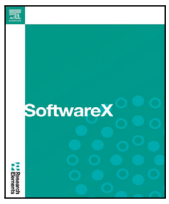
Traffic weaver: Semi-synthetic time-varying traffic generator based on averaged time series

SoftwareX 28 (2024) 101946

Contents lists available at ScienceDirect

SoftwareX

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



Original software publication

Traffic weaver: Semi-synthetic time-varying traffic generator based on averaged time series

Piotr Lechowicz [a,b,*], Aleksandra Knapińska [a], Adam Włodarczyk [a], Krzysztof Walkowiak [a]

[a] Department of Systems and Computer Networks, Wrocław University of Science and Technology, Wrocław, Poland
[b] Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden

ARTICLE INFO

Keywords:
Time varying traffic
Telecommunication network
Semi-synthetic traffic generator

ABSTRACT

Traffic Weaver is a Python package developed to generate a semi-synthetic signal (time series) with finer granularity, based on averaged time series, in a manner that, upon averaging, closely matches the original signal provided. The key components utilized to generate the signal encompass oversampling, recreating from average with a given strategy, stretching to match the integral of the original time series, interpolating, smoothing, repeating, applying trend, and adding noise. The primary motivation behind Traffic Weaver is to furnish semi-synthetic time-varying traffic in telecommunication networks, facilitating the development and validation of traffic prediction models, as well as aiding in the deployment of network optimization algorithms tailored for time-varying traffic.

Code metadata

Current code version	1.5.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00196
Permanent link to Reproducible Capsule	https://codeocean.com/capsule/8804531/tree/v3
Legal Code License	GNU AGPL
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	Python ≥ 3.9
If available Link to developer documentation/manual	http://w4k2.github.io/traffic-weaver/
Support email for questions	piotr.lechowicz@pwr.edu.pl

1. Motivation and significance

In telecommunication networks, such as backbone optical networks, many small end-to-end transmissions between individual users and devices combine into time-varying traffic, representing aggregated traffic over time. Thus, daily and weekly patterns can be observed in network traffic due to increased user activity in certain periods. Driven by the paradigm of self-driving and self-healing networks, traffic prediction, and anomaly detection gained significant research community attention in recent years. However, the community faces the problem of lacking real data, allowing for thorough experiments. Network operators are often constrained by legal aspects and cannot share the details of traffic generated by their customers. In turn, many researchers can have access either to small exemplary data or to averaged data without sufficient quality. To this end, the community relies on artificially generated data with various distributions and patterns based on their domain knowledge (e.g., [1–6]). Exemplary packet generating tools are iPerf [7] and Cisco TRex [8]. TrafPy [1] generates data-center networks traffic. Several works present only the methodology of generating the traffic without supplying the software. A 5G network traffic generator is presented in [9]. A self-similar traffic-generator with heavy tails based on wavelets theory is discussed in [10]. An AI-based generators have been presented in [11,12]. However, predicting and detecting changes in real data can bring significantly more challenges than artificially generated ones. Additionally, extensive experiments performed on a large pool of appropriately diverse datasets are necessary for the development and thorough evaluation of the designed algorithms [13].

* Corresponding author at: Department of Systems and Computer Networks, Wrocław University of Science and Technology, Wrocław, Poland.
E-mail address: piotr.lechowicz@pwr.edu.pl (Piotr Lechowicz).

https://doi.org/10.1016/j.softx.2024.101946
Received 26 March 2024; Received in revised form 2 October 2024; Accepted 20 October 2024
Available online 30 October 2024

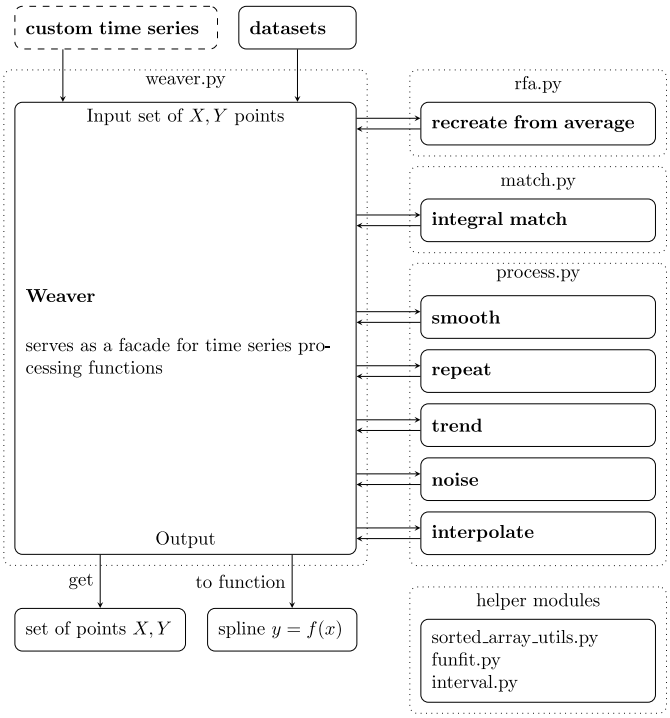


Fig. 1. Software architecture.

Therefore, some of the tools rely on captured real data statistics. In [14], a packet generator is presented based on the on/off sources with various distributions. Swing [15] is a tool capturing stationary empirical cumulative distributions of packet parameters from packet traces and generating traffic according to that. However, the available solutions do not recreate dynamic properties of the traffic over time.

The purpose of Traffic Weaver is to generate new time-varying data based on an already available sample of data, i.e., to create semi-synthetic data when the size of real data is either insufficient or the time points at which the data were measured are too rare. In consequence, it can generate larger and diverse datasets with similar traffic patterns based on the original traffic. In particular, the software has been used in scientific research to create semi-synthetic time-varying traffic to develop and evaluate traffic prediction models [16,17] and multi-layer network optimization algorithms using generated time-varying connection requests (intents) [18,19]. Semi-synthetic data allowed a thorough evaluation of the developed algorithms in real-world settings and various desired characteristics.

The aim of Traffic Weaver is to read averaged time series and to create a semi-synthetic signal with finer granularity that, after averaging, matches the original signal provided. The following tools are applied to recreate the signal: recreating from average with a given strategy, stretching to match the integral of the original time series, interpolating, smoothing, repeating, applying trend, and adding noise. Software users may provide exemplary data for the investigated problem or use one of the available datasets to create semi-synthetic time-varying traffic by applying various trends and noise profiles. The increased input data size allows for a more thorough investigation of the problem. Moreover, the ability to create datasets with specific characteristics enables detailed testing of the developed algorithms in various conditions [20].

2. Software description

2.1. Software architecture

Fig. 1 presents an overview of the software architecture. *Weaver*, located in the *weaver.py* module, wraps the supplied signal (time series) data, and provides an interface for processing functionalities. The time series can be either specified by the user or obtained from the embedded example datasets. Individual functionalities provided by the *Weaver* are delegated to other modules, e.g., recreating from average functionality is located in the *rfa.py* module. However, it is possible to use individual functionalities from the corresponding modules regardless of wrapping the time series into *Weaver*. *Weaver* allows retrieving the processed data either as sampled points or as a continuous spline function.

2.2. Software functionalities

This section describes the main functionalities provided by the Traffic Weaver. In the below description, the term *interval* refers to the distance between two sampled points in the input time series. The aim of the *Weaver* is to create an output time series with multiple points inserted in each interval.

- Class *Weaver(x, y)*

  Weaver is an interface for recreating the signal. It takes as an input a time series provided as two lists containing values of independent and dependent variables. It delegates processing to other modules and allows to retrieve the recreated signal either as lists of values of independent and dependent variables or as a spline, using the *get()* and *to_function()* methods, respectively.
- Recreating from average

  Recreating from average is a recreation of a signal with finer sampling granularity based on the supplied strategy. The number of created points between each interval (pair of points in the original time series) is provided as a parameter. The strategy determines how the created time series transits between points, i.e., how the new points are located. The software provides several strategies, namely, *ExpAdaptiveRFA()*, *ExpFixedRFA()*, *LinearAdaptiveRFA()*, *LinearFixedRFA()*, *PiecewiseConstantRFA()*, *CubicSplineRFA()*. E.g., *ExpAdaptiveRFA()* creates an adaptive transition window for each interval by combining linear and exponential functions. The size of the window is inversely proportional to the change of the function value on both edges of the interval, i.e., if the function value has a higher change on the right side than on the left side of the interval, the right side transition window is smaller than the left one.

  The *Weaver* class provides the *recreate_from_average(n, rfa_class)* method that delegates the execution to the *rfa.py* module and takes as an input number of samples *n* in each interval after oversampling, recreating from average strategy *rfa_class* inheriting *AbstractRFA()* class, and a dictionary of parameters passed to the selected strategy.
- Integral matching

  It aims to reshape the time series to match its integral to the integral of the reference function over the same domain (the original time series). It does that by stretching the signal in intervals such that the integral in the interval of the current time series is equal to the integral of the same interval in the reference function. The points in each interval are transformed inversely proportionally to the exponential value of distance from the interval center. The integral for the recreated function and for the original function can be calculated using either trapezoidal or rectangular rule.

  The *Weaver* class provides the *integral_match(target_function_integral_method, reference_function_integral_method)* method that delegates the execution to the match module and takes as an input a dictionary of parameters passed to the matching function. The time series currently stored in the Weaver is matched with a reference to the originally passed function to the class. *target_function_integral_method* and *reference_function_integral_method* specifies how the integral is calculated for the target and reference function, respectively.

- Smoothing

  It smooths a function using smoothing splines.

  The *Weaver* class provides the *smooth(s)* method to delegate the execution to the smoothing function and takes *s* as an argument. The argument *s* is a smoothing condition that controls the tradeoff between closeness and smoothness of the fit. Larger *s* means more smoothing, while smaller values of *s* indicate less smoothing. If *s* is None, its "good" value is calculated based on the number of samples and standard deviation.
- Repeating

  It repeats the time series a given number of times, resulting in a long-term time series containing periodic, e.g., daily or weakly, patterns.

  The *Weaver* class provides the *repeat(n)* method to repeat the time series. *n* is an argument passed to the function, defining how many times to repeat the time series.
- Trending

  It applies a trend to the time series according to the specified function. It allows adding a long-term trend to the time series, e.g., constant dependent variable increase over time.

  The *Weaver* class provides the *trend(trend_func, normalized)* method to apply a trend to the processed time series. The argument *trend_function* is a callable that shifts the value for the dependent variable based on the value of the independent variable. The callable takes one argument – the value of the independent variable – and has to return the shift value for the dependent variable. Argument *normalized* is a boolean determining if the trend function is normalized to the range of $[0, 1]$.
- Noising

  It applies a constant or changing over time Gaussian noise to the time series, expressed as signal to noise ratio.

  The *Weaver* class provides the *noise(snr)* method to apply noise to the signal. The argument *snr* defines the signal-to-noise ratio of a function either as a scalar value or as a list of changing values over time whose size matches the size of the independent variable.
- Interpolating

  It applies an interpolation of time series using specified points.

  The *Weaver* class provides the *interpolate(n, new_x, method)* method to interpolate the time series. The argument *n* is the number of fixed space samples in the new interpolated function. *new_x* is a list of points where to evaluate the interpolated function. It overrides the *n* parameter. Range should be the same as the original function domain. Interpolation is done according to the *method* parameter. Supported strategies are *linear*, *constant*, *cubic* and *spline*.
- Truncating

  It truncates a time series to a specified range. If specified points are not present in the time series, the closest points are selected such that the specified range is included.

  The *Weaver* class provides the *truncate(x_left, x_right, x_left_as_ratio, x_right_as_ratio)* method to truncate time series. Arguments *x_left* and *x_right* are values in the independent variable array to which truncate its content from the left and right side, respectively. Arguments *x_left_as_ratio* and *x_right_as_ratio* are boolean that determine if *x_left* and *x_right* are treated as ratios of the independent variable range to truncate from the left and right, respectively.
- Normalizing

  It normalizes the independent and dependent variable to the specified range.

  The *Weaver* class provides the *normalize_x(min_val, max_val)* and *normalize_y(min_val, max_val)* method to normalize the independent and dependent variable, respectively. Arguments *min_val* and *max_val* are the minimum and maximum values for normalization.
- Datasets

  The *datasets* module provides collected network traffic datasets.

  Network operators often collect data about traffic generated by their customers. However, due to legal aspects, exact values are not shared with the public. However, the community can access averaged or summary data presented in a form of plots. This module provides a set of datasets recreated from graphical plots which can be further resampled and regenerated using Traffic Weaver.

3. Illustrative examples

Fig. 2 shows a general usage example. Based on the provided original averaged time series (a), the signal is *n*-times oversampled and recreated from average values with a predefined strategy (b). Next, it is stretched to match the integral of the input time series function (c). Further, it is smoothed with a spline function (d). In order to create weekly semi-synthetic data, the signal is repeated seven times (e), applying a long-term trend consisting of sinusoidal and linear functions (f). Finally, the noise is introduced to the signal, starting from small values and increasing over time (g). To validate the correctness of the applied processing, (h) presents the averaged two periods of the created signal, showing that they closely match the original signal (except the applied trend).

3.1. Minimal processing example

Traffic Weaver is an open-source Python module released under MIT license and versioned in the public *Python Package Index* (PyPI) repository. It can be installed using *pip* package manager.

```
pip install traffic-weaver
```

Traffic Weaver import is done with the standard import command.

```
import traffic_weaver
```

To load one of the exemplary datasets, use the *load_dataset* function specifying the name of the dataset.

```
# load example dataset with average measurements over 1h
data = traffic_weaver.datasets.\
    load_dataset('sandvine_tiktok')
```

The *traffic_weaver* module provides the *Weaver* class that serves as an API to other processing capabilities. The *Weaver(x, y)* constructor takes the time series independent and dependent variables as arguments, denoted as *x* and *y*, respectively. Alternatively, the *Weaver.from_2d_array(data)* factory method takes a 2-D array as argument containing dependent and independent variables.

```
# create Weaver instance
wv = traffic_weaver.Weaver.from_2d_array(data)
```

Further signal processing is applied through the *Weaver* methods. Most of the methods return an instance to the *Weaver* itself, allowing for chaining the processing commands.

```
# process it creating samples every minute
wv.recreate_from_average(60).\
    integral_match().smooth(1.0).noise(snr=30)
```

To obtain the created new time series, call either *Weaver's get()* or *to_function()* methods. Next, visualize time series with matplotlib – the original data, created semi-synthetic traffic, and averaged semi-synthetic traffic to verify that the integral of semi-synthetic traffic does not differ much from the one in original signal. The result of the below listing is presented in Fig. 3.

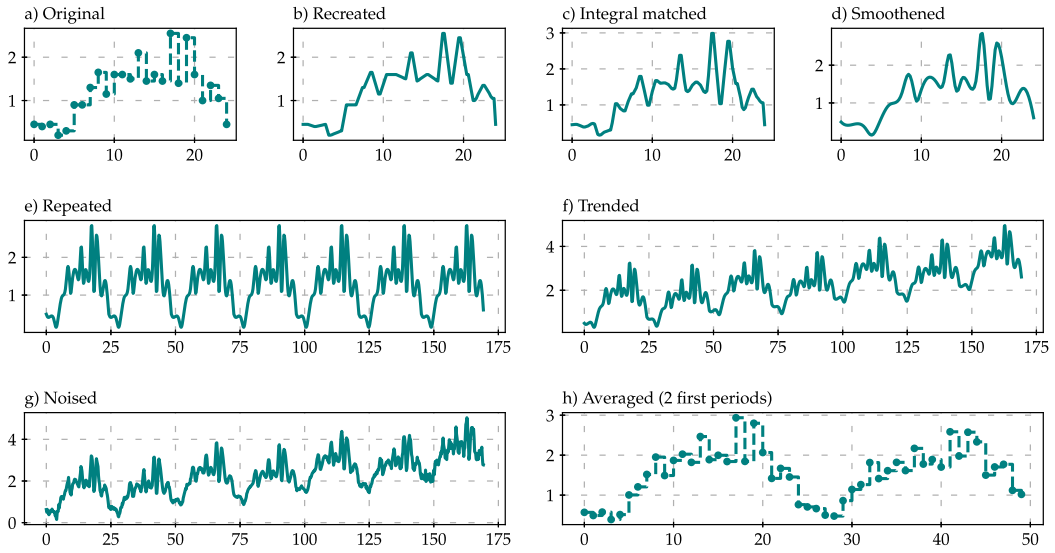


Fig. 2. Regenerating time-varying traffic from the averaged traffic sample: original traffic (a); recreated from average (b); matched with the integral of the original signal (c); smoothed (d); repeated 7 times (e); trended with sinusoidal and linear function (f); noised (g); averaged (h).

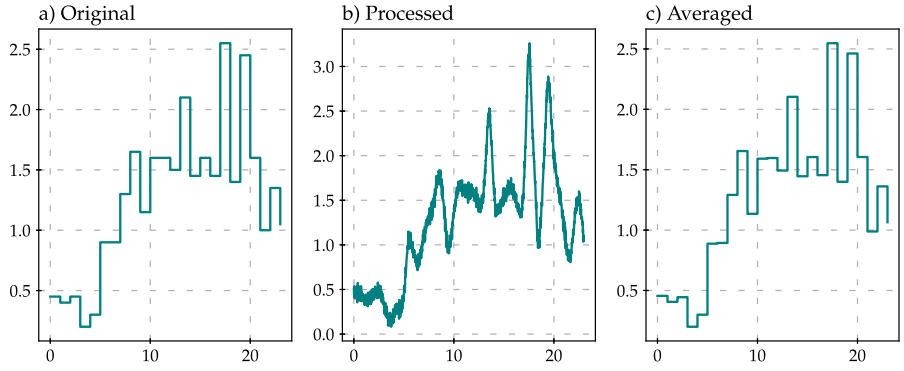


Fig. 3. Minimal processing example of recreating time-varying traffic: original traffic (a); processed by recreating from average, matching integral with the original signal, smoothing and noising (b); averaged (c).

```
import matplotlib.pyplot as plt

# plot original signal
fig, axes = plt.subplots(nrows=1, ncols=3,
                         figsize=(14, 4))
axes[0].plot(*wv.get_original(), drawstyle="steps-post")

# plot modified signal
axes[1].plot(*wv.get())

# plot averaged signal
x, y = traffic_weaver.process.average(*wv.get(), 60)
axes[2].plot(x, y, drawstyle="steps-post")

axes[0].set_title("a) Original", loc="left")
axes[1].set_title("b) Processed", loc="left")
axes[2].set_title("c) Averaged", loc="left")
plt.show()
```

4. Impact

The networking community lacks a public data repository for research purposes and the development of new optimization methods based on network traffic. Existing analyses of real traffic data (e.g., [21–23]), collected by the authors over long periods, usually stop at the data characterization stage and are not further used in networking research, nor are they easily accessible. Traffic Weaver closes this gap, allowing easy access to data and enabling a thorough evaluation of the developed algorithms. Using various options provided in Traffic Weaver, the created methods can be tested in diverse traffic conditions representing actual traffic patterns. In turn, the package allows a fair and versatile algorithm development, evaluation, and comparison with the existing solutions. It also helps in gaining insights into the operation of various methods in specific traffic conditions considering parameters such as noise levels, trends, and traffic types. These parameters are impossible to steer using the available sparse raw data. Moreover, Traffic Weaver is implemented in Python, which is the primary programming language used for the development of machine learning methods [24]. The proposed tool does not contain an embedded traffic generator based

on the analytical formulas. Therefore, it requires as an input either user-provided traffic or one of the supplied datasets. Moreover, signal recreation from average with user-specified parameters for probability distributions is not possible.

5. Conclusions

This article presents Traffic Weaver – a semi-synthetic time-varying traffic generator. The software creates new datasets based on either existing examples of real data supplied as datasets or user-specified data and enables adding desired characteristics. Through a variety of processing methods, including recreating from average, interpolating, integral matching, smoothing, repeating, trending, and noising methods, the package allows a thorough evaluation of created optimization and prediction methods based on the network traffic. The available example datasets provide a versatile entry for networking research. In the future, we plan to extend the software with signal recreation with various probability distributions varying over time.

CRediT authorship contribution statement

Piotr Lechowicz: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Aleksandra Knapińska:** Writing – review & editing, Writing – original draft, Visualization, Validation, Resources, Methodology, Investigation, Data curation, Conceptualization. **Adam Włodarczyk:** Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology. **Krzysztof Walkowiak:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Project administration, Methodology, Funding acquisition, Formal analysis, Conceptualization.

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

This work was supported by the National Science Center, Poland under Grant 2019/35/B/ST7/04272.

Data availability

Data included in the package.

References

[1] Parsonson CW, Benjamin JL, Zervas G. Traffic generation for benchmarking data centre networks. Opt Switch Netw 2022;46:100695. http://dx.doi.org/10.1016/j.osn.2022.100695.

[2] Valkanis A, Papadimitriou G, Nicopolitidis P, Beletsioti GA, Varvarigos E. A traffic prediction assisted routing algorithm for elastic optical networks. In: International conference on communications, computing, cybersecurity, and informatics. IEEE; 2021, p. 1–6. http://dx.doi.org/10.1109/CCCI52664.2021.9583188.

[3] Włodarczyk A, Lechowicz P, Szostak D, Walkowiak K. An algorithm for provisioning of time-varying traffic in translucent SDM elastic optical networks. In: 22nd international conference on transparent optical networks. IEEE; 2020, p. 1–4. http://dx.doi.org/10.1109/ICTON51198.2020.9203045.

[4] Petale S, Lin S-C, Matsuura M, Hasegawa H, Subramaniam S. PRODIGY: A progressive upgrade approach for elastic optical networks. In: IEEE global communications conference. IEEE; 2023, p. 2129–34. http://dx.doi.org/10.1109/GLOBECOM54140.2023.10437935.

[5] Han Y, Yoo J-H, Hong JW-K. Poisson shot-noise process based flow-level traffic matrix generation for data center networks. In: 2015 IFIP/IEEE international symposium on integrated network management. 2015, p. 450–7. http://dx.doi.org/10.1109/INM.2015.7140322.

[6] Han Y, Seo S-s, Hwang C, Yoo J-H, Hong JW-K. Flow-level traffic matrix generation for various data center networks. In: 2014 IEEE network operations and management symposium. 2014, p. 1–6. http://dx.doi.org/10.1109/NOMS.2014.6838394.

[7] iPerf3. https://software.es.net/iperf/. [Accessed 21 September 2024].

[8] Cisco TRex. https://trex-tgn.cisco.com. [Accessed 21 September 2024].

[9] Ziazet JM, Jaumard B, Duong H, Khoshabi P, Janulewicz E. A dynamic traffic generator for elastic 5G network slicing. In: 2022 IEEE international symposium on measurements & networking (M&N). 2022, p. 1–6. http://dx.doi.org/10.1109/MN55117.2022.9887734.

[10] Savu-Jivanov A, Isar A, Stolojescu-Crisan C, Gal J. Network self-similar traffic generator with variable hurst parameter. In: 2020 international symposium on electronics and telecommunications. 2020, p. 1–4. http://dx.doi.org/10.1109/ISETC50328.2020.9301120.

[11] Alsulami K, Zhang J, Ye F. Improvement on a traffic data generator for networking AI algorithm development. In: 2021 IEEE global communications conference. 2021, p. 1–6. http://dx.doi.org/10.1109/GLOBECOM46510.2021.9685616.

[12] Bikmukhamedov RF, Nadeev AF. Multi-class network traffic generators and classifiers based on neural networks. In: 2021 systems of signals generating and processing in the field of on board communications. 2021, p. 1–7. http://dx.doi.org/10.1109/IEEECONF51389.2021.9416067.

[13] Hoffmann F, Bertram T, Mikut R, Reischl M, Nelles O. Benchmarking in classification and regression. Wiley Interdiscip Rev Data Min Knowl Discov 2019;9(5):e1318. http://dx.doi.org/10.1002/widm.1318.

[14] Varet A, Larrieu N. How to generate realistic network traffic? In: 2014 IEEE 38th annual computer software and applications conference. 2014, p. 299–304. http://dx.doi.org/10.1109/COMPSAC.2014.40.

[15] Vishwanath KV, Vahdat A. Swing: Realistic and responsive network traffic generation. IEEE/ACM Trans Netw 2009;17(3):712–25. http://dx.doi.org/10.1109/TNET.2009.2020830.

[16] Knapińska A, Lechowicz P, Spadaro S, Walkowiak K. Agnostic prediction of multiple types of time-varying traffic in optical networks. In: IEEE global communications conference. IEEE; 2023, p. 1125–30. http://dx.doi.org/10.1109/GLOBECOM54140.2023.10436763.

[17] Ułanowicz B, Dopart D, Knapińska A, Lechowicz P, Walkowiak K. Combining random forest and linear regression to improve network traffic prediction. In: 23rd international conference on transparent optical networks. IEEE; 2023, p. 1–4. http://dx.doi.org/10.1109/ICTON59386.2023.10207506.

[18] Knapińska A, Lechowicz P, Spadaro S, Walkowiak K. On advantages of traffic prediction and grooming for provisioning of time-varying traffic in multilayer networks. In: 27th international conference on optical network design and modeling. IEEE; 2023, p. 1–6.

[19] Knapińska A, Lechowicz P, Spadaro S, Walkowiak K. Performance analysis of multilayer optical networks with time-varying traffic. In: 23rd international conference on transparent optical networks. IEEE; 2023, p. 1–4. http://dx.doi.org/10.1109/ICTON59386.2023.10207179.

[20] Stapor K, Ksieniewicz P, García S, Woźniak M. How to design the fair experimental classifier evaluation. Appl Soft Comput 2021;104:107219. http://dx.doi.org/10.1016/j.asoc.2021.107219.

[21] García-Dorado JL, Finamore A, Mellia M, Meo M, Munafo M. Characterization of ISP traffic: Trends, user habits, and access technology impact. IEEE Trans Netw Serv Manag 2012;9(2):142–55. http://dx.doi.org/10.1109/TNSM.2012.022412.110184.

[22] 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.

[23] Goścień R, Knapińska A, Włodarczyk A. Modeling and prediction of daily traffic patterns—WASK and SIX case study. Electronics 2021;10(14):1637. http://dx.doi.org/10.3390/electronics10141637.

[24] Nguyen G, Dlugolinsky S, Bobák M, Tran V, López García Á, Heredia I, et al. Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey. Artif Intell Rev 2019;52:77–124. http://dx.doi.org/10.1007/s10462-018-09679-z.