A Network Traffic Processing Library for ICS Anomaly Detection

Ondřej Ryšavý
Petr Matoušek
rysavy@vutbr.cz
matousp@fit.vutbr.cz
Brno University of Technology
Brno, CZ

ABSTRACT

Anomaly detection in industrial control systems based on traffic monitoring is one of the key components in securing these critical cyber-physical environments. Many anomaly detection methods have been proposed in the past decade. They are based on various principles stemming from signature detection, statistical analysis, or machine learning. Because of the lack of ICS communication datasets, their evaluation and mainly comparing their performance is problematic. If provided as a prototype implementation, the methods are implemented in various languages and require different input formats. In the present paper, we propose a library that can process ICS communication, extract required information, e.g., various packet-level or flow-level features, and provide the data to a user-specified anomaly detection method. It is possible to integrate the library in the system that automates the entire processing pipeline enabling us to conduct experiments with different methods while saving the time needed for manual data preparation.

CCS CONCEPTS

• **Security and privacy** → **Network security**; • **Networks** → **Network monitoring**.

KEYWORDS

Anomaly Detection, Industrial Control Systems, Network Traffic Classification, Network Traffic Processing

ACM Reference Format:
Ondřej Ryšavý and Petr Matoušek. 2021. A Network Traffic Processing Library for ICS Anomaly Detection. In *ECBS 2021: Engineering Computer Based Systems, May 26–27, 2021, Novi Sad, Serbia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3459960.3459963

1 INTRODUCTION

Detection of security threats in industrial networks became critical after criminals committed several serious attacks against critical systems worldwide. Research on anomaly detection for the ICS environment has produced numerous methods to identify the non-standard network traffic possibly indicating the attack. Though authors commonly evaluate published research methods, it is often difficult to reproduce them and compare them with other existing approaches. This is mainly because not all authors publish implementations of their experimental methods or datasets using for evaluation.

Recently, most approaches to network traffic classification and anomaly detection have been based on machine learning algorithms. Regardless of the algorithm used, heavy data preprocessing is required to extract features from the source data. In the case of network data, we are speaking about extracting either statistical information about the communication and/or selected fields and computed values from the network protocols. Researchers usually build their own ad-hoc solutions containing a wide range of existing single-purpose tools to implement network traffic processors.

Once the input data is ready, the data science part begins. The anomaly detection method's design means applying selected algorithms and repeating experiments until all parameters are set. The method provides the best result on datasets available. In the initial phase of the design, the interactive approach using, for instance, Jupyter Notebooks provides a suitable environment for researchers. Later, when the method's major contours are fixed, the automation of the process is required.

Anomaly detection methods are acceptable for deployment if their accuracy, reliability, and robustness can be demonstrated and thoroughly tested. Methods that work well in experiments may suffer various issues when applied in the real-world environment. In particular, an excessive false-positive rate may be very annoying for operators, eventually leading to disabling the method. To test the method, a rich collection of datasets is necessary. While the carefully collected ICS communication traffic from a controlled environment is used for design and immediate evaluation of the method, datasets collected in a real environment are necessary for further testing.

1.1 Motivation

While several data science libraries and frameworks offer rich environments for data analysis, visualization, and design of classification and anomaly detection methods, the data preparation phase usually requires the application of various CLI tools, which is a time-consuming and tedious activity. The data preparation phase's output can be a dataset represented as a CSV file or in some other suitable format. Sometimes the dataset needs to be rebuilt, which entails another cycle of data preparation activity. For the network-based anomaly detection methods, the data source is represented by packet traces. To extract the features from packet traces, packets

ECBS'21, May 26–27, 2021, Novi Sad, Serbia
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-9057-6/21/05…$5.00
https://doi.org/10.1145/3459960.3459963

need to be decoded, which requires specialized tools implementing the required protocol parsers.

The presented work's motivation is to improve the development life cycle of anomaly detectors that integrates the data preparation phase. It involves at least two different patterns of activities that are usually done using different sets of tools:

- The interactive pattern, which is suitable in the initial phases for performing exploratory data analysis identifying acceptable anomaly detection models.
- The batch pattern, which is used in the following stages of development when different methods and parameters are tested on large datasets to select the best available model.

With a carefully crafted environment, we believe that both activities can be supported by the same collection of tools, thus saving the transition between design phases and avoiding learning different tools.

1.2 Contributions

Our contributions are as follows:

- We design and implement a new packet capture preprocessing library that can be referenced from the anomaly detection design projects. The library provides functionality that covers packet capture reading, packet decoding, feature extraction, conversation tracking, and data exporting to a representation suitable for integration with machine learning toolkits.
- We perform a preliminary evaluation of the library's performance in terms of execution time for the most expected tasks. We compare it with the baseline represented by the tshark tool, one of the most used applications for packet traces processing.
- We provide a demonstration of the easy to use through the example of two AD methods. These methods work on different data sources, namely, flow records and timed packet series. It is shown that the library is flexible enough to support both cases.

While the library is still under development, we believe researchers can already use it for AD methods experiments. The implementation is publicly available on GitHub.

1.3 Paper Organization

The paper's remainder is organized as follows: Section 2 overviews the anomaly detection methods focused specifically on the ICS domain. Section 3 provides a discussion of related works, listing the known environments supporting the traffic classification development. Section 4 presents the architecture and main components of the proposed library. Section 5 provides the evaluation and demonstration of two use cases. The paper is concluded in section 6 by summarizing the current state and discussing the future work.

2 ANOMALY DETECTION IN ICS NETWORKS

Intrusion detection systems (IDS) have been adapted to protect industrial control systems. Anomaly-based detection systems that analyze network communication are specific types of IDS. The anomaly detection system is supposed to reveal differences between the actual system behavior and the one considered to be normal. Anomaly detection systems can work at various layers of the ISO/OSI model, for instance:

- A system observes the network-layer communication only. The normal ICS system behavior is determined in terms of, e.g., the common number of packets or amount of data transferred, expected hosts, and communication protocols.
- A system can learn or be instrumented [24] with the particular ICS model of the control process to detect any deviation from the expected behavior. It is required to decode ICS application protocols and provide other system-specific information for modeling the normal behavior and detecting anomalies.

Some systems do not perform full packet inspection, but they work with network flow records instead. While these systems' advantage is performance and scalability, their accuracy may be affected by the lack of information available.

Various methods and algorithms have been used for anomaly detection in ICS environments. Ahmed et al. [18] presented a survey with the generic framework of AD methods and categorized methods as classification-based, statistical, clustering, and those based on information theory. In the rest of this section, we present the selected methods roughly grouped according to their major characteristics.

2.1 Classification-based Methods

One-Class Support Vector Machine (OCSVM) was employed by Shang et al. in [19] to detect anomalies in Modbus communication. A normal communication behavior model is based on observation of Modbus function codes. Inoue et al. [20] compared Deep Neural Networks and one-class Support Vector Machines applied on logs from the Secure Water Treatment (SWaT) testbed showing that DNN has a better false-positive ratio while the SVM was able to detect more attacks. A method called TABOR was introduced by Lin et al. in [21]. It consists of timed automata and Bayes networks for profiling the normal operational behavior of SCADA systems. The normal behavior models are used as one-class classifiers to identify irregular behavioral patterns and dependencies. Support Vector Machine (SVM), Random Forest, k-nearest neighbor, and k-means clustering were considered for anomaly detection methods and compared using the synthetic ICS dataset by Anton et al. [13]. Anomaly detection employing neural networks has become popular recently [22]. Siegel [17] reported improvements over traditional machine learning methods between 5%-15% for two sensor datasets. Kreimel et al. gave a comparison of neural networks with a model-checking approach for monitoring SCADA systems using a real testbed representing the substation domain in [16]. Their experiments demonstrate that both methods provide accuracy greater than 90%.

2.2 Model-based Methods

Finite-automata are often considered as a suitable behavior model for ICS communication. Faisal, Cardenas, and Wool [24] proposed a DFA-based model for Modbus using the specification approach. Contrary to traditional learning approaches, the model is created mostly manually from the design documents. Zhanwei and Zenghui [25] used a discrete multi-input and multi-output (MIMO) system model to represent Modbus communication. Elbez et al. [23] applied an Auto-Regressive Fractionally Integrated Moving Average (ARFIMA) model to describe the GOOSE communication. It was shown that the statistical AD based on the Generalized Likelihood Ratio Test (GLRT) and the cumulative sum (CUSUM) is capable of detecting anomalies in GOOSE communication.

2.3 Time-series Analysis

Time series analysis methods aim at exploiting the characteristics of ICS communication, namely periodicity. Barbosa et al. [26] introduced a novel approach for learning periodic traffic patterns. The traffic is decomposed to a sequence of tokens consumed by a learner module, which processes each token to identify and characterize periodic activities. Learned cycle models are used to detect deviations in the traffic. The method is demonstrated on Modbus and MMS traffic. Lai et al. [12] built a traffic model based on a structural time series model for a chemical industry system. The structural model is given in terms of a trend item, a seasonal item, a cyclical item, and an irregular item. Model parameter estimation is done using standard Kalman filter recursions and the EM algorithm. Lin and Nadjm-Tehrani [14] dealt with the timing characteristics of spontaneous events in an IEC-104 network. The Inter-arrival time model and correlation model are created and consumed by the anomaly detector that generates alarms for unusual network behaviors.

3 RELATED WORK

The need to compare different traffic classification techniques was imminent about a decade ago when the pace of introducing new methods was significantly rising. As the answer to this concern, traffic classification design and evaluation systems were introduced. This section lists the most relevant systems, followed by a brief comparison with our library.

NeTraMark [2] was the first Internet traffic classification benchmark where eleven different state-of-the-art traffic classifiers were integrated. It enables the creation of new classifiers and their comparison by basic performance metrics. Also, the rich GUI with multiple visualization views is provided.

A long-term effort led to the development of the Traffic Identification Engine (TIE) [7]. TIE is an open-source tool for network traffic classification developed between 2008-2014, focusing on evaluation, comparison, and combination of different traffic classification techniques. The TIE focuses only on traffic classification, but it offers different operating modes, including real-time. The TIE was popular in the community and has been widely used to design and evaluate traffic classifiers.

A similar system, TrafficS [3], is a real-time network traffic classification and benchmarking tool. It features a traffic sampling method suitable to design and evaluate high-performance classifiers. The system tracks changes in statistical features to discover the most significant ones that affect the classification performance.

The Piper framework [7] was recently developed, which implements a unified network traffic processing pipeline to supply data for different ML applications. By consolidating data preprocessing, it is possible to deploy multiple ML-based analysis modules for real-time applications.

Some other tools exist that provide an environment suitable for classifier construction and evaluation. For instance, Network Traffic-based Application Identification (netAI) has been developed for identifying the applications of the corresponding network flows. The environment is a collection of standalone tools, including machine learning suite WEKA. The environment is quite flexible though it requires the deep knowledge of several different tools and some effort to integrate them correctly into the working processing pipeline.

The idea of our library is close to NeTraMark and TIE. Contrary to NeTraMark, our library does not provide a rich user interface in itself. Instead, the integration with Jupyter Notebook is considered, enabling us to use any available visualization package for data presentation and graphing. Contrary to TIE, TrafficS, and Piper, we do not primarily aim at providing an environment for real-time classifiers. Instead, we focus on providing a unified data processing pipeline that integrates the entire design lifecycle into a single development environment. We also support both packet-level and flow-level data sources. The full packet content is retained in the traffic storage. Also, the interactive design and experimentation is a key principle offered. The system offers comparable performance with the existing solutions when integrated with an efficient machine learning library.

4 LIBRARY OVERVIEW

The library is written in C# language targeting .NET Core framework, which enables it to run on all major OSes. The library implements the first two stages of the processing pipeline, as depicted in Figure 1. The individual components can also be used independently from projects that require packet capture processing functionality. The library can be referenced from a command-line tool or interactive Jupyter notebooks with .NET Core support. The first option is useful for implementing standalone tools in the later stages of the development. The second option is mainly utilized during the method design and prototyping, including interactive code execution and modification.

The main components of the library are as follows:

- Capture trace providers implement loading and storing packet from and to packet capture files.
- Protocol decoders parse the most common network protocols and selected ICS protocols (DLMS, DNP3, MODBUS/TCP, S7COMM). It is possible to implement new protocol parsers when required.
- Traffic store is the heart of the library. It is a (persistent) key-value database for storing packets and conversation, based on the FASTER key-value store providing high performance and supporting data larger than a memory.
- Conversations and packet processors provide methods that consume conversations or packets, respectively, and produce user-defined results.

The library is primarily used for packet trace preprocessing. The output data is produced by conversation and packet processors. When developing a new AD method, the user usually provides a conversation or packet processor implementation to obtain the

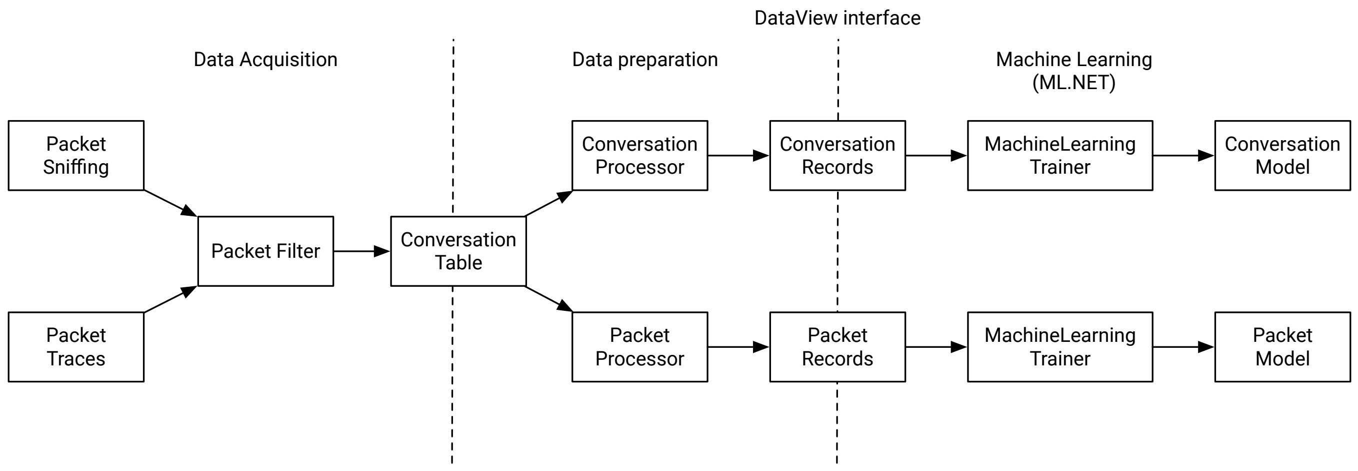

Figure 1: Processing pipeline of the training phase

required type of output data. The main phases of this procedure are described in the remainder of this section.

4.1 Injecting Packets

The processing pipeline starts by consuming a stream of packets, either read from a packet capture file or captured on a network interface. Often, not all traffic is interesting, and the BPF can be set to filter input packets. When the packets are injected, they are grouped into conversations. The conversation represents the bidirectional flow. Thus, the original packets and identified conversations are stored in the traffic data repository. This store provides fast insertion and updates. The implementation uses techniques to reduce memory allocations to improve performance.

4.2 Processing Conversations

When packets are injected into the library, they are immediately collected in conversations. The conversation is a transport-level bidirectional session defined as the tuple consisting of the following field: transport protocol, source IP address, source port, destination IP address, and destination port. No checking is made on whether the connection is the correct TPC session, i.e., filtering retransmitted segments, detecting missing segments, or checking SYN, FIN, and RST flags are not performed. Conversation processors can further analyze the conversations, possibly extracting valid TCP sessions.

Once packets are in the traffic store, they can be accessed directly or via their parent conversations. In both cases, a processor is applied on packets to produce the required results. There are two types of processors:

- A conversation processor is used to analyze and extract features from the whole conversation. It takes a conversation as an input and produces the output of the specified type. For instance, the *NetFlowProcessor* generates NetFlow-like records from conversations.
- A packet processor is used to compute resulting records from individual packets. A processor provides a method that takes a packet as an input and generates the specific output. For instance, the MODBUS packet processor parses the packet and produces a collection of selected fields, e.g., function code and address of fields referenced.

The library enables batch processing of conversations and packets using processors. By this means, features can be extracted for either individual packets or conversations. Since conversation processors have access to full packets, they can also use packet processors to obtain individual packets' values. For instance, considering MODBUS/TCP communication, it is possible to extract MODBUS function codes, enabling to enrich a set of features with application-level information. This is employed in Modbus Packet and Modbus Conversation processors that export features for individual MODBUS packets and bidirectional flows, respectively.

We have implemented several conversation processors, but the user can implement her own by extending provided abstract classes. Implementing the new processor amounts to specify the result type and a function that computes the output value for each input record. To avoid repetitions and code duplication, the conversation processors can be used within other processors or pipelined using basic composition operations. Also, it is possible to apply multiple processors to the same source data and then aggregate their results.

4.3 Exporting Features

The processors' main goal is to extract information relevant for further processing in AD method training or testing. The result of applying the processor to an input collection is a set of records. The library provides functions that turn resulting records into the input compatible with the machine learning library ML.NET [8]. The ML.NET uses abstract data type *DataView* for defining the input and output of all operations. A generated record can be represented by any struct or class type with arbitrary nested values. Because *DataView* is a collection of simple records, our library performs automatic flattening using information obtained from reflection. The access methods are generated in runtime for all types used as *DataView* objects to improve the performance. The further feature manipulation is then done by applying methods from *DataOperationsCatalog* on *DataView* input provided by the ML.NET library.

4.4 Model Training

The anomaly detection model training is done primarily in the realm of the ML.NET library. The choice of this library is obvious. It provides an off-the-shelf collection of high-performance machine learning algorithm implementations and enables integration with some major ML frameworks. It is also possible to use tools other than ML.NET. The data interface consists of a collection of records produced by conversation or packet processors. All produced collections have pull-based access, which means that data are computed on demand, improving the performance and reducing the memory overhead. To integrate with other tools, it is enough to implement a connector similar to that for *DataView*.

4.5 Interactive Mode

Jupyter Notebook provides a suitable environment for interactive data exploration and experimental anomaly detection method design. The library's integration is possible thanks to .NET Interactive package that implements a .NET support for Jupyter notebooks.

In the interactive mode, the user mainly calls functions from a uniform API surface provided by the library. This API offers the library's most helpful functions that implement common usage patterns, e.g., loading packet trace to a conversation table, application of conversation processor, and storing conversation records to the CSV file. The standard libraries, e.g., LINQ, Collections, I/O, offer additional operations to manipulate data enumerations and access files. Machine learning and data visualization is enabled by referencing ML.NET and XPlot.Plotly packages.

4.6 Batch Mode

Usually, in the batch mode, the new (console) application is written. This application references the necessary libraries for data preprocessing, machine learning, and output generating. The application is then repeatedly executed in experiments with different methods and parameters. All public classes of the library that implement specific parts of the data processing functionality can be directly used from the console application too. It is possible to tune data processing performance using optimized data loaders, parameters of conversation tables that control the allocated storage space, different processors, etc. Because of using the uniform development environment, the most of the source code (including data preprocessing) is the same as in the interactive phase.

5 EVALUATION AND USE CASES

The evaluation presented in this section aims to demonstrate that the proposed library can improve the data preparation phase in terms of performance and user experience.

In the performance evaluation, the proposed tool is compared with tshark, commonly used for extracting data from packet capture files during the data preparation phase. Both tshark and the library are asked to perform the same tasks: data extraction from the source capture file. The amount of time required for various sizes of inputs is then compared.

The possible application scenarios of the proposed library are demonstrated using two simple anomaly detection methods. The first method consumes network flow data to learn the model using a one-class K-means algorithm. The normal behavior is represented through the expected number of operations within a fixed interval window. The second method works on packet-level and builds the model that performs time series anomaly detection. Spikes and change points identified in the series of packets are reported as anomalies.

The purpose of these AD implementations is only to demonstrate the library's principles and its smooth integration with a machine learning library. For this reason, we have not considered evaluating the performance of demonstrated anomaly detection methods in terms of accuracy, precision, recall, and other relevant metrics. Instead, the purpose is to show that the data preprocessing library easily integrates with the machine learning framework.

The primary datasets considered for evaluation are from the industrial cybersecurity conference 4SICS, which hosts ICS/SCADA cybersecurity stakeholders across critical industries. When compared to other available datasets, the 4SICS packet traces contain the richest set of ICS communication. We tested the environment with other available MODBUS datasets [27], but the results are only presented for 4SICS data. The environment can also decode other ICS traffic (DNP3, S7, DLMS), but we left experiments with these protocols for future work mainly because of the lack of suitable datasets.

5.1 Data preprocessing performance

experiments to measure the computation time required for the data processing in a typical application scenario. All experiments are conducted on 2-processor 16-cores Intel Xeon CPU E5-2620 machine running at 2.1GHz, with 128GB RAM running 64-bit Windows 10. We measured data load time and data processing time. We present the result only for three representative datasets containing MODBUS/TCP communication. The size of source capture files was 25MB, 140MB, and 200MB containing 246k, 1.2M, and 2.2M packets, respectively. The results are shown in Table 1. It consists of three rows each for the dataset used. The first four columns give information on the datasets: dataset name, file size in bytes, number of packets, and the duration. The next columns are running times for different experiments.

To set a baseline we used a tshark tool that generates CSV files for provided capture files consisting of selected MODBUS/TCP fields (column tshark). The library was benchmarked separately for the two phases:

- Packet trace ingestion - packets from the source capture file are loaded in the storage performing conversation identification (column Ingest). The packet storage stands for the data source in interactive and batch mode instead of the original capture files.
- Feature Export - packets from the store are processed using Modbus processors. Either packet-level features (column Exp/P) or conversation-level features (column Exp/C) are extracted.

Each benchmark was executed several times to get the average running time. In addition to the similar output as tshark (CSV file of Modbus packet fields), the pipeline can produce the IPFIX records of MODBUS/TCP conversations. As seen, the overall processing is an order of magnitude faster using the library compared to tshark.

Table 1: Dataset parameters and processing time

Dataset	Size[B]	Packets	Duration	tshark[ms]	Ingest[ms]	Exp/P[ms]	Exp/C[ms]	Pipeline[ms]
4SICS-151020	25,711,082	246,137	6:59:57	15764	782	396	473	1650
4SICS-151021	139,998,821	1,253,100	24:00:02	83217	3089	1621	2293	7004
4SICS-151022	209,236,002	2,274,747	15:02:02	145320	6338	2469	3159	11966

Listing 1: An example implementation of the AD trainer

```
// PART 1: Data ingestion
using var conversationTable = FasterConversationTable
    .Create($"{sourceFile}.db");
using var loader = conversationTable.GetStreamer();
using var pcapReader = new SharpPcapReader(sourceFile);
while (pcapReader.GetNextFrame(out var rawFrame)) {
    loader.AddFrame(rawFrame);
}
loader.Close();

// PART 2: Data preparation
var processor = new Modbus.ModbusBiflowProcessor();
var windows = conversationTable.Conversations
   .GroupByWindow(start, duration);

var modbusProcessor = new ModbusBiflowProcessor();
var records = windows.SelectMany(conv =>
    conversationTable.ProcessConversations(conv,
    modbusProcessor.ApplyToWindow(start, duration)));

// PART 3: Model Training
var trainingData = records.AsDataView();
var mlContext = new MLContext();
var normalize = mlContext
    .Transforms
    .NormalizeMinMax("Features");
var options = new KMeansTrainer.Options {
    NumberOfClusters = numberOfClusters
};
var pipeline = normalize
    .Append(mlContext.Clustering.Trainers.KMeans(options));

var model = pipeline.Fit(trainingData);
```

5.2 OC-KMeans Anomaly Detection

This method implements a clustering algorithm to model normal traffic as a collection of clusters. Based on identified clusters of normal flows, it is possible to classify the new flow either as normal or unknown.

In the training phase (see the algorithm in Listing 1), IPFIX records representing the normal behavior are clustered using the K-means algorithm. Each cluster identifies a conversation pattern. The input to the K-means algorithm is a feature vector that is created from selected Modbus IPFIX fields. The code of the trainer consists of three parts. The first part implements data loading to the store. This needs to be done only once for every dataset used for training. The second part implements data preparation. *ModbusBiflowProcessor* processes the conversations from the library store. Before *ModbusBiflowProcessor* is applied, the stored conversations are ordered and grouped in the defined duration windows as required by the AD method. This amounts to i) compute intervals and identify included conversations by applying the *GroupByWindow* operator, and ii) limit the application of the conversation processor only to a part of the conversation that appears in the current window (operator *ApplyToWindow*). The third part is the trainer implementation using ML.NET. The result is the model that can be saved and used for anomaly detection. The training data is transferred to ML.NET using the *AsDataView* method, which maps the records from the conversation processor to the *IDataView* type required by the ML.NET. The created model can be further used for evaluation and parameter tuning.

Listing 2: An implementation of Time Series AD

```
// PART 1: DATA PREPARATION
var frames = table.ProcessFrames(table.FrameKeys,
    new TimedFrames());
var intervals = frames.GroupBy(x =>x.Ticks/timeInterval.Ticks));
var framesValues = intervals.Select(x => new PacketCountData {
    Timestamp = x.Key * timeInterval.Ticks,
    Value = x.Count() });

// PART 2: TIME SERIES AD
var mlContext = new MLContext();
var dataview = mlContext.Data.LoadFromEnumerable(framesValues);
int period = mlContext.AnomalyDetection
    .DetectSeasonality(dataview, "Value");
var options = new SrCnnEntireAnomalyDetectorOptions() {
    Threshold = 0.3, Sensitivity = 80.0,
    DetectMode = SrCnnDetectMode.AnomalyAndMargin,
    Period = period
};

// PART 3: Invoke SrCnn algorithm to detect anomaly
var outputDataView = mlContext.AnomalyDetection
  .DetectEntireAnomalyBySrCnn(dataview, "Prediction",
     "Value", options);
var predictions = mlContext.Data
  .CreateEnumerable<PacketCountPrediction>(outputDataView,
     reuseRowObject: false);
```

5.3 Time Series Anomaly Detection

To demonstrate the library's use for time series anomaly detection, the output from a packet processor is fed to SR-CNN anomaly detector [10]. The code snippet is shown in Listing 2. The detector expects a series of data. The first part prepares the data by collecting packets in defined time intervals. For each interval, we determine packet count. Although, any aggregation can be used, e.g., total octets transmitted, number of TCP segments, etc. The second part creates MLContext, loads prepared data, and computes the seasonality. Finally, in the third part, the anomalies are detected, and the result is provided in the predictions collection.

6 CONCLUSION

To evaluate the existing anomaly detection techniques and simplify the design of new methods, we implemented a library for network data preprocessing. Designers can utilize the library during the interactive design activities by referencing it from Jupyter Notebooks as well as for writing standalone anomaly detection tools.

The presented library implements a function for processing packet traces. In addition to common functions of similar tools, conversation tracking is implemented. It enables users to obtain arbitrary flow-level and packet-level features by applying the custom processors. The preliminary evaluation demonstrates that the library can be employed in the design pipeline providing the necessary performance, and it is easy to use.

We compared the proposed library to a tshark-based preprocessing pipeline commonly used to extract features from packet traces. We were able to improve the overall performance of the preprocessing phase. However, it does not mean that tshark is not a suitable tool for data preprocessing and feature extraction. Tshark offers much more functions than our library and supports hundreds of communication protocols. Our library offers a specialized collection of functions supporting only the limited set of use cases related to the design and development of anomaly detection methods. The aim was also to enable the tight integration with machine learning libraries avoiding generating and loading intermediate CSV files. Finally, we can use the library directly from Jupyter notebooks for interactive development, simplifying the entire process.

The proposed library is still in an early stage of development. Further performance improvements, new features, the unification of the preprocessing pipeline, and support for on-line processing are anticipated steps towards the full-fledged implementation. The source code of the library is publicly available[1].

ACKNOWLEDGMENTS

The presented work was carried under the project "Security monitoring of ICS communication in the smart grid" sponsored by grant no. VI20192022138.

REFERENCES

[1] Dainotti, A., De Donato, W., & Pescapé, A. (2009). TIE: A community-oriented traffic classification library. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5537 LNCS, 64–74. https://doi.org/10.1007/978-3-642-01645-5_8

[2] Lee, S., & Hyun-chul. (2011). NeTraMark A Network Traffic Classification Benchmark. ACM SIGCOMM Computer Communication Review, 41(1), 8. https://dl.acm.org/citation.cfm?id=1925865

[3] Yan, X., Liang, B., Ban, T., Guo, S., & Wang, L. (2012). TrafficS: A behavior-based network traffic classification benchmark system with traffic sampling functionality. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7666 LNCS(PART 4), 100–107. https://doi.org/10.1007/978-3-642-34478-7_13

[4] Jin, Y., Duffield, N., Erman, J., Haffner, P., Sen, S., & Zhang, Z. L. (2012). A modular machine learning system for flow-level traffic classification in large networks. ACM Transactions on Knowledge Discovery from Data, 6(1). https://doi.org/10.1145/2133360.2133364

[5] Donato, W., Pescapé, A., & Dainotti, A. (2014). Traffic identification engine: An open library for traffic classification. IEEE Network, 28(2), 56–64. https://doi.org/10.1109/MNET.2014.6786614

[6] Kim, Y. H., Konow, R., Dujovne, D., Turletti, T., Dabbous, W., & Navarro, G. (2015). PcapWT: An efficient packet extraction tool for large volume network traces. Computer Networks, 79, 91–102. https://doi.org/10.1016/j.comnet.2014.12.007

[7] Hu, B., Kamiya, K., Takahashi, K., & Nakao, A. (2020). Piper: A Unified Machine Learning Pipeline for Internet-scale Traffic Analysis. GLOBECOM - IEEE Global Telecommunications Conference, 1–6. https://doi.org/10.1109/globecom42002.2020.9322531

[8] Ahmed, Z., Amizadeh, S., Bilenko, M., Carr, R., Chin, W. S., Dekel, Y., Dupre, X., Eksarevskiy, V., Erhardt, E., Eseanu, C., Filipi, S., Finley, T., Goswami, A., Hoover, M., Inglis, S., Interlandi, M., Katzenberger, S., Kazmi, N., Krivosheev, G., & Zhu, Y. (2019). Machine learning at Microsoft with ML.NET. In KDD '19: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 2448–2458).

[9] Chandramouli, B., Goldstein, J., Barnett, M., DeLine, R., Fisher, D., Platt, J. C., Terwilliger, J. F., & Wernsing, J. (2014). Trill: A High-Performance Incremental Query Processor for Diverse Analytics. Proceedings of the VLDB Endowment, 8(4), 401–412. https://doi.org/10.14778/2735496.2735503

[10] Ren, H., Xu, B., Wang, Y., Yi, C., Huang, C., Kou, X., Xing, T., Yang, M., Tong, J., & Zhang, Q. (2019). Time-series anomaly detection service at Microsoft. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. https://doi.org/10.1145/3292500.3330680

[11] Darwish, I., & Saadawi, T. (2018). Attack detection and mitigation techniques in industrial control system-smart grid DNP3. Proceedings - 2018 1st International Conference on Data Intelligence and Security, ICDIS 2018, 131–134. https://doi.org/10.1109/ICDIS.2018.00028

[12] Lai, Y., Liu, Z., Song, Z., Wang, Y., & Gao, Y. (2016). Anomaly detection in Industrial Autonomous Decentralized System based on time series. Simulation Modelling Practice and Theory, 65, 57–71. https://doi.org/10.1016/j.simpat.2016.01.013

[13] Anton, S. D., Kanoor, S., Fraunholz, D., & Schotten, H. D. (2018). Evaluation of machine learning-based anomaly detection algorithms on an industrial modbus/TCP data set. ACM International Conference Proceeding Series, 1–9. https://doi.org/10.1145/3230833.3232818

[14] Lin, C., & Nadjm-tehrani, S. (2019). Timing Patterns and Correlations in Spontaneous SCADA Traffic for Anomaly Detection. 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019), 73–88.

[15] Zhanwei, S., & Zenghui, L. (2019). Abnormal detection method of industrial control system based on behavior model. Computers and Security, 84, 166–178. https://doi.org/10.1016/j.cose.2019.03.009

[16] Kreimel, P., Eigner, O., Mercaldo, F., Santone, A., & Tavolato, P. (2020). Anomaly detection in substation networks. Journal of Information Security and Applications, 54. https://doi.org/10.1016/j.jisa.2020.102527

[17] Siegel, B. (2020). Industrial Anomaly Detection: A Comparison of Unsupervised Neural Network Architectures. IEEE Sensors Letters, 4(8). https://doi.org/10.1109/LSENS.2020.3007880

[18] Ahmed, M., Naser Mahmood, A., & Hu, J. (2016). A survey of network anomaly detection techniques. Journal of Network and Computer Applications, 60, 19–31. https://doi.org/10.1016/j.jnca.2015.11.016

[19] Shang, W., Zeng, P., Wan, M., Li, L., & An, P. (2016). Intrusion detection algorithm based on OCSVM in industrial control system. Security and Communication Networks, 9(10), 1040–1049. https://doi.org/10.1002/sec.1398

[20] Inoue, J., Yamagata, Y., Chen, Y., Poskitt, C., & Sun, J. (2017). Anomaly detection for a water treatment system using unsupervised machine learning. IEEE International Conference on Data Mining Workshops, ICDMW, 2017-. https://doi.org/10.1109/ICDMW.2017.149

[21] Lin, Q., Verwer, S., Adepu, S., & Mathur, A. (2018). TABOR: A graphical model-based approach for anomaly detection in industrial control systems. ASIACCS 2018 - Proceedings of the 2018 ACM Asia Conference on Computer and Communications Security. https://doi.org/10.1145/3196494.3196546

[22] Kravchik, M., & Shabtai, A. (2018). Detecting cyber attacks in industrial control systems using convolutional neural networks. Proceedings of the ACM Conference on Computer and Communications Security. https://doi.org/10.1145/3264888.3264896

[23] Elbez, G., Keller, H. B., Bohara, A., Nahrstedt, K., & Hagenmeyer, V. (2020). Detection of DoS Attacks Using ARFIMA Modeling of GOOSE Communication in IEC 61850 Substations. Energies, 13(19), 5176. https://doi.org/10.3390/en13195176

[24] Faisal, M., Cardenas, A. A., & Wool, A. (2017). Modeling Modbus TCP for intrusion detection. 2016 IEEE Conference on Communications and Network Security, CNS 2016, 386–390. https://doi.org/10.1109/CNS.2016.7860524

[25] Zhanwei, S., & Zenghui, L. (2019). Abnormal detection method of industrial control system based on behavior model. Computers and Security, 84, 166–178. https://doi.org/10.1016/j.cose.2019.03.009

[26] Barbosa, R. R. R., Sadre, R., & Pras, A. (2016). Exploiting traffic periodicity in industrial control networks. International Journal of Critical Infrastructure Protection, 13, 52–62. https://doi.org/10.1016/j.ijcip.2016.02.004

[27] Choi, S., Yun, J. H., & Kim, S. K. (2019). A comparison of ICS datasets for security research based on attack paths. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 11260 LNCS. Springer International Publishing. https://doi.org/10.1007/978-3-030-05849-4_12

[1]https://github.com/rysavy-ondrej/traffix.net