Pixelizr: A Visual Demonstration of the FERRET Framework

Fabian Marquardt
University of Bonn
marquard@cs.uni-bonn.de

Christopher Schmidt
University of Bonn
schmidtc@cs.uni-bonn.de

Lennart Haas
University of Bonn
haasl@cs.uni-bonn.de

***Abstract*—Tasks such as network scanning, data scraping or other forms of data retrieval are the basis of many activities in computer networks research. To create flexible and efficient tools for such tasks we have created FERRET, a Python-based framework. FERRET uses a pipeline architecture to perform data retrieval and analysis tasks. Our demonstration provides an artificial, yet interesting showcase of this framework using individual pixels of an image file to make the parallel processing of data with a FERRET-based tool visible to the viewer.**

I. Introduction and Motivation

Many studies and research projects in the area of computer networks require building automated systems to conduct network scans, scrape data from a remote server, or perform other sorts of data retrieval. In many cases, the retrieved data must then be processed in order to gain relevant results and insights.

Hence, it should be a common goal of the computer networks research community to share the software and methodology used for such studies. In many scenarios this is already the case and sophisticated and well-tested tools have been developed. We can name the *zmap* network scanner [1] or the *scrapy* crawling framework [2] as two examples of such tools. But sometimes the functionality and concepts of these tools might be too constraining and the flexibility for own extensions to these tools might be limited.

On the other hand, starting new measurement tools from scratch for every research activity comes with many disadvantages as well: It involves solving basic programming tasks again and again, which leads to duplicated code and is inherently error-prone. Obviously it requires a lot of time and effort which is not spent working on the actual research subject, but instead just on creating the required tools.

To strike a balance between these two approaches, we have created the *Flexible and Efficient Resource Retrieval Toolchain* (FERRET), a rather minimalist Python-based framework. FERRET is used as a basis for our research activities within the *Web Application Security Lab* (WEASEL), including our recent work which has been accepted for the 45th IEEE LCN [3]. FERRET is built especially to support Internet measurement studies, but has a flexible architecture to also support other tasks in computer networks research. After a phase of internal development and testing we intend to provide FERRET as open source software in the near future so that other interested researchers can use and potentially even contribute to the project.

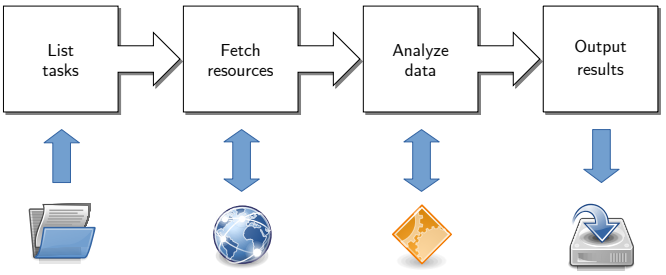

Fig. 1: Concept of a pipeline-based data retrieval tool

To show the features and performance of FERRET, we intend to give a live demonstration of our work at the 45th IEEE Conference on Local Computer Networks. This demonstration involves an artificial web server, which segments an image file into the individual pixels and serves only one pixel per HTTP request. Using a FERRET-based toolchain, we will show how we can parallelize fetching all the pixels from the server to reconstruct the original image quickly.

The remainder of this demonstration proposal is structured as follows: In Section 2 we introduce the foundational concept behind FERRET and give an example on how it suits our typical research tasks. Section 3 explains the technical architecture of FERRET in more detail. Section 4 focuses on the scope of the proposed demonstration. Finally, we conclude our work.

II. Pipeline Concept

To create an efficient and scalable system while maintaining flexibility for every step of the retrieval and analysis process, we propose to use a pipeline-based infrastructure. This means that instead of creating a monolithic program which executes all steps of the complete system, each single step is carried out by an individual program. Each step passes its resulting data to the next step by using message queues.

Figure 1 gives an example about how this principle can be applied in typical computer networks research scenarios: The first element of the pipeline is responsible for enumeration of all tasks which should be executed. This could e.g. be a number of different targets for a network scan, a number of URLs which should later be fetched from a website, etc. The information about which tasks are required could come from an external resource such as a configuration file and is subject to the specific implementation of this pipeline element. For

each executed task, the required information is packed into a suitable data structure and passed on to the next pipeline element. This could be an element which fetches some kind of information from the network for each task, e.g. by executing one or more HTTP requests. The resulting data is then again passed on to the next element. In our example, this element is responsible for analyzing and processing the fetched resources, potentially interfacing with other external tools. For example, it could be used to unpack the payload of the fetched resources or to search for certain patterns in the contained payload. Finally, the resulting data of each task is forwarded to the last element, which would persist the gained information by writing it to a log file, database, etc.

III. FERRET

Using the concept mentioned above, we created the FERRET framework. The goal was to provide a minimal set of functions for the pipeline infrastructure so that each element of the pipeline could be developed and maintained independently while not having to deal with typical management tasks such as connecting the pipeline elements or encapsulating and decapsulating data structures every time a new element is required. Since Python is the mainly used language in our research activities, FERRET is also implemented in Python.

A. Architecture

The architecture of a FERRET-based toolchain is visualized in Figure 2: Each step of the pipeline is performed by one logical pipeline element. When one element is done processing a certain task, it can pass this task to the next element by using a message queue. Each element of the pipeline can interact with several named queues so that branches or even more complex control and data flows can be realized. In our figure for example, element #1 has declared two queues A and B. The data of the tasks is encoded with the Python *pickle* [4] mechanism, meaning that even complex data structures can be passed from one pipeline element to the next without problems. The next pipeline elements will then pull the available data from the queue to continue working on the task. For elements that require a lot of processing power, it is possible to start several instances of one logical pipeline element, as is visible for element #2 in our example. Data from the queue will be distributed among all instances of one element, so that tasks can be processed in parallel.

B. Technical details

Since FERRET aims to provide high throughput and concurrency, it makes heavy use of Python's asynchronous programming concepts. Especially I/O-heavy tasks can be parallelized very efficiently when using *asyncio* [5]. This also means that other Python asynchronous frameworks such as *aiohttp* [6] can be integrated seamlessly into a FERRET-based toolchain.

The message queues used in FERRET are based on the *RabbitMQ* [7] technology. Connections to the RabbitMQ server are handled by the *aio-pika* module [8], which is able to provide high throughput and low overhead when passing data from

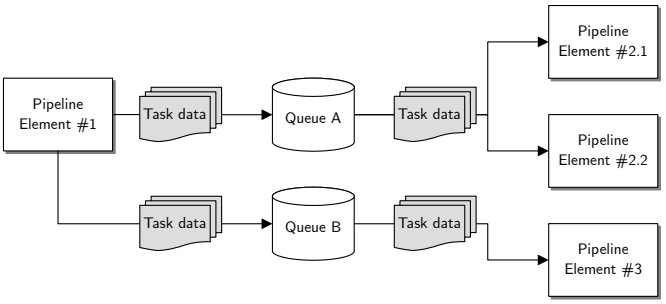

Fig. 2: FERRET architecture

one pipeline element to the next one. By using RabbitMQ it is also possible to create setups with FERRET-based pipeline elements running distributed across multiple hosts to maximize processing performance.

FERRET additionally includes a thin wrapper around the *configargparse* [9] module which facilitates configuration in various deployment scenarios. Using this wrapper, FERRET-based pipeline elements can receive configuration data in a unified way via several mechanisms, namely via the command line, a configuration file or environment variables. This makes it easy to test the elements locally during development, but also to create cloud-based deployments using e.g. Docker or Kubernetes.

IV. LCN Demo

When preparing the demonstration during the difficult times of the COVID-19 pandemic we thought about a good way to show the potential of FERRET, while also providing an engaging and interactive experience for the attendees of the virtual conference.

A. Pixelizr

Eventually we developed *Pixelizr*, which is a showcase project consisting of a small web server and a FERRET-based client. The basic idea of Pixelizr is that the server provides access to a number of stored image files, but instead of accessing the whole file at once it only allows the client to fetch one single pixel of the image with each HTTP request.

This means that the client, in order to download the full image, must launch a very high number of requests to the server's API, parse the contained information, and finally reassemble the original image.

Obviously, this would take a very long time when using a traditional, sequential client. With our FERRET-based client however the fetching process can be efficiently parallelized and the resulting image is reassembled quickly.

B. Demo scope

During the demonstration the conference attendees will be able to see the Pixelizr client running live via screensharing. While the Pixelizr client receives the image contents from the server, the rendered image is progressively updated, as is visible in Figure 3.

Fig. 3: Screenshot of the Pixelizr client

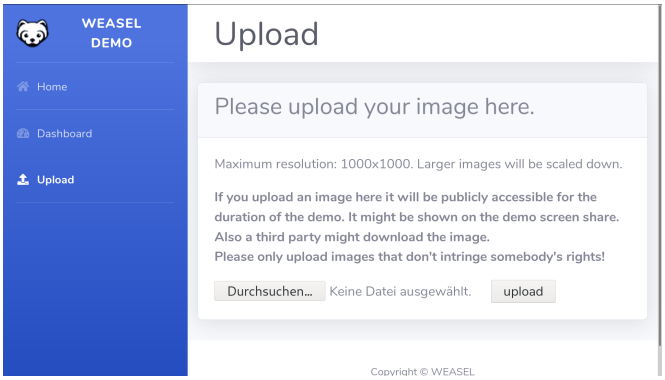

Fig. 4: Screenshot of the Pixelizr server image submission

Viewers of the demonstration will also be able to upload their own images via the submission site of the server (cf. Figure 4) and see them being transmitted to the client. Viewers will in addition get access to the Pixelizr server dashboard (cf. Figure 5), which provides metrics and statistics about the interaction between server and client.

We will also demonstrate the flexibility of a toolchain built with FERRET by adding additional processing steps to the client on the fly such as altering the colors of the image, performing geometric translations, etc.

V. Conclusion

In this demonstration proposal we presented our idea of developing typical software tools required for computer networks research by using a pipeline-based architecture. We introduced our Python-based framework called FERRET, which

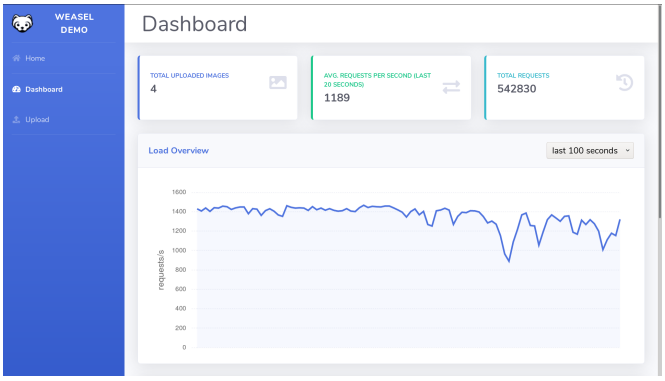

Fig. 5: Screenshot of the Pixelizr server dashboard

implements this idea and provides a foundation to create flexible and scalable tools for research. We presented Pixelizr as an artificial example application built with FERRET, which aims to visualize the data collection process and provide an interactive experience in a virtual conference scenario.

For the future we intend to publish more information about FERRET and eventually release the framework as open source software. We intend to continue using FERRET as a basis for future research activities and are very interested in getting valuable feedback from the conference attendees in order to further improve the framework.

References

[1] Z. Durumeric, E. Wustrow, and J. A. Halderman, "Zmap: Fast internet-wide scanning and its security applications," in *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C.: USENIX, 2013, pp. 605–620. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/durumeric

[2] D. Kouzis-Loukas, *Learning scrapy*. Packt Publishing Ltd, 2016.

[3] F. Marquardt and C. Schmidt, "Don't Stop at the Top: Using Certificate Transparency Logs to Extend Domain Lists for Web Security Studies," in *2020 IEEE 45th Conference on Local Computer Networks (LCN) (accepted)*, 2020.

[4] "pickle - Python object serialization," Python Documentation, 2020. [Online]. Available: https://docs.python.org/3/library/pickle.html

[5] "asyncio - Asynchronous I/O," Python Documentation, 2020. [Online]. Available: https://docs.python.org/3/library/asyncio.html

[6] "aiohttp," GitHub, 2020. [Online]. Available: https://github.com/aio-libs/aiohttp/

[7] D. Dossot, *RabbitMQ essentials*. Packt Publishing Ltd, 2014.

[8] "aio-pika AMQP client," GitHub, 2020. [Online]. Available: https://github.com/mosquito/aio-pika

[9] "ConfigArgParse," Python Package Index, 2020. [Online]. Available: https://pypi.org/project/ConfigArgParse/