

Department of Information Engineering and Computer Science

Bachelor’s Degree in

COMPUTER SCIENCE

final dissertation

a web gui for the

packet loss concealment testbench tool

|  |  |
| --- | --- |
| Supervisor  Prof. Luca Turchet  Co-Supervisor  Luca Vignati | Student  Stefano Dallona |

Academic year 22/23

**Acknowledgements**

To my wife and to my daughters

who constantly supported me during this journey

**INDEX**

[**1** **Packet Loss Concealing (PLC)** 7](#_Toc147254692)

[**1.1** **Digital audio fundamental concepts** 7](#_Toc147254693)

[**1.2** **Packet loss simulation** 8](#_Toc147254694)

[1.2.1 Binomial probability distribution 8](#_Toc147254695)

[1.2.2 Gilbert-Elliot loss model 8](#_Toc147254696)

[**1.3** **Packet loss concealment** 9](#_Toc147254697)

[**1.4** **Output analysis metrics** 9](#_Toc147254698)

[**2** **PLC Testbench UI** 11](#_Toc147254699)

[**2.1** **Web GUI’s main functions** 11](#_Toc147254700)

[**2.1.1** **Configuration of the input data** 11](#_Toc147254701)

[**2.1.2** **Execution management and monitoring** 13](#_Toc147254702)

[**2.1.3** **Analysis of the execution’s results** 13](#_Toc147254703)

[**2.1.4** **Elaboration search and inquiry** 16](#_Toc147254704)

[**2.2** **User Interaction and Experience** 16](#_Toc147254705)

[**2.3** **Technical choices** 16](#_Toc147254706)

[**2.3.1** **Project architecture** 16](#_Toc147254707)

[**2.3.2** **Adopted programming languages** 17](#_Toc147254708)

[**2.3.3** **Frameworks** 17](#_Toc147254709)

[**2.3.4** **Packaging and distribution** 18](#_Toc147254710)

[**2.3.5** **Software Development Process** 18](#_Toc147254711)

[**Results** 19](#_Toc147254712)

[**2.4** **Support to the validation and debugging of the PLC-TestBench tool** 19](#_Toc147254713)

[**2.5** **Highlighting improvement areas for the PLC Testbench tool** 19](#_Toc147254714)

[**2.6** **Extension of the tool usability** 19](#_Toc147254715)

[**2.7** **Improved user experience** 19](#_Toc147254716)

[**2.8** **Future work** 20](#_Toc147254717)

[**3** **Conclusion** 20](#_Toc147254718)

**Abstract**

*The purpose of this thesis was implementing a graphical user interface for the PLC (Packet Loss Concealment) Testbench Tool developed by the Ph.D student Luca Vignati, thus making the tool easier to use while at the same time empowering the interpretation and analysis of the tool’s outputs.*

*Before the implementation of this user interface, the only possible way to interact with the tool was by a Jupyter Notebook, where inputs must be given directly by modifying the code. This required the user to have a fully functional development environment, with all the complexity implied and it also required the user to be familiar with Python programming. Moreover, there was no aid in the interpretation of results, so the only option was to consult the raw output audio files and images. This type of interaction is suboptimal, especially in terms of consulting the results, as one is forced to consider one element at a time, precluding the possibility of increasing the quality of the analysis by considering multiple aspects at once.*

*PLC algorithms are typically needed in real-time data transmission contexts, and their role is to mask (“conceal”) any loss that may occur to one or more packets in a manner that is transparent to the application consuming the data stream.*

*The PLC Testbench Tool aims at providing a modular framework to support a qualitative and quantitative comparison of different PLC algorithms’ performance, by applying two or more of them to a set of audio files provided as an input to the tool. Packet loss is simulated by applying stochastic**mathematical models with configurable parametrization. The software package includes an implementation of the most frequently used packet loss simulation algorithms such as the Binomial PLS, based on binomial loss probability distribution and the Gilbert-Elliott PLS, based on a two-states Markov’s chain.*

*The qualitative analysis is supported by several visual representations of the audio signals’ features (lost packets masks, waveforms, spectrograms) and by the possibility to play any subset of each produced audio track to evaluate its perceived quality.*

*The quantitative analysis of the PLC algorithm outcome is supported by multiple metrics produced by the tool for each of the reconstructed audio files like the Mean Squared Error (MSE), the Mean Absolute Error (MAE), the Spectral Energy, the PEAQ (Perceptual Evaluation of Audio Quality).*

*As a framework, the tool can be easily extended by providing additional implementations of the different object types (PLS algorithm, PLC algorithm, output analysis metric).*

*The GUI can handle these extensions transparently exposing the new features without the need of any code modifications. This is achieved by exploiting the underlying programming language’s introspection capabilities and by scanning the PLC testbench library to discover the custom features.*

*Each elaboration of the tool produces a result dataset, conventionally called a “Run”, which contains a snapshot of all the input parameters and of all the output produced (lost packets masks, reconstructed audio, metrics’ file).*

*The elaboration can be represented as a forest of trees whose root node is associated to one of the audio file given as input. Each input file originates multiple loss simulations, one for each instance of a packet loss algorithm configured. Loss simulations can be mapped to nodes at depth one in the trees and the corresponding audio tracks are in turn reconstructed by applying each one of the PLC algorithms configured at Run level.*

*Reconstructed audio files are represented by nodes at depth two in the trees. Finally, each reconstructed algorithm is evaluated by applying each metric configured in the Run to the corresponding reconstructed audio track.*

*Multiple instances of each configuration object can be set on a Run, as long as the corresponding settings are unique within the Run.*

*During the design phase different types of GUI were evaluated and their pros and cons were carefully weighted up. Finally, the decision was made to go for the development of a web application because of the many advantages offered by the web technology, like a wide range of deployment modes (ranging from a local standalone environment to big, distributed infrastructures), the portability of the application on different platforms, the ease of installation. All of this comes at the price of a slightly increased effort in the implementation.*

*In order to make the distribution of the software as easy as possible, despite the very large number of dependencies in terms of required libraries, it was decided to leverage containers technology.*

*The design of the GUI was made with a particular focus on modularity to make maintenance and future enhancements easier. Modularity was achieved by building graphical widgets that can be reused across the application.*

*The development of the application presented several challenges, especially because of the large amount of data inherently involved by the audio processing. Even short audio tracks are composed of millions of samples that needed to be represented visually as waveforms and/or streamed as sound in real-time. In order to reach the desired performance subsampling techniques and multiple optimizations had to be adopted.*

*Another challenging aspect was the wide spectrum of technologies and programming languages to be learnt in a relatively short period of time.*

1. **Packet Loss Concealing (PLC)**

Most of the widely adopted tools for any kind of remote interaction need to be deployed over the public Internet. While our everyday experience with the public Internet (web browsing, content consumption) may give the feeling that the Internet is a dependable medium, it is not. The transmission protocols that power the Internet at the lower levels follow the so-called “best effort” design philosophy, and only higher lever protocols like TCP introduce the reliability needed for most web applications to work. However, reliability comes at the cost of higher and highly unpredictable latency. For this reason, since most remote interaction tools strive towards that “sense of presence”, they cannot afford to pay the latency price, therefore, they cannot benefit from the reliable communication channel that most other Internet services are built upon. The inevitable implication of dealing with an unreliable channel in a time-sensitive application is packet loss, that is the unavailability of network packets when they are needed by the application. Packet loss results in perceivable glitches and artifacts in the signal on the receiving side, regardless of the signal type. Video signals experience blocking artifacts or even frame freeze, while audio signals experience pops and crackles. These artifacts significantly impact the user experience thwarting any effort to achieve the “sense of presence”. Redundancy in the transmission can mitigate packet loss to an extent, but only a very effective Packet Loss Concealment (PLC) algorithm can prevent any packet loss to become perceivable.

The domain where PLC techniques were applied throughout this work it that of non-speech audio signals, as speech reconstruction requires peculiar techniques that were out of scope for the research from which this thesis originated.

Currently only WAV audio format is supported.

* 1. **Digital audio fundamental concepts**

Sound transmission is a phisical phenomenon consisting of a pressure wave propagating across gaseous, liquid or solid medium. The frequency, amplitude and phase of this wave can change over time, originating a frequency spectrum. Sound can be perceived by the human ear only if any of the frequencies in the spectrum falls within a certain range, called human auditory field.

Digital audio is a representation of sound in a digital form. Typically the pressure wave amplitude is sampled at regular intervals and its value is mapped to a discrete range of values, thus originating a stream of numbers.

According to the Nyquist–Shannon sampling theorem, in order to sample all frequencies included in the human auditory field an audio signal must be sampled at a rate of 44.1 kHz .

A diagram of a wave

Description automatically generated with medium confidence A graph with red and blue dots

Description automatically generated

* 1. **Packet loss simulation**

The packet loss can be simulated by randomly dropping some packets from the original audio stream based on a given probability distribution. The random variable represents a packet transmission whose possible outcomes are successful transmission or packet loss.

La perdita di pacchetti può essere simulata eliminando casualmente alcuni pacchetti dal flusso audio originale in base a una determinata distribuzione di probabilità. La variabile casuale associata alla distribuzione di probabilità rappresenta la trasmissione di un pacchetto, i cui possibili esiti sono la trasmissione riuscita o la perdita del pacchetto.

### Binomial probability distribution

In this model the probability of the random variable to take the value corresponding to a lost packet follows a binomial probability distribution, which represents the sum of a series of multiple independent and identically distributed Bernoulli trials. A Bernoulli trial is a random experiment with exactly two possible outcomes (success/failure), whose probabilities are p [0, 1] and q = 1 – p [0, 1].

The resulting probability to get N success/failure events in a row is described by the following formula based on the binomial coefficients.

A math equations on a white background

Description automatically generated

### Gilbert-Elliot loss model

Compared to models based on a single probability distribution, Gilbert-Elliot model has the big advantage of keeping into account the correlation between loss events. While in the binomial distribution loss model each loss event is unrelated to the previous one and therefore its probability is independent, in the Gilbert-Elliot model the probability of each loss event is conditioned by the previous one.

This behaviour is in line with the assumptions of the Markov’s chains and in fact Gilbert-Elliot model can be considered a two-state Markov’s chain whose graph is the one depicted below.

A diagram of a circular object

Description automatically generated

This model is much more accurate compared to single distribution models, because in the real-world packet loss is usually caused by failures or congestion in the transmission network and therefore tends to happen in bursts.

So, when a first loss is experienced, it is more likely that subsequent packets are lost compared to a transmission success scenario, because the system has usually transitioned from a working to a non-working state. In the same way, when the last packet has been transmitted successfully this is an indication that the system is working properly and therefore the probability that the next packet will not be lost is higher than when the system is in a non-working state.

* 1. **Packet loss concealment**

PLC algorithms are typically needed in real-time data transmission contexts, and their job is to mask (“conceal”) any loss that may occur to one or more packets in a manner that is transparent to the application consuming the data stream. Any PLC algorithm needs to satisfy the following requirements:

* Its execution time needs to be smaller than the period between the consumption of packets.
* Its output doesn’t need to match exactly the content of the corresponding packet. However, when put back in place of the lost packet, it needs to be perceptually indistinguishable from the original audio stream.

There are many ways to perform PLC on audio signals. From simpler waveform-based solutions such as waveform repetition and PSOLA to more advanced autoregressive models. Recently, Deep Learning based PLC has begun to arise in the literature with some noticeable examples both in speech and music.

Regardless of the type, they all use the audio data received immediately before the lost packet as the input to the algorithm, which will output the number of samples contained in the lost packet.

* 1. **Output analysis metrics**

The metrics implemented by the last version PLC Testbench tool at the time of writing this thesis are:

**Mean Squared Error (MSE):** is a measure representing the mean value of the squared differences between paired observations of the same phenomenon. The observation pairs can be represented by an estimatated value versus an actual value or by measures of the same parameter collected at different times, or any other pair of values that make sense in the context of the statistical analysis being performed. It is computed as:

A number and equation with numbers and symbols

Description automatically generated with medium confidence

**Mean Absolute Error (MAE):**

A mathematical equation with black text

Description automatically generated

**Spectral Energy: ???**

**Perceptual Evaluation of Audio Quality:** is a standardized algorithm for objectively measuring perceived [audio quality](https://en.wikipedia.org/wiki/Audio_quality). Among all the metrics implemented out-of-the-box in the PLC Testbench tool this is certainly the most specialized and most effective one for the domain of audio signals processing because it is based on a [psychoacoustic model](https://en.wikipedia.org/wiki/Psychoacoustics) of the human ear and brain. The model takes into consideration human auditory field and its limits and compares a reference signal with a test signal, ignoring differences that are not considered as perceptible according to the model’s rules. PEAQ transforms the inputs by applying filter banks to the Discrete Fourier Transform (DFT) of the signals, producing an intermediate output in the form of a set of variables, each one capturing a different psychoacoustic dimension.

Finally the output variables are used to feed a neural network that simulates the cognitive processes happening inside the human brain which translates the input variables into an overall quality score named Objective Difference Grade (ODG).

The algorithm comes in two different versions: the basic one, less computationally expensive, and the advanced one, more accurate but computationally heavier and therefore not always applicable to real-time signals.

A diagram of a process

Description automatically generated

1. **PLC Testbench UI**
   1. **Web GUI’s main functions**

The purpose of the user interface for the PLC Testbench is to make the tool easier to use, reduce the time required to consult results and increase the quantity and quality of information obtained from the analysis of results.

Before the implementation of this user interface, the only available interaction took place directly in a Jupyter Notebook, where inputs had to be provided directly by modifying the code. Moreover, there was no aid in the interpretation of results, so the only way was to consult the raw output audio files and images. This type of interaction is suboptimal, especially in terms of consulting the results, as one is forced to consider one element at a time, precluding the possibility of increasing the quality of the analysis by considering multiple aspects at once.

The user interface exposes all the functions provided by the underlying PLC Testbench and can be conceptually divided into four modules:

1. Configuration of the input data
2. Progress monitoring
3. Results analysis
4. Elaborations search and inquiry
   * 1. **Configuration of the input data**

The purpose of this module of the GUI is to allow the selection of the files to be processed by a Run of the PLC Testbench tool and to allow the configuration of the sets of packet loss, reconstruction and evaluation algorithms to be applied to each input file with the corresponding parametrizations.

Given the multi-step nature of the task and the high number of selections and choices involved in the process supported by this part of the interface, while designing it the focus was put on making the interaction as simple and intuitive as possible.

In order to achieve this goal, it was decided model the main steps of the process through a wizard based interaction where each step has a clear and small purpose and the user goes the sequence of steps in a logical manner. Splitting a complex task into multiple simpler tasks has several advantages like decreasing the likelihood of mistakes, giving a clearer context for error reporting, making it easier for the user to develop a more accurate mental model of the software, giving the users an update on their progress in the procedure, reducing the cognitive load.

At each step the coherence of the data is enforced reporting any error to the user before the next step is undertaken. This way any wrong or missing information is pointed out in a limited scope and therefore the problem is easier to understand and fix.

From a technical side, this was identified from the very beginning as the part of the GUI which is more likely to be impacted by extensions or future developments, so it looked crucial to make it as flexible and auto-adaptive as possible.

This meant that the interface had to be able to deal with the addition of new algorithms with arbitrary parameters in a transparent way, without requiring any code adjustment.

The fulfillment of this requirement was obtained by exploiting the powerful introspective functions of python programming language, that allow to analyze code structure at runtime. Through these APIs it is possible to discover relationships between software components.

In the PLC Testbench UI it is enough for any implementation of a new algorithm to inherit from the base class of if its own category, be it a loss simulation or a packet loss concealing or an output analysis one, for it to get automatically exposed in the corresponding screens with all its properties.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* + 1. **Execution management and monitoring**

This module of the GUI is intended to allow the user to monitor the state of an in-progress elaboration, providing some sort of progress bar and information like the ETA (Estimated Time of Arrival) for the sub-steps.

Each elaboration of the PLC Testbench tool is named a “Run” and it contains a snapshot of all the input data and all the generated output results, like the loss simulation masks, the reconstructed files and the metrics evaluated on each reconstructed file. Each of the elaboration’s steps can be mapped to a tree node where the root node represents an input file. The overall Run is therefore a forest of trees.

Since the elaboration is identical for all the input files in terms of the algorithms that are executed on the original track and it is sequential, displaying the entire structure of a Run all the time would imply cluttering the interface with redundant information. It was therefore decided to represent progress at two different levels, that is within each input file tree and globally.

A computer screen shot of a network

Description automatically generated

* + 1. **Analysis of the execution’s results**

This module was considered from the very beginning as the most important one, so a great part of the time spent in the analysis and development of the PLC Testbench UI was dedicated to it.

In this case the focus was on providing an easy way to analyze the results from both qualitative and quantitative perspectives.

Qualitative analysis is mainly represented by browsing and visually comparing the waveforms in the regions where packets have been lost and by listening to the audio files generated from the PLC algorithms.

Quantitative analysis is mainly represented by the numerical comparison of the performance achieved by the PLC algorithms according to a given metric like the Mean Squared Error, the Mean Absolute Error, the Spectral Energy, the Perceptual Evaluation of Audio Quality (PEAQ) with respect to the specified input parameters.

Another extremely important aspect was providing the capability to integrate and correlate as much as possible the different types of information to get deeper insight into the behavior of the algorithms being tested.

The main requirements in this case were:

* clearly show the lost portions of the original file
* allow easy comparison of the waveforms ranging from very high level to sample level detail
* give the possibility to display every combination of waveforms in the comparison
* give the possibility to play any audio track, original or reconstructed setting the playhead at any position by simply clicking on the waveform
* support a very wide range of zoom levels including a sample level one
* display visually the metrics calculated on the reconstructed audio files
* display a spectrogram of the audiofile, be it the original or a reconstructed one

The need to display multiple information at the same time made a dashboard interface more appropriate and appealing in this case. The dashboard was organized in 4 sections where the first one is focused on representing the waveforms of all the audio versions of an input file and providing navigation across the files and zoom functions. Lost samples region are displayed together with the waveform and when clicked the corresponding detailed data are loaded in the section below.

The second section provides the ability to inspect the waveform at sample level in any region were some samples were lost. This is extremely useful to evaluate from a qualitative point of view the behaviour of the algorithm and to cross check the correctness of the algorithm’s implementation as visual representation makes any misbehaviour much more apparent.

The third section is intended to display the metrics calculated on the reconstructed versions of the analyzed file and is made by two different charts, one containing the figures from metrics producing time series, the other representing the figures from metrics producing single values.

Finally the fourth section displays the spectrogram of the current audio file, representing the distribution of the frequencies composing it and their relative and absolute intensity in the form of a color map.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated



* + 1. **Elaboration search and inquiry**
  1. **User Interaction and Experience**

<https://marvelapp.com/blog/design-principles-reducing-cognitive-load/>

<https://pencilandpaper.io/articles/ux-pattern-analysis-data-dashboards/>

* 1. **Technical choices**

During the design phase different types of GUI were evaluated and their pros and cons were carefully weighted up. Finally, the decision was made to go for the development of a web application because of the many advantages offered by the web technology, like a wide range of deployment modes (ranging from a local standalone environment to big, distributed infrastructures), the portability of the application on different platforms, the ease of installation.

Comparison with peak.js and wavesurfer

* + 1. **Project architecture**

The user interface application is made of a Web application composed of two layers: one managing the user interface representation and the interaction with the user, the other providing the backend services to be consumed by the graphical components.

The application therefore can be used both as a standalone application by deploying it on the user machine or in a multi-user environment where the application is deployed on one or more remote servers.

The two layers exchange information via a RESTFul API which, if needed, can be exploited also by third-party services. The API is secured by using OAuth2 protocol.

In the presentation layer, modularity has been achieved through the creation of specialized but highly customizable components that can easily be reused across different pages.

In the backend layer, modularity is provided by the extensibility of the RESTFul API where new functions can be “plugged in” by simply adding new endpoints. This layer is also hiding the details of the interaction with the PLC Testbench from the presentation layer, providing an abstract and stable view of the library functions.

Persistence is based on a NoSQL database, where the data are stored in JSON format in order to provide higher flexibility to accommodate future evolution of the schema. Both on-premise and cloud databases are supported.

The GUI adapts automatically to the PLC Testbench library by using introspection to retrieve the list of the algorithms used for packet loss, PLC, and output analysis. This way any extension made to the underlying tool does not require any manual change to the UI code.

The application architecture has been designed to be as stateless as possible and thus supports very well horizontal scalability, which means it is possible to split the workload among as many instances of the application as it is need by providing proper load balancing in front of them.

Scalability together with containerized distribution makes the application suitable for deployment in an orchestrated environment.

* + 1. **Adopted programming languages**

The criteria that drove the selection of the programming languages to be used in the project were the following in order of importance:

1. More portable language are to be preferred over less portable language.
2. More largely adopted and supported language are to be preferred over less widespread language.
3. The languages already used by the PLC Testbench are to be preferred in order to reduce the project maintenance complexity.
4. Languages with better support for scalability are to be preferred over less scalable languages.

Based on this criteria Javascript popped up as the best choice for the core UI layer because of its out-of-the-box integration with all browsers, its being the de-facto standard for the web and the huge number of GUI widgets libraries freely available. On the other side Python seemed like the most natural choice for the Rest API and services’ layer, given its tighter integration with the PLC Testbench tool and the availability of several libraries and frameworks inherently designed for that purpose.

Adopting the same language used by the PLC Testbench tool helped a lot in keeping the project complexity as low as possible, both in terms of languages and in terms of final artifact’s distribution.

* + 1. **Frameworks**

Although frameworks in some cases tend to increase the complexity of software projects, in this case it was judged to be worth adopting some. In fact, leveraging available solutions to common problems was seen in this case not only as a way to avoid “reinventing the wheel” and speed up the project delivery, but also as a mean to standardize the code organization and incorporate some important best practices and design-patterns.

The frameworks used in the PLC Testbench UI codebase are the following:

**React JS:** probably the most important one; the library makes declarative, component base development of GUIs much easier, by providing support for components’ lifecycle management, state management, UI rendering management and optimization.

**PrimeReact:**

**Node js:** used essentially to manage javascript dependencies through the npm package manager.

**D3js:** for the highly specialized tasks related to visual representation of data in most cases D3js library was used behind the scene by wrapping the d3js code into brand new custom React components.

**Wavejs:** a flexible graphical library to represent layered audio waveforms by SVG (Scalable Vector Graphics).

**Flask:** a lightweight framework providing all the functionalities needed by a REST API based web application.

**MongoDB:** persistence of the data was achieved by means of MongoDB, which is one of the most widespread opensource no-SQL databases. The choice of a no SQL database was mainly influenced by the fact that this technology does not impose a rigid schema on the data and therefore looked more flexible and adaptable in case of future extensions of the software.

**PyMongo:** a library for Python programming language that makes interacting with MongoDB much more convenient by providing higher level abstractions over the data layer.

Comparison with other libraries like Peak.js o Wavesurfer.

Realtime vs pre-computed zoom, …

* + 1. **Packaging and distribution**

With regard to packaging and distribution, the objective was a zero-setup installation of the tool and the proper technology to achieve it was considered to be Docker containers.

The PLC Testbench UI is available as a docker public image built from the master branch of the publicly available GitHub repository of the project. The only prerequisite to get the tool running in standalone mode is having a Docker installation on the target machine.

In order to start the application two containers have to be created, one to host a MongoDB instance and one to host the Flask web application. The template for command lines to launch each container are provided in the README.md file of the PLC Testbench UI’s GitHub project and must be customized in the parts which are necessarily specific to each user’s environment, like volume or SSL certificates setup.

This means that the PLC Testbench UI can run on any platform where Docker can run, like all Linux distributions, Windows and MacOS.

The Docker containerization, together with the packages installer tool from Python also helped in managing the huge amount of transitive dependencies brought into the project by specialized libraries for audio manipulation like gstreamer, librosa or by others for more general purposes like tensorflow.

* + 1. **Software Development Process**

At the beginning of the thesis work my knowledge about the domain of digital audio signals processing was extremely limited and my knowledge on the specific subdomain of packet loss concealing was close to zero. Therefore, a major concern was keeping software delivery iterations as short as possible in order to avoid wasting time on useless features and in order to allow fast feedback on the developed features.

A software development process based on Kanban methodology seemed to be a perfect match for this agile way of working and GitHub issue tracking features proved to be like the ideal tool to enforce the process, combining at the same time simplicity, expressiveness, and tight integration with the source code management.

The need to release code frequently also led to the decision to setup a Continuous Integration/Continuous Delivery infrastructure to automate the release of new features in the test environment. CI/CD pipelines allow describing the build and deploy process in the shape of declarative files that can be automatically parsed and executed by CI/CD tools. The tools can also execute different test suites to address unit testing, regression testing and integration testing before actually deploying the artifacts, thus ensuring a proper quality of the versions deployed. On top of all the usual advantages coming from automation, like increased reliability, higher standardization, increased traceability of the process, being simple text files, pipelines descriptors can be managed like any other source file in the project. This means they can be stored in the source code repository, thus getting the benefits of versioning and more reliable management in collaborative environments where multiple developers work on the same project.

Automated deployment to the target environment can be automatically triggered when source code is committed to the repository, so that any code commit that is successfully build and tested is immediately propagated to the test environment.

Circle CI cloud was selected as the CI/CD platform because it is very powerful and it is freely available for small teams and non-commercial use.

Having a CI/CD infrastructure in place helped a lot in shortening the release cycles’s duration and in preventing deployment mistakes related to the differences between the local development environment located on a Windows laptop and the test environment running on a Mac OS server.

1. **Results**

The most important results achieved through the PLC Testbench UI development can be summarized as follows:

* Support to the validation and debugging of the PLC Testbench tool
* Highlighting improvement areas for the PLC Testbench tool
* Extension of the tool usability
* Improved user experience for the PLC Testbench tool
  1. **Support to the validation and debugging of the PLC-TestBench tool**

During the development of the Graphical User Interface, it has become evident how the GUI could support and speed-up the debugging of the PLC algorithm, thus acting also as a validation tool for the PLC Testbench itself. The possibility to compare the waveforms of the original files with those of the reconstructed files at different level of details and focusing only on the regions affected by packets loss made it much easier to confirm the compliance of the PLC algorithms with some essential invariants.

One of these invariants is for example the fact that signals must being identical to the original one in every region where packets have not been lost except for some deviations in the area immediately following a packet loss where there can be some border effects caused by a gradual fading of the PLC reconstruction to make transition smoother.

The detailed visual representation of the signals also made it easier to discover any weird behavior in the PLC algorithms. Without a visual representation more sophisticated than the raw images generated natively by the PLC Testbench tool, finding out such situations would probably have been much more time consuming or even not possible.

* 1. **Highlighting improvement areas for the PLC Testbench tool**

Before the development of the GUI the PLC Testbench tool had only a minimal persistence layer built around filesystem storage for audio file output and .pickle files for configurations and packet loss masks. Pickle is a serialization format natively supported by python which allows to persist memory objects into a file and rebuild them when needed.

The inputs and outputs of each elaboration were grouped by means of filesystem directories and to avoid collisions in the folder names naming conventions had been adopted to encode the parameters. Naming conventions, together with the elaboration logs also represented the main criteria to find out the location where the outputs had been stored.

Every elaboration was unrelated from the others and the inquiry functions were completely unsupported.

While developing the GUI slightly the “Run” concept emerged as a container for the metadata related to an elaboration and together with it the need to move to a more sophisticated persistence layer became clear.

At the beginning a raw set of inquiry functions was built on top of the existing persistence but they were extremely limited due to the poor information available about the “Run”. Finally the decision was made to exploit a database to collect the relevant metadata of the elaborations.

Database based persistence paved the way for the development of a much more powerful inquiry module, exploiting a query engine to dynamically build queries. It also led to further insights, like the possibility to store into persistence the hashes of the files and configuration parameters used by each elaboration and use them to enable performance optimization in the PLC Testbench by completely removing redundant processing across different elaboration. Thanks to the introduction of the hashing feature it was also possible to build a validation layer to detect and prevent data inconsistencies.

* 1. **Extension of the tool usability**

Providing a GUI for the Testbench Tool improves the overall user experience, makes the learning curve of the tool less steep and extends its usability to users not having a programming background.

* 1. **Improved user experience**

Compared to the out of the box representations provided by the PLC Testbench tool which are essentially static images, the PLC Testbench UI allows real-time exploration of the data at different level of detail. The detail level can be selected interactively, depending on the needs and the data are more integrated by being displayed, when comparable, as multiple time-series on the same graph.

In order to improve the user experience the GUI has been made responsive so that it can automatically adjust to different devices (PC, tablet, mobile).

The interface has been designed to be user-friendly, intuitive, and as fast as possible, considering the large dataset inherently involved in audio processing applications and the overhead related to charting functions. Whenever possible subsampling or caching techniques have been applied to minimize latency and network bandwidth waste.

* 1. **Future work**

Despite providing an implementation for all the basic recognized use cases and being fully operational, the PLC Testbench UI has obviously still a lot of room for improvements. Future enhancements of the GUI could obviously be dictated by the evolution of the PLC Testbench itself, but could also be more strictly connected to the user interaction and could be represented for example by:

* extending support to additional audio file formats, like mp3
* providing a function to expose the PLC Testbench logs
* implementing search functions on output results
* implement different user interface themes
* internationalize the UI by adding support for additional languages
* implementing user collaboration use cases
* extending supported identity providers for login (currently only Google)
* developing export functions for the most widely used formats (Excel, CSV, etc.)

1. **Conclusion**

I would like to thank Pr. Luca Turchet and Luca Vignati for constantly supporting me during the development and providing precious suggestions and feedback.

**Bibliography**

[1] Elliott, Edwin O. "Estimates of error rates for codes on burst-noise channels." The Bell System Technical Journal 42.5 (1963): 1977-1997.

[2] Fink, Marco, and Udo Zölzer. "Low-Delay Error Concealment with Low Computational Overhead for Audio over IP Applications." DAFx. 2014.

[3] Verma, Prateek, et al. "A deep learning approach for low-latency packet loss concealment of audio signals in networked music performance applications." 2020 27th Conference of open innovations association (FRUCT). IEEE, 2020.

[4] Thiede, Thilo, et al. "PEAQ-The ITU standard for objective measurement of perceived audio quality." Journal of the Audio Engineering Society 48.1/2 (2000): 3-29.

***Note***

*In the bibliography, all the sources consulted for the dissertation have to be cited and listed in alphabetical order by the first author's surname.*

*According to the source material, the quotation has to be as follows:*

*BOOKS*

*Surname and initial/s of the name/s of the author/s, date of edition, publishing house and (if applicable) number of edition.*

*JOURNAL ARTICLES*

*Surname and initial/s of the first name/s of the author/s, title of the article, name of the journal, volume number, issue number and page numbers.*

*CONFERENCE PAPERS*

*Surname and initial/s of the name/s of the author/s, year of the conference, title of the article, name of the conference, place of the conference, conference dates, page numbers.*

*CITING WEB RESOURCES*

*The consulted webpages have to be listed in alphabetical order.*

*It is necessary to:*

* *Copy the specific URL (the web address) of the consulted webpage*
* *If available, indicate the surname and first name of the author/s, the title and subtitle of the text*
* *If available, indicate the last date you retrieved the webpage (day/month/year).*

**Attachment A Source code repository**

The source code of the project is available on the following GitHub public repository:

<https://github.com/stefano-dallona/plc-testbench-ui>

**PLC Testbench UI’s GitHub source code repository** ….

**Attachment B Docker image**

**PLC Testbench UI’s Docker image**

A pre-built image of the application is publicly available at the following URL:

<https://hub.docker.com/r/stdallona/plc-testbench-ui>

**Running pre-built image and building custom images from source code**

If the pre-built image does not fit your needs you can build a suitable image by customizing the Dockerfile located in the root directory of the source code project on published on GitHub.

**Attachment C CI/CD Pipelines**

In order to make release cycles faster and more reliable a Continuous Integration/Continuous Deployment pipeline has been setup on the free cloud-based tool Circle CI.

The pipeline has been developed exploiting the pipeline as code feature of Circle CI, based on yaml language and is stored in the ***.circleci/config.yml*** file in the GitHub source code repository of the PLC Testbench UI project.

**CircleCI CI/CD Pipelines**

# Use the latest 2.1 version of CircleCI pipeline process engine.

# See: https://circleci.com/docs/configuration-reference

version: 2.1

# Define a common Docker container and environment for jobs

executors:

  docker-publisher:

    # Define the image tag

    environment:

      IMAGE\_TAG: stdallona/plc-testbench-ui:1.0.3

    # Use `docker:stable` as the Docker image for this executor

    docker:

      - image: docker:stable

# Define a job to be invoked later in a workflow.

# See: https://circleci.com/docs/configuration-reference/#jobs

jobs:

  build:

    # Use docker-publisher from above as the Docker container to run this job in

    executor: docker-publisher

    # Add steps to the job

    # See: https://circleci.com/docs/configuration-reference/#steps

    steps:

      - checkout

      # Set up a separate Docker environment to run `docker` commands in

      - setup\_remote\_docker

      - run:

          name: Build Docker image

          command: docker build --tag "${IMAGE\_TAG}" .

      # Archive and persist the Docker image

      - run:

          name: Archive Docker image

          command: docker save --output image.tar "${IMAGE\_TAG}"

      - persist\_to\_workspace:

          root: .

          paths:

            - ./image.tar

  push:

    # Use docker-publisher from above as the Docker container to run this job in

    executor: docker-publisher

    steps:

      # Set up a separate Docker environment to run `docker` commands in

      - setup\_remote\_docker

      # Load and un-archive the Docker image

      - attach\_workspace:

          at: /tmp/workspace

      - run:

          name: Load Docker image

          command: docker load --input /tmp/workspace/image.tar

      # Log in to Docker Hub and push the image

      - run:

          name: Publish Docker image

          command: |

            echo "${DOCKERHUB\_PASS}" | docker login --username "${DOCKERHUB\_USERNAME}" --password-stdin

            docker push "${IMAGE\_TAG}"

# Run the two different jobs as a sequenced workflow

workflows:

  version: 2

  build-push:

    jobs:

      # Run the build first

      - build

      # Push the image second

      - push:

          # Build needs to finish first

          requires:

            - build

          # Only push images from the master branch

          filters:

            branches:

              only: master