ESDS: Extensible Simulator for Distributed Systems

Loic GUEGAN

August 31, 2022

1 Simulation Architecture

The ESDS simulator comprises two major components: 1) The Simulation Orchestrator(SO) 2) The Simulated Nodes (SN). This architecture is depicted in Figure 1.

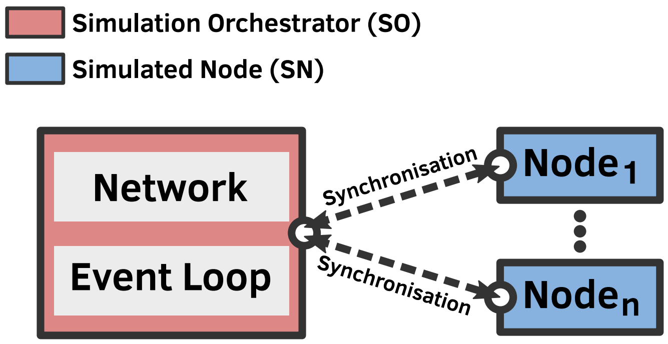

Figure 1: Architecture of ESDS

The SO is the main process in charge of implementing the simulation main loop. It instantiates the network (e.g bandwidths andlatencies), collects and processes the events (e.g communications,turn on/off). The nodes on the other hand are threads that simulate the nodes behaviors.

2 Example

To run a simulation, you need to provide at least 2 files. The first one instantiate the orchestrator and the second one will simulate the node. In this section, you will learn how to write both files.

The simulated scenario comprises 2 nodes that wakes up randomly every hour for a duration called "uptime". The sender try to transmit his data during that uptime. The other node is a receiver that have similar random wake up parterns and strive to receive data from the sender.

2.1 Orchestrator

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

import esds # Load ESDS
import numpy as np # Use numpy to construct bandwidth and latencies matrix
```

Table 1: Simulated Nodes blocking and non-blocking API calls

Call	Blocking	Description
`send(<int>,<data>,<size>,<dst>,<rdst>)!`	yes	Send `<data>! of size <size>! on interface <int>!`
`sendt(<int>,<data>,<size>,<dst>,<t>,<rdst>)!`	yes	Send `<data>! of size <size>! on interface <int>! with a timeout of <t>!`
`receive(<int>)!`	yes	Wait for and fetch incoming data on interface `<int>!`
`receivet(<int>,<t>)!`	yes	Wait for and fetch incoming data on interface `<int>! with a timeout of <t>!`
`wait(<t>)!`	yes	Wait for a specific amount of simulated time `<t>!`
wait$_e nd()$!	yes	Wait until the end of the simulation
`log(<message>)!`	no	Report `<message>! to the SO that will print it to the standard output`
`read(<register>)!`	no	Read in the SO registers (e.g *clock* to get the current simulated time)
turn$_o ff()/turn_o n()$!	no	Change the node state to "*off*" or "*on*" respectively

```python
##### Bandwidth matrix
# Bandwidth value can be 0 for unreachable nodes
# Regarding wireless interfaces the diagonals of the bandwidth and latency matrices are
↪  very important.
# They determine the duration of the tranmission for THE SENDER. It allows to have a
↪  different tx
# duration per node and per interface. Please cf esds.py for more informations.
n=2 # Number of nodes including the sender
B=np.full((n,n),5) # 5Mbps

##### Latency matrix
# If the latency entries match one with a bandwidth of 0
# then it will be ignore since node is unreachable.
L=np.full((n,n),0) # 0s

##### Create the simulator
# esds.Simulator take at least a dictionnary as a parameter
# This dictionnary contains all the network interfaces (name as a key) of each node
s=esds.Simulator({"wlan0":{"bandwidth":B, "latency":L,
↪  "is_wired":False},"eth0":{"bandwidth":B, "latency":L, "is_wired":True}})

##### Instantiate nodes
uptime=180 # 180s uptime
s.create_node("sender",args=uptime) # Load sender.py for the first node with 5 as
↪  argument (first row in B and L)

# Aguments can be passed to nodes via: s.create_node("sender",args="my argument")
for n in range(0,n-1): # Load receiver.py for the remaining nodes
    s.create_node("receiver",args=uptime)

##### Run the simulation
s.run()
```

2.2 Nodes

To implement a node, you should create a python file with the method execute(api). This method will be called by the orchestrator to execute the code of your node. The api parameter provide you access to the following esds API:

2.2.1 Sender

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

import random

# Note that the following is required to have different instance from thread to thread
lr=random.Random(6)

def execute(api):
    uptime=api.args
    endoff=0
    for i in range(0,24):
        startoff=random.randint(0,3600-uptime)
        api.turn_off()
        api.wait(startoff+endoff)
        api.turn_on()
        wakeat=api.read("clock")
        wakeuntil=wakeat+uptime
        # Send until uptime seconds if elapsed
        while api.read("clock") < wakeuntil:
            api.sendt("wlan0","hello",10,None, wakeuntil-api.read("clock"))
        api.log("Was up for {}s".format(api.read("clock")-wakeat))
        endoff=3600*(i+1)-api.read("clock")
    api.turn_off()
    api.wait(endoff)
    api.turn_on()
```

2.2.2 Receiver

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

import sys, random, time

lr=random.Random(6)

def execute(api):
    uptime=api.args
    endoff=0
    for i in range(0,24):
        startoff=random.randint(0,3600-uptime)
        api.turn_off()
        api.wait(startoff+endoff)
        api.turn_on()
        wakeat=api.read("clock")
        wakeuntil=wakeat+uptime
        # Receive until uptime seconds if elapsed
        while api.read("clock") < wakeuntil:
            code, data=api.receivet("wlan0",wakeuntil-api.read("clock"))
            if code == 0:
                api.log("Receive "+data)
        api.log("Was up for {}s".format(api.read("clock")-wakeat))
        endoff=3600*(i+1)-api.read("clock")
    api.turn_off()
    api.wait(endoff)
    api.turn_on()
```

2.3 Simulation Output

Here is part of the simulation output:

```
[t=82626.000,src=n0] Send 10 bytes on wlan0
[t=82630.000,src=n0] Was up for 180.0s
[t=82630.000,src=n0] Turned off
[t=83083.000,src=n1] Turned on
[t=83263.000,src=n1] Was up for 180.0s
[t=83263.000,src=n1] Turned off
[t=85910.000,src=n0] Turned on
[t=85910.000,src=n0] Send 10 bytes on wlan0
[t=85926.000,src=n0] Send 10 bytes on wlan0
[t=85942.000,src=n0] Send 10 bytes on wlan0
[t=85958.000,src=n0] Send 10 bytes on wlan0
[t=85974.000,src=n0] Send 10 bytes on wlan0
[t=85990.000,src=n0] Send 10 bytes on wlan0
[t=86006.000,src=n0] Send 10 bytes on wlan0
[t=86022.000,src=n0] Send 10 bytes on wlan0
[t=86038.000,src=n0] Send 10 bytes on wlan0
[t=86054.000,src=n0] Send 10 bytes on wlan0
[t=86070.000,src=n0] Send 10 bytes on wlan0
[t=86086.000,src=n0] Send 10 bytes on wlan0
[t=86090.000,src=n0] Was up for 180.0s
[t=86090.000,src=n0] Turned off
[t=86400.000,src=n0] Turned on
[t=86400.000,src=n1] Turned on
[t=86400.000,src=esds] Simulation ends
```

Brackets indicate additional informations related to the message (e.g source and simulated time). All the send and received events are reported automatically by esds.