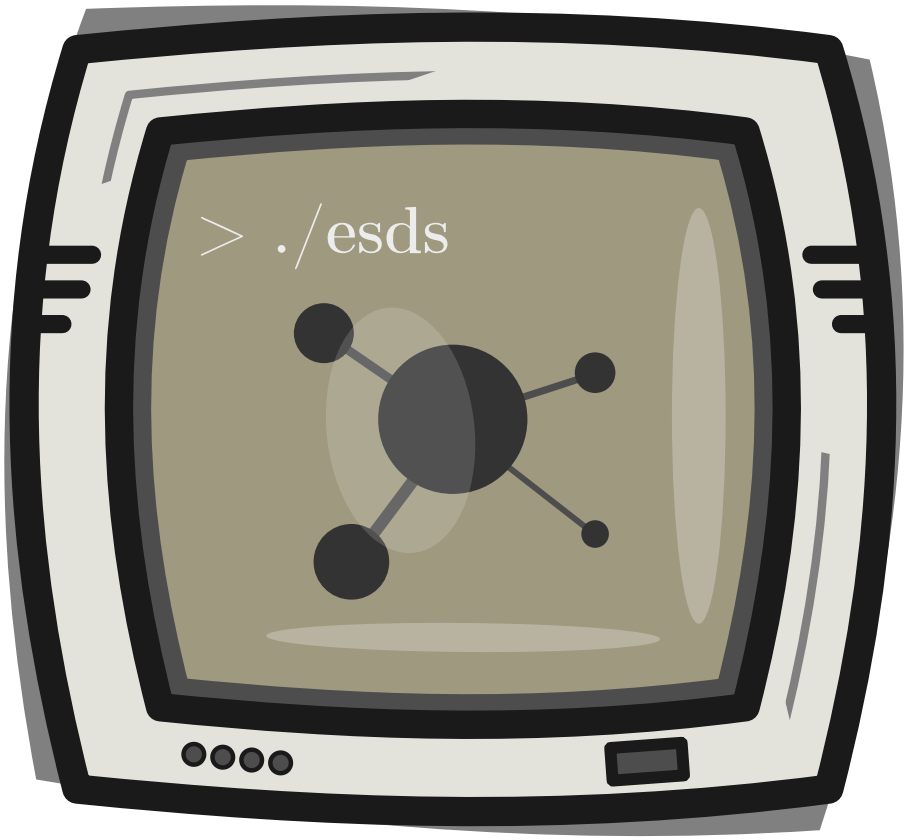
User Manual

— ESDS v0.0.3 —

August 23, 2023



ESDS an Extensible Simulator for Distributed Systems

Written by Loic Guegan and Issam Raïs

Contents

1 Architecture of ESDS

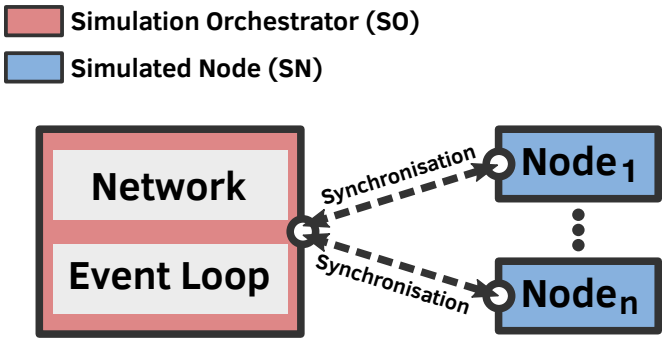


Figure 1: Simulation architecture used by ESDS

The ESDS simulator comprises two major components: 1) The Simulation Orchestrator (SO) 2) The Simulated Nodes (SN). This architecture is depicted on Figure 1. The SO is the main process in charge of implementing the simulation main loop. It instantiates the network (e.g bandwidths and latencies matrices), collects and processes the events (e.g communications,turn on/off). On the other hand, the SNs are threads that implement the node behaviors by following so called *node API*.

2 Getting started

To run a simulation, at least 2 files are required: 1) a platform file 2) a node implementation source code. The platform file defines the simulated network platform (network links and performances etc.) and sets various simulation parameters. The node implementation source code provides the logic of the simulated nodes.

2.1 Platform file

Platform files are written in YAML and contains 3 sections namely: 1) *general* 2) *nodes* 3) *interfaces*. The *general* section is optional but all the other sections must be present. Here is an example of a simple platform file to simulate 2 wireless nodes:

assets/platform.yaml

```yaml
general:
    interferences: on # Turns on interferences

nodes:
    count: 2
    implementations:
        - all node.py
    arguments:
      0: "sender"
      1: "receiver"

interfaces:
```

```
    wlan0:
        type: "wireless"
        nodes: all # All nodes are connected to this network
        links:
            - all 50kbps 0s all # All nodes are reachable by each other
        txperfs:
            - all 50kbps 0s
```

2.2 Node implementation file

Nodes implementations are written using python. Here is the implementation of the node mentioned in the last `platform.yaml` file:

assets/node.py

```python
def execute(api):
    role=api.args # "sender" or "receiver" cf. platform.yaml
    if role == "sender":
        api.send("wlan0","MY MESSAGE",10,None)
    else:
        api.receive("wlan0")
```

More details about implementations of nodes are availablein the *example code* and the *ESDS implementation of the SNs interface*.

2.3 Execution

To run our first simulation, the following command can be run: that contains `platform.yaml` and `node.py`:

```
> esds run platform.yaml
```

Here is the output of the simulation:

```
[t=0.000,src=n0,grp=def] Send 10 bytes on wlan0
[t=0.002,src=n1,grp=def] Receive 10 bytes on wlan0
[t=0.002,src=esds] Simulation ends
```

In this case, simulation tooks $0.002s$ and 10 bytes were sent on the wlan0 interface from node 0 (src=n0) to node 1 (src=n1).

2.4 Custom orchestrator instantiation

Instead of using a `platform.yaml` file, it is possible to instantiate manually the esds orchestrator. To do so, you need to implement that procedure in a python file. Here is and example that performs the exact same simulation presented in Section 2.3 but with a custom instantiation of the orchestrator:

assets/orchestrator.py

```python
#!/usr/bin/env python

import esds
import numpy as np

n=2 # 2 nodes
B=np.full((n,n),50*1000) # Bandwith+txperfs 5bps
L=np.full((n,n),0) # Latency 0s

s=esds.Simulator({"wlan0":{"bandwidth":B, "latency":L, "is_wired":False}})

##### Instantiate nodes with their implementation
s.create_node("node",interfaces=["wlan0"],args="sender") # Use node.py for the first node, specify the
↪  vailable communication interfaces and use "sender" as argument
s.create_node("node",interfaces=["wlan0"],args="receiver") # Now the second node

##### Run the simulation
s.run(interferences=True)
```

Next we can run the simulation:

```
> ./orchestrator.py
```

3 Platform file

As explain in Section 2.1, esds platform files comprise 3 sections:

1. **general:** to settings up esds
2. **nodes:** to configure the simulated nodes
3. **interfaces:** to create network the interfaces available for each nodes

Lets see in details the format of each section.

3.1 The general section

This section is used to settings up the overall parameters of esds. Table 1 reference all the keywords for this section of the platform file.

Keyword	Description	Example
interferences	Turn on/off interferences detection during wireless communications.	`interferences: on`
debug	Turn on/off esds debugging mode (generate a debug file).	`debug: on`
debug_file	Specify the file to use as output for the debugging.	`debug_file: "./myfile.txt"`
breakpoints	Specify a list of simulated time (in seconds) at which esds must interrupt and call the specified callback.	`breakpoints: [5, 6, 7]`
breakpoints_every	Specify an interval of time (in seconds) at which esds will interrupt and call the specified callback.	`breakpoints_every: 5`

breakpoints_callback	Tell esds where how to reach the callback used during breakpoints.	`breakpoints_callback:`
		`  file: "platform_callback.py"`
		`  callback: "callback"`

Table 1: Usable keywords in the general section of a esds platform file.

3.1.1 Notes

The **interferences** keywords allows to control interference detection. It works as follow. If at any point in time during a wireless communication, another communication happens, on the same wireless interface, in the range of the receiver, then the interference return code will be given to the receiver upon reception.

3.2 The node section

The node section is used configure the simulated node of esds. Table 2 references all the keywords used in the nodes section.

Keyword	Description	Example
count	Number of simulated nodes.	`nodes: 5`

implementations	Bind each node to their respective implementation (uses **the range syntax**).	`implementations:`
		`  - 0 sender.py`
		`  - 1-@ receiver.py`

groups	Specify the group to use in the nodes log reports. Useful to filter logs. By default, nodes are in the *def* group.	`groups:`
		`  - 0 groupA`
		`  - 1-@ groupB`

arguments	Define the arguments that will be passed to each node implementation (keys of each element uses **the range syntax**).	`arguments:` `  all: 2`

Table 2: Usable keywords in the nodes section of a esds platform file.

Several entries in the platform file use a **range syntax** to bind data (node implementations, links etc.) to node *ids*. For a simulation with p nodes, each node will have an allocated id such that $id \in [0, 1, ..., p-1]$. Here are examples of valid range syntax for a simulation that uses 5 nodes:

- **0,1,2,3** Node 0,1,2 and 3
- **0-2** Node 0,1 and 2
- **all** Node 0,1,2,3 and 4
- **2-@** Node 2,3 and 4
- **0-@** Node 0,1,2,3 and 4 (same as **all**)

3.3 The interfaces section

The interfaces section allows to define the network interfaces that will be usable by each node during the simulation. For each interface the Table 3 references the available keywords.

Keyword	Description	Example
type	Interface is "wireless" or "wired".	`type: "wireless"`
nodes	Specify the nodes that have access the current network/interface (uses the **range syntax**)	`nodes: 2-@`

links	List all the links between nodes on the interface (uses the **range syntax**).	`links:`
		`  # Link node 0 to node 1:`
		`  - 0 10Mbps 0s 1`
		`  # Link node 1 and 2 to node 3 and 4:`
		`  - 1,2 1 Mbps 5s 3,4`

txprefs	Define the transmission performance of each wireless node (keys of each element uses **the range syntax**). This keyword is only available for wireless interfaces.	`txperfs:`
		`  - 0 10kbps 0s`
		`  - 1 10kbps 5s`

Table 3: Usable keywords for each interface in the interfaces section of a esds platform file.

In esds, txperfs (or transmission performance) corresponds to the transmission performance on the wireless interface. It is used to compute the transmission duration of wireless communications. When using custom orchestrator instantiation, the txperfs parameter of a given node can be assigned by fixing the diagonal components of the bandwidth and latency matrices.