flow-models: A framework for analysis and modeling of IP network flows

Piotr Jurkiewicz

Department of Telecommunications, AGH University of Science and Technology, Kraków, Poland

Abstract

Recent developments in SDN technologies have resulted in flow-based forwarding becoming are a feasible alternative for traditional mechanisms. However, as the efficiency of flow-based solutions strongly depends on traffic characteristics, they need to be evaluated using realistic and accurate flow models. In this paper, we present a software framework, called flow-models, for creating precise and reproducible statistical flow models from NetFlow/IPFIX flow records. The flow-models package can be used to merge split records, calculate histograms of flow features and create General Mixture Models fitting them. The models created by flow-models can be used both as an input in analytical calculations and to generate realistic traffic in simulations.

Keywords: flow model, distribution fitting, traffic engineering, SDN

1. Motivation and significance

Fixed-function switches and routers are the foundation of traditional networking. These network devices are custom-built, monolithic proprietary boxes. They consist of a vertically integrated application-specific integrated circuit (ASIC) and proprietary, closed source software. Network administrators are allowed by device vendors to configure only selected parameters, without the possibility to change the device operation in a significant way. Moreover, such devices need manual configuration, usually through CLI, which is a slow and error-prone process.

To overcome these drawbacks, the software-defined networking (SDN) concept emerged around 2010 as an alternative. SDN decouples the network control and forwarding functions, and makes network control directly programmable. In SDN, forwarding devices are programmed with flow forwarding rules using standardized mechanisms like OpenFlow and P4. Network intelligence is centralized in a software controller which maintains a global view of the network. Administrators can dynamically adjust network-wide traffic flow to meet changing traffic patterns with automated SDN programs they write, which do not depend on proprietary software.

Although the SDN concept originated in academia, its advantages resulted in the development of programmable switching chips and the subsequent introduction of SDN switches to the market. Due to these technological advancements, flow-based networking is gaining increasingly more attention as a feasible alternative for traditional solutions [1].

Flow in computer networks is defined as a unidirectional sequence of related packets. In Internet Protocol (IP) networks, the most common approach, used for example in NetFlow/IPFIX (IP Flow Information Export) [2], is to associate flows with transport layer connections. This means that flow is a sequence of packets that share the same address fields up to the transport layer (so-called 5-tuple): IP source address, IP destination address, source port, destination port, and transport layer protocol type.

SDN and flow-based networking can ease network management, and can overcome many limitations of existing approaches, especially in traffic engineering (TE) [3], quality of service (QoS) provisioning [4] or network security [5]. The most promising area for flow-based networking is flow-based forwarding.

Currently, the most common approach is still packet-based forwarding. In such an approach, each packet is forwarded by a router independently, according to a per-destination IP prefix routing table. Routing tables are usually populated by routing protocol daemons, which run on each router, and use a distributed algorithm to discover the network topology and shortest paths between particular IP destinations. However, due to its distributed nature, per-packet routing imposes significant limitations on multi-path transmission capabilities, as well as adaptivity to changing traffic patterns.

Distributed routing protocols can discover and use the shortest path between selected nodes in the network. Where more than one path with a cost equal to the shortest one exists, all of them can be used. Such an approach is called equal-cost multi-path (ECMP) and is common in current networks. Its usability is however limited in wide-area networks due to their asymmetric nature and resulting lack of multiple shortest paths. For example in the *nobel_eu* topology from SNDLib[1], the average number of disjoint short-

Email address: `jurkiew@agh.edu.pl` (Piotr Jurkiewicz)

[1] http://sndlib.zib.de

Table 1: Code metadata

Current code version	1.2
Permanent link to code/repository used for this code version	https://github.com/piotrjurkiewicz/flow-models/tree/v1.2
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
Link to developer documentation/manual	https://flow-models.readthedocs.io
Support email for questions	piotr.jurkiewicz@agh.edu.pl

est paths between any nodes is 1.20, whereas the average number of all disjoint paths is 2.61. Unequal cost multipath (UCMP) assumes usage of additional paths with cost higher than the shortest one's cost. However, this is not trivial, as it may lead to routing loops. Loop-free UCMP requires specific metrics and constraints in routing protocol. The only protocol currently in usage supporting UCMP is EIGRP. However, to maintain its distributed nature, it has to enforce conservative conditions while calculating the set of feasible paths and discard some of the existing paths [6]. This significantly limits the number of additional paths which can be used. To use all available disjoint paths and achieve maximum flow between selected nodes, techniques not based on distributed algorithms have to be used.

Adaptive (load-sensitive) routing is also impossible in the per-packet approach, as the dynamic alteration of link costs leads to instability, which ultimately deteriorates network performance. This has been shown by early ARPANET attempts [7] [8]. Flow-based forwarding enables multipath and adaptive approaches, which are impossible to achieve in per-packet routing due to routing loops and route-flapping constraints, respectively [9].

However, the efficiency of most flow-based solutions strongly depends on traffic characteristics. Therefore, they should be evaluated using realistic flow models. A perfect example of such solutions are traffic engineering mechanisms exploiting the heavy-tailed nature of IP flows, which reroute significant flows over alternative paths, balancing that way the network's load [10]. To reliably evaluate such ideas, accurate distributions of flows' length and size must be used. The lack of such models negatively impacts the credibility of experiments results.

Realistic models can be created from traffic traces, previously collected in the network. The well-known solutions for flow records collection are NetFlow, IPFIX, and sFlow. Packages like `flow-tools` or `nfdump` [11] provide tools for filtering and calculating simple statistics from collected flow records. `flow-tools` generates usage reports for flow data sets by IP address, IP address pairs, ports, packets, bytes, interfaces, next hops, autonomous systems, ToS bits, exporters, and tags. On the other hand, `nfdump` allows filtering and calculating simple summary/top-N statistics from network flow records. The aforementioned packages lack, however, any capabilities for the analysis and modeling of flow features. This includes flow's length (number of packets), size (amount of bytes), duration, and rate distributions. Moreover, they do not provide any tool for merging flow records split due to active timeout. The goal of this framework is to fill this gap.

Specifically, the presented framework can be used to merge flow records split during the collection process, calculate histograms of flow features distributions and finally to create General Mixture Models fitting those distributions. Afterward, created models can be used in networking research, both as an input in analytical calculations and to generate realistic traffic in simulations. The framework also provides several auxiliary functionalities, like flow records sorting or histogram distribution and model plotting.

2. Software architecture

The framework consists of several tools, which provide features not available in the existing flow record processing packages. Following the Unix philosophy, each tool is a separate Python module aimed at a single purpose. Tools are tailored to be used sequentially in data-processing pipelines and features provided by them are orthogonal. The scheme of such a pipeline is shown in Figure 1. Tools being part of the framework are shown in black, while tools from the external packages are colored in blue.

The existing solutions can be used in the initial steps of the pipeline. First, all flow records have to be collected. Hardware NetFlow exporters or software ones (`ulogd2` or `nfpcapd`) can be used to create flow records from packet traffic. Such flow records are collected in the nfcapd format. Next, before any further processing, the data need to be cleaned and filtered. Irrelevant or erroneous flow records can be filtered out with the `nfdump` tool. Subsequent steps require the usage of tools provided by our package, as necessary functionalities are not provided by existing solutions. Since long-lasting flows may be reported multiple times due to triggering procedures in the exporters, such flow records have to be found and merged back. The next step is the reduction of data passed to

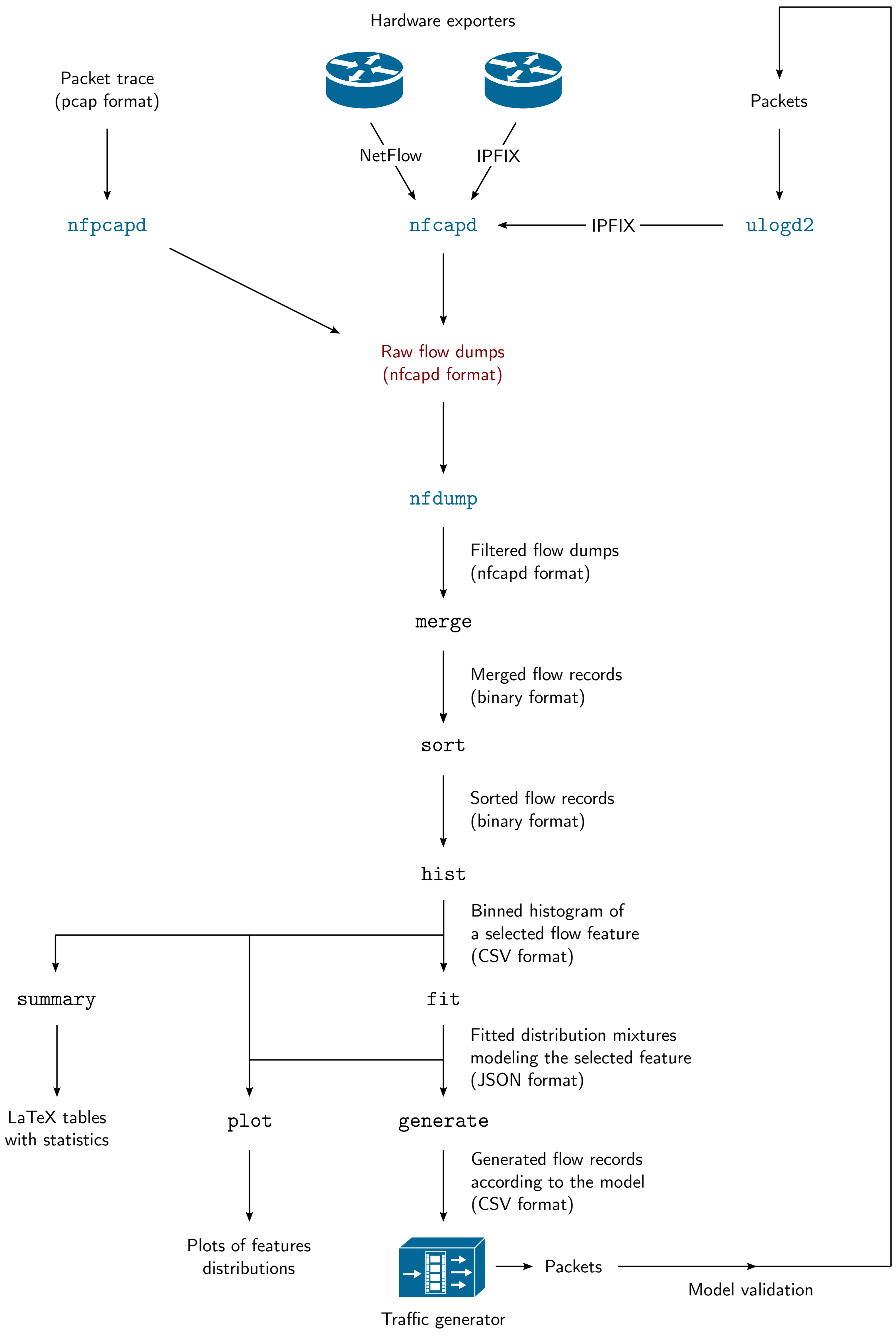

Hardware exporters
Packet trace (pcap format)
Packets
NetFlow
IPFIX
nfpcapd
nfcapd
IPFIX
ulogd2
Raw flow dumps (nfcapd format)
nfdump
Filtered flow dumps (nfcapd format)
merge
Merged flow records (binary format)
sort
Sorted flow records (binary format)
hist
Binned histogram of a selected flow feature (CSV format)
summary
fit
Fitted distribution mixtures modeling the selected feature (JSON format)
LaTeX tables with statistics
plot
generate
Generated flow records according to the model (CSV format)
Plots of features distributions
Packets
Model validation
Traffic generator

Figure 1: The scheme of data processing pipeline.

the modeling by binning it. The fitting of a general mixture model, approximating the collected data, follows afterward. The fitted model can be used to mimic real traffic in simulators or traffic generators. The framework currently includes the following tools:

- `merge` – merges flow records that were split across multiple records due to active timeout
- `sort` – sorts flow records according to specified fields (requires *numpy*)
- `hist` – calculates histograms of flows length, size, duration or rate
- `hist_np` – calculates histograms using multiple threads (requires *numpy*, much faster, but uses more memory)
- `fit` – creates General Mixture Models (GMM) fitted to flow records (requires *scipy*)
- `plot` – generates plots from flow records and fitted models (requires *pandas* and *scipy*)
- `generate` – generates flow records from histograms or mixture models
- `summary` – produces TeX tables containing summary statistics of flow dataset (requires *pandas*)
- `convert` – converts flow records between supported formats

The framework is implemented in Python. It has been published on GitHub and can be installed as a package from the PyPI (The Python Package Index) using the `pip install flow-models` command. The package consists of executable tool modules listed above and subpackage `lib` containing modules providing common functionalities. All tool modules are registered as entry points, so after package installation, a user can easily call them from the shell command line (e.g. `flow-models-merge`).

The package makes an extensive use of NumPy [12] and SciPy [13] numerical processing libraries. Matplotlib [14] is used for plots generation and Pandas [15] data analysis library for CSV file handling and summary tables generation. Overall, the framework consists of 15 Python files, containing 1988 lines of code and 176 lines of comments, according to the `cloc` utility.

The purpose of the framework is to provide flow analysis and modeling functionalities not present in the existing packages. Therefore, it was designed to be used on the top of `nfdump` toolset, which is a standard solution for flow data collection and filtering. The nfcapd file format from that toolset is an input format for our framework. To exchange data between tools included in the framework, two intermediate flow records formats can be used: comma-separated values (CSV) format and binary format. A flow record contains the following fields:

- `af`, `prot` – address family, IP protocol number
- `inif`, `outif` – input and output interface numbers
- `sa0:sa3`, `da0:da3` – consecutive 32-bit words forming source and destination IP addresses
- `sp`, `dp` – source and destination transport layer ports
- `first`, `first_ms` – timestamp of a first packet (seconds and milliseconds components)
- `last`, `last_ms` – timestamp of a last packet (seconds and milliseconds components)
- `packets` – number of packets (flow length)
- `octets` – number of octets (bytes) (flow size)
- `aggs` – number of aggregated flow records forming this record

The *binary* file format can be used as an effective alternative to the textual CSV format. Each binary file stores one field as an array of binary values of a specified type. The file name contains field name (as listed above) and data type, which specifies the type of binary object stored in a file. Such a storage schema has several advantages:

- Fields can be distributed independently (for example, one can share flow records without sa* and da* address fields for privacy reasons).
- Fields can be compressed/uncompressed selectively (important when processing data which barely fits on disks).
- Additional or custom fields can be trivially added or removed.
- Supports storage of any field using any object type (signedness, precision).
- Files can be memory-mapped as numerical arrays (unlike IPFIX, nfcapd or any other structured/TLV format).
- Memory-mapping is IO and cache efficient (columnar memory layout allows applications to avoid unnecessary IO and accelerate analytical processing performance on modern CPUs and GPUs).

3. Software functionalities

First, flow records have to be collected. In the case of hardware NetFlow/IPFIX exporters, the `nfcapd` tool can be used for that purpose. Alternatively, `ulogd2` can be used on Linux systems to export flows observed on the machine by the Netfilter/Conntrack subsystem into IPFIX records. These records are also collected by `nfcapd` and saved into its on-disk format. It is also possible to convert a packet capture file into flow records with the `nfpcapd` tool. This brings the possibility to analyze and model time-related features (flow durations and rates), which is impossible with data originating from hardware exporters due to timekeeping issues [16].

After the collection, the data have to be cleaned. This is especially important in the case of data originating from hardware exporters, as they can provide corrupted flow records characterized by implausible durations. Moreover, the flow records file may contain records originating from multiple devices, which also need to be filtered out. The `nfdump` command-line tool can be used for that purpose. It supports powerful and flexible filter syntax similar to tcpdump. It is written in C and is very well optimized for filtering tasks.

3.1. merge

In all hardware and many software exporters, long-lasting flows may become split due to *active timeout* and reported as multiple flow records. Such flow records have to be found and merged back in order to obtain accurate flow length, size, or duration values. The `merge` tool available in our framework can be used for that purpose. Additionally, it filters out erroneously split records. The tool processes all flow records sequentially and performs all calculations using only integers to ensure precision and reproducibility. This is possible thanks to Python's unlimited width integer support.

The tool takes flow records in any supported format as an input and outputs merged flow records in binary or CSV format. Each merged flow record contains *aggs* field, which tells how many flow records were merged back into that particular aggregate flow record. A user should specify both *active* and *inactive* timeouts used in the collection process when calling the command to ensure the correctness of merge operation.

3.2. sort

During the merging process, flow records may become reordered. This applies especially to long flows, which in some circumstances may stay cached until the end of the merge process. Such flows are dumped at the end of output files. The purpose of `sort` tool is to reorder flow records in a file according to specified keys, usually flow start or flow end times. This step is unnecessary when further operations will be performed on the whole file. However, in a case when only a part of a record file will be used, sorting is required.

3.3. hist

Fitting of mixture models does not have to be performed on complete flow records. Instead, it can be performed on histograms, calculated by binning flow records into buckets according to the selected parameter (e.g. flow length or size). Histogram files can also be easily published as they are many orders of magnitude smaller and, unlike flow records, do not contain private information such as IP addresses. We provide a tool called `hist` which performs flow binning.

The tool takes flow records in any supported format as an input and outputs a histogram file in a CSV format. A user should specify the parameter to be binned (flow length, size, duration, or rate) and additional columns to be summed in a histogram (by default packets and octets are counted, additional fields can be rate and duration). The user can also specify a parameter, which is a power-of-two defining starting point for logarithmic binning. Logarithmic binning significantly reduces the size of histogram files without affecting the quality of the fitting process noticeably.

Two implementations of the tool are available: `hist` and `hist_np`. The former is a pure Python implementation that takes advantage of unlimited width integer support in Python to perform more accurate calculations. The latter uses the NumPy package to perform binning, which can utilize SIMD instructions and multiple threads and is many orders of magnitude faster, but requires more memory and can introduce rounding errors due to the operation on doubles having limited precision.

3.4. fit

The `fit` tool is the key component of the framework. Its purpose is to find a mixture of distributions (along with their parameters) matching accurately the selected flow feature. We have implemented the Expectation–Maximization (EM) algorithm [17] to estimate the parameters of a statistical model composed of mixture components.

The tool takes a flow histogram CSV file as an input and performs distribution mixture fitting. JSON file, describing shares of separate distributions in the mixture and their parameters, is an output. To start the EM algorithm, an initial distribution mixture has to be provided. Its parameters are then iteratively refined in order to find the local optimum. Our tool can receive an initial distribution mixture from a user, but it can also generate an initial mixture for a particular dataset on its own, which means that the user has to only provide the number and types of distributions used in a mixture.

Currently, *uniform*, *normal*, *lognormal*, *Pareto*, *Weibull* and *gamma* distributions can be used in mixtures fitted by our tool. However, the *uniform* and *lognormal* distributions are usually sufficient to provide an accurate mixture model. They have the advantage of being fast to fit, since their maximization steps have analytical solutions, whereas some other distribution parameters (*Weibull* or *gamma*) must be calculated using numerical optimization methods. Another advantage is that they are widely implemented, so distribution mixtures composed of them can be usable in various network simulators and traffic generators.

The `fit` tool can operate in command line mode and graphical interactive mode (GUI). In the case of batch operation, fitting is performed according to provided command line parameters and the result is saved in a JSON file in the working directory. In the case of interactive operation, the user can observe the fitting process in real-time on a GUI. After its completion, he can examine the model quality on plots and, if necessary, refine the number of distributions and their initial parameters and repeat the fitting. The video showing the interactive fitting process is provided in Section 4.

3.5. plot

An important part of any modeling task is the visualization of both input data and resulting models. The `plot` tool can be used for that purpose. It can generate probability density (PDF), cumulative distribution function (CDF), average packet size, and packet interarrival time plots. It takes CSV histogram files and mixture model JSON files as input. The input histogram data can be visualized on a PDF plot as points, 2-dimensional histogram, or kernel density estimation (KDE) contour plot. Model mixtures are presented as lines. Additionally, components of a mixture can be plotted, both separately and in stacked mode. The tool automatically normalizes data points in the case of logarithmically-binned histograms. Moreover, the framework contains a custom fast Fourier transform (FFT) based implementation of weighted KDE computation.

3.6. generate

In order to be used for benchmarking network mechanisms, models must enable the generation of traffic matching the mixtures. The tool `generate` provides a reference for how to properly generate flows from distribution mixtures. It takes a path to the directory containing JSON mixture models as input and outputs flow records. Additionally, a CSV histogram file can be used instead of a mixture model as an input, to generate flows exactly matching the particular dataset.

3.7. summary

The purpose of the `summary` tool is to generate traffic characteristics and features distribution tables in TeX or HTML format from CSV histogram files. Generated summary tables can be easily published in an article or shared as a web page.

3.8. convert

The `convert` tool can be used to convert flow records between supported formats (`nfcapd` format and CSV and binary formats flow described in Section 2).

4. Illustrative examples

Here we provide a step-by-step guide through the distribution fitting and model creation process. In order to be able to follow the tutorial with the data publicly available in the project's repository, it starts after flow binning (that is with a flow histogram CSV file). To get a reference on how to create histogram files from flow records, framework documentation and Makefile should be consulted.

First, it must be ensured that the required Python's standard library modules are installed. This applies in particular to `tkinter` and `venv` modules, which in some distributions are not installed by default with the Python's binaries. We will conduct our experiments in a virtual environment. Alternatively, the flow-models package can be installed systemwide with the `pip` command, in which case the `numpy`, `scipy`, `pandas` and `matplotlib` should be present on the system.

Listing 1: Installation of the framework and its dependencies.

```
$ python3 -m venv test
$ cd test
$ bin/pip install flow-models numpy scipy pandas
    matplotlib
```

In this tutorial, we will use the dataset `agh_2015`, which is provided in the project's Git repository. The provided dataset does not contain flow records due to privacy and size concerns. Therefore, we will start the fitting process with the histogram file, which was previously created from flow records with the `hist` command.

Firstly, we will use the `summary` tool to get a grasp on the data. The tool generates TeX tables containing overall traffic characteristics and distributions of selected features. Specifically, with the command below, a distribution table for the flow length of all transport layer protocol flows will be created.

Listing 2: Usage of the `summary` tool.

```
$ cd agh_2015
$ flow-models-summary -x length histograms/all/
    length.csv
```

Similarly, a distribution table for flows depending on their size can also be generated:

Listing 3: Usage of the `summary` tool for a flow size.

```
$ flow-models-summary -x size histograms/all/size
    .csv
```

Now having the grasp on basic data properties, let us get an insight into the details. This can be done by plotting empirical probability distribution functions and cumulative distribution functions of flow features. The `plot` tool can be used for that purpose. The command below will generate plots of PDF and CDF of a number of flows, packets, and octets (bytes) in the function of flow length. Additionally, plots presenting an average number of packets, octets, and packet size depending on flow length will be created.

Listing 4: Plotting a histogram of a flow length.

```
$ flow-models-plot histograms/all/length.csv
```

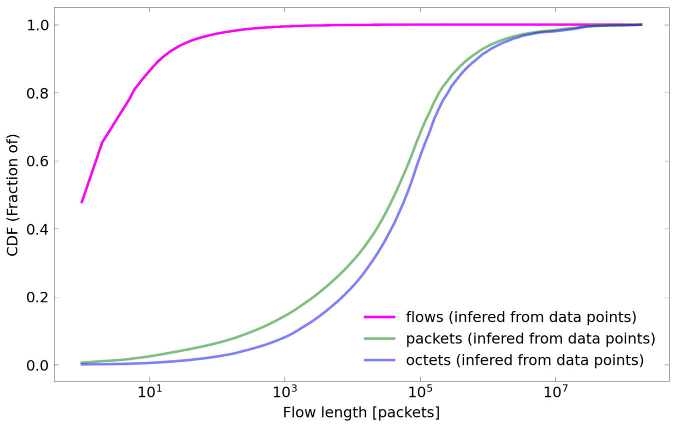


Figure 2: The cumulative distribution function (CDF) plots.

Plots were created in the current working directory as PNG files. The Figure 2 presents the CDF plot (cdf.png).

Now let us move to the fitting process, which is the core operation of the framework. It can be performed with the `fit` tool. Similarly as with previous commands, flow histogram CSV should be given as an input. The `fit` tool can operate in the batch and the interactive mode. Below we will perform mixture fitting for the flows distribution in function flow's length in the batch mode. In such a case, the number of iterations (i) and the number of distributions of each type (U and L) should be specified as command-line parameters.

Listing 5: Fitting of a distribution mixture.

```
$ flow-models-fit -i 400 -U 6 -L 4 -y flows
    histograms/all/length.csv
0.00 Processing: histograms/all/length.csv
0.05 Iteration: 0
...
12.29 Iteration: 399
12.29 Saving: flows

$ cat flows.json
{
  "sum": 4032376751,
  "mix": [
    [0.3050265769901237, "uniform", [0, 1]],
    [0.24841988004416196, "uniform", [0, 2]],
    [0.06366063664158104, "uniform", [0, 3]],
    [0.04921649965932878, "uniform", [0, 4]],
    [0.00931559166293734, "uniform", [0, 5]],
    [0.08217474157187263, "uniform", [0, 6]],
    [0.1312637498251484, "lognorm",
        [0.5207023493412831, 0,
        7.805599279070412]],
    [0.07328615421743442, "lognorm",
        [0.7701056575379248, 0,
        22.10972501544735]],
    [0.02928912648715966 2, "lognorm",
        [1.1252645297342552, 0,
        128.6451515069823]],
    [0.008347042900250784, "lognorm",
        [1.9838369452408506, 0,
        1084.4707584768773]]
  ]
}
```

The fitted mixture is saved into a JSON file in the current working directory. In this case, this is the file flows.json, which contents is listed above. Alternatively, a user can perform interactive fitting with the GUI. In such a case, the tool should be started with the `--interactive` parameter.

Listing 6: Interactive fitting of a distribution mixture.

```
$ flow-models-fit -i 100 -L 1 -y packets --
    interactive histograms/all/length.csv
```

The Figure 3 shows a screenshot of the interactive fitting tool. Users can change parameters (number of iterations and number of distributions) on the right panel. After clicking the fit button, the process starts. Its progress can be monitored on the progress bar. Additionally, after checking the animate checkbox, the current distribution mixture will be plotted after each iteration. This allows observing the fitting process in real-time. The window on the bottom-right shows the current distribution mixture in JSON format. After achieving a satisfying result, the mixture can be saved to a JSON file using the save button.

We also provide an illustrative video presenting the interactive fitting process. The video is visible next to the article, in the right-hand side panel. The user is trying to find a mixture model describing flow size (octets number) in the function of flow length (packets number). On the upper plot, the probability density function is shown, for data, mixture, and all its components separately. The bottom plot shows cumulative distribution functions. The user repeats the fitting process with a different number of distributions and algorithm iterations until the satisfying result is obtained. Model's accuracy can be examined on presented plots, in which fragments can be zoomed in.

The `plot` tool can be used not only to plot empirical distribution functions from histogram files but also mixture models. Both can be plotted on the same figure in order to enable their comparison. For this, a path to a directory containing model JSON files should be provided after the path to a histogram CSV file.

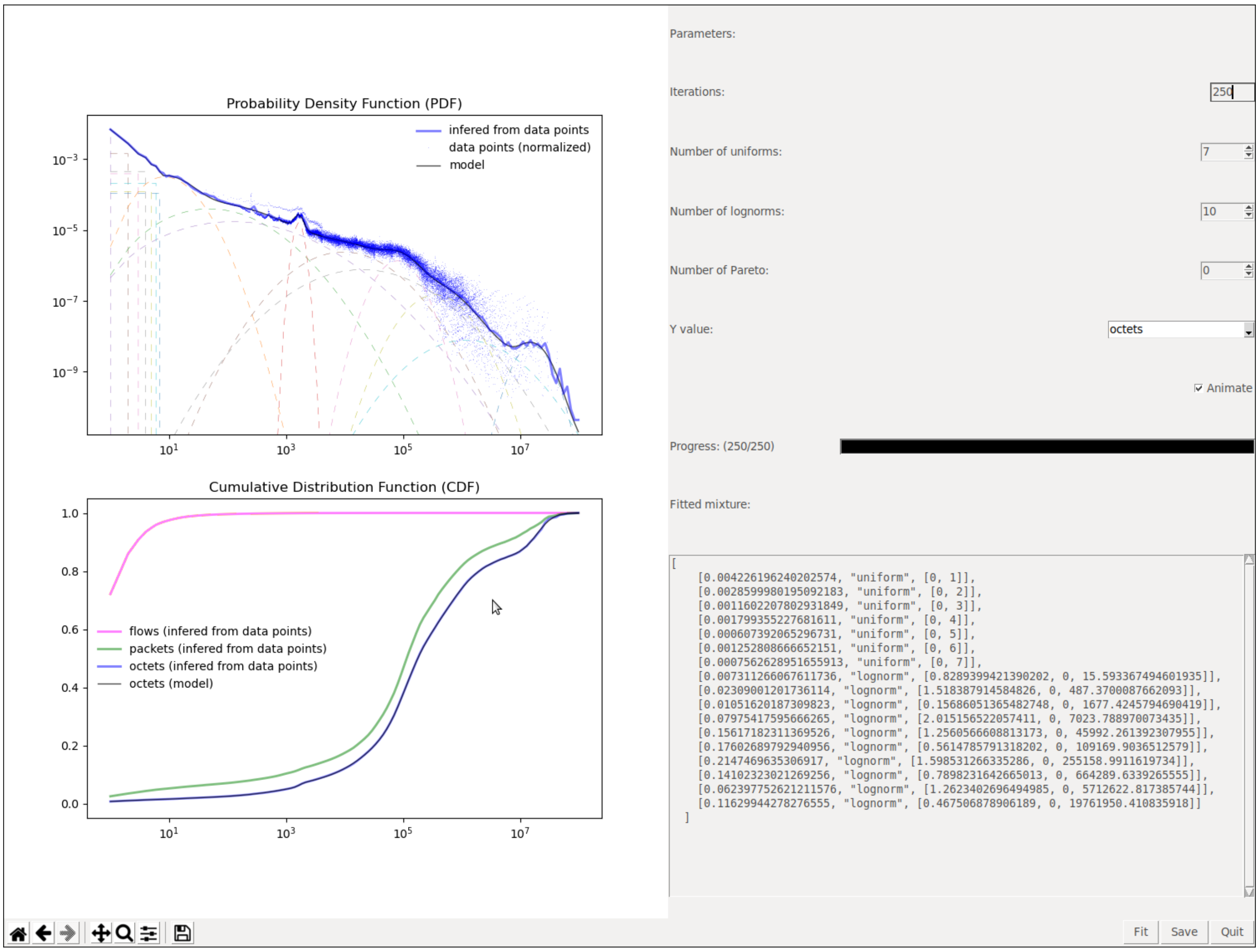
Listing 7: Plotting empirical flow distributions and mixture models.

```
$ flow-models-plot histograms/all/length.csv
    my_mixture/
```

Finally, the constructed model can be used to generate flows. The tool `generate` can be used for that purpose. It is not particularly useful on its own. Instead, its goal is to provide a reference and serve as an example of how to properly generate traffic from flow models. In the case presented below, the model from my_mixture directory is used to generate 20 flow records in `csv_flow` format using two different seeds.

Listing 8: Generating artificial flows from a model.

```
$ flow-models-generate -x length -s 20 --seed 0
    my_mixture/
$ flow-models-generate -x length -s 20 --seed 1
    my_mixture/
```


Probability Density Function (PDF)
infered from data points
data points (normalized)
model
10^-3
10^-5
10^-7
10^-9
10^1
10^3
10^5
10^7
Cumulative Distribution Function (CDF)
flows (infered from data points)
packets (infered from data points)
octets (infered from data points)
octets (model)
1.0
0.8
0.6
0.4
0.2
0.0
10^1
10^3
10^5
10^7
Fit
Save
Quit

Parameters:

Iterations: 250

Number of uniforms: 7

Number of lognorms: 10

Number of Pareto: 0

Y value: octets

☑ Animate

Progress: (250/250)

Fitted mixture:

```
[
    [0.004226196240202574, "uniform", [0, 1]],
    [0.0028599980195092183, "uniform", [0, 2]],
    [0.0011602207802931849, "uniform", [0, 3]],
    [0.001799355227681611, "uniform", [0, 4]],
    [0.000607392065296731, "uniform", [0, 5]],
    [0.001252808666652151, "uniform", [0, 6]],
    [0.0007562628951655913, "uniform", [0, 7]],
    [0.007311266067611736, "lognorm", [0.8289399421390202, 0, 15.593367494601935]],
    [0.02309001201736114, "lognorm", [1.518387914584826, 0, 487.3700087662093]],
    [0.01051620187309823, "lognorm", [0.15686051365482748, 0, 1677.4245794690419]],
    [0.07975417595666265, "lognorm", [2.015156522057411, 0, 7023.788970073435]],
    [0.15617182311369526, "lognorm", [1.2560566608813173, 0, 45992.261392307955]],
    [0.17602689792940956, "lognorm", [0.5614785791318202, 0, 109169.9036512579]],
    [0.214746963530917, "lognorm", [1.598531266335286, 0, 255158.9911619734]],
    [0.14102323021269256, "lognorm", [0.789823164266501З, 0, 664289.6339265555]],
    [0.06239775262121157б, "lognorm", [1.2623402696494985, 0, 5712622.817385744]],
    [0.11629944278276555, "lognorm", [0.467506878906189, 0, 19761950.410835918]]
  ]
```

Figure 3: The interactive `fit` tool.

5. Impact

The uninterrupted growth of Internet traffic is being observed for many years. Recently, the forced transition to remote work and education boosted its rate even more [18]. As limits of physical infrastructure are being reached, efficient traffic engineering is becoming a necessity for network operators to maintain the desired quality of service. Fine-grained flow traffic engineering is considered to be one of the most promising solutions [19].

However, the efficiency of flow-based control strongly depends on traffic characteristics, and thus, should be assessed based on realistic and accurate flow models. To reliably evaluate such ideas, realistic distributions of flows' length and size must be ensured. Due to a lack of adequate models, researchers make arbitrary and oversimplified assumptions, which often do not correspond to reality. This negatively impacts the credibility of results presented in numerous papers. Moreover, different and arbitrary assumptions in various works exclude the possibility to effectively compare different solutions. Thus, despite many SDN-based TE solutions that have been proposed recently [19], their adoption by operators is very limited.

We expect that the presented framework will help to solve these issues. It goes along with the reproducible research effort, which is still being neglected in the computer networking field [20]. It is also designed with big data analysis capabilities in mind. Specifically, it supports out-of-core computing, making it possible to analyze data that exceeds available memory. Moreover, most processing steps can be scaled horizontally using the map-reduce technique. Therefore, provided implementation is not limited in terms of the number of processed flow records, which makes it suitable for researchers and network operators who experience significant traffic volumes. We hope that they will share their models.

The framework was already used to construct flow length and size models from a trace of 4 billion flows recorded in a campus network [21]. Statistics and models provided in that paper are already being used in various research areas, including protocol performance [22], deep learning traffic prediction [23], intrusion detection [24] and server CPU scheduling algorithms [25].

However, other types of networks (e.g. datacenter networks) can exhibit significantly different traffic characteristics. This means that models presented in [21] could not be suitable for all research scenarios. Thus, in this publication, we want to provide the framework itself to a wider audience of researchers and network operators. This should allow them to create similar models based on their networks' traffic. We expect that they will share their models and thus provide a set of research data to be used in a variety of networking experiments, improving their credibility, comparability and reproducibility. Usage of realistic, accurate, and repeatable traffic models will be a key factor in providing comparability and reproducibility in flow-based networking research.

6. Limitations and further research

The framework operates on flow records, which are collected according to some specific flow definition. Whereas NetFlow flow is a sequence of packets in the transport layer, network layer flows (defined only by source and destination IP addresses) are also used in some research areas. Flow timeout rules are also strictly associated with flow definition. This means that using the framework to model flows present in a packet trace with different definitions (e.g. various timeout values) requires repeated creation of several flow record files which would be the input for the framework. Such files can consume a lot of storage.

Another limitation of the framework is that General Mixture Models created with the `fit` tool model all flow features independently. Such models are sufficient for many use cases. Nevertheless, more advanced models, incorporating relationships between features and header values, would be useful in machine learning applications, like the prediction of flow features based on header values in the first packet [23].

The framework can also be further extended with capabilities to model or simulate particular mechanisms and techniques. This includes for example elephant-based flow table management [10] or techniques like mirroring first-N packets of a flow to CPU for inspection. A very promising direction would be creating an interface to connect the framework with machine learning libraries, like *Scikit-learn*, *Keras* or *OpenAI Gym*.

7. Conclusions

In this paper, we introduced the *flow-models* framework, a toolset for creating statistical models of IP network flows. The software provides flow features distribution analysis and modeling capabilities, not present in the exiting packages. In particular, the framework allows to merge flow records split during the collection process, calculate histograms of flow features distributions, and finally to create General Mixture Models fitting those distributions. The framework also provides several auxiliary functionalities, like flow records sorting or histogram distribution and model plotting. With the framework, researchers and network operators can create flow models based on their networks' traffic. Created models can be used in networking research, both as an input in analytical calculations or to generate realistic traffic in simulations. We expect that the presented framework will significantly enhance the quality of networking research by improving its reproducibility and comparability.

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.

Acknowledgments

The research was carried out with the support of the project "Intelligent management of traffic in multi-layer Software-Defined Networks" founded by the Polish National Science Centre under project no. 2017/25/B/ST6/02186.

References

[1] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-Defined Networking: A Comprehensive Survey, Proceedings of the IEEE 103 (1) (2015) 14–76. doi:10.1109/JPROC.2014.2371999.

[2] J. Quittek, J. Zseby, B. Claise, S. Zander, IPFIX Requirements, RFC 3917 (2010).
URL http://tools.ietf.org/html/rfc3917

[3] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, W. Chou, A roadmap for traffic engineering in SDN-OpenFlow networks, Computer Networks 71 (2014) 1–30. doi:10.1016/j.comnet.2014.06.002.

[4] R. Wójcik, A. Jajszczyk, Flow Oriented Approaches to QoS Assurance, ACM Comput. Surv. 44 (1) (Jan. 2012). doi:10.1145/2071389.2071394.

[5] S. Shin, L. Xu, S. Hong, G. Gu, Enhancing Network Security through Software Defined Networking (SDN), in: 2016 25th International Conference on Computer Communication and Networks (ICCCN), 2016, pp. 1–9. doi:10.1109/ICCCN.2016.7568520.

[6] J. J. Garcia-Lunes-Aceves, Loop-free routing using diffusing computations, IEEE/ACM Transactions on Networking (TON) 1 (1) (1993) 130–141. doi:10.1109/90.222913.

[7] D. Bertsekas, Dynamic behavior of shortest path routing algorithms for communication networks, IEEE Transactions on Automatic Control 27 (1) (1982) 60–74. doi:10.1109/TAC.1982.1102884.

[8] Z. Wang, J. Crowcroft, Analysis of Shortest-Path Routing Algorithms in a Dynamic Network Environment, ACM SIGCOMM Computer Communication Review 22 (2) (1992) 63–71. doi:10.1145/141800.141805.

[9] P. Jurkiewicz, R. Wójcik, J. Domżał, A. Kamisiński, Testing implementation of FAMTAR: Adaptive multipath routing, Computer Communications 149 (2020) 300–311. doi:10.1016/j.comcom.2019.10.029.

[10] P. Jurkiewicz, Boundaries of Flow Table Usage Reduction Algorithms Based on Elephant Flow Detection, in: 2021 IFIP Networking Conference (IFIP Networking), 2021, pp. 1–9. doi:10.23919/IFIPNetworking52078.2021.9472832.

[11] P. Haag, Watch your Flows with NfSen and NFDUMP, in: 50th RIPE Meeting, 2005.

[12] S. v. d. Walt, S. C. Colbert, G. Varoquaux, The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering 13 (2) (2011) 22–30. doi:10.1109/MCSE.2011.37.

[13] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, et al., SciPy 1.0: fundamental algorithms for scientific computing in Python, Nature Methods 17 (3) (2020) 261–272. doi:10.1038/s41592-019-0686-2.

[14] J. D. Hunter, Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering 9 (3) (2007) 90–95. doi:10.1109/MCSE.2007.55.

[15] W. McKinney, et al., pandas: a foundational python library for data analysis and statistics, Python for High Performance and Scientific Computing 14 (9) (2011).

[16] R. Hofstede, I. Drago, A. Sperotto, R. Sadre, A. Pras, Measurement artifacts in netflow data, in: International Conference on Passive and Active Network Measurement, Springer, 2013, pp. 1–10. doi:10.1007/978-3-642-36516-4_1.

[17] A. P. Dempster, N. M. Laird, D. B. Rubin, Maximum Likelihood from Incomplete Data via the EM Algorithm, Journal of the Royal Statistical Society. Series B (Methodological) 39 (1) (1977) 1–38. doi:10.1111/j.2517-6161.1977.tb01600.x.

[18] CISCO, Cisco Annual Internet Report (2018–2023) White Paper (2020).
URL https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[19] A. Mendiola, J. Astorga, E. Jacob, M. Higuero, A Survey on the Contributions of Software-Defined Networking to Traffic Engineering, IEEE Communications Surveys Tutorials 19 (2) (2017) 918–953. doi:10.1109/COMST.2016.2633579.

[20] Q. Scheitle, M. Wählisch, O. Gasser, T. C. Schmidt, G. Carle, Towards an ecosystem for reproducible research in computer networking, Reproducibility '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 5–8. doi:10.1145/3097766.3097768.

[21] P. Jurkiewicz, G. Rzym, P. Boryło, Flow length and size distributions in campus Internet traffic, Computer Communications 167 (2021) 15–30. doi:10.1016/j.comcom.2020.12.016.

[22] V. Adarsh, P. Schmitt, E. Belding, MPTCP Performance over Heterogenous Subpaths, in: 2019 28th International Conference on Computer Communication and Networks (ICCCN), 2019, pp. 1–9. doi:10.1109/ICCCN.2019.8847086.

[23] C. Hardegen, B. Pfülb, S. Rieger, A. Gepperth, Predicting Network Flow Characteristics Using Deep Learning and Real-World Network Traffic, IEEE Transactions on Network and Service Management 17 (4) (2020) 2662–2676. doi:10.1109/TNSM.2020.3025131.

[24] L. Han, Y. Sheng, X. Zeng, A Packet-Length-Adjustable Attention Model Based on Bytes Embedding Using Flow-WGAN for Smart Cybersecurity, IEEE Access 7 (2019) 82913–82926. doi:10.1109/ACCESS.2019.2924492.

[25] A. Rucker, M. Shahbaz, T. Swamy, K. Olukotun, Elastic RSS: Co-Scheduling Packets and Cores Using Programmable NICs, in: Proceedings of the 3rd Asia-Pacific Workshop on Networking 2019, APNet '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 71–77. doi:10.1145/3343180.3343184.