# Wishful/IBFD Tutorial

# Initialization

The following example shows how the experimenter can create IBFD UPI and initiate it. The parameters below must be set prior to the experiment and before **start** command: Frequency, Transmitter Shift Frequency, Tx Gain, Rx Gain, Tx Sample Rate and Device Address. The UPI instructor initiates this parameters with the default values.

from IBFD\_agent import \*

Agent = IBFDAgent(ip\_address='10.33.136.138', ip\_port = 5022)

Agent.start()

# Write the experiment code here

Agent.stop()

# Start/Stop USRP

After initialization, **start** command runs the USPR and the parameters blow are still adjustable: Destination Address, Payload, collision detection average length, collision detection threshold and MAC settings. To terminate the experiment, the USRP should be stopped by **stop** command.

# Retuning duplexer

Prior to the experiment, the analog self-interference canceller should be tuned. This is needed only once after **start** command and can be done as following:

Agent.start()

Agent.si\_enable()

time.sleep(1)

Agent.retune\_duplexer(True)

time.sleep(2)

Agent.retune\_duplexer(False)

# Train the collision detector

The transmitting signal can be provided form either the host or the Microblaze. Due to the backoff time in CSMA/CD deployed on the Microblaze, for training the collision detector, the host should provide the transmitting signal. The following commands shows how one can train the collision detector and read the percent of the collision. The collision detection makes use of an average block. The averaging length and the detection threshold are adjustable through **set\_cd\_avg\_length** and **set\_cd\_threshold** commands.

Agent.start()

Agent.set\_cd\_avg\_length(8)

Agent.set\_cd\_threshold(6)

Agent.set\_mac(1) # set host as the source of transmitting data

Agent.si\_enable() # start transition

time.sleep(1)

Agent.set\_collision\_detection('reset') # reset the collision detector

time.sleep(2)

Agent.set\_collision\_detection('train') # train the detector

time.sleep(3)

Agent.set\_collision\_detection('fix\_alpha') # fix the detector alpha parameter

time.sleep(1)

Agent.set\_collision\_detection('start') # start detection

Agent.set\_mac(0) # set Microblaze as the source of transmitting data

time.sleep(3)

CD\_res = Agent.req\_cd\_result() # reading the detected value in percent

print(CD\_res)

# MAC setting

The MAC works in two modes; CSMA and CSMA/CD. To enable collision detection in the MAC protocol, the CD training phase should be down primitively as it is explained above. The CSMA/CD protocol is applicable only when the Microblaze is the source of the data, i.e. **Agent.set\_mac(2).**

# Setting host payload

The host payload is adjustable through the following commands. The experimenter can set an array of bytes (at most 118 elements). The following sample code shows how one can set the host payload.

Note: The Microblaze utilizes a ramp of size 118 as its payload.

Data = bytearray([0,1,2,3,4])

Agent.send\_payload(Data)

# Data request

After initialization and calling the start command, the experimenter can send request for IQ data or the received packet via **req\_iq\_data()** and **req\_recv\_pkt()** commands.

# Example

The Python code below, establishes a CSMA/CD protocol experiment with 2 terminals.

import time

from IBFD\_agent import \*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Initialization

Agent1 = IBFDAgent(ip\_address='10.33.136.138', ip\_port = 5022)

Agent2 = IBFDAgent(ip\_address='10.33.136.139', ip\_port = 5022)

Agent1.device\_address(0)

Agent2.device\_address(1)

Agent1.dest\_address(1)

Agent2.dest\_address(0)

Agent1.set\_tx\_power(5)

Agent2.set\_tx\_power(5)

Agent1.set\_rx\_gain(0)

Agent2.set\_rx\_gain(0)

Agent1.set\_cd\_avg\_length(8)

Agent1.set\_cd\_threshold(4)

Agent2.set\_cd\_avg\_length(8)

Agent2.set\_cd\_threshold(4)

Agent1.set\_collision\_detection('reset')

Agent2.set\_collision\_detection('reset')

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Start USRPs

Agent1.start()

Agent2.start()

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Retuning Duplexer

Agent1.si\_enable()

Agent1.retune\_duplexer(True)

time.sleep(3)

Agent1.retune\_duplexer(False)

Agent1.si\_disable()

Agent2.si\_enable()

Agent2.retune\_duplexer(True)

time.sleep(3)

Agent2.retune\_duplexer(False)

Agent2.si\_disable()

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Train the collision detectors

Agent1.si\_enable()

Agent1.set\_mac(1)

time.sleep(1)

Agent1.set\_collision\_detection('reset')

time.sleep(2)

print('... Training')

Agent1.set\_collision\_detection('train')

time.sleep(3)

print('... Fix Alpha')

Agent1.set\_collision\_detection('fix\_alpha')

time.sleep(2)

Agent1.set\_collision\_detection('start')

Agent1.set\_mac(0)

Agent1.si\_disable()

Agent2.si\_enable()

Agent2.set\_mac(1)

time.sleep(1)

Agent2.set\_collision\_detection('reset')

time.sleep(2)

print('... Training')

Agent2.set\_collision\_detection('train')

time.sleep(3)

print('... Fix Alpha')

Agent2.set\_collision\_detection('fix\_alpha')

time.sleep(2)

Agent2.set\_collision\_detection('start')

Agent2.set\_mac(0)

Agent2.si\_disable()

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Experiment: data exchange with CSMA/CD

Agent1.set\_mac(2) #enable CSMA/CD

Agent2.set\_mac(2) #enable CSMA/CD

Agent1.si\_enable()

Agent2.si\_enable()

Data1 = Agent1.req\_iq\_data()

print(Data1[‘IQ-COMPLEX’]) #print captured data

Data2 = Agent2.req\_recv\_pkt()

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Terminate the experiment

Agent1.si\_disable()

Agent2.si\_disable()

Agent1.stop ()

Agent2.stop ()

# Using Wishful UPIs

In order to use wishful UPIs you need Python 3.5 or higher running on a unix system and a system gcc installation.

On your system you also need to install the following packages:

* Wishful\_upi
* Wishful\_framework
* Wishful\_controller
* Wishful\_agent
* Module\_discovery\_pyre

You can obtain those packages directly from the wishful github repositories:

<https://github.com/wishful-project>

Once you clone the repositories, generate a virtual environment:

Python –m venv *foldername*

Where foldername is the folder of your virtual environment.

For example:

#python –m venv wishful\_python

Creates a folder called wishful\_python containing a python installation independent from the global system one. The python binaries are in the subfolder “bin”.

All the repositories contain a setup.py file. It is important to run it to get all the dependencies.

The main dependencies are:

* Gevent
* pyYaml
* google
* zmq
* docopt

The setup.py script shall be launched as:

#: ~/wishful\_python/bin/python3 setup.py install

Do this for all the repositories you cloned.

If the installation of some dependencies does not work you can use:

#: ~/wishful\_python/bin/pip install *missing\_dependency*

Once everything is installed and setup it’s time to write the controller to run your experiment. You also need to write a configuration file in yaml language.

An example controller looks like the following:

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Usrp\_controller: Wishful controller for the USRP setup in ku leuven

Usage:

usrp\_controller [options] [-q | -v]

Options:

--logfile name Name of the logfile

--config configFile Config file path

Example:

./ usrp\_controller -v --config ./config.yaml

Other options:

-h, --help show this help message and exit

-q, --quiet print less text

-v, --verbose print more text

--version show version and exit

"""

import sys

import datetime

import logging

import wishful\_controller

import gevent

import yaml

import wishful\_upis as upis

\_\_author\_\_ = "Franco Minucci, Seyed Ali Hassani"

\_\_copyright\_\_ = "Copyright (c) 2017, KU Leuven"

\_\_version\_\_ = "0.1.0"

\_\_email\_\_ = "franco.minucci@esat.kuleuven.be"

log = logging.getLogger('wishful\_agent.main')

controller = wishful\_controller.Controller()

nodes = []

#callback to add the node received from the agent

@controller.new\_node\_callback()

def new\_node(node):

nodes.append(node)

print("New node appeared:")

print(node)

#function called when the execution of the node ends

@controller.node\_exit\_callback()

def node\_exit(node, reason):

if node in nodes:

nodes.remove(node);

print("NodeExit : NodeID : {} Reason : {}".format(node.id, reason))

@controller.set\_default\_callback()

def default\_callback(group, node, cmd, data):

print("{} DEFAULT CALLBACK : Group: {}, NodeName: {}, Cmd: {}, Returns: {}".format(datetime.datetime.now(), group, node.name, cmd, data))

def print\_response(group, node, data):

print("{} Print response : Group:{}, NodeIP:{}, Result:{}".format(datetime.datetime.now(), group, node.ip, data))

def main(args):

log.debug(args)

config\_file\_path = args['--config']

config = None

with open(config\_file\_path, 'r') as f:

config = yaml.load(f)

#Used to load configuration from the yaml file

controller.load\_config(config)

controller.start()

gevent.sleep(10)

print("\n")

print("Connected nodes", [str(node.name) for node in nodes])

#set usrp parameters

controller.blocking(False).node(nodes[0]).radio.iface("usrp0").set\_parameters({'DEV\_ADDRESS':10,'TX\_POWER':9,'RX\_GAIN':1,'SET\_CD':'reset'})

gevent.sleep(2)

controller.blocking(False).node(nodes[0]).radio.iface("usrp0").set\_parameters({'DEST\_ADDRESS':11,'CD\_AVG\_LEN':15,'CD\_THRESHOLD':10,'SI\_ENABLE':True,'RETUNE\_DUPLEXER':True})

gevent.sleep(4)

#now the duplexer is tuned

controller.blocking(False).node(nodes[0]).radio.iface("usrp0").set\_parameters({'RETUNE\_DUPLEXER':False})

print('Mac Setting for CD training should be 1')

controller.blocking(False).node(nodes[0]).radio.iface("usrp0").set\_parameters({'SET\_MAC':1})

gevent.sleep(4)

print('Train CD')

print('... Training')

controller.blocking(False).node(nodes[0]).radio.iface("usrp0").set\_parameters({'SET\_CD':'train'})

gevent.sleep(4)

print('... Fix Alpha')

controller.blocking(False).node(nodes[0]).radio.iface("usrp0").set\_parameters({'SET\_CD':'fix\_alpha'})

gevent.sleep(2)

print('... Start CD')

controller.blocking(False).node(nodes[0]).radio.iface("usrp0").set\_parameters({'SET\_CD':'start'})

gevent.sleep(3)

controller.blocking(False).node(nodes[0]).radio.iface("usrp0").set\_parameters({'SET\_MAC':1})

#control loop that requests parameters and measurements periodically

while True:

gevent.sleep(5)

if nodes:

#execute non-blocking function immediately

#request parameters

gevent.sleep(2)

controller.blocking(False).node(nodes[0]).radio.iface("usrp0").get\_parameters(('RX\_GAIN','DEV\_ADDRESS','TX\_POWER','SET\_CD'))

#request measurements

gevent.sleep(2)

controller.blocking(False).node(nodes[0]).radio.iface("usrp0").get\_measurements(('CD\_RESULT','IQ\_DATA'))

if \_\_name\_\_ == "\_\_main\_\_":

try:

from docopt import docopt

except:

print("""

Please install docopt using:

pip install docopt==0.6.1

For more refer to:

https://github.com/docopt/docopt

""")

raise

args = docopt(\_\_doc\_\_, version=\_\_version\_\_)

log\_level = logging.INFO # default

if args['--verbose']:

log\_level = logging.DEBUG

elif args['--quiet']:

log\_level = logging.ERROR

logfile = None

if args['--logfile']:

logfile = args['--logfile']

logging.basicConfig(filename=logfile, level=log\_level,

format='%(asctime)s - %(name)s.%(funcName)s() - %(levelname)s - %(message)s')

try:

main(args)

except KeyboardInterrupt:

log.debug("Controller exits")

finally:

log.debug("Exit")

controller.stop()

This code must be coupled with the corresponding configuration file written in YAML:

## WiSHFUL Controller's config file

controller:

name: "Controller"

info: "WiSHFUL Controller"

dl: "tcp://127.0.0.1:8990" ##address and port used for downlink

ul: "tcp://127.0.0.1:8989" ##address and port used for uplink

modules:

discovery:

module : wishful\_module\_discovery\_pyre

class\_name : PyreDiscoveryControllerModule

##iface parameter must be the name of the network interface on which the communication with

##the agent happens. In this case is set to local loop because agent and controller where on the same machine

kwargs: {"iface":"lo", "groupName":"wishful\_1234", "downlink":"tcp://127.0.0.1:8990", "uplink":"tcp://127.0.0.1:8989"}