to configure the timing information that the current DGI needs. The file name should be the same as the one included in the /config directory. One example is:



```
# Filename of the timing configuration
timings-config=config/timings-slow-30.cfg
```

4. Logger.cfg
    a. Does not need to modify the current file. Basically, it is a file with all the lines commented out
5. Timings.cfg
    a. This file configures the execution speed of the DGI code by setting different expiration time to different timers
    b. Currently using *timings-slow-30.cfg*
6. Topology.cfg
    a. Includes the information of the defined communication topology for consensus network
    b. This is a workaround as this file defines the physical topology of the system. In DGI environment, the peers are communicating through an Ethernet with each instance configured with a certain IP address with a certain port number. Thus, it is essentially a fully connected communication topology.
    c. By using this configuration file, essentially we are forcing the communication topology to be the same as the physical topology.
    d. Each DGI instances should have the same topology.cfg file.
    e. The file is currently configured as:



```
edge 1 7
edge 2 6
edge 3 7
edge 3 8
edge 4 6
edge 5 7
edge 6 8

sst 1 ubuntu:51871
sst 2 ubuntu:51872
sst 3 ubuntu:51873
sst 4 ubuntu:51874
sst 5 ubuntu:51875
sst 6 ubuntu:51876
sst 7 ubuntu:51877
sst 8 ubuntu:51878
```

For further details, please refer to DGI manual.

4. DGI code Compilation

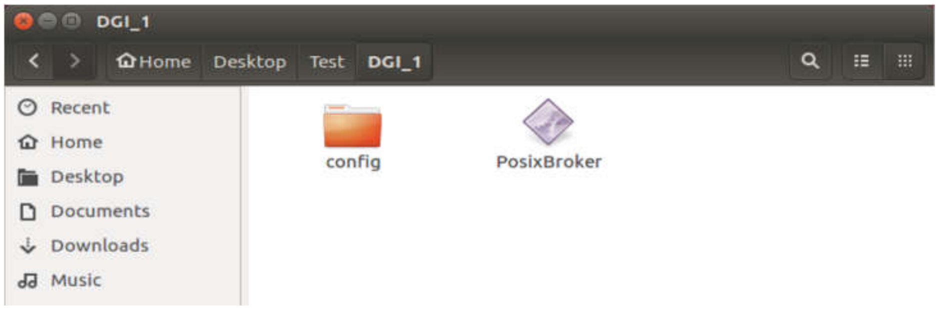
Download and save the DGI source code into your local machine and follow the steps below to compile the DGI source code in the Linux environment that you have built:

```
1. cd your_local_directory/CoDES_DGI/Broker
2. cmake -DCMAKE_BUILD_TYPE=Release
3. make
```

By now, the source code should have been compiled and an executable file named "PosixBroker" should have been generated under the Broker directory.

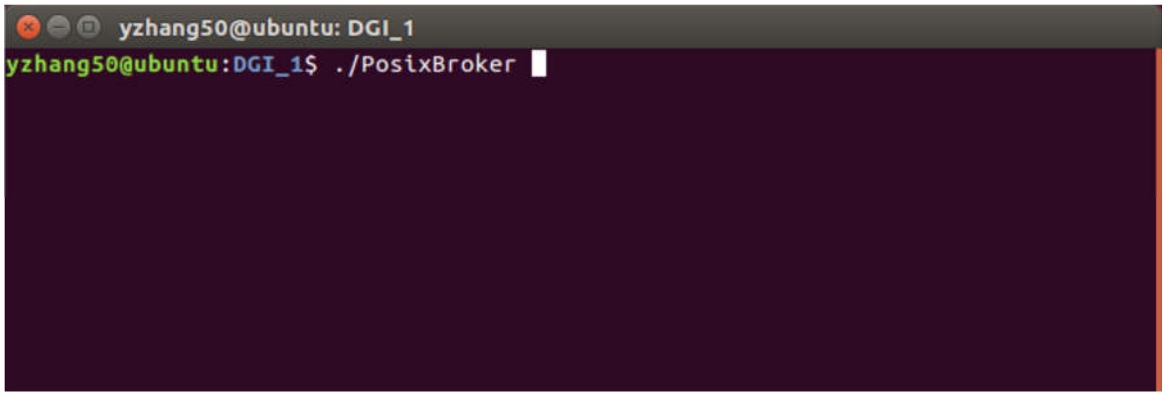
5. Execute the DGI code

To execute the DGI code for this case, you will need to create 8 independent folders for each DGI instance. Each folder should include the executable file "*PosixBroker*" and the */config* folder that contains all the configuration files.



Open 8 terminal sessions and make the current directory of each session corresponds to each DGI instance by using the "`cd`" command.

Finally, by executing (`./ProsixBroker`) the executable file one by one, the CoDES algorithm should be running and each terminal session will print out algorithm information as the algorithm executes.

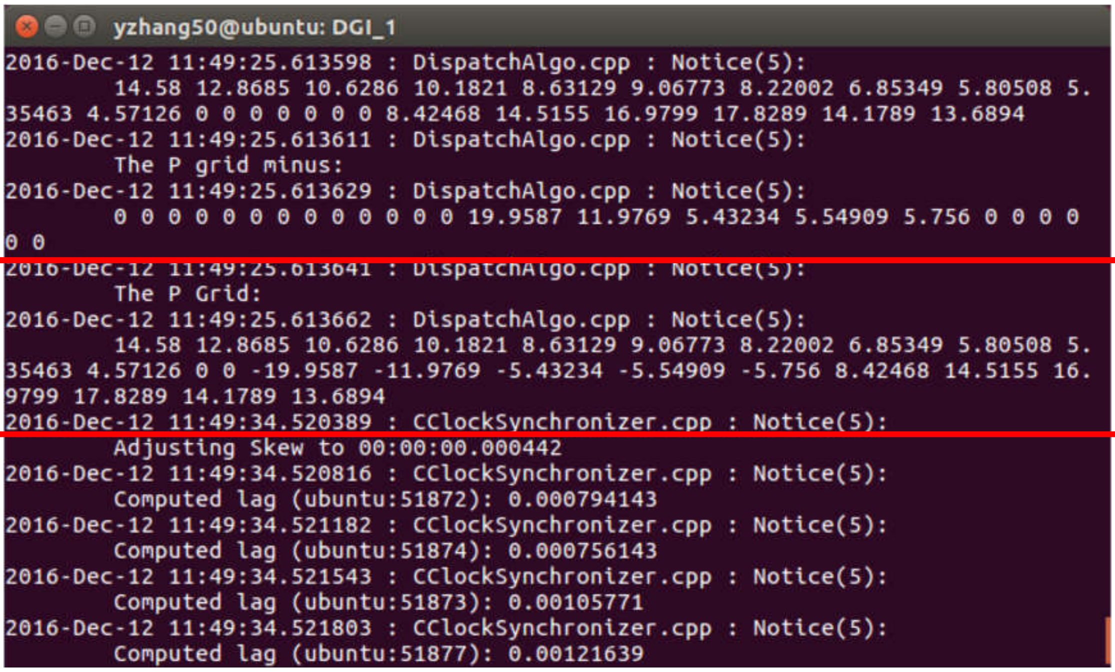


```
yzhang50@ubuntu:DGI_1$ ./PosixBroker
```

6. Execution result example

- Power trade with grid

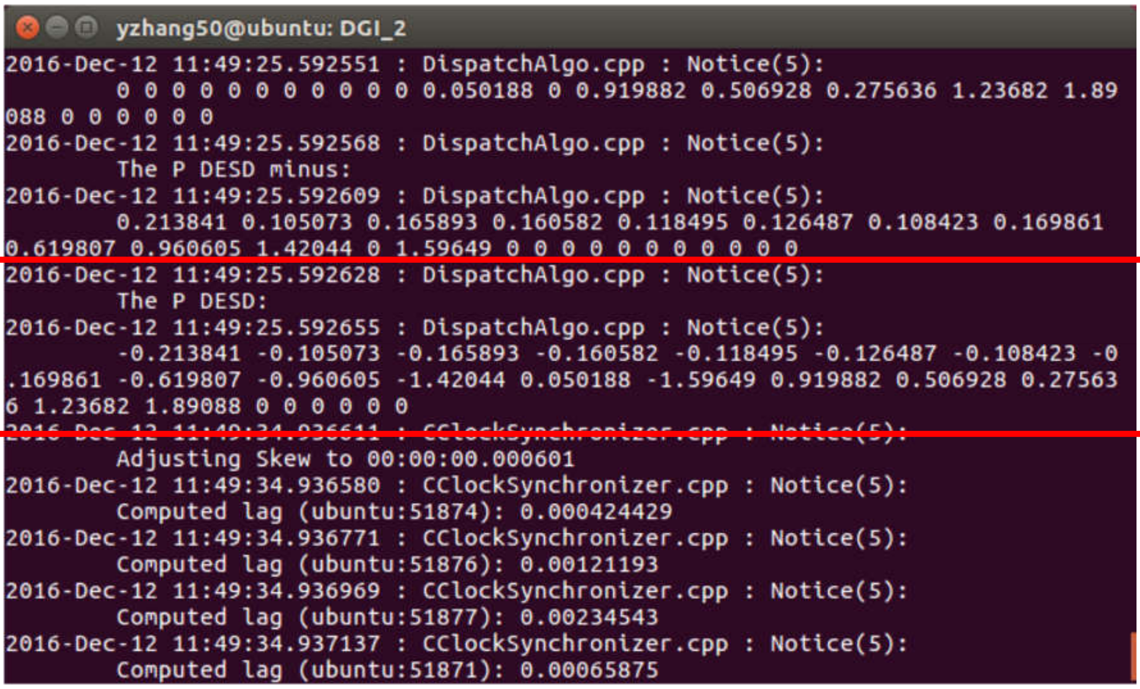
The printed values “The P Grid: 14.58 12.8685 …..” are the power trade with grid.



```
2016-Dec-12 11:49:25.613598 : DispatchAlgo.cpp : Notice(5):
        14.58 12.8685 10.6286 10.1821 8.63129 9.06773 8.22002 6.85349 5.80508 5.
35463 4.57126 0 0 0 0 0 0 0 8.42468 14.5155 16.9799 17.8289 14.1789 13.6894
2016-Dec-12 11:49:25.613611 : DispatchAlgo.cpp : Notice(5):
        The P grid minus:
2016-Dec-12 11:49:25.613629 : DispatchAlgo.cpp : Notice(5):
        0 0 0 0 0 0 0 0 0 0 0 0 19.9587 11.9769 5.43234 5.54909 5.756 0 0 0 0
0 0
2016-Dec-12 11:49:25.613641 : DispatchAlgo.cpp : Notice(5):
        The P Grid:
2016-Dec-12 11:49:25.613662 : DispatchAlgo.cpp : Notice(5):
        14.58 12.8685 10.6286 10.1821 8.63129 9.06773 8.22002 6.85349 5.80508 5.
35463 4.57126 0 0 -19.9587 -11.9769 -5.43234 -5.54909 -5.756 8.42468 14.5155 16.
9799 17.8289 14.1789 13.6894
2016-Dec-12 11:49:34.520389 : CClockSynchronizer.cpp : Notice(5):
        Adjusting Skew to 00:00:00.000442
2016-Dec-12 11:49:34.520816 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51872): 0.000794143
2016-Dec-12 11:49:34.521182 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51874): 0.000756143
2016-Dec-12 11:49:34.521543 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51873): 0.00105771
2016-Dec-12 11:49:34.521803 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51877): 0.00121639
```

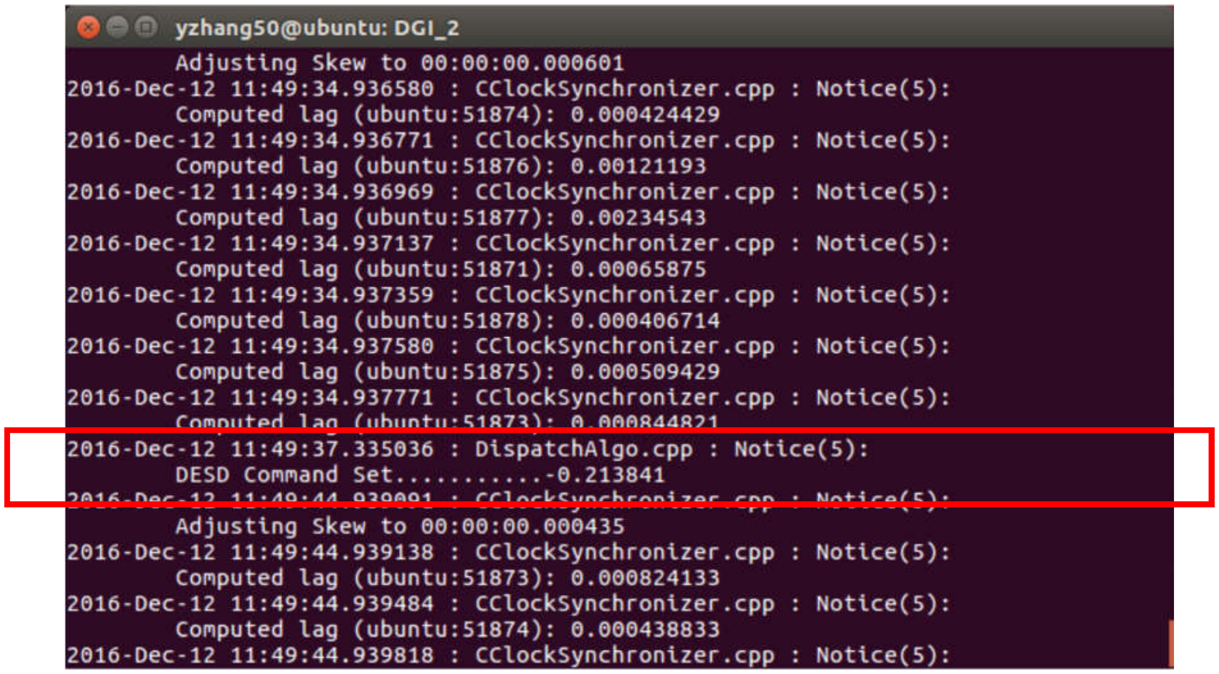
- DESD charging/discharging command

The printed values “The P DESD: -0.213841 -0.105073 …..” are the DESD commands:



```
2016-Dec-12 11:49:25.592551 : DispatchAlgo.cpp : Notice(5):
        0 0 0 0 0 0 0 0 0 0 0 0.050188 0 0.919882 0.506928 0.275636 1.23682 1.89
088 0 0 0 0 0 0 0
2016-Dec-12 11:49:25.592568 : DispatchAlgo.cpp : Notice(5):
        The P DESD minus:
2016-Dec-12 11:49:25.592609 : DispatchAlgo.cpp : Notice(5):
        0.213841 0.105073 0.165893 0.160582 0.118495 0.126487 0.108423 0.169861
0.619807 0.960605 1.42044 0 1.59649 0 0 0 0 0 0 0 0 0 0 0
2016-Dec-12 11:49:25.592628 : DispatchAlgo.cpp : Notice(5):
        The P DESD:
2016-Dec-12 11:49:25.592655 : DispatchAlgo.cpp : Notice(5):
        -0.213841 -0.105073 -0.165893 -0.160582 -0.118495 -0.126487 -0.108423 -0
.169861 -0.619807 -0.960605 -1.42044 0.050188 -1.59649 0.919882 0.506928 0.27563
6 1.23682 1.89088 0 0 0 0 0 0
2016-Dec-12 11:49:34.936611 : CClockSynchronizer.cpp : Notice(5):
        Adjusting Skew to 00:00:00.000601
2016-Dec-12 11:49:34.936580 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51874): 0.000424429
2016-Dec-12 11:49:34.936771 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51876): 0.00121193
2016-Dec-12 11:49:34.936969 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51877): 0.00234543
2016-Dec-12 11:49:34.937137 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51871): 0.00065875
```

- Sending Commands to the corresponding DESD device (we just print it on the screen now, there is no actual commands sent to the devices)



```
        Adjusting Skew to 00:00:00.000601
2016-Dec-12 11:49:34.936580 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51874): 0.000424429
2016-Dec-12 11:49:34.936771 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51876): 0.00121193
2016-Dec-12 11:49:34.936969 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51877): 0.00234543
2016-Dec-12 11:49:34.937137 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51871): 0.00065875
2016-Dec-12 11:49:34.937359 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51878): 0.000406714
2016-Dec-12 11:49:34.937580 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51875): 0.000509429
2016-Dec-12 11:49:34.937771 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51873): 0.000844821
2016-Dec-12 11:49:37.335036 : DispatchAlgo.cpp : Notice(5):
        DESD Command Set...........-0.213841
2016-Dec-12 11:49:44.939091 : CClockSynchronizer.cpp : Notice(5):
        Adjusting Skew to 00:00:00.000435
2016-Dec-12 11:49:44.939138 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51873): 0.000824133
2016-Dec-12 11:49:44.939484 : CClockSynchronizer.cpp : Notice(5):
        Computed lag (ubuntu:51874): 0.000438833
2016-Dec-12 11:49:44.939818 : CClockSynchronizer.cpp : Notice(5):
```