pyPMU – Open Source Python Package for Synchrophasor Data Transfer

Stevan Šandi, Božo Krstajić, *Member, IEEE*, and Tomo Popović, *Senior Member, IEEE*

***Abstract*—pyPMU is an open source Python package library that implements the IEEE C37.118 standard protocol for synchrophasor data transfer. Synchrophasor measurements provide a precise and time-synchronized phasor measurements collected throughout power grid. The pyPMU library offers various functions for supporting synchrophasor applications and it enables rapid synchrophasor test bed implementations. The paper provides detailed overview of the pyPMU package, its installation, and usage examples. The most interesting use scenarios include PMU simulator and PMU data stream splitter, which have been described in the paper. The discussion illustrates initial benchmarking results and current issues. Finally, the paper outlines future work on the improvements and practical applications. The pyPMU project is published under 3-clause BSD license and made available at GitHub.**

***Keywords*—Synchrophasor; Phasor Measurement Unit; Phasor Data Concentrator; Stream Splitter; IEEE C37.118.**

I. Introduction

Synchrophasors represent a precise and time-synchronized phasor measurements collected throughout power grid. Satellite synchronized clocks are used to enable time-stamped measurements, which make synchrophasor data aligned to the reference time base. Synchrophasor measurements provide an accurate, dynamic and comprehensive view of a power system [1]. Synchrophasor support tools play an important role in the development of Wide Area Monitoring, Protection, and Control (WAMPAC) solutions, because they enable validation of Phasor Measurement Units (PMUs), Phasor Data Concentrators (PDCs), communication network and resources, phasor estimation algorithms and end-to-end tests [2][3][4][5]. The review of existing synchrophasor support tools revealed opportunities for the development of IEEE C37.118 support tools suite independent of the platform and operating system [6][7][8]. Furthermore, such tools are necessary to support synchrophasor application development and their evaluation [9]. An example is PMU simulator capable of sending realistic synchrophasor data streams, which may include phasor data corresponding to various test scenarios such as pre-defined measurement, power system faults, synchrophasor commands, etc. As for network testing, it is important to evaluate latencies, errors, and throughput during the transmission of synchrophasor data. The evaluation tools need to be simple to use and install with minimal dependency on third party libraries which sometimes might have problems with installation. The *pyPMU* is made to meet requirements mentioned above [10]. Python programming language, which is extremely popular and easy for scripting tasks, will ensure cross-platform compatibility. The well-documented library should enable simple application or test bed development.

The pyPMU library should represent solid ground for building complete suite for synchrophasor tools for simulation, communication and evaluation tools. The project aims are:

- To provide high quality, well-documented and simple-to-use implementation of IEEE C37.118 synchrophasor communication protocol: the library is providing building elements needed to implement various test scenarios, scalable test beds and evaluation tools for synchrophasor applications. The library, in the form of a Python package, can be used to support the development of synchrophasor applications as well.
- To facilitate research and education in synchrophasors: *pyPMU* allows researchers and students in power systems to learn more about synchrophasor in general. It enables researchers to gain hands-on experience with synchrophasor networking, testing, and applications. Researchers can further adjust and develop the source code, which allows them to learn about synchrophasors and software engineering.
- To address industry challenges: if further developed, the *pyPMU* software package may provide the industry with a set of accessible and reliable synchrophasor tools, without having to invest significant energy into development or utilization of expensive commercial packages. Companies may find it useful for development of supporting script and evaluation tools.

The paper is organized as following: after the Introduction section, Section 2 provides the project overview, which discusses the structure of the Python package. In Section 3, the use of the library is illustrated with implementations of a PMU simulator and PMU data stream splitter. The Discussion section covers known issues and the current roadmap for the project. Conclusions and references are given at the end.

Stevan Šandi and Božo Krstajić are with Faculty of Electrical Engineering, University of Montenegro, Džordža Vašingtona bb, 81000 Podgorica, Montenegro (e-mails: stevan.sandi@gmail.com, bozok@ac.me).

Tomo Popović is with University of Donja Gorica, Donja Gorica bb, 81000 Podgorica, Montenegro (e-mail: tomo.popovic@udg.edu.me).

978-1-5090-4086-5/16/$31.00 ©2016 IEEE

II. The Project Overview

Project pyPMU became available to the public in March of 2016 in a form of GitHub repository. The main idea behind pyPMU project is to offer various set of applications and libraries for supporting the measurement of synchrophasors. The software package is licensed under 3-clause BSD (BSD-3) license. Publishing the pyPMU software package as open-source project should enable inclusion of additional individual and institutional contributors.

The pyPMU package source-code is well-documented and Python Enchantment Proposals (PEP) compliant. Cross platform compatibility and minimal third-party library dependency principals ensure easy installation and usage. Project structure is simple and it should be kept for the future updates. Brief description of the project structure shown in Fig. 1 is given below.

```
schandi@argo:~/PycharmProjects$ tree pyPMU/
pyPMU/
├── apps
│   ├── pmy.py
│   └── splytter.py
├── docs
│   ├── frame.html
│   └── pycco.css
├── examples
│   ├── pyPMU.py
│   ├── randomPMU.py
│   ├── streamSplitter.py
│   ├── tinyPDC.py
│   └── tinyPMU.py
├── LICENSE.txt
├── README.md
├── synchrophasor
│   ├── frame.py
│   ├── __init__.py
│   ├── pdc.py
│   ├── pmu.py
│   ├── splitter.py
│   └── utils.py
├── tests
│   ├── benchPDC.py
│   └── validate_frames.py
└── TODO.md

5 directories, 20 files
```

Figure 1. The pyPMU package project structure.

Each file inside **`synchrophasor`** folder is single Python module. Module **`frame`** represents the implementation of the IEEE C37.118 synchrophasor communication protocol [11]. Modules **`pmu`**, **`pdc`** and **`splitter`** utilize **`frame`** module to implement required functionalities. Inside the **`app`** folder, ready to use scripts are provided and the folder **`examples`** contains a set of useful snippets, which explain appropriate usage of modules available in package. The folder **`tests`** is meant to be a folder for various test and validation scripts.

III. Usage Examples

In this section, snippets for creating PMU, PDC and data-stream splitter are illustrated. The complete source code with comments can be found inside the **`examples`** folder.

A. PMU simulator

A simple PMU simulator that sends constant values can be found inside **`pmu`** module. The code snippet in Fig. 2. illustrates how to create a PMU simulator using **`pmu`** module.

```
from synchrophasor.pmu import Pmu

pmu = Pmu(ip='127.0.0.1', port=1410)
pmu.set_configuration()
pmu.set_header()
pmu.run()

while True:
    if pmu.clients:
        pmu.send(pmu.ieee_data_sample)

pmu.join()
```

Figure 2. Creating a PMU simulator using the **`pmu`** module.

If no argument is provided for methods **`set_configuration()`** and **`set_header()`**, the default configuration and header message will be loaded defined in module (Fig. 2).

To validate that it is really working, a third-party tool called PMU Connection Tester is used to preview the received measurements in real-time [12]. Figures 3 and 4 show that the PMU Connection Tester correctly understood messages sent by PMU simulator and successfully displayed their content.

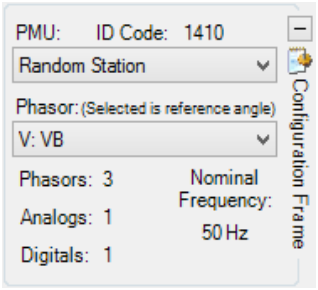

Figure 3. Validating the configuration message.

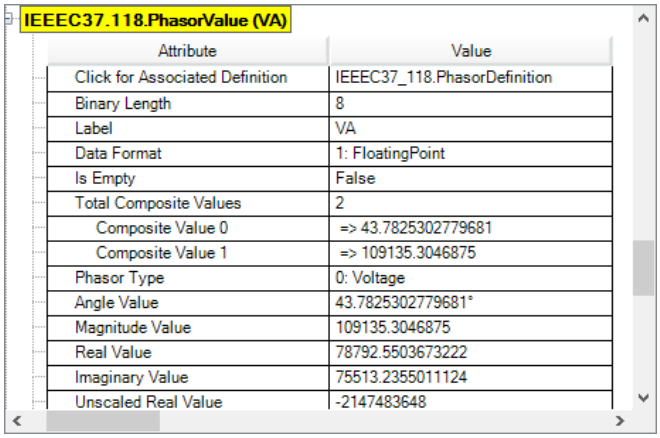

Figure 4. Validating measurements sent by the PMU simulator.

B. Phasor Data Concentrator

Using the **`pdc`** module, a simple PDC can be created. Although it still has limited functionalities, this PDC will be able to understand received messages and report if CRC of newly arrived message fails. Current implementation does not have a feature to aggregate multiple data streams into one and send it to other destinations, which is left for future

developments. The code shown in Fig. 5 creates a PDC that acts as a client and receives measurements.

```
from synchrophasor.pdc import Pdc

pdc = Pdc(pdc_id=9991,
          pmu_ip='127.0.0.1',
          pmu_port=1410)

pdc.run()
header = pdc.get_header()
config = pdc.get_config()
pdc.start() # Request to start sending data

while True:
    data = pdc.get()
    if not data:
        pdc.quit()
        break
```

Figure 5. Creating PDC using **pdc** module.

C. *Data-stream splitter*

One of the most interesting practical uses of the pyPMU at the moment is implementing a synchrophasor data splitter (Fig. 6), which is available in the **splitter** module.

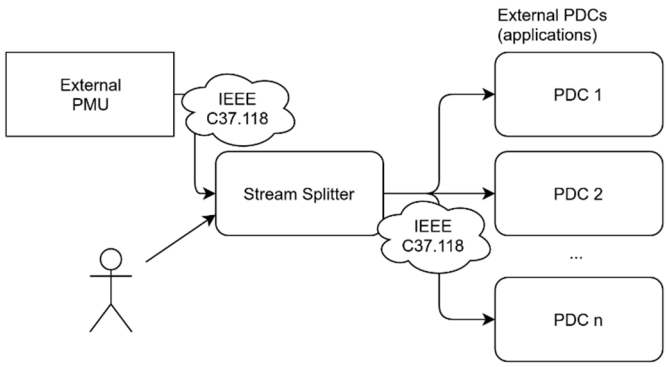

Figure 6. Practical use case: data-stream splitter.

Using the **splitter** module with just four lines of code a data stream splitter will be up and running, ready to pass data frames received from **source_ip** and **source_port** address to any PDC client accepted through a listener on **listener_ip** and **listener_port** (see Fig. 7).

```
from synchrophasor.splitter \
import StreamSplitter

stream_splitter =
StreamSplitter(source_ip='127.0.0.1',
               source_port='1410',
               listen_ip='127.0.0.1',
               listen_port='1502')

stream_splitter.run()
stream_splitter.join()
```

Figure 7. Creating data-stream splitter using **splitter** module.

A sample of real-time debug output for the data stream splitter with two connected PDCs is shown in Fig. 8. The first line illustrates how to run the splitter script. The source IP and port are set to 127.0.0.1 and 1410 and these parameters identify the source PMU. The listener IP and port are set to 127.0.0.1 and 1411 and these parameters tell the splitter to simulate a PMU on that address and port. In the debug log in Fig. 8, it can be seen that the first PDC has established communication with the splitter and started receiving the data. Please note that the IP addresses for all devices in the example are 127.0.0.1 since this was a test simulation test on a single PC. For the practical purposes, it is recommended to disable the debug output feature in order to achieve a significant performance improvement in terms of data transmission latency.

IV. Discussion

In its current version, the pyPMU can be useful for researchers to quickly implement Python scripts to create and manage synchrophasor data streams. One particularly interesting usage of the library is the simple creation of PMU simulator instances, which can play a major role in creation various synchrophasor evaluation testbeds. Another interesting application of the module is implementing PMU data stream splitters, which may be useful both for testbeds and future industry use. The library is expected to continue growing and gain additional functionalities. The initial benchmarking results showed that choosing multiprocessing over multithreading was a good decision since the latency in data transmission is constant even for bigger reporting rates and multiple connected clients (i.e. PDC simulators). There is a multiprocessing issue on the Windows platforms caused by missing **fork()** feature [13]. Once this issue is resolved, the pyPMU project will ensure promised cross-platform compatibility.

The pyPMU package has the potential to fill in the gap in the domain of synchrophasor related tools, especially for research and education purposes. The rising popularity of Python as a programming language and open platform, together with a large availability of scientific tools, gives the pyPMU edge in the domain of creation of synchrophasor test beds. For example, using the pyPMU library, one can implement synchrophasor stream splitters to interconnect between various synchrophasor applications. The implementation of the communication protocol can be extended to implement protocol mappers, various visualizations, real-time synchrophasor data analytics, evaluation tools, etc. Making the project open source made it possible for additional researchers and developers to use it, test it, provide constructive feedback, as well as to contribute to it and expand it.

There are a few crucial functionalities that are missing in the current release. First of all, User Datagram Protocol (UDP) transmission method for synchrophasors is not yet supported, but it is the main topic on the development team's roadmap. Another important issue is that the current version only supports Configuration Frame version 2 from the standard. Version 1 and 3 need to be added and the class **ConfigFrame** needs to be refactored and to be used as an interface between all three versions. This will lead to

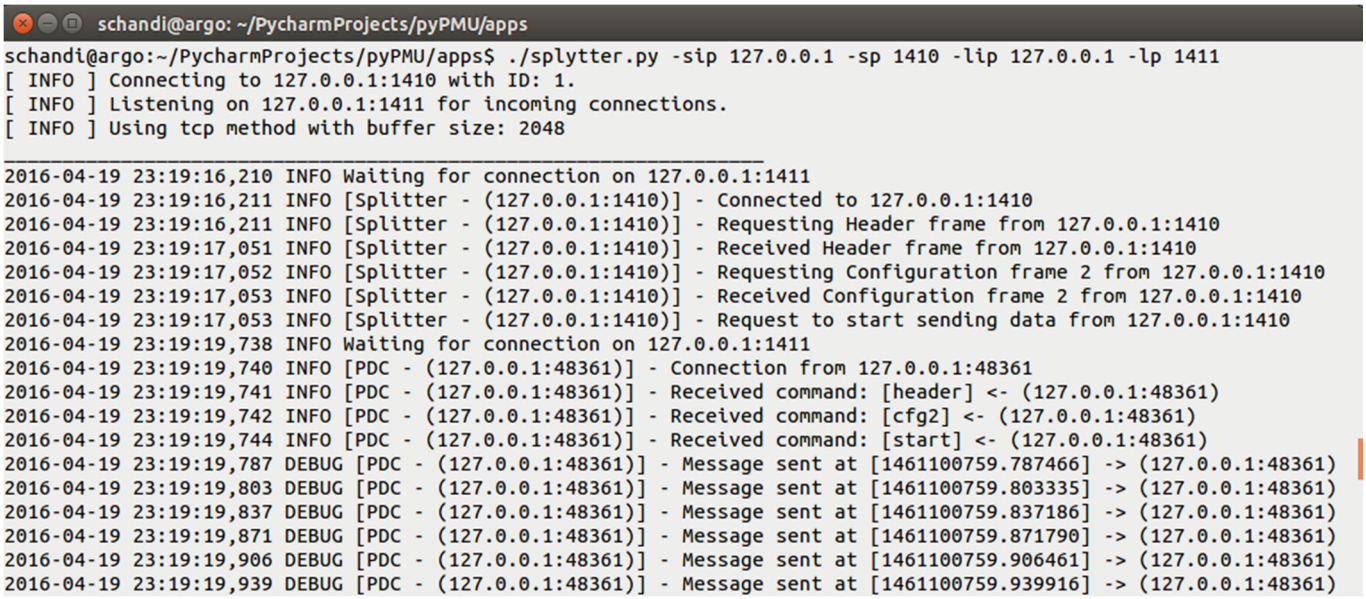

Figure 8. Data-stream splitter application real-time debug output.

additional simplification of the **`pmu`**, **`pdc`** and **`splitter`** module. At the moment, the PDC acts as client only, not capable for data streaming toward other PDCs (also known as super PDCs). Future development targets additional features such as aggregating data streams and interfacing synchrophasor database, which will enable implementations of a fully functional PDCs. Future releases may also benefit from methods for measurement extraction and simple manipulations, capturing the data streams for replaying, storing the measurements into a database, as well as from GUIs for example applications [14].

For the future work, the library will be expanded to include timing functionalities and synchrophasor algorithms in order to grow into a full blown suite of synchrophasor tools.

V. Conclusion

This paper discusses an open source Python package called pyPMU. The pyPMU package implements IEEE C37.118 synchrophasor communication protocol and it can be used in various research, development, and educational scenarios.

The main contributions of the presented research are:

- a Python library that implements IEEE C37.118 data transfer and works as a cross-platform software with minimum dependency on third-party tools is provided;
- the library supports various usage scenarios such as PMU simulators, simple PDCs, synchrophasor data splitter, which have been illustrated in this paper;
- the use of the pyPMU enables implementation of various synchrophasor test beds and provides support for prototyping synchrophasor applications, thus may have an important role in research and education;
- the project is available as an open source software under the liberal BSD license, which allows users to use it freely, modify it, contribute to it, and even fork it and create their own projects.

References

[1] IEEE Power Engineering Society, *IEEE Std C37.118.1, "IEEE Standard for Synchrophasor Data Transfer for Power Systems,"*, 2011.

[2] IEEE Standard, *Test Suite Specification, "IEEE Test Suite Specification – Version 2,"* pp. 1-43, 2015.

[3] I. C. Decker, M. N. Agostini, A. S. e Silva, D. Dotta, *"Monitoring of a large scale event in the Brazilian power system by WAMS,"* in Proc. VIII iREP, pp 1-8, 2010.

[4] G. Heydt. M. Kezunovic, P. Sauer, A. Bose, J. McCalley, C. Singh, W. Jewell, D. Ray and V. Vittal, *"Professional resources to implement the Smart Grid, in North American Power Symposium (NAPS),"* pp 1-82011, 2009.

[5] P. Sauer, *Educational needs for the Smart Grid workforce, "Power and Energy Society General Meeting,"* pp 1-3, 2010.

[6] Grid Protection Alliance, *openPDC*, *https://github.com/GridProtectionAlliance/openPDC*, last accessed: 9.4.2016.

[7] iPDC, *https://ipdc.codeplex.com*, last accessed: 9.4.2016.

[8] S. Šandi, T. Popović, B. Krstajić, *"Alati za podršku mjerenju sinhrofazora,"* IT'14, Žabljak, 2014.

[9] E. National Electric Sector Cybersecurity Organization Resource, *"Wide Area Monitoring, Protection and Control Systems (WAMPAC),"* 2012.

[10] pyPMU – Github repository, https://github.com/iicsys/pypmu, last accessed: 9.4.2016.

[11] IEEE Power Engineering Society, *IEEE Std C37.188.2, "IEEE Standard for Synchrophasor Data Transfer for Power Systems,"* pp. 9-43, 2011.

[12] Pmu Connection Tester, *https://pmuconnectiontester.codeplex.com/*, last accessed: 9.4.2016.

[13] Porting UNIX applications directly to Win32, *https://msdn.microsoft.com/en-us/library/y23kc048.aspx*, last accessed: 9.4.2016.

[14] pyPMU Roadmap, https://github.com/iicsys/pypmu/blob/master/TODO.md, last accessed:9.4.2016