SoftwareX 24 (2023) 101506

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Software update

Flow-models 2.0: Elephant flows modeling and detection with machine learning

Piotr Jurkiewicz

Institute of Telecommunications, AGH University of Krakow, Poland

ARTICLE INFO

Article history:
Received 14 July 2023
Accepted 16 August 2023

Keywords:
Elephant flows
Mice flows
Traffic engineering
SDN
Machine learning

ABSTRACT

This article presents the new version of the `flow-models` IP network flow modeling framework. The improved features include flow skipping and counting, flow filtering, IP address anonymization, and time series data calculation. The new version also enables simulation of the first packet mirroring feature and provides tools for modeling the detection of elephant flows. It includes examples of using the `scikit-learn` library to build machine learning models for elephant flow detection based on the first packet. Furthermore, it provides an anonymized flow dataset, enabling researchers to train and validate machine learning models for traffic analysis in a reproducible and comparable manner.

Refers to

Piotr Jurkiewicz, flow-models: A framework for analysis and modeling of IP network flows, SoftwareX, Volume 17, 2022, 100929, ISSN 2352-7110, 10.1016/j.softx.2021.100929.

Code metadata

Current code version	2.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00447
Code Ocean compute capsule	N/A
Legal Code License	MIT
Code versioning system used	Git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	NumPy, SciPy, Matplotlib, pandas, cryptography
Link to developer documentation/manual	https://flow-models.readthedocs.io
Support email for questions	piotr.jurkiewicz@agh.edu.pl

1. Introduction

Flow-based traffic engineering is a promising approach for managing the increasing network demand without compromising quality of service or requiring costly infrastructure investments [1,2]. It involves dynamically selecting paths for IP flows based on the network's current or predicted load. This allows flows between the same endpoints to follow any number of alternative paths. Adaptive routing of flows also offers greater stability compared to dynamic load balancing in classic IP prefix-based routing [3].

In [4], we introduced flow-models, an open-source Python framework designed to create precise and reproducible statistical IP network flow models from NetFlow/IPFIX flow records. This framework offers advanced capabilities that are not found in other packages, including the ability to merge split flow records, calculate histograms of flow feature distributions, and create General Mixture Models fitting those. Researchers and network operators can utilize these models for analytical calculations and generate realistic traffic in simulations. Furthermore, the framework provides additional functionalities such as sorting flow records and plotting histograms and distribution models.

DOI of original article: https://doi.org/10.1016/j.softx.2021.100929.
E-mail address: piotr.jurkiewicz@agh.edu.pl.

https://doi.org/10.1016/j.softx.2023.101506

In this paper, we present a new version of the framework that introduces new functionalities specifically focused on modeling and detecting elephant flows. Despite technological advancements, the number of simultaneous flows in networks still exceeds the capacities of switch flow tables [5]. However, recent analysis shows that a small percentage of flows, representing only 0.2–0.4% of the total, account for 80% of network traffic [6,7]. By implementing individual handling for this subset of flows and routing the remaining flows over the shortest paths, significant improvements in network performance can be achieved while minimizing overhead and the number of flow table entries.

The challenge lies in detecting elephant flows, preferably from their initial packets. To address this, we introduce the `first_mirror` and `elephants` subpackages in the new version of framework. These subpackages can be used to model various elephant detection mechanisms and techniques. We provide examples how to train machine learning models detecting elephant flows with the first packet using the `scikit-learn` library. We also publicly share a dataset of flow records in a binary format, anonymized using the prefix-preserving encryption algorithm, which can be utilized for training and testing machine learning models. Additionally we describe improvements and fixes in the existing framework functionalities introduced since the previously presented version, including three new tools in the main framework package.

2. Improvements of existing features

Since the version 1.2, which was described in the previous paper, several improvements and fixes has been made in the framework. This included 92 commits, 4788 lines of code added and 2003 lines deleted. Overview of the most significant changes is provided below:

Common flow skipping and counting capability. In the previous version, the `convert` tool had the capability to skip a specified number of flows on its input or output, or to process only a requested number of flows instead of all flows. This functionality has been moved to the input generator functions within the `flow_models.lib.io` library. As a result, this feature is now available in all tools. Users can utilize the `--skip-in`, `--skip-out`, `--count-in`, and `--count-out` parameters to control this behavior.

Flow filtering capability. The `convert` tool also had the capability to filter flows according to a specified Python expression. This allowed, for example, selecting only flows with certain source addresses or longer than N packets. In the current version, this functionality was also moved to the common I/O library, which means that now any tool can perform filtering during its operation. Filters can be specified using the `--filter-expr` command line parameter.

Fast flow filtering with NumPy. Filtering is done by executing a filter Python expression when the flow tuple is being read by the input function. However, when using the binary flow record input format, it can be performed much faster using NumPy masking. We have implemented this in the new version of the package. Consequently, starting from version 2.0, bitwise operators should be used instead of logical ones in filter expressions to ensure compatibility with both the Python interpreter and NumPy:

For example, a filter selecting HTTPS flows:

`--filter-expr ''prot==6 and (sp==443 or dp==443)''`

should be rewritten as:

`--filter-expr ''(prot==6) & ((sp==443) | (dp==443))''`

Common argument parser. In the current version of the framework, we have standardized the command line argument syntax across different tools. We achieved this by implementing a common argument parser using the `argparse` library, which is now utilized by all tools.

Update of read_pipe to the new nfdump format. Recently, the `nfdump` package made a change to its pipe format syntax, adopting a single field that represents the flow start and end timestamps as milliseconds since the start of the Unix epoch. Consequently, we have updated the function responsible for reading `nfdump` pipe formatted flow records in our framework to accommodate this change.

API documentation improvements. Since the previous version, we have made significant improvements to the documentation of our tools and API. We have added usage examples to the help messages of our tools, as well as parameter types and descriptions to the functions in the framework library via docstrings. These are accessible through the automatically generated API reference at: https://flow-models.readthedocs.io.

Automatic testing and deployment. The previous version did not contain any functional tests of the code. In the current version, we have introduced tests to validate the main features of the framework using the `pytest` package. Furthermore, we have implemented continuous integration (CI) using GitHub Actions. This allows for the automatic building of the framework and running tests on each commit and pull request. Additionally, we have included an action that automatically builds and publishes a Python package to PyPI whenever new releases of the framework are made.

Bug fixes. Several bugs, also reported by other users of the framework, have been fixed since the previous release. They were mostly related to type casting problems or inconsistent parameter names usage across different tools.

3. New features

Several new features have been also added in the current version. This includes three new tools in the main `flow_models` package, and two `first_mirror` and `elephants` subpackages aimed at specific applications.

3.1. New tools

`cut` tool efficiently splits binary flow record files without the need for serialization. It utilizes the dd Unix command for optimal performance in this task.

`anonymize` tool anonymizes IPv4 addresses in flow records using Crypto-PAn [8] prefix-preserving algorithm. It works only for IPv4 flows. Therefore, after processing by this tool all flows of other address families will be filtered out.

`series` tool is designed to calculate the number of flows, packets, and bytes transmitted on a link in each second based on flow records. It generates time series data for these features and writes them in CSV format.

3.2. First packet mirroring simulation

The `first_mirror` subpackage enables simulations and analytical calculations of the first N packets mirroring feature found in the latest generation of SDN switches. This feature involves the switch's CPU or SDN controller receiving copies of the initial packets from each new flow in the switch's dataplane. The controller can then perform packet inspection and flow identification. By analyzing various aspects of the new flows, including the connection setup procedure, packet size, and time gaps between

the first packets, the controller can determine the associated application, even for encrypted connections.

This capability is also valuable for early detection of elephant flows. By classifying flows based on their first packets, the need for mid-flow rerouting is eliminated. Furthermore, it ensures that for the majority of a flow's lifespan, it will be subject to traffic engineering mechanisms specifically designed for elephant flows, such as individual routing paths. Additionally, the controller can continuously learn and refine its detection models based on the stream of first packets from flows.

We provide a script to simulate this feature at the packet level and compute the count of flows, packets, and bytes mirrored to the controller for each second, using flow records as input.

3.3. Elephants modeling and detection with machine learning

The `elephants` subpackage provides functionalities related to *elephant flows* modeling. It offers tools for simulating and calculating reduction in flow table occupancy and number of flow entry operations for a desired fraction of traffic, which can be achieved using various elephant detection methods. These methods include classifying flows based on the first packet header (including machine learning-based algorithms), reaching a predefined counter threshold (including inexact counters like sketches or Bloom filters), or detecting elephants through sampling.

The `simulate` tool performs simulations at the packet level. It reads flow statistics from a histogram file or generates flows from a JSON mixture, and then generates sample packets for these flows. Simulations can be repeated to obtain confidence intervals. On the other hand, the `calculate` tool can analytically calculate flow table occupancy reduction curves based on provided flow records, histograms, or mixtures.

Additionally, we provide the `elephants.sklearn` subpackage, which offers examples on how to use the `scikit-learn` library [9] to train machine learning algorithms for detecting elephant flows based on the first packet. Several recent papers have focused on this topic, for example [10–14]. However, none of these works analyze metrics such as flow table reduction or the amount of traffic transmitted after flow classification, which we believe are crucial from the perspective of traffic engineering and QoS. These studies primarily focus on classification accuracy, measured by parameters like true positive rate, true negative rate, and accuracy of flow size and duration prediction. They provide limited insight into the effectiveness of the analyzed algorithms in our specific application. For example, misclassifying the largest flow in the network has a much greater impact on the change in traffic coverage than misclassifying a small flow. The metrics presented so far do not account for this difference. Our proposal is to use novel metrics for evaluating ML algorithms in the context of elephant flow detection for TE purposes, specifically flow table occupancy reduction and fraction of traffic covered. There is a tradeoff between those two: increasing the elephant detection threshold leads to greater flow table reduction but decreases the fraction of covered traffic.

The examples in the `elephants.sklearn` subpackage demonstrate how to train and validate classifiers and regressors from the `scikit-learn` library to obtain reduction curves and how to tune the hyperparameters of these algorithms. Additionally, we provide the `elephants.ml` utility module, which contains functions useful in machine learning applications, such as data preparation and calculation of flow table reduction scores.

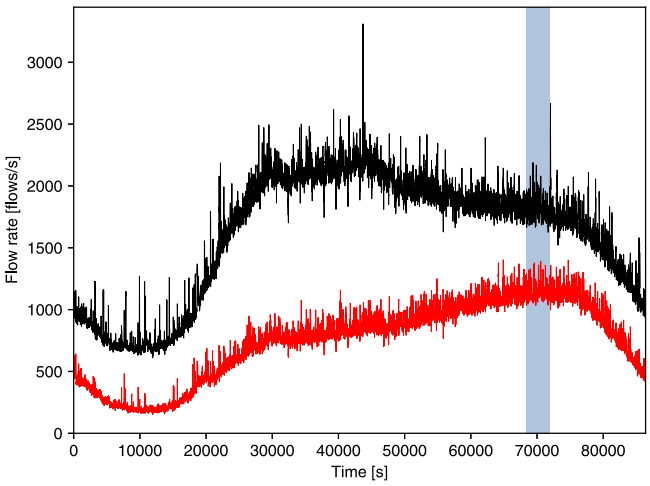

Fig. 1. Flow rate on June 10th.

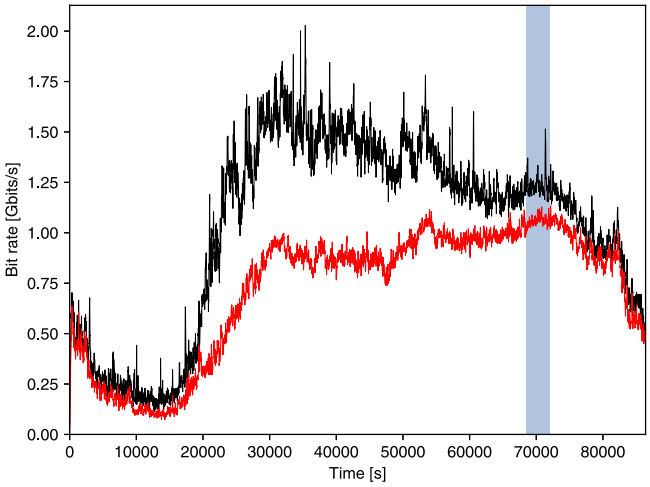

Fig. 2. Bit rate on June 10th.

4. Anonymized flow dataset

In the previous version of the package, we provided CSV histograms of flow features derived from the `agh_2015` dataset, as well as JSON files containing distribution mixtures fitted to these histograms. However, the original flow records used to generate these histograms were not included in the package. The complete set of flow records for this dataset occupies approximately 278 GB in binary format. Furthermore, the records contain real source and destination IP addresses, so it was not possible to share them both due to practical and privacy reasons.

However, for training machine learning algorithms, individual flow records that include IP addresses, port numbers, and flow sizes are required. As a result, we have made the decision to publicly share an anonymized subset of flow records from the `agh_2015` dataset. This will allow other researchers to train and validate their models in a reproducible and comparable manner.

4.1. Selection process

We examined the whole dataset in order to select the most suitable time period to share. Using the `series` tool, we generated separate plots of flows, packets, and bytes per second for each of the 30 days in the dataset. We analyzed these plots,

looking for anomalies or irregularities that may indicate unusual network activity. For instance, a sudden spike in flow rate without a corresponding increase in bit rate could be indicative of a DDoS or SYN flood attack. Finally we decided to select Wednesday, June 10th for sharing. This particular day was a working day with normal university operations, including the presence of students in dormitories, which contributes to the majority of traffic. Network traffic is shown in Fig. 1 and Fig. 2. The **red line** represents traffic from/to dormitories, and the **black line** is the overall campus traffic.

However, even after excluding non-IPv4 flows, the size of the flow records for that day amounted to 6.4 GB. Therefore, we divided the chosen day into 24 one-hour periods based on the flow start time (`first`, `first_ms`). For each of these hours, we calculated the theoretical flow table occupancy reduction rate curves of a detection algorithm which can perfectly detect elephant on the first packet, as described in [15]. Finally, we determined that the most suitable hour to share is 19:00 UTC. It is marked on Fig. 1 and Fig. 2 in blue. The reduction rate curve calculated from the flow records data during that hour closely resembles that of the mixture derived from the entire 30-day dataset, as shown in Fig. 3. Moreover, the traffic generated by dormitories accounted for 90% of traffic during that hour. Therefore, the selected dataset can be regarded as a representative for residential traffic.

4.2. Anonymization

We applied the prefix-preserving Crypto-PAn algorithm [8] to anonymize the source and destination IPv4 addresses in the dataset using the `anonymize` tool. This approach is commonly employed in research, as demonstrated in previous works such as [16,17]. However, it is important to consider that anonymization may impact the performance of machine learning models. To investigate this, we conducted experiments using the `scikit-learn` library. The dataset was divided into five folds, and we trained and tested regressors to predict flow sizes and detect elephant flows using both the original and anonymized data. The results, presented in Fig. 3, indicate that the achieved reduction in flow table occupancy is similar across the same folds, regardless of whether the IP addresses were anonymized or not.

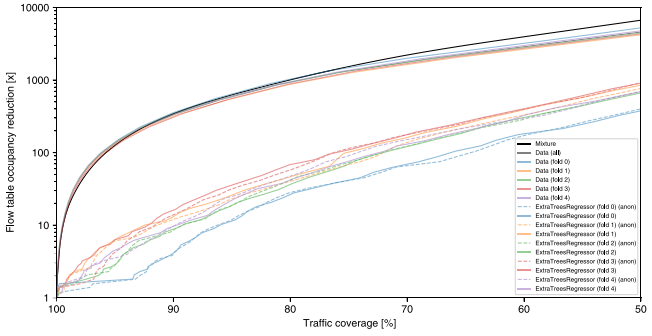

Fig. 3. Flow table occupancy reduction curves for the original and anonymized dataset.

5. Conclusions

The new version of the framework includes several improvements and bug fixes, enhancing its overall functionality and usability. Notable enhancements consist of flow skipping and counting, flow filtering, anonymization of IP addresses using the Crypto-PAn algorithm, and calculation of time series data for flows, packets, and bytes transmitted on a link.

The new `first_mirror` subpackage allows for simulations of the first packet mirroring feature found in recent SDN switches. The new `elephants` subpackage focuses on modeling the detection of elephant flows. It offers tools for simulating and calculating flow table occupancy reduction for various detection methods and provides examples of using the `scikit-learn` library to build machine learning models for elephant flow detection based on the first packet. We emphasize the importance of specific metrics, such as flow table reduction and fraction of traffic covered, for evaluating machine learning algorithms in the context of elephant flow detection for traffic engineering purposes.

Furthermore, we share an anonymized flow records dataset, allowing other researchers to train and validate their models in a reproducible and comparable manner. We describe the data selection process and anonymization method used to ensure privacy while maintaining the dataset's utility for machine learning applications. The framework's open-source nature and the availability of the anonymized dataset promote collaboration and reproducibility in network traffic research.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data is available in the GitHub repository.

References

[1] Kreutz D, Ramos FMV, Veríssimo PE, Rothenberg CE, Azodolmolky S, Uhlig S. Software-defined networking: A comprehensive survey. Proc IEEE 2015;103(1):14–76. http://dx.doi.org/10.1109/JPROC.2014.2371999.

[2] Mendiola A, Astorga J, Jacob E, Higuero M. A survey on the contributions of software-defined networking to traffic engineering. IEEE Commun Surv Tutor 2017;19(2):918–53. http://dx.doi.org/10.1109/COMST.2016.2633579.

[3] Jurkiewicz P, Wójcik R, Domżał J, Kamisiński A. Testing implementation of FAMTAR: Adaptive multipath routing. Comput Commun 2020;149:300–11. http://dx.doi.org/10.1016/j.comcom.2019.10.029.

[4] Jurkiewicz P. flow-models: A framework for analysis and modeling of IP network flows. SoftwareX 2022;17:100929. http://dx.doi.org/10.1016/j.softx.2021.100929.

[5] Shen G, Li Q, Ai S, Jiang Y, Xu M, Jia X. How powerful switches should be deployed: A precise estimation based on queuing theory. In: IEEE INFOCOM 2019-IEEE conference on computer communications. IEEE; 2019, p. 811–9. http://dx.doi.org/10.1109/INFOCOM.2019.8737629.

[6] Megyesi P, Molnár S. Analysis of elephant users in broadband network traffic. In: Meeting of the European network of universities and companies in information and communication engineering. Springer; 2013, p. 37–45. http://dx.doi.org/10.1007/978-3-642-40552-5_4.

[7] Jurkiewicz P, Rzym G, Boryło P. Flow length and size distributions in campus internet traffic. Comput Commun 2021;167:15–30. http://dx.doi.org/10.1016/j.comcom.2020.12.016.

[8] Xu J, Fan J, Ammar M, Moon SB. On the design and performance of prefix-preserving IP traffic trace anonymization. In: Proceedings of the 1st ACM SIGCOMM workshop on internet measurement. New York, NY, USA: Association for Computing Machinery; 2001, p. 263–6. http://dx.doi.org/10.1145/505202.505234.

[9] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12:2825–30.

[10] Durner R, Kellerer W. Network function offloading through classification of elephant flows. IEEE Trans Netw Serv Manag 2020;17(2):807–20. http://dx.doi.org/10.1109/TNSM.2020.2976838.

[11] da Silva MVB, Jacobs AS, Pfitscher RJ, Granville LZ. Predicting elephant flows in internet exchange point programmable networks. In: Advanced

information networking and applications: Proceedings of the 33rd international conference on advanced information networking and applications, vol. 33. Springer; 2020, p. 485–97. http://dx.doi.org/10.1007/978-3-030-15032-7_41.

[12] Hardegen C, Pfülb B, Rieger S, Gepperth A, Reißmann S. Flow-based throughput prediction using deep learning and real-world network traffic. In: 2019 15th international conference on network and service management. IEEE; 2019, p. 1–9. http://dx.doi.org/10.23919/CNSM46954.2019.9012716.

[13] Hardegen C, Pfülb B, Rieger S, Gepperth A. Predicting network flow characteristics using deep learning and real-world network traffic. IEEE Trans Netw Serv Manag 2020;17(4):2662–76. http://dx.doi.org/10.1109/TNSM.2020.3025131.

[14] Gomez JG, Hernandez V, Ramirez-Gonzalez G. Traffic classification in IP networks through machine learning techniques in final systems. IEEE Access 2023. http://dx.doi.org/10.1109/ACCESS.2023.3272894.

[15] Jurkiewicz P. Boundaries of flow table usage reduction algorithms based on elephant flow detection. In: 2021 IFIP networking conference. 2021, p. 1–9. http://dx.doi.org/10.23919/IFIPNetworking52078.2021.9472832.

[16] Luxemburk J, Hynek K, Cejka T, Lukačovič A, Šiška P. CESNET-QUIC22: a large one-month QUIC network traffic dataset from backbone lines. Data Brief 2023;46:108888. http://dx.doi.org/10.1016/j.dib.2023.108888.

[17] Griffioen H, Doerr C. Examining Mirai's battle over the Internet of Things. In: Proceedings of the 2020 ACM SIGSAC conference on computer and communications security. New York, NY, USA: Association for Computing Machinery; 2020, p. 743–56. http://dx.doi.org/10.1145/3372297.3417277.