BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 71(4), 2023, Article number: e146285
DOI: 10.24425/bpasts.2023.146285

ARTIFICIAL AND COMPUTATIONAL INTELLIGENCE

Smart network anomaly detection software architecture for network-enabled ubiquitous devices

Mariusz PELC[1,2], **Dawid GALUS**[1], **Mariusz GOLA**[1], and **Aleksandra KAWALA-STERNIUK**[1]*

[1] Faculty of Electrical Engineering, Automatic Control and Informatics, Opole University of Technology, Opole, Poland
[2] School of Computing and Mathematical Sciences, University of Greenwich, London, UK

Abstract. In this paper we present an architecture for run-time reconfiguration of network-enabled ubiquitous devices. The whole idea is based on a policy-based system where the whole decision-making (e.g. anomaly detection-related) logic is provided in a form of an externally loaded policy file. The architecture is verified through real-life implementation on an embedded system whose sensitivity can be easily modified should a need arise in run-time without affecting network device/segment (and thus potentially a number of network services) so that they continue working while the re-configuration process is triggered.

Key words: network anomaly/threats detection; smart systems; policy-based computing.

1. INTRODUCTION

Typically, various kinds of anomaly detection algorithms are physically implemented as a piece of software running on a network device or a system. In this paper an architecture supporting smart and flexible anomaly detection systems for network-enabled ubiquitous (e.g. IoT) devices is proposed. **Methodology:** Typically, many network-enabled devices are usually considered as resource-constrained systems (with limited processing power, storage, memory, etc.). At the same time they are exposed to various kinds of threats which specifics and nature may change in time. This means that these devices may greatly benefit from running a system / middleware (meant by middle layer software) which will allow them to adjust their network anomaly detection capability in run-time and potentially trigger in response some sort of self-protection mechanism / behaviour. **Results:** Taking into account the aforementioned devices, some benefits resulting from decoupling the decision-making logic (responsible for anomaly detection) from the software component (application) that will be presented. As a result, the software component will decrease in size (reducing memory-related requirement) and at the same time, the decision-making logic stored in a form of a separate policy file in the device file system) may be easily updated with a newer version. **Conclusions:** The solution proposed in this paper can be implemented on various kinds of devices, starting from the core network components and devices (e.g. routers) through consumer-grade devices with internet connectivity up to various kind of network-enabled devices to guarantee their increased security and enable their easy yet very effective run-time reconfiguration.

Nowadays, when networks and systems security becomes a paramount, it is of utmost importance to guarantee that the anomaly detection or threats analysis system are able to deal with ongoing security-related events as effectively and efficiently as possible. Unfortunately, due to a huge variety of possible anomalies and threats (routing-related, buffer overflow based, protocol-specific, traffic flowing, trojans, worms, etc.) and their changing characteristics (different protocols, ports, traffic volumes, etc.) it is not very likely that a single, even best-trained / best-tuned, algorithm would perform equally well and guarantee the same level of security over a long period of time or for all potential threats. In case it would no longer be able to work up to the expected standard, there are actually only a few things that could potentially be done to improve its performance:

- update the currently used monitoring algorithms(s) with respect of its core parameters (e.g. neuron weights in ANN) to guarantee its increased efficiency and accuracy,
- update the currently used monitoring algorithm(s) so that it would become aware of some other anomaly/attacks types (e.g. increase the number of rules in the fuzzy system, etc.),
- replace the currently used monitoring algorithm(s) with an alternative in case the type of anomaly/attacks fall beyond their detection capability or to provide more fine-grained classification (e.g. thresholds).

Typically, anomaly detection or threats analysis systems are implemented as some kind of a program embedded into a network device(s) or a designated computer(s) and/or workstation(s), subject of the anomaly detection scope, volume of data that needs to be analysed or complexity of the anomaly detection algorithm(s). Such a program usually needs to be (and in most cases is) tuned to make it suitable for a specific network or for a specific solution. Tuning process may incorporate directly an expert knowledge about network and/or system behaviour or implement some techniques allowing the anomaly detection

*e-mail: a.kawala-sterniuk@po.edu.pl

Manuscript submitted 2023-01-15, revised 2023-06-05, initially accepted for publication 2023-06-09, published in August 2023.

algorithm to learn this by itself to be consequently able to detect any activity that deviates from the normal behaviour pattern. There are many technologies allowing to properly reflect the normal network behaviour, including machine/deep learning techniques [1, 2], statistical methods [3–5], etc.

The system we are going to present is based on MAPE (Monitor-Analyse-Plan-Execute) architecture [6] that allows us to equip various kinds of systems with the capability to act autonomously or with limited human supervision. So the autonomous behaviour is the novelty of the paper because systems which will incorporate our solution will be able to perform various tasks with context awareness and in case the environmental conditions change – adapt their behaviour to the changed working conditions. Alternatively, such an ability could be implemented using ML-based methods (see e.g. [7, 8]), but the problem with ML-based approach is that it needs a proper data set to train the network, it is difficult to estimate time needed to come up with a decision (reduces applicability in the real-time systems) and most of all - any reconfiguration requires the process to be started over (contrary to the proposed solution where the only thing needed for the system to re-configure is a newer version of policy provided by the system expert.) For the purpose of anomaly detection all the MAPE architecture components will be based on AGILE [9] and AGILE-FUZZY PDL (Policy Definition Language) [10] that can monitor a system behaviour through a number of environment (process) variables, then it can perform analysis, plan actions in response to a specific system behaviour based on expert-provided reasoning mechanism (can be AGILE policy, can be FUZZY rules-based policy, etc.) and finally, it can action (execute) decisions to alter the system behaviour (to mitigate the identified problem).

Structure of the remaining part of this paper is the following: Section 2 provides literature review, in Section 3 one can find description of the policy-based anomaly detection software architecture, Sections 4 provides explanation of how the proposed software architecture contributes to run-time reconfiguration and smart behaviour of the target system whilst Section 5 details features of the fuzzy system used for the anomaly detection purposes as well as includes description of the test cased test environment and test data. Section 6 shows the resulting system behaviour with the emphasis on how its sensitivity may change when run-time policy-based reconfiguration is triggered. Section 7 includes concluding statement.

2. RELATED WORK

Widely understood anomaly detection systems constitute one of the most rapidly developing research domains. It can apply to many potential applications, including network anomaly detection, intrusion detection, etc. Depending on the application domain different methods are used to identify any deviation from what is considered to be the normal network/system behaviour.

Among the typical approaches that are being used for network anomaly detection are those originating from the statistical sciences. In these solutions, typically, the network behaviour is analysed from the point of view of its statistical or probability features. Very often the appropriate mathematical apparatus is also used to identify potential anomalies. Such an approach was undertaken in [4] where the authors provided comparison of methods to identify anomalies including Principal Component Analysis, Ant Colony Optimisation and AutoRegresive Moving Average Forecasting. The same approaches can be found in [11], where the authors have used Chi-square based statistical anomaly detection for VMWare performance data or in [12], where Bayesan (statistical) model with time slicing was used in order to detect network anomalies. Similarly, in [13] the authors have provided appropriate analytical framework applied for anomaly-detection in network traffic. The specific goal was to provide a method that would, e.g. reduce the human involvement in the anomaly detection process. The methods of interest were Seasonal Autoregressive Integrated Moving Average (SARIMA) times series model and Long Short-Term Memory (LSTM).

Alternative approach may also use some mathematical modelling which is specifically used for traffic classification (clustering) purposes. In this approach the focus is to detect outliers meaning all the events that fall outside of a group that reflects traffic patterns classified as normal. Following this method, the authors in [14] have proposed NADO method (Network Anomaly Detection using Outlier Approach) for identifying outliers. Outlier detection technique using fuzzy neural network and k-means algorithm was also presented in [15]. In order to classify traffic to the normal cluster k-means clustering technique was used. Clustering-based approach to network anomaly detection was also described in [16] and in [17]. In [18] certain type of Intrusion Detection System was proposed. This IDS system used Support Vector Data Description (SVDD) to train the system to cluster normal user behaviour.

Unfortunately, the formal methods based on mathematical modelling or statistical analysis are not always applicable, especially whenever expressing the network behaviour in mathematical terms constitutes a very complex task. In such a case the solution might be based on employing, e.g. artificial intelligence based methods which usually require knowledge about the typical network behaviour (learnt by themselves through observation or provided in a form of traffic patterns) in order to be able to identify any network traffic that stands out and does not match the pattern. These solutions include machine learning or deep learning techniques. Majority of the latest research papers targeting network anomaly detection are actually following this way, e.g. [19], where the authors use machine learning approach to prevent compromising network-enabled devices on the network. The learnt behaviour allowed the network-enabled devices to detect DDoS attacks. Deep Learning approach was used for the purpose of network traffic anomaly detection was presented in [20]. Recurrent Neural Networks were used for the network anomaly detection purposes in [21]. Alternatively to using ANN-based approaches, in some papers the Fuzzy Logic methodology was used to identify anomalous behaviour, e.g. in [22] IDs system was proposed where the fuzzy logic was used to describe different security attacks.

A natural consequence of existence of both, the mathematical and artificial intelligence-based methods was that there were also several solutions presented where the two approaches were combined in order to use their strengths for even better and more accurate identification of anomalous behaviour. In [23] the authors have proposed a combined approach where Back Propagation Neural Network is using Genetic Algorithms along with Simulated Annealing Algorithm to identify anomalies. In [24] the authors propose IDS which is use decision tree and Gaussian Mixture Model (GMM) for detection of anomalies. In [25] there is a combination of neural deviation learning and enforcement of statistically significant deviations from norm.

Quite an interesting trend in the network anomaly detection are different kinds of bio-inspired algorithms. For example, in [26] a version of Cuckoo search algorithm was used for the DDoS attacks detection. Another approach, where non-parametric, distributed, bio-inspired algorithm has been used for anomaly detection in networks applying observation on self-organization principle similar to the one used in honey bees colonies, was presented in [27].

As one can see from the selection of related work, there are many existing and some new emerging approaches to solve the problem of network anomaly detection, which includes some examples of research involving different artificial intelligence, fuzzy logic and expert knowledge-based methods (as well as formal methods using mathematical apparatus for anomaly detection task) for addressing the problem of anomaly detection in network. In this context the proposed solution, although using expert knowledge and/or fuzzy logic reasoning, is of a different nature. It neither emphasises the accuracy of the applied anomaly detection methods (though it is clearly visible that the anomaly detection algorithm we use has the ability to react to some deviations from a typical traffic characteristics within a given time window), nor it includes a comparison to some alternative solutions. What we have proposed is rather focusing on the implementation aspects where, no matter whichever anomaly detection method is finally used, the proper design of the software components would guarantee much better utilisation of these methods through providing an interface for easy cooperation of different anomaly detection technologies (e.g. policy-based computing and Fuzzy Logic systems) as well as it equips anomaly detection systems with additional degree of freedom regarding their run-time re-configuration.

3. ANOMALY DETECTION SOFTWARE ARCHITECTURE FOR NETWORK-ENABLED UBIQUITOUS DEVICES

Typically, the software used for anomaly detection that is installed and/or embedded into a network device implements a specific detection algorithm where the crucial parameters (detection rules, thresholds values, etc.) are hard-coded into the software itself. Such software benefits from its simple configuration but in case there is a need to update the anomaly detection algorithm, this actually can only be done through whole firmware update and cannot often be done “on the fly”. As a result, during the update process some of the network components may become temporarily unavailable but in the worst case scenario this may affect whole network segments (subnets) and render them inaccessible until the network device reconfiguration has finished.

The alternative solution that is proposed in this paper will deal with the re-configuration problem in a completely different way. Rather than re-uploading the whole firmware and/or software to the network device requiring its anomaly detection algorithm only to be updated, the firmware might have a modular architecture with the anomaly detection-related module (and hence the anomaly detection algorithm itself) able to be changed and/or replaced “on the fly” without the need of re-installing the whole device firmware. Obviously, such a software architecture will become more complex and ultimately it may require (in some cases) more resources (such as processing power, memory, etc.). But the potential benefits are quite obvious and justify this increased complexity.

One of the technologies that may be very useful in case such a software architecture was to be implemented is policy-based computing and specifically, ODP (Open Decision Point) software architecture [28, 29]. This software architecture allows constructing software components where the key decision-making logic is provided externally as a run-time loadable policy. The policy is usually specified by system expert(s) in a form of XML-compliant policy file written in AGILE PDL or AGILE-FUZZY PDL and by-design supports self-* / smart features (in our case self-* is mostly related to self-configuring).

In order to implement the policy-based computing technology within a network device, a complete re-design (from the architecture viewpoint) of the software component is required. Typically, in case the network device provides any kind of intrusion network/host anomaly detection (or e.g. intrusion detection) features, they usually are fixed as just one of the functional parts of the whole software component. Hence whenever it is required to make changes to the anomaly detection part only, then, since this part is inseparable from a bigger whole, this kind of update operation is doable only via downloading a newer version of software component program as whole and restart it after the update/upgrade operation has been completed. One can easily notice that updates of the anomaly/intrusion detection part (due to nature of the threats which are subject to frequent change) will be performed much more often compared to the updates of the core network device functionality (e.g. transmitting/receiving packets, packets routing, etc.). As a result, the architecture of the software component must be of modular structure and there must be some kind of additional mechanisms in place to support only partial reconfiguration of the software component allowing to update the core component logic while keeping the other parts up and running not to cause network or services unavailability during the update process. Example design for such an architecture is shown in Fig. 1.

As one can see in Fig. 1, the fixed section represents all the core functionality that is usually altered via whole software update. In a typical network device software component architecture this section is everything that a network device is running. In the proposed software component architecture the Fixed Section represents mainly the functionality that relates to handling networking while the Decision Section block represents solely the anomaly detection/decision-making related functionality that can be dynamically altered “on the fly” (in run-time) without the need for re-installing remaining part of

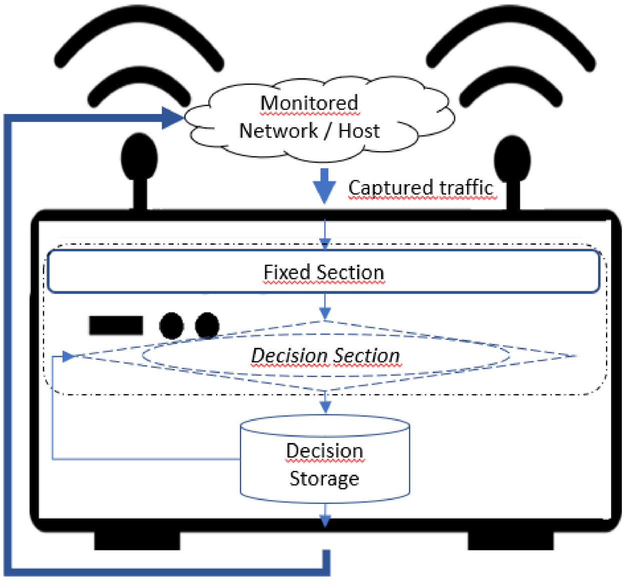


Fig. 1. Proposed architecture for network device software

the software. This approach would guarantee that the networking functionality is available all the time and there is no disruption in the network operation whilst the capability to better recognise/react to anomaly is improved. The dashed line around those two blocks indicate that in the typical architecture those two blocks are integrated. Additionally, the proposed architecture includes a Decision Storage which allows the system to conform to the MAPE-K [6] which is foundation for various kind of self-* and autonomous systems. This storage can store new decisions (being result of recognised network/host anomaly) and allows us to find out whether specific anomalies have been previously detected and load corresponding decisions rather than evaluate them every time. Such an approach may save time and potentially reduce the effect of an intrusion, etc. Each decision is applied to the monitored network/host as a reaction to the detected anomaly.

4. SMART BEHAVIOUR THROUGH POLICY-BASED RE-CONFIGURATION

The key point of this paper is to propose a flexible software architecture allowing a network-enabled device meaning any device but especially the resource-constrained ones to adjust its behaviour in response to various kinds of changes related to the device working environment or improve its operation in case new information or a better way of processing available information is available. And to do it not through re-loading the whole device software (e.g. through firmware update) but rather through altering the decision-making logic part and leaving rest of the device software as it is.

The proposed architecture was shown in Fig. 1. On the other hand, the architecture is supposed to support smart behaviour of such a network device and allow the software component to be re-configured without the need of restarting the network device. What we propose as the solution to the problem is policy-based re-configuration supported by *AGILE_Lite* library. The library supports Open Decision Point (ODP) [28] architecture which allows any software component to implement the architecture to be re-configurable in run-time. Open Decision Points are host place holders for loadable policy files containing decision-making logic which is formed based on the knowledge of a system expert about the monitored/supervised/controlled system. If, apart from using typical AGILE policies [9], the reasoning is based on fuzzy logic (supported by AGILE-FUZZY extension) [10], then such a system incorporates AI technology into the decision-making logic and thus allows to implement smart behaviour.

Traditional software components also include decision-making logic but contrary to the proposed approach, the logic is integral part of the whole software. All programming languages provide various kinds of conditional constructs, e.g. *if-else* conditional instructions, *while/until* conditional loops, etc. and these instructions constitute part of the software component code and as such are they not separable from the rest of the function or future-specific code. *AGILE_Lite* library [30] provides API interface through which such a separation is doable. The only change on the programming side is very little compared to the typical software component code and rather than arranging all those conditional instructions into a decision-making logic, using the *AGILE_Lite* library requires the software developer to follow the below three-step routine:

- assign a set of *EnvironmentVariables* to reflect the current system state and provide information essential to make a decision,
- evaluate the appropriate policy,
- read the policy decision and only provide a piece of code that is actuating the decision (e.g. call corresponding functions).

Such an approach results also in one more but crucial change, a new programming paradigm, in comparison to the typical approach: it is possible to decouple the software developer and policy developer roles. In the traditional approach the software developer is also coding the decision-making logic (this can be done in co-operation with the system expert to properly reflect the desired system behaviour). In the proposed approach the only thing the software developer needs to know is what kind of information is needed on the policy side to make the decision, what the potential decisions are and how they should translate into changing the system behaviour. Then the code and the policy/policies can be developed separately and brought together at the very end of the target system.

The AGILE/AGILE-FUZZY policy as such maps a set of *EnvironmentVariables* containing all necessary information about a system state into one of *ReturnValues*. Additionally, policy can return one or more *OutputVariabies* to provide more information about the decision (e.g. the *ReturnValue* reflecting the policy decision may require increasing temperature (e.g in a room) whilst through an *OutputVariable* it can additionally be specified by how much the temperature should be increased. Such a policy reflection is shown in Fig. 2.

All the rhomboidal shapes seen in Fig. 2 represent policy decision-making blocks, respectively *Rules*, *ToleranceRangeChecks* or *UtilityFunctions* which allow different

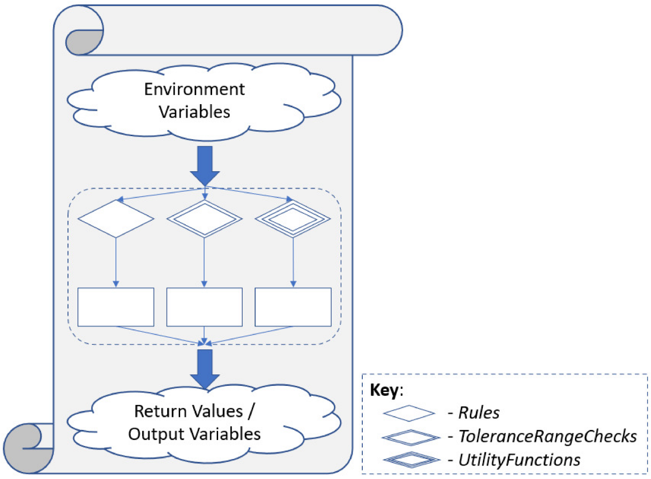


Fig. 2. Policy representation

way of processing the context information to reach a decision whilst the rectangular blocks represent, e.g. *Action* blocks corresponding to some specific actions to be taken. The context information is processed through these policy objects in order to make sense of what it means in the context of monitored/controlled system state and which actions are supposed to be taken as a result. The more such objects are used within a policy the better use of the context information can be made and, consequently, the more precise and correct the policy decision will be. And since policies are formed usually by system experts and reflect their knowledge about their specifics, then policy-based reasoning and decision-making translates in a natural way into smartness of such a system.

5. SMART ANOMALY DETECTION SYSTEM

For the purpose of demonstrating how the smart features could be embedded into a network device/system through we have developed a software that implements the policy-based computing solution to the problem. For that reason we have designed a software component with two Decision Points:

- AGILE FUZZY type Decision Point that will be monitoring the number of sent/received segments from/to a host and deciding the extent of anomaly.
- AGILE type Decision Point that will be using the decision/information provided by the AGILE FUZZY Decision Point and after combining the information with a trend analysis outcome will make a decision regarding the system operational mode. For simplicity of the demonstration only 2 modes will be implemented:
  1) *unrestricted*, with all services available from all locations;
  2) *restricted*, where certain services will only be accessible from selected locations only. The restrictions will be applied through appropriate firewall re-configuration using a Bash script.

For the trend analysis the Mann-Kendall method [31,32] will be used which is implemented in *pyMankendal* package for *Python* programming language [33].

The resulting software component architecture is shown in Fig. 3 originates from the one shown in Fig. 1.

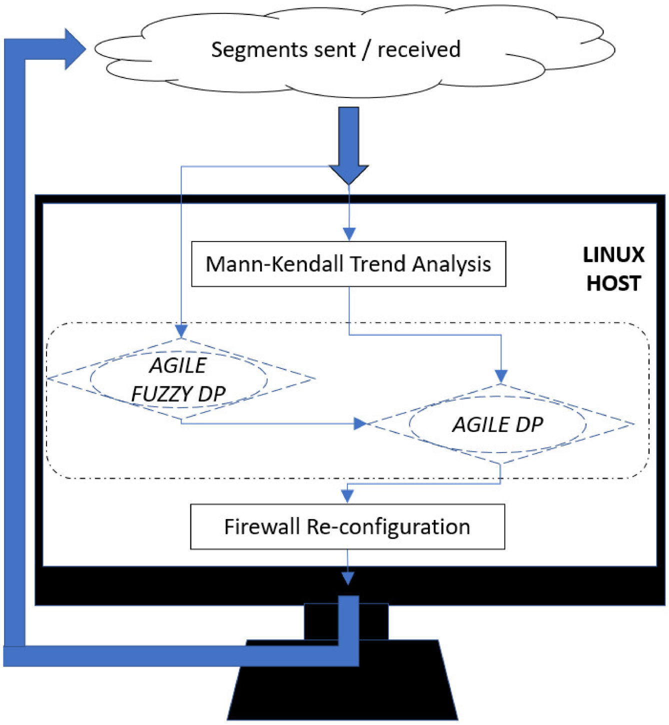


Fig. 3. Implementation architecture for host smart anomaly detection

It contains two elements representing the fixed section of the code:

1. Mann-Kendall trend analysis algorithm (which is fixed and does not require real-time changes).
2. Firewall re-configuration code that will, depending on the policy decision, switch the system (firewall) into one of two previously described operational/security modes (this section also does not need to be altered in real-time as the only thing it does is trigger a Bash script execution. The Bash script can be altered externally, if needed, to include/exclude restricted services.

The Decision Section from Fig. 1 is reflected by two Decision Points, one running AGILE FUZZY policy and one running AGILE policy. The policies are provided in GitHub repository at [34].

5.1. Test environment and setup

In order to present the implementation architecture in operation for host anomaly detection purposes, the test environment was located on a real production server. The server is normally used to provide some teaching resources for students, hence, taking into segments sent, segments received, etc.

5.2. Role of AGILE FUZZY and AGILE POLICIES

In the presented software component architecture (see: Fig. 3, both Decision Points must be populated with appropriate versions/types of policies written in AGILE Policy Definition Language (PDL). AGILE PDL was first presented in [9] as a language used to express a decision-making logic. Fuzzy logic extension of AGILE PDL allows defining fuzzy sets and specify parameters of fuzzy systems as well as define fuzzy rules.

The two above-mentioned policy types have different purpose in the proposed system. From the hierarchy point of view, the AGILE FUZZY policy is at lower level and its main role is to match input data into defined membership functions representing corresponding fuzzy sets. In the considered system the input data was assumed to be segments sent out (SSO) and change in segments sent out (SSODT) reflecting by how much the number of SSO changed compared to its previous level. The output variable is the level of risk based on which decisions may be taken regarding system re-configuration, applied security measures, etc. All those input and output data are assumed to be normalised (with values $[-1.0\dots 1.0]$. The specifics of the fuzzy sets definition in the AGILE FUZZY policy is shown as it is in Fig. 4.

```
<LinguisticVariables>
 <LVar Name="SSOVal" Type="Input"/>
 <MembershipFunctions>
 <MF Name="Small" Type="Mamdani"
 Value="0,0,0.25,0.5"/>
 <MF Name="Medium" Type="Mamdani"
 Value="0.25,0.5,0.5,0.75"/>
 <MF Name="High" Type="Mamdani"
 Value="0.5,0.75,1.0,1.0"/>
 </MembershipFunctions>
 </LVar>
 <LVar Name="SSODTVal" Type="Input"/>
 <MembershipFunctions>
 <MF Name="Minus" Type="Mamdani"
 Value="-1.0,-1.0,-0.5,0"/>
 <MF Name="Zero" Type="Mamdani"
 Value="-0.5,0,0,0.5"/>
 <MF Name="Plus" Type="Mamdani"
 Value="0,0.5,1.0,1.0"/>
 </MembershipFunctions>
 </LVar>
 <LVar Name="Risk" Type="Output"/>
 <MembershipFunctions>
 <MF Name="Small" Type="Mamdani"
 Value="0,0,0.25,0.5"/>
 <MF Name="Medium" Type="Mamdani"
 Value="0.25,0.5,0.5,0.75"/>
 <MF Name="High" Type="Mamdani"
 Value="0.5,0.75,1.0,1.0"/>
 </MembershipFunctions>
 </LVar>
</LinguisticVariables>
```

Fig. 4. Fuzzy sets definition

Fuzzy sets definition shown in Fig. 4 assumes their even distribution, any tuning procedure was applied. The rule base is shown in Table 1.

Table 1
Rule base

SSO \ SSODT	Small	Medium	High
Minus	Small	Small	Medium
Zero	Small	Medium	High
Plus	Medium	High	High

The role of the AGILE policy working at a higher level of the decision-making hierarchy is on one hand to get the information about the risk resulting from the actual values of the relevant metrics and on the other hand, it can utilise another type of information, e.g. trend analysis, to alter the risk assessment returned by the fuzzy system and provide its final evaluation. For example, a high value of segments sent out/segments received and a high value of its change will have different significance in case the trend is raising or falling.

In order to perform the final risk assessment, the AGILE policy shown in Fig. 5 defines a *ToleranceRangeCheck SSOLevelTRC* which checks how the risk assessment provided by the fuzzy system (represented by textutFLRisk variable) compares to a *FLRiskReference* reference value located in the middle of the output space. Tolerance value of *0.2* defines a dead zone which allows performing the same action *AInZone* for a range of values (in this case $[0.2\dots 0.3]$), Some alternative actions may be taken in case the risk assessment is located above or below the dead zone (respectively, *AHigher* and *ALower*.

```
<ToleranceRangeChecks>
 <TRC Name="RiskLevelTRC" Check="FLRisk"
 Compare="FLRiskReference"
 Tolerance="0.2" ActionInZone="AInZone"
 ActionLower="ALower" ActionHigher="AHigher"/>
</ToleranceRangeChecks>
```

Fig. 5. Fuzzy sets definition

Once the information (risk level assessed by the fuzzy logic system) is processed in the TRC, the next step is to put the information into a wider context, which is in relation to the trend detected in the monitored data. This is due to the fact that, as mentioned before, the situation is different if the changes are high but the trend is raising or falling (e.g. falling trend and high values of the risk do not require equally firm response from the system compared to the situation when the risk is high and the trend is already increasing). Hence, the next step is to consider the *FLRisk* value (representing the risk level returned by the fuzzy inference system) in relation to the trend. This is being done in a series of *Rules*, as shown in Fig. 6.

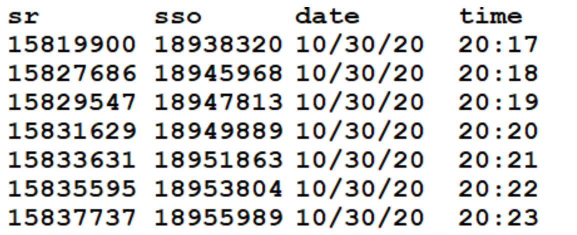
Just to remind, the *RuleInZone(...)* are processed in case the *FLRisk* value is in the dead zone defined in the TRC, the *RuleLower(...)* when the *FLRisk* value is above the defined dead zone and the *RuleHigher(...)* are processed otherwise.

```
<Rules>
 <Rule Name="RuleLowerFalling" LHS="Trend"
 Op="EQ" RHS="Falling"
 ActionIfTrue="RetLowAction"
 ElseAction="null"/>
 <Rule Name="RuleLowerNoTrend" LHS="Trend"
 Op="EQ" RHS="NoTrend"
 ActionIfTrue="RetLowAction"
 ElseAction="null"/>
 <Rule Name="RuleLowerRaising" LHS="Trend"
 Op="EQ" RHS="Raising"
 ActionIfTrue="RetMediumAction"
 ElseAction="null"/>
 <Rule Name="RuleInZoneFalling" LHS="Trend"
 Op="EQ" RHS="Falling"
 ActionIfTrue="RetLowAction"
 ElseAction="null"/>
 <Rule Name="RuleInZoneNoTrend" LHS="Trend"
 Op="EQ" RHS="NoTrend"
 ActionIfTrue="RetMediumAction"
 ElseAction="null"/>
 <Rule Name="RuleInZoneRaising" LHS="Trend"
 Op="EQ" RHS="Raising"
 ActionIfTrue="RetHighAction"
 ElseAction="null"/>
 <Rule Name="RuleHigherFalling" LHS="Trend"
 Op="EQ" RHS="Fallling"
 ActionIfTrue="RetLowAction"
 ElseAction="null"/>
 <Rule Name="RuleHigherNoTrend" LHS="Trend"
 Op="EQ" RHS="NoTrend"
 ActionIfTrue="RetHighAction"
 ElseAction="null"/>
 <Rule Name="RuleHigherRaising" LHS="Trend"
 Op="EQ" RHS="Raising"
 ActionIfTrue="RetHighAction"
 ElseAction="null"/>
 </Rules>
```

Fig. 6. Final risk assessment in a series of *Rules*

5.3. Test data

In order to gather some relevant statistics, a designated bash script was running every minute for over 2 weeks time on the server through the *crond* daemon . Internally, the script was running *netstat -s* and then was filtering some relevant metrics, out of which the segments sent out and segments received were the most important. Example chunk of data showing results from 10-minute monitoring is shown in Fig. 7.



```
sr       sso      date     time
15819900 18938320 10/30/20 20:17
15827686 18945968 10/30/20 20:18
15829547 18947813 10/30/20 20:19
15831629 18949889 10/30/20 20:20
15833631 18951863 10/30/20 20:21
15835595 18953804 10/30/20 20:22
15837737 18955989 10/30/20 20:23
```

Fig. 7. Interface statistics – segments sent out and segments received

These values were used to calculate the *SSO* and *SSODT*. Firstly, as the *netstat -s* command provides the segment sent out and segment received values at the given time measured from the interface activation time. As one can see, they increase in time. So, to get the information how many packets were sent out/received over a period of time it was necessary to take out the value at the beginning of the period from the value at the end of the period. This produced the SSO (and SR) values. Now, in order to know the change between the two succeeding periods it was necessary to take out the *SSO* value at the beginning of the period from the value at the end of the period. This produced the SSODT (and SRDT) values. As mentioned before, the granularity for getting the network statistics readings through the *crond* daemon was set to 1 minute. But due to the fact that this produced huge amounts of data, we decided to process every fifth of the recorded samples for the anomaly detection purposes (which corresponds to getting the network statistics readings every 5 minutes. Diagram showing both metrics are shown in, respectively, Fig. 8 and Fig. 9.

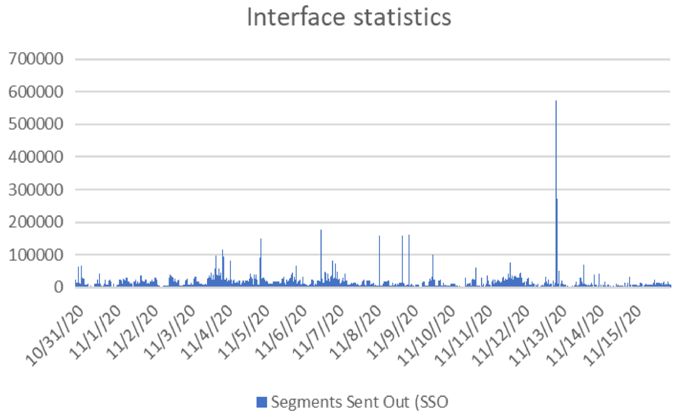


Fig. 8. Interface statistics – segments sent out (SSO)

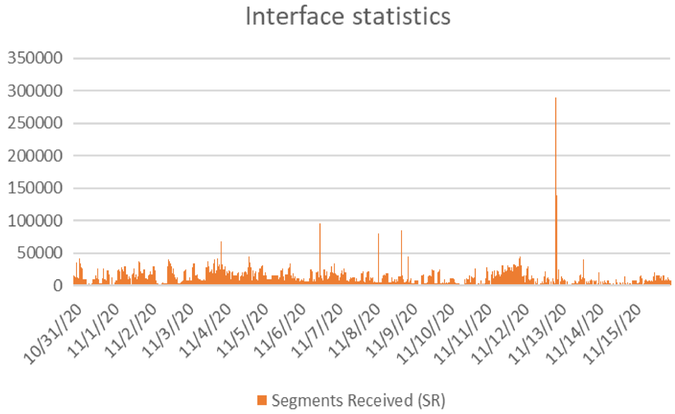


Fig. 9. Interface statistics – segments received (SR)

5.4. Test cases

As it was mentioned before, the AGILE policy uses trend analysis to make the final decision regarding final risk assessment. Trend analysis requires a number of data to work on and is strongly dependant on the sample size. In our simulation we decided to use a sliding window of width of 500 samples to calculate the trend. This means that the whole method will reach its nominal anomaly detection capability as soon as the required number of data is captured. Also, since the AGILE FUZZY policy requires normalised values of the *SSO* (*SR*) and *SSODT* (*SRDT*), all data were passed to the policy relative to maximum value within the window. This is the only solution for this kind of on-line algorithms where global maximum (maximum value that would ever appear) is unknown. The window size, apart from policy-related choices, in itself is actually one of the crucial parameters when it comes to the anomaly detection system sensitivity which makes it a very relevant test case for the simulation purposes.

In order to show the whole anomaly detection system in operation, two are three scenarios provided showing its main advantages.

- *Case I*: In this case the system is processing the data using different combination of AGILE and AGILE FUZZY policies for different sizes of moving windows: 200, 250, 300, 350, 400 and 450 samples. So in this case the effect of improved versions of policies will be demonstrated.
- *Case II*: In this case the system will in run-time switch at some point (around half of the simulation time) from one combination of AGILE and AGILE Fuzzy policy to another (to simulate the case where the system's decision making logic gets changed "on the fly").

For the simulation purpose there were in total of 2 AGILE and 2 AGILE FUZZY policies developed.The difference between the AGILE policies is that the second policy has different value of *FLRiskReference* variable (change from 0.5 to 0.8) which results in qualifying only high risk values (being outcome of the AGILE FUZZY policy) as those potentially requiring attention. And the difference between the two mentioned versions of AGILE FUZZY policies is that in the second policy the fuzzy system was tuned (*SOOVal* input linguistic variable was changed) so that it would qualify as really risky only those *SSO* values which are relatively substantially different from the rest. The tuned version of the *SSOVal* input linguistic variable definition is show in Fig. 10. Other variables remained as in the first version of AGILE FUZZY policy.

```
<LinguisticVariables>F
 <LVar Name="SSOVal" Type="Input"/>
 <MembershipFunctions>
 <MF Name="Small" Type="Mamdani"
 Value="0,0,0.4375,0.6875"/>
 <MF Name="Medium" Type="Mamdani"
 Value="0.4375,0.6875,0.6875,0.9375"/>
 <MF Name="High" Type="Mamdani"
 Value="0.6875,0.9375,1.0,1.0"/>
 </MembershipFunctions>
 </LVar>
</LinguisticVariables>
```

Fig. 10. *SSOVal* variable after Tuning

6. EVALUATION OF THE PROPOSED SYSTEM

In order to present the behaviour of the anomaly detection system in case it uses different versions of AGILE and AGILE FUZZY policies, a range of tests were carried out where each potential combination of the two AGILE and two AGILE FUZZY policies was applied in the system with different observation window sizes. The results are gathered in Table 2.

Table 2
Gathered results

window width	A1F1	A1F2	A2F1	A2F2
s = 200	570	277	181	84
s = 250	522	201	131	67
s = 300	482	154	95	53
s = 350	435	134	93	50
s = 400	398	108	69	38
s = 450	376	107	61	26

As one can see from Table 2, widening the window size results in reducing the number of situations when the system thought there is some sort of deviation from the typical behaviour. Differences are really spectacular, the system was able to reduce the number from 507 cases to only 26 leaving e.g. system admin or some other systems significantly less cases to analyse. So, even if the system would not be the final one on the decision making path, it would still be useful as some sort of screening system that identifies the most suspicious traffic. The units for the data shown in 2 (as well as 3) are the following: in the first column the window width is expressed in the number of samples while in the all remaining columns the results reflect the number of system reactions 9meaning how many times the proposed system qualified the processed data as indication some sort of anomaly).

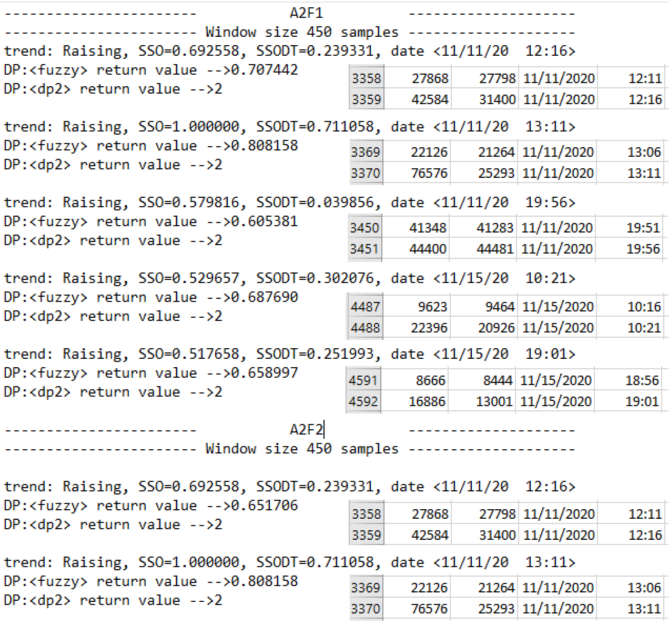
However, as indicated at the very beginning of this paper, our main goal was actually not to provide a new method for anomaly detection (though policy-based anomaly detection satisfies this requirement very well) but rather to show flexibility of the system resulting from the possibility of updating its anomaly detection logic in run-time. After doing so it is possible to improve the system efficiency in case there is a new knowledge about anomalies characteristic provided (in a form of a new policy) by a system expert. For that reason we have run a few tests showing the system behaviour in case the either, the AGILE or AGILE FUZZY policy gets updated on the fly. In table Table 3 we have shown some results for the scenario when the policy update is triggered around the middle of the simulation time. We tested a few configurations of AGILE and AGILE FUZZY policies as far as the switching procedure is concerned to show the effect of each policy substitution by a different version. The window sizes were assumed to be the same as in the previous set of simulations. For all simulations we assumed that the system starts with the first version of AGILE policy (which is denoted in the table as *A1*) with the *FLRiskReference* set to 0.5 and first version of AGILE FUZZY policy (denoted in the table as *F1* with evenly distributed (not tuned) membership functions for the input linguistic variable *SSOVal*.

Table 3
Selected results

window width	A1F1:A1F2	A1F1:A2F2	A1F1:A2F1
s = 200	404	291	356
s = 250	316	277	293
s = 300	300	276	279
s = 350	277	257	259
s = 400	242	225	227
s = 450	228	214	217

As one can see in Table 2, the number of system reactions clearly indicates that every time there was a switching from the A1F1 policy configuration to any other, the system precision in detecting potential anomalies was increasing. This clearly shows that policy update helps to improve the anomaly detection system accuracy.

A better understanding in terms of in which situations the system reacts is shown in Fig. 11.



```
----------------------          A2F1          --------------------
---------------------- Window size 450 samples --------------------
trend: Raising, SSO=0.692558, SSODT=0.239331, date <11/11/20  12:16>
DP:<fuzzy> return value -->0.707442      3358   27868   27798 11/11/2020   12:11
DP:<dp2> return value -->2               3359   42584   31400 11/11/2020   12:16

trend: Raising, SSO=1.000000, SSODT=0.711058, date <11/11/20  13:11>
DP:<fuzzy> return value -->0.808158      3369   22126   21264 11/11/2020   13:06
DP:<dp2> return value -->2               3370   76576   25293 11/11/2020   13:11

trend: Raising, SSO=0.579816, SSODT=0.039856, date <11/11/20  19:56>
DP:<fuzzy> return value -->0.605381      3450   41348   41283 11/11/2020   19:51
DP:<dp2> return value -->2               3451   44400   44481 11/11/2020   19:56

trend: Raising, SSO=0.529657, SSODT=0.302076, date <11/15/20  10:21>
DP:<fuzzy> return value -->0.687690      4487    9623    9464 11/15/2020   10:16
DP:<dp2> return value -->2               4488   22396   20926 11/15/2020   10:21

trend: Raising, SSO=0.517658, SSODT=0.251993, date <11/15/20  19:01>
DP:<fuzzy> return value -->0.658997      4591    8666    8444 11/15/2020   18:56
DP:<dp2> return value -->2               4592   16886   13001 11/15/2020   19:01

----------------------          A2F2          --------------------
---------------------- Window size 450 samples --------------------

trend: Raising, SSO=0.692558, SSODT=0.239331, date <11/11/20  12:16>
DP:<fuzzy> return value -->0.651706      3358   27868   27798 11/11/2020   12:11
DP:<dp2> return value -->2               3359   42584   31400 11/11/2020   12:16

trend: Raising, SSO=1.000000, SSODT=0.711058, date <11/11/20  13:11>
DP:<fuzzy> return value -->0.808158      3369   22126   21264 11/11/2020   13:06
DP:<dp2> return value -->2               3370   76576   25293 11/11/2020   13:11
```

Fig. 11. Sensitivity Difference

As one can see, the Fig. 1, shows differences in the systems reactions for two different policy configurations, respectively A2F1 and A2F2. So, the difference between these configurations is that the AGILE FUZZY policy changes from the *F1*, which is qualifying slightly lower values of the *SSO* data as constituting a potential risk into *F2* policy, which treats as potentially risky only those *SSO* values that really stand out from others within the observation window. This means that, with the *SSODT* value in both versions of the AGILE FUZZY policy having identical impact on the policy decisions (identical fuzzy sets), the only responsibility for the whole system decisions are mainly resulting from the values *SSO* values. Having a look at the policy decisions and the actual values of the *SSO* shown for each decision, one can see, e.g. for the decision made regarding data registered at 11/11/20 at 12:11–12:16, that for the *F1* AGILE POLICY the relative *SSO* value equal *0.692558* resulted in overall risk assessment at the level of *0.707442* whilst for the *F2* policy, for which the membership functions were shifted toward higher values, the same relative value of *SSO* resulted in overall risk assessment at the level of *0.651706*. This means that what was qualified as abnormal for the *F1* policy, was now equally bad for the *F2* policy. By analogy, the lower relative values of the *SSO* data, e.g. *0.597816* registered at *11/11/20 17:51-17:56* fell below the threshold defined in the AGILE policy *A2* (which was the same in both configurations) hence as a result the final decision of the system was to ignore them as not potentially anomalous.

The sequence of data processing in both policies is shown in Fig. 12.

```
~~~~ Loading script <"anomaly_f1.xml">
~~~~
~~~~ <"anomaly_f1.xml"> script loaded
successfully ~~
~~~~ DP:<"fuzzy"> policy evaluation ~~~~~~
 -->Evaluate Policy: "Policy1"<--------
 -->Execute Action: "Logic"<-----------
 -->Evaluate FuzzyRule: "FR1"<--
 -->Evaluate FuzzyRule: "FR2"<--
 -->Evaluate FuzzyRule: "FR3"<--
 -->Evaluate FuzzyRule: "FR4"<--
 -->Evaluate FuzzyRule: "FR5"<--
 -->Evaluate FuzzyRule: "FR6"<--
 -->Evaluate FuzzyRule: "FR7"<--
 -->Evaluate FuzzyRule: "FR8"<--
 -->Evaluate FuzzyRule: "FR9"<--
 -->Evaluate ReturnValue: "Out"<-------
DP:<fuzzy> return value -->0.707442
~~~~ Loading script <"anomaly_a2.xml">
~~~
~~~~ <"anomaly_a2.xml"> script loaded
successfully ~~
~~~~ DP:<"dp2"> policy evaluation ~~~~~~
 -->Evaluate Policy: "Policy1"<------
 -->Evaluate Template: "T1"<---------
 -->Execute Action: "Start"<---------
 -->Evaluate TRC: "RiskLevelTRC"<----
 -->Execute Action: "AInZone"<-------
 -->Evaluate Rule: "RuleInZoneNoTrend"<--
 -->Evaluate Rule: "RuleInZoneRaising"<--
 -->Execute Action: "RetHighAction"<--
 -->Evaluate ReturnValue:
 "OverallRiskHigh"<--
DP:<dp2> return value -->2
```

Fig. 12. Policy evaluation trace

The above trace shows that at first all the rules in the AGILE FUZZY policy are processed and the outcome of the policy is a specific risk level. That risk level is compared to the threshold value in the AGILE policy (specifically, in the *RiskLevelTRC*) and, as one can see in the specific trace, the *FLRisk* value for which the trace was recorded appeared to be in the dead zone. Next, the trend value was checked and as it is visible in the trace, the *RuleInZoneRaising* triggered the *RetHighAction* indicating that in this given circumstances the overall risk is high.

We can summarise the obtained results briefly by stating that the proposed system is fit for purpose – on one hand, it allows relatively easy system re-configuration which improves its efficiency and on the other hand, it is not resource-hungry which makes it implementable even in relatively resource-constrained systems (e.g. networks of IoT devices). The system performance strongly relies on expert knowledge expressed in the form of a policy which constitutes core logic of a network-enabled device – structure and policy objects used in the policy allow us to describe the system behaviour while various kind of threshold values guarantee fast adaptation – in case the traffic/parameters change, the logic may remain the same and the only thing needed adjustments is the threshold values.

7. CONCLUSION

In this paper we proposed a software architecture for smart network anomaly detection or threats analysis systems. The solution was thoroughly described as well as its main advantages in comparison to systems that are not equipped with such features. It was explained how such an architecture would allow expert knowledge to be easily expressed in a form of a high-level user-defined policy containing the main decision-making logic. This logic being the core element of the whole system can be subject of ‘on-the-fly’ (or scheduled) updates allowing to easily adapt to any changes regarding the tools and methods the attackers/hackers could use to compromise a system/device. Such an architecture is especially beneficial for resource-constrained systems which do not have enough resources to host software able to deal with all possible kinds of problems (e.g. detect anomaly or threat) but instead can be tuned up to become application- or mission-specific (the software hosted on the device may load logic from a policy which perfectly deals with a very specific problem leaving out all others).

We also provided a number of test cases to clearly demonstrate all the advantages the architecture can potentially bring and implemented the whole solution by developing a working program in C/Python programming languages running on a production server. Especially, we showed how much the anomaly detection system behaviour and accuracy may change in response to run-time policy changes.

Nevertheless, although the main goal of this paper was to focus on the architectural side of the solution, the final product – a working application implemented on a Raspberry PI device – proved ability to identify changes in the network traffic that clearly stand out from the rest. So, despite being only a proof of concept implementation it already demonstrated relatively good accuracy.

We did not test the system from the point of view of resource usage which would be a critical factor allowing to assess a potential application domain. But nowadays even some basic network devices (not to mention devices like routers or network servers) seem to be equipped with enough resources to be considered as capable to host a system implementing the software architecture we propose. Smart architecture and smart homes can play a crucial role in the near future [35–37].

REFERENCES

[1] P. Mulinka and P. Casas, “Stream-based machine learning for network security and anomaly detection,” in *Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, ser. Big-DAMA ’18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 1–7.

[2] A. Meyer-Berg, R. Egert, L. Böck, and M. Mühlhäuser, “Iot dataset generation framework for evaluating anomaly detection mechanisms,” in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ser. ARES ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 30.

[3] I.C. Paschalidis and Y. Chen, “Statistical anomaly detection with sensor networks,” *ACM Trans. Sen. Netw.*, vol. 7, no. 2, p. 17, Sep. 2010.

[4] G. Fernandes, E.H.M. Pena, L.F. Carvalho, J.J.P.C. Rodrigues, and M.L. Proença, “Statistical, forecasting and metaheuristic techniques for network anomaly detection,” ser. SAC ’15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 701–707.

[5] T.L. Fond, J. Neville, and B. Gallagher, “Designing size consistent statistics for accurate anomaly detection in dynamic networks,” *ACM Trans. Knowl. Discov. Data*, vol. 12, no. 4, p. 46, Apr. 2018.

[6] J.O. Kephart and D.M. Chess, “The vision of autonomic computing.” *Computer*, vol. 1, pp. 41–50, 2003.

[7] G. Pang, C. Shen, L. Cao, and A.V.D. Hengel, “Deep learning for anomaly detection: A review,” *ACM Comput. Surv.*, vol. 54, no. 2, p. 38, mar 2021, doi: 10.1145/3439950.

[8] J. Arevalo-Herrera, J.E. Camargo Mendoza, and J.I. Martinez Torre, “Network anomaly detection with machine learning techniques for sdn networks,” in *Proceedings of the 7th International Conference on Information and Education Innovations*, ser. ICIEI ’22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 129–135. [Online]. Available: https://doi.org/10.1145/3535735.3535750

[9] R.J. Anthony, “A policy-definition language and prototype implementation library for policy-based autonomic systems,” in *Proc. of 3rd International Conference on Autonomic Computing*. IEEE Computer Society, 2006, pp. 265–276.

[10] M. Pelc, “Context aware fuzzy control systems.” *Int. J. Softw. Eng. Knowl. Eng.*, vol. 24(5), pp. 825–856, 2014.

[11] M. Solaimani, M. Iftekhar, L. Khan, and B. Thuraisingham, “Statistical technique for online anomaly detection using spark over heterogeneous data from multi-source vmware performance data,” in *2014 IEEE International Conference on Big Data (Big Data)*, 2014, pp. 1086–1094.

[12] T. Liu, A. Qi, Y. Hou, and X. Chang, "Method for network anomaly detection based on bayesian statistical model with time slicing," in *2008 7th World Congress on Intelligent Control and Automation*, 2008, pp. 3359–3362.

[13] P. Kromkowski, S. Li, W. Zhao, B. Abraham, A. Osborne, and D.E. Brown, "Evaluating statistical models for network traffic anomaly detection," in *2019 Systems and Information Engineering Design Symposium (SIEDS)*, 2019, pp. 1–6.

[14] M.H. Bhuyan, D.K. Bhattacharyya, and J.K. Kalita, "Nado: Network anomaly detection using outlier approach," in *Proceedings of the 2011 International Conference on Communication, Computing & Security*, ser. ICCCS'11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 531–536.

[15] P. Kaur, "Outlier detection using kmeans and fuzzy min max neural network in network data," in *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)*, 2016, pp. 693–696, doi: 10.1109/CICN.2016.142.

[16] J. Mazel, P. Casas, Y. Labit, and P. Owezarski, "Sub-space clustering, inter-clustering results association & anomaly correlation for unsupervised network anomaly detection," in *Proceedings of the 7th International Conference on Network and Services Management*, ser. CNSM '11. Laxenburg, AUT: International Federation for Information Processing, 2011, pp. 73–80.

[17] K. Flanagan, E. Fallon, P. Connolly, and A. Awad, "Network anomaly detection in time series using distance based outlier detection with cluster density analysis," in *2017 Internet Technologies and Applications (ITA)*, 2017, pp. 116–121.

[18] T. Kenaza, K. Bennaceur, and A. Labed, "An efficient hybrid svdd/clustering approach for anomaly-based intrusion detection," ser. SAC '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 435–443.

[19] R. Bhatia, S. Benno, J. Esteban, T.V. Lakshman, and J. Grogan, "Unsupervised machine learning for network-centric anomaly detection in iot," ser. Big-DAMA '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 42–48.

[20] X. Lu, P. Liu, and J. Lin, "Network traffic anomaly detection based on information gain and deep learning," in *Proceedings of the 2019 3rd International Conference on Information System and Data Mining*, ser. ICISDM 2019. New York, NY, USA: Association for Computing Machinery, 2019, pp. 11–15.

[21] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 2828–2837.

[22] P.P. Chapke and R.R. Deshmukh, "Intrusion detection system using fuzzy logic and data mining technique," ser. ICARCSET '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 63.

[23] Z. Chiba, N. Abghour, K. Moussaid, A.E. Omri, and M. Rida, "A hybrid optimization framework based on genetic algorithm and simulated annealing algorithm to enhance performance of anomaly network intrusion detection system based on bp neural network," in *2018 International Symposium on Advanced Electrical and Communication Technologies (ISAECT)*, 2018, pp. 1–6.

[24] M. Bitaab and S. Hashemi, "Hybrid intrusion detection: Combining decision tree and gaussian mixture model," in *2017 14th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC)*, 2017, pp. 8–12.

[25] G. Pang, C. Shen, and A. van den Hengel, "Deep anomaly detection with deviation networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD'19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 353–362.

[26] K.M. Prasad, A.R.M. Reddy, and K.V. Rao, "Bartd: Bio-inspired anomaly based real time detection of under rated app-ddos attack on web," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 32, no. 1, pp. 73–87, 2020.

[27] L.K.G. Gonzalo P. Suárez and N.H. Fefferman, "A case study in tailoring a bio-inspired cyber-security algorithm: Designing anomaly detection for multilayer networks," *J. Cyber Secur. Mobil.*, vol. 8, no. 1, pp. 113–132, 2019.

[28] P. Ward, M. Pelc, J. Hawthorne, and R.J. Anthony, "Embedding dynamic behaviour into a self-configuring software system," in *Proceedings of 5th International Conference on Autonomic and Trusted Computing*. Springer LNCS, 2008, pp. 373–387.

[29] R.J. Anthony, M. Pelc, P. Ward, and J. Hawthorne, "A run-time configurable software architecture for self-managing systems," in *Proc. of ICAC 2008*. IEEE Computer Society, 2008, pp. 207–208.

[30] M. Pelc and R. Anthony, "Towards policy-based self-configuration of embedded systems," *SIWN Syst. Infor. Sci. Notes*, vol. 2, no. 1, pp. 20–26, 2007.

[31] H.B. Mann, "Non-parametric test against trend," *Econometrica*, vol. 13, pp. 245–256, 1945.

[32] M.G. Kendall, *Rank Correlation Methods*. Charles Griffin, 1975.

[33] M. Hussain and I. Mahmud, "Pymannkendall: a python package for non parametric mann kendall family of trend tests." *J. Open Source Softw.*, vol. 4, no. 39, p. 1556, 2019.

[34] M. Pelc, "Github policies repository," https://github.com/mariusz- pelc/policies, 2023 (accessed January 14, 2023).

[35] E. Dostatni, D. Mikołajewski, J. Dorożyński, and I. Rojek, "Ecological design with the use of selected inventive methods including ai-based," *Appl. Sci.*, vol. 12, no. 19, p. 9577, 2022.

[36] I. Rojek, E. Dostatni, D. Mikołajewski, L. Pawłowski, and K.M. Węgrzyn-Wolska, "Modern approach to sustainable production in the context of industry 4.0," *Bull. Pol. Acad. Sci. Tech. Sci.*, p. e143828, 2022.

[37] J. Vanus, J. Kubicek, O.M. Gorjani, and J. Koziorek, "Using the ibm spss sw tool with wavelet transformation for co2 prediction within iot in smart home care," *Sensors*, vol. 19, no. 6, p. 1407, 2019.