WirelessEye - an Interactive Real-Time Workflow for WiFi-Based Sensing

2020 - 2022, Philipp H. Kindt philipp.kindt@informatik.tu-chemnitz.de

with contributions from: - Cristian Turetta cristian.turetta@univr.it - Scripts to interface TensorFlow - Florenc Demrozi florenc.demrozi@univr.it - Specification of timestamp format

This software tool accompanies our paper titled “WiFiEye – Seeing over WiFi Made Accessible”, to appear at the WiSense workshop at PerCom 2024

License

When using WirelessEye together with Nexmon, please also pay attention to the Nexmon CSI license, see here.

If you use this software in your research, please consider citing our paper.

Getting Started

Preqrequisites

To perform WiFi-based sensing using WirelessEye, you need - A Raspberry PI 4B with Nexmon firmware (see below) - A Linux PC or Laptop (e.g., Linux Mint) - the GNU C compiler (GCC) - GNU make - QT library version 5 - One or multiple standard WiFi APs to create some WiFi signals to capture

Components

WirelessEye consists of the following two pieces of software

1. CSIServer_ng: A simple TCP server to be run on the Raspberry PI. It will receive the UDP broadcasts from Nexmon and make them available over the network via TCP port 5501
2. WirelessEye Studio: A Qt GUI to display, record and export CSI data in real-time. To be run on any Linux PC from which the Raspberry Pi that runs the CSIServer is reachable over the network.

Preparing the Raspberry Pi

1. Installing Nexmon Before using WirelessEye, the Raspberry Pi has to be prepared to run Nexmon firmware patches. For this purpose, configure and run the nexmon CSI tool as describe here: Nexmon CSI Repository

By default, no RSSI is obtained from this firmware. WirelessEye expects RSSI by default and will hence fail if none is received. Therefore, either switch off RSSI processing in networkThread.h by setting the CSI_CONTAINS_RSSI to false:

#define CSI_CONTAINS_RSSI false

Alternatively, install a patched version of Nexmon that supports RSSI. This is usually the much better alternative. This works as follows.

1. Download the following files from here (don't clone using git - just download from there)
   - rc/csi.ucode.b_cm43455c0.7_45_189.patch
   - src/csi_extractor.c
2. Replace these files in patches/bcm43455c0/7_45_189/nexmon_csi/src of your nexmo installation
3. Recompile the firmware
   - in your nexmon folder: `source setup_env.sh`
   - in /patches/bcm43455c0/7_45_189/nexmon_csi/: `make install-firmware`

2. Compiling and running CSIServer_ng

   WirelessEye contains a TCP server to access the CSI data from another computer, which is called CSIServer_ng. It needs to be compiled and run. For this purpose, do the follwoing on the Raspberry PI:

   1. Copy te CSIServer_ng folder to the Raspi
   2. In the CSIServer_ng folder, type `make`.
   3. Run the CSI server by the command `./CSIServer`
   4. It is recommended to configure Nexmon and run the CSI Server at startup of the Raspberry Pi. We recommend using rc.local. See here for how to do this.

3. Compiling and Running WirelessEye Studio

   1. In the folder `WirelessEye`, type `make`
   2. Run WirelessEyeStudio by typing `./WirelessEye`

Using WirelessEye Studio

You can find a good overview on the functionality provided by WirelessEye in our paper. When hovering the mouse over some object, a quick description in shown in the status bar.

Here's a quick how-to on using WirelessEye:

1. In the tab *settings->connection*, enter the IP address of hostname of your Raspberry Pi

2. In the tab *visualization*, click *connect*. Upon success, the text in the button will change to "connected" and data is being streamed from the Raspberry Pi
3. In the *visualization tab*, empirically select the range of CSI values in which you can see your events of interest
4. When pressing the *record* button, the CSI data is stored into a file. The filename can either be selected in the *settings* tab, or will be automatically assigned based on the time and date. The actual filename is shown in the console when recording starts. WirelessEye supports 3 different formats for recording, which can be selected in the *settings* tab. The actual file format is documented in doc/fileFormats.pdf.
5. Real-Time export of the CSI data, e.g., to a classifier, can be initiated in the *Real-Time Classification* tab. More on this is written below in a separate section.

Real-Time Export

WirelessEye can stream the preprocessed CSI data to any external program, e.g., a classifier that uses machine learning methods. This is controlled form the *Real-Time Classification* tab. Here, any external program can be executed. It is possible to specify the command to be executed and its command line parameters.

The CSI data is written to the standard input of the launched executable. The format is the *Simple CSI format*, which is documented in doc/fileFormats.pdf. The classifier can write its classification results to its standard output, which is imported back into WirelessEye. WirelessEye can annotate these results in the real-time visualization. Though it can only launch one executable, this executable can run multiple classifier. WirelessEye supports displaying the results of multiple classifiers. Hence, the launched executable needs to launch additional classifiers, or needs to include multiple of them in one executable. The data fromat for importing results back into WirelessEye is also documented in doc/fileFormats.pdf.

A pair of scripts for accessing tensorflow to 1) train a classifier using previously recorded data and 2) perform live classification using the real-time export mechanism is included in WirelessEye. They are described below.

Building Classification Models using Tensorflow

The follwoing two python scripts for creating and using a classification model using the Tensorflow machine learning framework are provided along with WirelessEye.

- scripts/model_generation.py: This script is used for creating a machine learning model, e.g., for recognizing human activity using CSI data. It

contains multiple configuration parameters that need to be adjusted in the script.
- scripts/realtime_classification.py: This script can be executed from with WirelessEye to perform real-time classification. It reads the CSI data in real-time and queries a previously
generated model

Prerequisites

We recommend Python 3.8. The following python modules need to be installed for these scripts to work. (The first three are included in Linux Mint, the others need to be installed via pip) - matplotlib - pandas - seaborn - tensorflow (type `python3.8 -m pip install tensorflow` in a terminal for installation) - imblearn (type `python3.8 -m pip install imblearn` in a terminal for installation)

Using model_generation.py

`model_generation.py` provides multiple configurable parameters, which need to be adjusted first. A (mostly reasonable) default value is assigned to each of them. The parameters to be adjusted can be found under the comment # *Settings* in the script code. The following parameters need to be adjusted: - *sampling_frequency*: The sampling frequency of the CSI data, i.e., the (average) number of WiFi frames per second. - *seconds*: A time window (in seconds) of CSI data to be fed into the model - *overlap*: The time by which two adjacent CSI data time windows overlap - *training_epochs*: The number of training epochs - *labels*: An array of strings that contain all labels
- *path_to_file*: Path to a file recorded in WirelessEye to be analyzed. It needs to be in the simple CSV format.

After adjusting these options, the script needs to be executed repeatedly with different previously recorded CSI files. Different files are specified by changing the *path_to_file* parameter each time. It is assumed that the data contained in a single CSV file belongs to one specific label.

Starting from a file like this example:

timestamp	amplitude_sub1	amplitude_sub2	amplitude_sub3	...	amplitude_subn
2021-10-22 14:29:39.314396	1.3608	1.3839	1.4003	...	1.4230
2021-10-22 14:29:39.416798	1.3718	1.3932	1.4565	...	1.4887
...	...	...	...	...	...
2021-10-22 14:32:23.519207	1.3527	1.3794	1.4233	...	1.4434
...	...	...	...	...	...

to put labels on a CSI record, you just have to add a column with the desired label. The label marks the activity/status of the environment in a certine time

window. The label will be used by the model to recognize the CSI pattern for the respective activity/status of the environment described by the label. The final file sholud be similar to:

timestamp	amplitude_sub1	amplitude_sub2	amplitude_sub3	...	amplitude_subn
2021-10-22 14:29:39.314396	1.3608	1.3839	1.4003	...	1.4230
2021-10-22 14:29:39.416798	1.3718	1.3932	1.4565	...	1.4887
...	...	...	...	...	...
2021-10-22 14:32:23.519207	1.3527	1.3794	1.4233	...	1.4434
...	...	...	...	...	...

Make sure that data is recorded in the *simple* CSV format. The model is stored in a file called `model.h5`. The model stored in this file can be queried using `realtime_classification.py`, which is described next.

Using realtime_classification.py

`realtime_classification.py` is called from within WirelessEye Studio. This can be done in the 'Real-Time Classification' tab. Use `python3.8` as the executable and `realtime_classification.py` as the argument. Adjust the number of classes to the length of the *labels* array in `model_generation.py` (for the default value in the script, this would be 4). The script will assign a counting number to the sting lables, which are imported back into WirelessEye. E.g., the first label in the *labels* string of `model_generation.py` will be *0*, the second one *1*, the third one *2*,...

After `realtime_classification.py` has been executed, the *Classifier Output* plot will be available in WirelessEye Studio and in sync with the CSI data.

Developing Plugins

WirelessEye supports plugins to process CSI data. A plugin is a simple C-file. It is complied independently from WirelessEye. Developing filter plugins is simple and can be learned within minutes. A filter plugin has to provide a couple of functions, which are being called trough WirelessEye using dynamic linking.

The only code a plugin needs to import from WirelessEye is the structure *CSIData* from studio/src/CSIData.h. A function called *filter_run()* obtains a pointer to a filled *CSIData* structure. The processing plugin can modify the data in this structure. WirelessEye studio will read the changes back after *filter_run()* has finished.

Learning how a filter is written can be done by examining the extensively commented file studio/src/filters/sample_filter.c. This file implements a fully-functional sample filter with minimalistic code effort. Each function that

is contains a detailed description as a comment. It can also serfe as a sceleton for writing a custom filter. With studio/src/filters/sample_filter.c, it is straight-forward two develop a custom plugin - no additional documentation needed.

Developing for WirelessEye studio

If you want to modify or extend WirelessEyeStudio, you find a full Doxygen documentation of all files of WirelessEye Studio in the doc subdirecory. To build this documentation, go to the *doc/* subdirectory. Then type *doxygen* for building the documentation. Next, go to the *doc/latex/* subdirectory and type *make* to compile a PDF document. Next, you find a documentation in the file *doc/latex/refman.pdf*.

Please also let us know if you'd like to contribute some code to our repository.

Acknowledgements

Thanks to Alejandro Masrur, Florenc Demrozi, Cristian Turetta, Graziano Pravadelli, Samarjit Chakraborty and Shengjie Xu for the pleasant interactions related to WirelessEye.

Known Issues

- (Uncofirmed) In a longer experiment (continuously recording for 70+ hours), the resulting data file appeared to be malformed. It has yet to be confirmed whether this can be reproduced.