BLEPal

Mohammed Farouk Akli, Nedir Nour El Amine Boukerras, Nesrine Derradji, Dyhia Laga, Mohammad Imran Syed

BLEPal

Bluetooth Trace Synchronization and Merging Python Tool

Mohammed Farouk AKLI,
Nedir Nour El Amine BOUKERRAS,
Nesrine DERRADJI, Dyhia LAGA

(All authors contributed equally to this work)

Supervisor : **Mohammad Imran SYED**

SCIENCES SORBONNE UNIVERSITÉ

Table of Contents

1. Introduction

The application we developed is written in python and shell script, it encompasses every aspect of the process : from the capture procedure until the final result. Every capture result will be written on a .csv file, the user would not need to do anything manually. Our application is based on an automation philosophy (everything is happening in an automatic way transparent to the end user) and a user friendly dialog inbox which will accompany the user for the whole process.
We decided to name it BLEPal, this name is inspired by the tool WiPal [1] that was made for Wi-Fi, as BLEPal is a tool made for Bluetooth and pal means a friend so our application is metaphorically a Bluetooth's friend as it offers a trace completeness for the protocol so we can higher the performances and reduce data loss when applying the synchronizing and merging process.
To use BLEPal, you will only need the python script of the application and a configuration file named file.cfg. The python script of BLEPal and the file.cfg need to be in the same directory.
The application uses either the nrf52840 dongle or the Ubertooth one dongle to capture the Bluetooth traffic.

1.1. Synchronization Process

Our synchronization method is based on synchronizing an asynchronous environment.
Our environment is a set of Raspberry Pi 4 devices and Bluetooth sniffers (either **Nordic** or **Ubertooth**), a Bluetooth headset and a PC. we establish our topology as following (caricatured by a graph) :

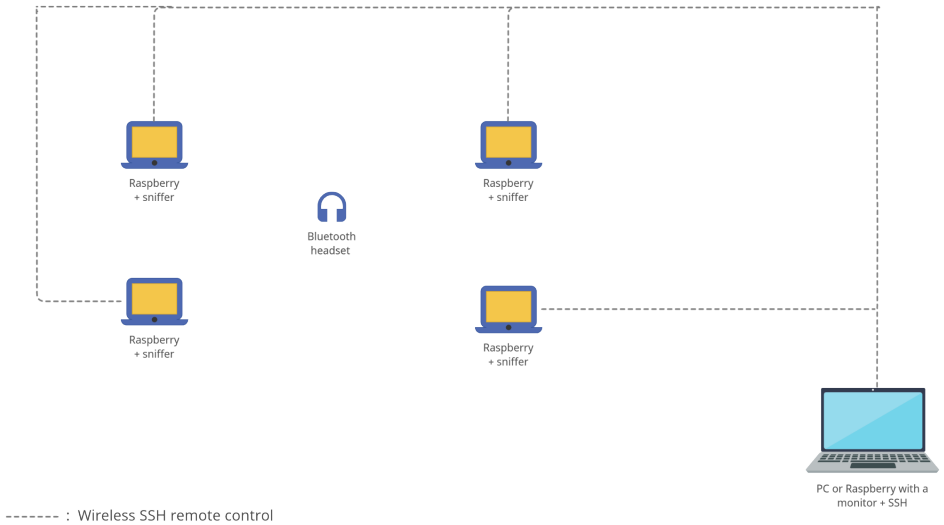

Figure 1.1 : Capture's graph

A timestamp is associated with every **Raspberry + sniffer** set, the timestamp of the sniffer will be equal to the timestamp of the Raspberry, we would not assume that we are in a synchronous environment (every sniffer can have its own timestamp).
Let us associate a timestamp to every sniffer of the four sniffers that we have at the exact moment before launching the capture :
Timestamp of sniffer 1 : t_1

Timestamp of sniffer 2 : t_2

Timestamp of sniffer 3 : t_3

Timestamp of sniffer 4 : t_4

After launching the capture and the Bluetooth traffic, a specific frame has its own timestamp (the instant the frame was captured by the sniffer). This timestamp will be, obviously, bigger than the initial timestamp before launching the capture. Let us associate a timestamp to a frame of every sniffer of the four sniffers that we have (for simplification we considered here a single frame for every sniffer but in reality we have many frames within a sniffer's trace) :
Timestamp of the frame of the sniffer 1 : $t_1 + x_1$

Timestamp of the frame of the sniffer 2 : $t_2 + x_2$

Timestamp of the frame of the sniffer 3 : $t_3 + x_3$

Timestamp of the frame of the sniffer 4 : $t_4 + x_4$

Now and by doing a subtraction between the two set of equations we will have a new timestamp, :
New frame's timestamp of the sniffer 1 : $t_1 + x_1 - t_1 = x_1$

New frame's timestamp of the sniffer 2 : $t_2 + x_2 - t_2 = x_2$

New frame's timestamp of the sniffer 3 : $t_3 + x_3 - t_3 = x_3$

New frame's timestamp of the sniffer 4 : $t_4 + x_4 - t_4 = x_4$

These new timestamps x_i are absolute timestamps rather than the relative timestamps we had before the subtraction operation, they do not depend on the timestamp of the machine (the timestamp of the sniffer).
We can now establish an order to these absolute timestamps, for example we can have this following order :
$x_1 < x_2 < x_3 < x_4$
So the frame with timestamp x_1 is the first captured frame of the experiment, the frame with the timestamp x_2 is the second frame captured, the frame with the timestamp x_3 is the third frame captured and the frame with the timestamp x_4 is the last frame captured.

1.2. Merging Process

The merging process will lean on the extraction of unique frames. When the capture scenario uses **Nordic** sniffers, we will remove the duplicates according to the couple [CRC,Channel Index]. When the capture scenario uses **Ubertooth** sniffers, we will remove the duplicates according to the value of the CRC.

Here is an example of how the merging process works (example given for 4 traces) :

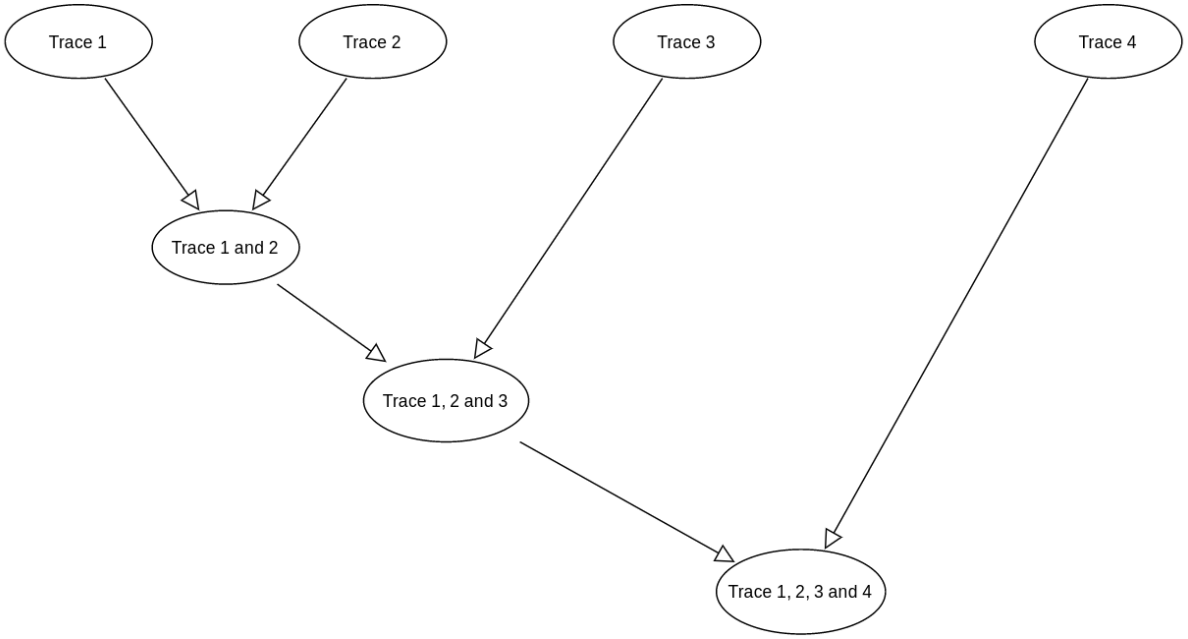

Figure 1.2 : Merging process

The final result will be the trace containing all four traces synchronized and merged (trace 1, 2, 3 and 4).

2. How to use the tool

On a console, type python3 BLEPal.py then a dialog inbox will guide you through the whole process[1].

2.1. Libraries required

You need to have the following libraries installed:

- os
- pandas
- subprocess
- re
- signal
- time
- matplotlib.ticker
- tkinter
- sys
- csv

2.2. The configuration file file.cfg

the content of file.cfg :

```
Mac address of the bluetooth device / mac:address:of:your:bluetooth:device
IP address of the raspberries /
ip.address.Raspberry.1
ip.address.Raspberry.2
.
.
.
ip.address.Raspberry.n
Password of the raspberries /
password : password_1
password : password_2
.
.
.
password : password_n
```

1. It is necessary to use Python3.

3. Link for the tool

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

References

[1] Thomas Claveirole and Marcelo Dias de Amorim. 2010. WiPal: efficient offline merging of IEEE 802.11 traces. SIGMOBILE Mob. Comput. Commun. Rev. 13, 4 (March 2010), 39–46. DOI:https://doi.org/10.1145/1740437.1740443