# PyPal: Wi-Fi Trace Synchronization and Merging Python Tool

Mohammad Imran Syed, Anne Flandenmuller, Marcelo Dias de Amorim



Sorbonne Université
LIP6

# PyPal

Mohammad Imran Syed
mohammad-imran.syed@lip6.fr

Paris, March, 2022

Wi-Fi Trace Synchronization and Merging Python Tool

Supervisors: Dr. Anne Flandenmuller and Dr. Marcelo Dias de Amorim

# Contents

# 1 Introduction

PyPal is an updated and Python version of WiPal, which was part of Thomas Claveirole's Ph.D. thesis back in 2010 [1]. The main idea is to synchronize the traces captured by different sniffers at the same time and to be able to merge them by removing the duplicate frames. The process is composed of five modes: (i) identifying reference frames, (ii) extraction of unique frames, (iii) intersection of unique reference frames, (iv) synchronization and (v) merging. We explain each of these modules in the following sub-sections.

## 1.1 Identifying reference frames

A frame is said to be unique when it appears "in the air" once and only once for the whole duration of the measurement. A frame that is unique within each trace but that actually appeared twice on the wireless medium should not be considered as unique. The process of extracting unique frames finds candidates to become reference frames.

The process of intersecting unique frames identifies then identical unique frames from both traces to become reference frames.

## 1.2 Extraction of unique frames

We consider every beacon frame and non-re-transmitted probe response as unique frames. These are management frames that access points send on a regular basis (e.g., every 100 ms for beacon frames). The uniqueness of these frames is due to the 64-bit timestamps they embed.

## 1.3 Intersection

The intersection process intersects the sets of unique frames from both input traces.

## 1.4 Synchronization

Synchronizing two traces means mapping trace one's timestamps to values compatible with trace two's. We compute this mapping with an affine function *t2 = at1 + b*. It estimates *a* and *b* with the help of reference frames as the process runs.

The synchronization process operates on windows of *w*+1 reference frames. For each reference frame $R_i$, the process performs a linear regression using reference frames $R_{\lfloor i-w/2 \rfloor}$ , . . . , $R_{\lceil i+w/2 \rceil}$. At the beginning and at the end of the trace, we use $R_1$, . . . , $R_w$ and $R_{N-w}$ , . . . , $R_N$ (N is the number of reference frames). The result gives a and b for all frames between $R_i$ and $R_{i+1}$.

## 1.5 Merging

The role of merging is to copy frames from synchronized traces to the output trace. Of course, it must organize its output correctly while avoiding duplicate frames. Algorithm 3 gives the original WiPal merging algorithm that we also use in PyPal.

**Algorithm 3** WiPal's merging algorithm.

**Input:** two synchronized traces $T_1$ and $T_2$.
**Output:** the merge of $T_1$ and $T_2$.

```
1:  procedure ADVANCE(f: frame, T: trace)
2:      Append f to output; f ← T's next frame (or nil)
3:  end procedure

4:  f1 ← T1's first frame; f2 ← T2's first frame
5:  while f1 ≠ nil or f2 ≠ nil do
6:      if f1 = nil then ADVANCE(f2, T2)
7:      else if f2 = nil then ADVANCE(f1, T1)
8:      else
9:          t_f1 ← f1's time of arrival
10:         t_f2 ← f2's time of arrival
11:         if f1 = f2 and |t_f1 − t_f2| < 106μs then
12:             Append either f1 or f2 to output.
13:             f1 ← T1's next frame (or nil)
14:             f2 ← T2's next frame (or nil)
15:         else if t_f1 < t_f2 then ADVANCE(f1, T1)
16:         else ADVANCE(f2, T2)
17:         end if
18:     end if
19: end while
```

# 2 PyPal's specifities

## 2.1 Fields required in the trace

The tool takes two traces (in csv or txt format) as input and then performs the option you select. You would need to have the following fields in the traces[1]:

- frame.number: Frame_number
- frame.time_epoch: Frame_time_epoch
- wlan.fixed.timestamp: Fixed_timestamp
- wlan_radio.signal_dbm: RSSI_dBm
- wlan_radio.channel: Channel
- wlan.fc.type: Frame_type
- wlan.fc.type_subtype: Frame_subtype
- wlan.fc.retry: Retransmission
- wlan.fcs: Checksum
- wlan.sa: Source_MAC_address
- wlan.seq: Sequence_number
- wlan.frag: Fragment_number

## 2.2 tshark command to extract the required fields from a pcap trace

You can use the following tshark command to extract the above mentioned fields from a pcap file.

tshark -r pcap_input_file -Y '!_ws.malformed and wlan_radio.channel==1'
-T fields -E header=y -E separator=/t -e frame.number -e frame.time_epoch
-e wlan.fixed.timestamp -e wlan_radio.signal_dbm -e wlan_radio.channel
-e wlan.fc.type -e wlan.fc.type_subtype -e wlan.fc.retry -e wlan.fcs -e wlan.sa
-e wlan.seq -e wlan.frag > csv_or_txt_output_file

[1]It is, however, essential to clearly define which data one can sniff depending on the location of the measurement campaign to preserve the privacy of the users. It is also necessary to carry out hashing of MAC addresses to preserve the privacy.

## 2.3 Steps involved in synchronization

The beacons are the closest representatives of real-time clocks. We use these frames as a base for the synchronization of traces. Two traces are used as input, one as a reference trace and the second trace is the one which has to be synchronized. The first step is to independently extract the beacons that are common in both traces. Hence, the coverage areas of the sniffers capturing these traces should overlap to perform this step. The common frames are referred to as reference frames. In the next step, the timestamps of reference frames are synchronized using linear regression over a sliding window of 3 frames. The synchronized reference frames are then used to synchronize the complete trace. The tool provides an additional option of concatenating or merging the synchronized traces[2].

# 3 How to use the tool

python3 pypal.py -h will also show you the information on how to operate the tool[2].

## 3.1 Libraries required

You need to have the following libraries installed:

- numpy
- pandas
- scikit-learn

## 3.2 Input arguments of the tool

The tool has to positional arguments and those are the two traces:

- trace1: trace to be synchronized
- trace2: reference trace

There are several optional arguments but you have to tell the tool which one you want to use. You can use only one optional argument at a time. The arguments are given below:

- -U : extract unique frames
- -R : extract unique reference frames
- -SR : synchronize unique reference frames
- -S : synchronize traces
- -C : concatenate traces (and keep the duplicate frames)
- -M : merge the traces and remove the duplicate frames within a time difference of 106µs.

## 3.3 Time synchronization error

The time synchronization error (the difference between two timestamps of different sniffers for the same frame) has to be less than half the minimum gap between two valid IEEE 802.11 frames. In the IEEE 802.11b protocol, the minimum gap can be calculated as the 192 microsecond preamble delay plus 10 microsecond SIFS (Short Inter-Frame Space) plus 10 microsecond minimum transmission time for a MAC frame, to be a total of 212 microsecond. So the precision is 212/2 = 106µss [3].

[2]It is preferable to use Python3.

# 4 Link for the tool

The tool can be found on the following link:
https://gitlab.lip6.fr/syed/pypal

# References

[1] T. Claveirole, “Activités wi-fi en environnement ouvert : outils, mesures et analyses,” Ph.D. dissertation, 2010, thèse de doctorat dirigée par Dias De Amorim, Marcelo Informatique, télécommunications et électronique Paris 6 2010. [Online]. Available: http://www.theses.fr/2010PA066020

[2] T. Claveirole and M. Dias de Amorim, “Wipal: Efficient offline merging of ieee 802.11 traces,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 13, no. 4, p. 39–46, mar 2010. [Online]. Available: https://doi.org/10.1145/1740437.1740443

[3] J. Yeo, M. Youssef, and A. Agrawala, “A framework for wireless lan monitoring and its applications,” in *Proceedings of the 3rd ACM Workshop on Wireless Security*, ser. WiSe ’04. New York, NY, USA: Association for Computing Machinery, 2004, p. 70–79. [Online]. Available: https://doi.org/10.1145/1023646.1023660