Learning Probabilistic Automata in the Context of IEC 104

Technická zpráva FIT VUT v Brně

Vojtěch Havlena, Lukáš Holík, Petr Matoušek

Technická zpráva č. FIT-TR-2020-01
Fakulta informačních technologií, Vysoké učení technické v Brně

Last modified: May 18, 2021

Learning Probabilistic Automata in the Context of IEC 104

Vojtěch Havlena, Lukáš Holík, Petr Matoušek

FIT, IT4I Centre of Excellence, Brno University of Technology, Czech Republic

Abstract. Industrial Control System (ICS) communication transmits monitoring and control data between industrial processes and the control station. ICS systems cover various domains of critical infrastructure such as the power plants, water and gas distribution, or aerospace traffic control. Security of ICS systems is usually implemented on the perimeter of the network using ICS enabled firewalls or Intrusion Detection Systems (IDSs). These techniques are helpful against external attacks, however, they are not able to effectively detect internal threats originating from a compromised device with malicious software. In order to mitigate or eliminate internal threats against the ICS system, we need to monitor ICS traffic and detect suspicious data transmissions that differ from common operational communication. In our research, we obtain ICS monitoring data using standardized IPFIX flows extended with meta data extracted from ICS protocol headers. Unlike other anomaly detection approaches, we focus on modelling the semantics of ICS communication obtained from the IPFIX flows that describes typical conversational patterns. This paper presents a technique for modelling ICS conversations using frequency prefix trees and Deterministic Probabilistic Automata (DPA). As demonstrated on the attack scenarios, these models are efficient to detect common cyber attacks like the command injection, packet manipulation, network scanning, or lost connection. An important advantage of our approach is that the proposed technique can be easily integrated into common security information and event management (SIEM) systems with Netflow/IPFIX support. Our experiments are performed on IEC 60870-5-104 (aka IEC 104) control communication that is widely used for the substation control in smart grids.

1 Introduction

Protection of the critical infrastructure that includes smart grids, water treatment, gas and oil distribution, railways or aerospace traffic control has become a challenge for security experts during past years [1, 2]. Cyber security is essential to the safe and reliable operation of modern industrial processes. Industrial Control Systems (ICS) and Supervisory Control and Data Acquisition (SCADA) systems are typically used in many industries to monitor and control physical processes. With adoption of IT technologies like TCP/IP or Ethernet, cyber attacks against ICS/SCADA systems become easier. The attacks on the industrial systems from the outside can be effectively filtered out on the perimeter of an

ICS network by ICS-enabled firewalls or IDS systems. This protection is, however, ineffective against the attacks originating from the inside of the network. Such attacks can be initiated by a malware installed on a control station, from a compromised host or a rogue device connected to the internal network. Attackers first scan the ICS network in order to identify potential attack targets and then they launch an attack that can control industrial processes, steal sensitive data or damage functionality of the system [3, 4], e.g., the cyber attack against the Ukrainian power grid in 2016 [5], cyber-espionage group APT33 targeting aerospace and energy sector in the U.S., Saudi Arabia and South Korea in 2017 [6], or the attack against pharmaceutical company Bayer in 2019.

In order to identify and eliminate internal cyber threats against the ICS system, we need to monitor ICS communication and detect suspicious behavior [7,8]. As showed in our previous work [9], high visibility of ICS communication can be achieved using IPFIX flow monitoring extended with meta data extracted from ICS protocol headers on the application layer. These so called ICS flow records contain flow properties extracted from the IP layer (source and destination IP addresses), transport layer (source and destination ports), and application layer (e.g., object ID, operation type, response type) [10]. ICS flow records also include statistical properties of the flow, e.g., the starting and ending time, the number of transmitted bytes, packets, etc., which make them a valuable source of data for anomaly detection [11]. Flow records are usually collected on the network management system where they are analyzed for security purposes [12].

In this paper, we apply a probabilistic approach to model the ICS traffic. The traffic is seen as a sequence of "conversations" between pairs of ICS devices. Each conversation is understood as a string with certain probability of occurrence in a typical traffic. Our approach is based on learning a deterministic probabilistic automaton (DPA) that describes the distribution of the occurrence probability over the conversations. For that, we use the learning algorithm [13]. A typical ICS traffic between two ICS/SCADA devices is stable, predictable, and uses a limited set of commands [14–16]. This makes it possible to learn a DPA that represents the ICS traffic accurately, and use it effectively to detect anomalies.

Anomaly detection (AD) compares a DPA representing an input network traffic with the previously learnt model. If these models differ, it means that either unknown conversations were found in the input data or that the legitimate communication strings appeared with an unusual frequency. This points either to malfunctioning of the network or a cyber attack.

The proof of concept of this technique was demonstrated in our previous work [10]. In this paper, we focus on effectiveness and accuracy of the method that is demonstrated on typical classes of cyber attacks [17]. This technical report is an extended version of paper published in the proceedings of IM'21 [18].

Contribution The main contribution of this paper is a technique that effectively models ICS communication using probabilistic automata. We consider two probabilistic models: deterministic probabilistic automata and frequency prefix trees. While prefix trees are easy and fast to construct, DPAs provide more compact

representation which is generated in polynomial time [13]. The second contribution involves anomaly detection using DPAs. We introduce two methods: the first one is based on computing the probability of a single conversation wrt. DPA, the second one compares two probabilistic distributions representing the learnt model and the input traffic. The proposed technique was designed so that (i) it is effective in detection of common cyber attacks on ICS networks, and (ii) can be easily implemented into a SIEM system. Anomaly detection using DPAs is demonstrated on IEC 104 traffic.

Structure of the Paper After introduction, Sec. 2 gives an overview of the recent research related to the anomaly detection of ICS and SCADA systems. Sec. 3 gives preliminaries on probabilistic automata. Sec. 4 describes a process how DPAs are generated from ICS flow records. Sec. 5 presents anomaly detection using DPAs. Results of our experiments with IEC 104 communication are given in Sec. 6. The last section concludes our work and discusses further research.

2 Related Work

Anomaly detection (AD) of ICS/SCADA communication has been explored by many research teams in previous years as a response to the increasing threats of cyber attacks against the critical infrastructure [7,16]. Unlike signature-based approach, anomaly detection creates a model of the legitimate behavior of an ICS system during normal operations. Then, AD system observes deviations of an input traffic wrt. the normal behavior model. If the deviation is higher then a given threshold, the input communication is marked as anomalous.

Rakas et al. [16] divide AD systems into three groups: statistical-based (univariate, multivariate, time series, cumulative sum), knowledge-based (finite automata, description scripts, expert systems), and machine learning-based (using Bayesian networks, Markov models, neural networks, fuzzy logic, etc.). Our approach is a combination of knowledge-based and machine learning-based techniques because we employ probabilistic approach as in Markov models and the model is implemented as a (probabilistic) automaton.

Similar approach to ours was explored by Lin and Nadjm-Tehrani [19,20] who observed three attributes of IEC 104 communication (AsduType, CoT, IOA) and created a probabilistic suffix tree (PST) that represented underlying timing patterns of spontaneous events for each attribute class. Using the changes of distribution of inter-arrival times, they categorized the traffic into five different groups based on periodicity and stability of observed times. They used PSTs to predict the future behavior of communication and detect possible changes. Their method is computationally demanding and sensitive to network delays. Instead of modelling timing features we focus on semantics of IEC 104 conversations in order to detect irregularities in exchanged commands.

Martinelli et al. [21] employ a network of timed automata (TA) to model the SCADA water distribution system. Numerical values of water tank level are mapped into three classes. Time changes represent edges in the TAs. Anomaly

detection is implemented using formal verification of pre-defined temporal logic formulae over the model. This method has a limited usage due to the manual creation of the model and high demands on model checking computation.

Goldenberg and Wool [22] similarly to us model semantics of ICS protocol, more specifically, sequences of queries and responses of Modbus communication. Their model employs deterministic finite automata (DFA) where symbols of the alphabet represent a tuple of a transaction ID, function code, reference number, and bit/word count of the Modbus packet. DFA transitions express the predicted behavior of the system which can be either normal, retransmission, miss, or unknown. The created model is sensitive to out-of-order messages and is able to recognize invalid messages. In our work, we also observe probability of transmitted messages that is important for detection of command injection and replay attacks.

Probabilistic approach to SCADA communication was applied by Caselli et al. [23, 24] who introduce a sequence-aware intrusion detection system based on discrete-time Markov chains (DTMC). The modeling process clusters all messages with the same semantic meaning to one state, e.g., read coils from address 0. Transitions represent a sequence of messages with probability related to the jump from state A to B. In our approach, messages are represented as strings accepted by a DPA rather than states as in Caselli's approach.

An important advantage of our system is that input data is obtained using standardized IPFIX flow monitoring [25]. Input flow records extended with ICS header values are sufficient to create an accurate model of ICS communication suitable for anomaly detection. To our best knowledge, we are not aware of any published work on using probabilistic automata for modelling ICS/SCADA semantics for anomaly detection.

3 Preliminaries

3.1 Probabilistic Automata

We write Σ^* to denote the set of all finite strings over an alphabet Σ, with ϵ denoting the empty string. A *deterministic probabilistic automaton* (DPA) is a tuple $\mathcal{A} = (\Sigma, Q, \delta, q_0, \mathbb{F})$ where Σ is an alphabet, Q is a finite set of *states*, $\delta : Q \times \Sigma \times Q \to [0, 1]$ is a (total) *transition function* assigning probabilities from the interval $[0, 1]$ of rational numbers to *transitions*, $q_0 \in Q$ is the *initial state*, and $\mathbb{F} : Q \to [0, 1]$ is a mapping assigning the *acceptance probabilities* to states.

The probabilistic automaton must satisfy the *consistency* condition requiring that for each state q, the sum of probabilities of the outgoing transitions plus the probability of acceptance is 1, that is, $\mathbb{F}(q) + \sum_{a \in \Sigma, r \in Q} \delta(q, a, r) = 1$. Additionally, since the automaton is implicitly deterministic, every state $q \in Q$ must have a unique successor via every symbol a, that is, $\forall q \in Q, \forall a \in \Sigma : |\{r \mid \delta(q, a, r) > 0\}| = 1$. Moreover for a state $q \in Q$ we define the *probabilistic automaton of* q as $\mathcal{A}_q = (\Sigma, Q, \delta, q, \mathbb{F})$.

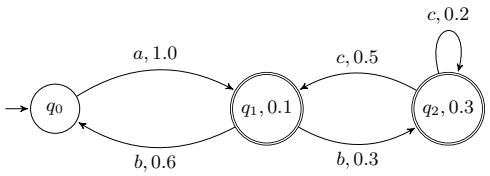


Fig. 1: Example of a probabilistic automaton. States are labeled with a state name and the accepting probability (no number corresponds to zero probability). Transitions are labelled with a symbol and the probability taking this transition.

The automaton defines a probability distribution $\mathcal{P}_\mathcal{A} : \Sigma^* \to [0, 1]$ over Σ^* as follows. Each string $w = a_1 \dots a_n \in \Sigma^*$ has its unique *trace*, the sequence $\pi = (q_0, a_1, q_1) \cdots (q_{n-1}, a_n, q_n)$ where $\delta(q_{i-1}, a_i, q_i) > 0$ for $1 \leq i \leq n$, and its probability is defined based on the trace as $\mathcal{P}_\mathcal{A}(w) = \mathbb{F}(q_n) \cdot \prod_{1 \leq i \leq n} \delta(q_{i-1}, a_i, q_i)$. Informally, $\mathcal{P}_\mathcal{A}(w)$ is the probability of the random walk through the automaton that respects the symbols of w and accepts at the end.

Example 1. Consider a DPA from Fig. 1. Then $\mathcal{P}_\mathcal{A}(abc) = 1.0 \cdot 0.3 \cdot (0.2 \cdot 0.3 + 0.5 \cdot 0.1) = 0.033$.

A *deterministic frequency finite automaton* (DFFA) is a tuple $\mathcal{A} = (\Sigma, Q, \delta, q_0, \mathbb{F})$ that differs from a probabilistic automaton only so that δ and $\mathbb{F}$ assign natural numbers representing frequencies to transitions and states, i.e., $\delta : Q \times \Sigma \times Q \to \mathbb{N}$ and $\mathbb{F} : Q \to \mathbb{N}$, and that *consistency* here means that there is no state with the *overall frequency* equal to 0, where the overall frequency of a state q is $\mathcal{C}(q) = \mathbb{F}(q) + \sum_{a \in \Sigma, r \in Q} \delta(q, a, r)$.

An DFFA can be *normalized* to an DPA by dividing the acceptance frequencies of each state q and frequencies of its outgoing transitions by its overall frequency $\mathcal{C}(q)$, see [13] (we denote this operation as Normalize).

3.2 Protocol IEC 104

For our experiments, we deal with the IEC 60870-5-104 (aka IEC 104) protocol [26] that is widely used in smart grids for substation control. IEC 104 is running on application layer of the TCP/IP model. The IEC 104 packet is formed by the fixed-length Application Protocol Control Information (APCI) header and Application Service Data Unit (ASDU) [27].

Control fields in the APCI define three types of the IEC 104 packet: *u-frames* used for tests, start and stop of data transfers, *s-frames* for supervisory function and *i-frames* that encapsulate ASDU data units exchanged between a central telecontrol station (master) and telecontrol outstation (slave).

The ASDU contains a type (e.g., single point of information, measured valued, single command, file ready), cause of transmission (periodic, spontaneous, activation, confirmation), information object address (IOA), and a list of information objects and elements with data transmitted to/from a substation.

In our work, we focus on i-frames only and two ASDU attributes: type (AsduType) and Cause of Transmission (CoT).

4 Modelling SCADA Communication Using Probabilistic Automata

In this section, we give a brief overview of the approach we use for learning probabilistic automata models of network communication, and of the specific techniques we use to pre-process the ICS flow records for the learning algorithm in order to provide meaningful results.

4.1 Learning Deterministic Probabilistic Automata

We first briefly outline the DPA learning algorithm Alergia from [13] that we use in our framework. Given a multiset S of strings on the input, the algorithm outputs a DPA that approximates the probabilities of the individual strings in S. The algorithm proceeds in the following steps:

1. Create a prefix tree with strings from S where each edge is labeled by the frequency of occurrences of the respective string prefix in S. Interpret the prefix tree as an DFFA.
2. Generalize and compact the DFFA by merging "similar" states. In our experiments, we consider two version of the algorithm, one which includes this step and one which does not. We call the former version (with merging) *Alergia* and the latter version *Prefix tree*.

Now we will describe steps 1 and 2 in more detail.

Prefix tree. The prefix tree is a compact (but still precise) representation of the multiset S. Its nodes are prefixes of strings in S (hence ϵ is the root) and there is an edge labeled by the symbol a from u to $u.a$ if and only if both u and $u.a$ are prefixes of strings from S, see Fig. 2. The edge is also labeled by the number of occurrences of the prefix $u.a$ in S, that is, by the number $\sum_{w \in S, \exists v: w = u.v} S(w)$. Note that by $S(w)$ we denote the number of occurrences of string w in S. DFFA

The prefix tree may be interpreted as a frequency automaton (this we denote as $\mathsf{Fpt}(S)$), where nodes are states, edges correspond to transitions, ϵ is the initial state, and the acceptance frequency of a each state w equals $S(w)$. Further, normalized $\mathsf{Fpt}(S)$ we call *prefix tree automaton of S*

Prefix tree minimization. The prefix tree automaton for a give set of strings may be large (basically, it is a tree-shaped PA) with some parts representing the same probabilistic distribution. In particular, given a PTA $\mathcal{A}$ we merge states p and q if $\mathcal{P}_{\mathcal{A}_p} = \mathcal{P}_{\mathcal{A}_q}$. Note that since only a finite number of strings in $\mathcal{P}_{\mathcal{A}_p}$ have nonzero probability, the states to be merged can be efficiently computed.

Generalization. Generalization is the main part of Alergia. Here, we will outline only the basic idea. Further details we provide in the following sections (or see [13]). The algorithm performs an exploration of the prefix tree automaton from the initial state (the root). While exploring the tree, it merges states r

on the frontier (*Blue* set) of the so far undiscovered part of the tree with the previously discovered states q (*Red* set).

Merging is a recursive procedure that merges the sub-tree rooted by r into the automaton reachable from q. The acceptance frequency of r is added to the acceptance frequency of q. Moreover, for each symbol a, the frequency of the outgoing a-transition of r is added to the frequency of the outgoing a-transition of q, and the merging procedure is recursively called on the target states of the two merged transitions (Merge and Fold operation).

Two states q and r are merged under the condition that they are sufficiently similar. Similarity here means that their acceptance frequencies are close enough as well as the frequencies of the outgoing a-transitions for each symbol a. What similarity is sufficient is controlled by the parameter α of the algorithm. α also corresponds to the probability that the merged automaton wrongly rejects a string from S. Additionally, states that are too insignificant, i.e., have a too small overall frequency, are excluded from merging no matter their similarity. The threshold overall frequency t_0 is the second parameter of the algorithm (Compatible operation).

Merging States The learning algorithm interatively merges a state from the *Blue* set and a state from the *Red* set. Recall that the *Blue* set contains still unprocessed states of the prefix tree. Therefore states from the *Blue* set have the only unique predecessor (for arbitrary symbol). If we hence wants to merge a state q_b from the *Blue* set to a state q_r from the *Red* set, we only need to find the unique predecessor r of q_b and change the destination of the transition going from r to q_b to q_r. See lines 1–3 in Alg. 1 for more details.

Algorithm 1: Merge operation

Input: A DFFA $\mathcal{A}$, states $q_r \in \mathit{Red}, q_b \in \mathit{Blue}$
Result: Modified DFFA with merged states q_r and q_b
1 Find a state $p \in Q$ and a symbol $a \in \Sigma$ s.t. $v > 0$ where $v = \delta_{fr}(p, a, q_b)$;
2 $\delta_{fr}(p, a, q_b) \leftarrow 0$;
3 $\delta_{fr}(p, a, q_r) \leftarrow v$;
4 **return** Fold$(\mathcal{A}, q_r, q_b)$;

However, the transition redirection described above is not sufficient for merging two states. When we redirect the transition, we loose the connection with subtree rooted at q_b (because it has the only predecessor and we removed this transition). Therefore, we need to insert (or fold) this subtree into the automaton starting from the state q_r. The insertion (or folding) of this subtree to the automaton is done recursively wrt. the subtree. In each recursive call we update the current transitions in the automaton according to current transitions in the subtree. This may lead to a possibly update of a frequency of the current transition in the automaton (if there is a matching transition in the current subtree

and in the current automaton) or to add a new transition to the automaton appending the remaining part of the subtree. See Alg. 2 for more details. This recursive operation is called as a last step of the Merge operation (Alg. 1).

Algorithm 2: Fold operation

Input: A DFFA $\mathcal{A}$, states q_1, q_2
Result: Modified DFFA with folded states q_1 and q_2 (the subtree rooted at q_2 is removed)

$\mathbb{F}_{fr}(q_1) \leftarrow \mathbb{F}_{fr}(q_1) + \mathbb{F}_{fr}(q_2)$;
foreach $a \in \Sigma$, $r_2 \in Q$ *s.t.* $\delta_{fr}(q_2, a, r_2) > 0$ **do**
  if $\exists r_1 \in Q$ *s.t.* $\delta_{fr}(q_1, a, r_1) > 0$ **then**
    $\delta_{fr}(q_1, a, r_1) \leftarrow \delta_{fr}(q_1, a, r_1) + \delta_{fr}(q_2, a, r_2)$;
    $\mathcal{A} \leftarrow \mathsf{Fold}(\mathcal{A}, r_1, r_2)$;
  else
    $\delta_{fr}(q_1, a, r_2) \leftarrow \delta_{fr}(q_2, a, r_2)$;
return $\mathcal{A}$;

The Algorithm *Alergia* In the last part we describe the details of the algorithm *Alergia*. The final tile of a puzzle is how to determine whether two states are compatible for merging (in other word whether they are "similar" as stated at point 2 in the high-level description above).

To determine two states q_1 and q_2 are compatible for merging we first check whether the final-state frequencies $\mathbb{F}_{fr}$ are similar wrt. the value of $\mathcal{C}$. For the testing whether the frequencies are similar the Hoeffding bounds are used (see the definition of Test in (1)). The test depends also on the parameter α. This parameters says how intensively the states should be merged. In the second step we compare the corresponding outgoing transitions of both states (the same test as for comparing the final-state frequencies is used). See Alg. 3 for more details.

$$\mathsf{Test}(f_1, n_1, f_2, n_2, \alpha) = \left| \frac{f_1}{n_1} - \frac{f_2}{n_2} \right| < \left(\sqrt{\frac{1}{n_1}} + \sqrt{\frac{1}{n_1}} \right) \cdot \sqrt{\frac{1}{2} \ln \frac{2}{\alpha}}. \qquad (1)$$

Now we have all ingredients for the algorithm *Alergia*. The algorithm works as described in high-level description in the beginning of this section. In the first step we create frequency prefix tree for a multiset of strings S. The *Red* set contains initially only the root of the prefix tree. The *Blue* set contains direct successors of the *Red* set. Then we iteratively select a state q_b from *Blue* set s.t. the value $\mathcal{C}(q_b)$ is above threshold t_0. This threshold determines the minimal number of strings that are necessary to be a state considered for merging. Then we try to find a red state q_r, which is compatible for merging with q_b. If we find such a state, we merge them together. If there is no such state we add q_b to the *Red* set. Finally we update the *Blue* set to contain direct successors of the red states. In the last step we normalize DFFA to obtain a DPA. Basically for

Algorithm 3: Compatible operation

Input: A DFFA $\mathcal{A}$, states q_1, q_2, $\alpha > 0$
Result: Are states q_1, q_2 compatible for merging?

if $\neg\mathsf{Test}(\mathbb{F}_{fr}(q_1), \mathcal{C}(q_1), \mathbb{F}_{fr}(q_2), \mathcal{C}(q_2), \alpha)$ **then**
  return *false*;
foreach $a \in \Sigma$ **do**
  Find $r_1 \in Q$, $r_2 \in Q$ s.t. $\delta_{fr}(q_1, a, r_1) > 0$ and $\delta_{fr}(q_2, a, r_2) > 0$;
  $v_1 \leftarrow \delta_{fr}(q_1, a, r_1)$;
  $v_2 \leftarrow \delta_{fr}(q_2, a, r_2)$;
  if $\neg\mathsf{Test}(v_1, \mathcal{C}(q_1), v_2, \mathcal{C}(q_2), \alpha)$ **then**
    return *false*;
return *true*;

each state q we normalize the value of $\mathbb{F}_{fr}(q)$ and the frequency of each outgoing transition with $\mathcal{C}(q)$. See Alg. 4 for more details.

Algorithm 4: The algorithm *Alergia*

Input: A multiset of strings S, $\alpha > 0$, $t_0 > 0$
Result: A DPA $\mathcal{B}$

$\mathcal{A} \leftarrow \mathsf{Fpt}(S)$;
$Red \leftarrow \{q_\varepsilon\}$;
$Blue \leftarrow \{q_a \mid a \in \Sigma \cap \mathsf{Pref}(S)\}$;
while *Choose* q_b *from Blue s.t.* $\mathcal{C}(q_b) \geq t_0$ **do**
  if $\exists q_r \in Red : \mathsf{Compatible}(\mathcal{A}, q_r, q_b, \alpha)$ **then**
    $\mathcal{A} \leftarrow \mathsf{Merge}(\mathcal{A}, q_r, q_b)$;
  else
    $Red \leftarrow Red \cup \{q_b\}$;
  $Blue \leftarrow \{q_{ua} \mid ua \in \mathsf{Pref}(S) \wedge q_u \in Red\} \setminus Red$;
return $\mathcal{B} = \mathsf{Normalize}(\mathcal{A})$;

4.2 Data Pre-processing

We will now describe the way in which we obtain ICS flow records and in which we pre-process them to prepare a suitable sample set S for the DPA learning algorithm.

Collecting ICS flows To collect ICS flow[1], we need to monitor ICS network by an IPFIX monitoring probe with ICS protocol support[2]. The probe observes

[1] ICS flow is an IPFIX flow extended with ICS meta data [28].
[2] This is supported by Flowmon probe, see `https://www.flowmon.com/en/solutions/solutions-by-industry/industrial-control-systems-scada` [Sept 2020]

passing traffic and creates ICS flow records[3] with meta data extracted from ICS headers. Flow records describing ICS communication within a given time window are transmitted to a IPFIX collector or SIEM system. Using ICS flows we learn a high-level communication model that includes ICS semantics, e.g., requested operations, device status, etc. In case of IEC 104 protocol, we focus on *i*-messages, i.e., IEC 104 messages that transmit application commands [27].

Partitioning the traffic by communication pairs Given a network flow records, our aim is to obtain an automaton for each pair of communicating devices describing the communication between the two. We therefore partition the traffic according to the communication pairs. This is easily done since each device is uniquely identified by a pair ⟨IP address, port⟩.

Splitting the traffic into conversations The learning algorithm from Sec. 4.1 takes a multiset of strings as the input. Network traffic is represented by ICS flow records which correspond to a single sequence of messages. Therefore, we first divide ICS flow records into a multiset of *conversations*, i.e., sequences of logically connected messages that correspond to one "communication session" of two devices. The sample set S then consists of the conversations and the learnt probabilistic automata denote a probability distribution over conversations. Recall that we work with messages on the application layer, thus, there can be multiple ICS conversation within one TCP session. This is typical for IEC 104 protocol.

Identification of a conversation in the sequence of flow records is based on the expert knowledge of the particular ICS protocol. In case of IEC 104 protocol, the conversation is started by messages with AsduType = 122 (select file), and packets with CoT = 7 (confirmation activation), 6 (activation), 3 (spontaneous). To check, if the conversation is *complete*, we test if the last packet meets the following criteria. Conversations are finished by messages with AsduType with 6 (end of initialization), 122–123 (last segment), 124 (ACK file), and packets with CoT = 7 (confirmation activation), 9 (confirmation deactivation), or 44–47 (unknown resource) [27].

Message abstraction To represent normal network communication using automata, we need to set a suitable level of abstraction and remove irrelevant details from the messages. Too much details would lead to an over-specialised learnt model that marks small nuances in communication as anomalies while too little details would blur the boundaries between normal communication and anomalies. For instance, each message (flow record) contains a timestamp, which makes the message unique. The learning procedure hence could hardly find any regular structure in the communication.

For IEC 104 protocol [27], we particularly take into account fields AsduType and CoT that determine the high-level communication model, and abstract from

[3] The flow record contains meta data about the flow, e.g., timestamp, src and dst address, msg length, duration, etc., see [25].

fields containing concrete data values, time, etc. A message (ICS flow record) after abstraction is modelled as a pair ⟨AsduType, CoT⟩. Thus, a conversation between two IEC 104 devices is a sequence of such pairs.

Example 2. Consider a sample of conversations S consisting of four conversations starting with prefix ⟨122,12⟩.⟨120,13⟩.⟨122,13⟩ . . . and four with only one message ⟨36,3⟩. Then the automaton constructed from this sample using the Prefix tree approach is shown in Fig. 2.

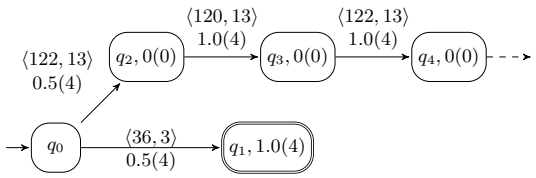


Fig. 2: Prefix tree automaton created from IEC 104 flows. The numbers in brackets denote labels of the prefix tree, numbers in parenthesis express the number of prefix occurrences

.

Packet Loss Detection When dealing with real network traffic, a natural question arise. What happen, if some packets are lost (not due to an attack/anomaly, but for instance due to a realibility of the network). In this part we describe how to cope with packet loosing. Our procedure works upon two assumptions: (i) such corrupted conversations occurr rarely, and (ii) there is some other conversation v that is "similar" to the corrupted one. Here the similarity means that we can add packets (symbols) at some positions of the corrupted conversation and obtain v. Ad (ii). For two strings $u = a_1 \dots a_n$ and v the number of symbols that we need to add to u in order to obtain v can be computed as follows.

$$dist(u, v) = \begin{cases} |v| - |u| & \text{if } v \in \mathcal{L}(\mathcal{A}_u^*), \\ \infty & \text{otherwise.} \end{cases} \tag{2}$$

where $\mathcal{A}_u^*$ is a NFA accepting the language $\Sigma^* a_1 \Sigma^* \dots \Sigma^* a_n \Sigma^*$.

Our procedure then checks if we are able to "repair" less frequent strings from S. An assumption is that among the less frequent strings (conversations) there could be the corrupted ones. For that we split the sample into two parts; high frequent conversations, i.e., $S^{high} = \{w \mid S(w) > \eta_0\}$ where η_0 determines maximum number of occurrences for a conversation to be considered as "less frequent", and low frequent confersations $S^{low} = S \setminus S^{high}$. The repairment then works as follows. If for a string $w \in S^{low}$ we find a string $v \in S^{high}$ s.t. $dist(w) \leq \eta_1$, we replace w in S with v. Note that η_1 is a parameter determining

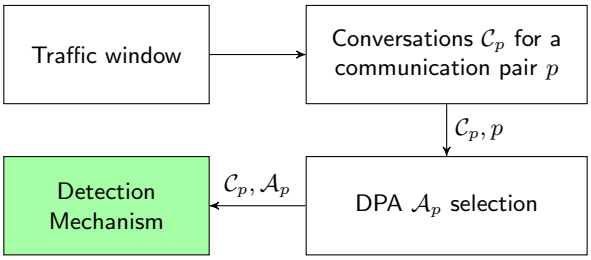


Fig. 3: Overview of the anomaly detection.

how many packets could be lost from a conversation. Such adjusted sample then serves as an input for learning/detecting anomalies.

In summary, the data pre-processing includes three steps:

1. From a given dataset, we extract only the IEC 104 flow records with i-messages and partition them by pairs of communication entities.
2. The modified traffic is further split into conversations.
3. We apply the abstraction on each message (possibly with packet loss detection).

The pre-processed data forms the input for learning as described in Sec. 4.1.

5 Anomaly Detection

Now we show how the learnt model of the network traffic is used to detect anomalies. Our detection mechanism works on the level of time windows of a fixed duration (particularly, 5 minutes) that are collected by the IPFIX monitoring probe. The length of a time window is not a fixed parameter and it may be changed. Its value corresponds to the Netflow export timeout recommended for flow monitoring in order to minimize flooding of the network by monitoring data. For time critical systems the timeout can be shortened. The detection has three consecutive phases, also shown in Fig. 3:

1. The time window is divided into a series of conversations $\mathcal{C}_p$ for each pair of communication devices p identified by end-to-end IP addresses and ports.
2. A learnt probabilistic automaton A_p describing the normal communication of p is selected using the end-to-end IP addresses and ports.
3. Anomalies are detected based on comparing $\mathcal{C}_p$ with $\mathcal{A}_p$.

The last step, anomaly detection based on a comparison of $\mathcal{C}_p$ and $\mathcal{A}_p$, is implemented as follows.

5.1 Anomaly Detection via Single Conversation Reasoning

The first mechanism for anomaly detection (we call it *Single*) is based on reasoning about individual conversations. For each conversation $c \in \mathcal{C}_p$, we compute

the probability $\mathcal{P}_{\mathcal{A}_p}(c)$ assigned to c by the probabilistic automaton $\mathcal{A}_p$ representing valid communication of the pair of devices p. If the probability is below the threshold μ, i.e., $\mathcal{P}_{\mathcal{A}_p}(c) \leq \mu$, an anomaly is detected. In this work, we set μ to 0, meaning that we are only interested in whether $\mathcal{A}_p$ marks c as possible (no matter how far), or not. The advantage of this mechanism is that it allows to point to the concrete conversation causing the anomaly.

5.2 Anomaly Detection via Distribution Comparison

The second mechanism focuses on evaluating each 5-minute traffic window as a whole (instead of on evaluating individual conversations in isolation). The probabilistic distributions of every window is compared to the probabilistic distribution of the learnt model of the normal communication traffic. This way, we can detect anomalies caused by missing conversations (e.g., a device stops responding) or by a change of a communication profile, which the method *Single* cannot detect.

The detection mechanism works as follows. We learn a DPA $\mathcal{A}'_p$ from a tested sequence of conversations $\mathcal{C}_p$ coming from the traffic window under scrutiny. We then compare $\mathcal{A}'_p$ with the DPA $\mathcal{A}_p$ (representing the normal traffic) and if the difference is too large, we report an anomaly. To quantify how much different is $\mathcal{A}_p$ from $\mathcal{A}'_p$, we use the 2-Euclid distance (or just Euclid distance), defined as

$$L_2(\mathcal{A}_p, \mathcal{A}'_p) = \sqrt{\sum_{w \in \Sigma^*} \left(\mathcal{P}_{\mathcal{A}_p}(w) - \mathcal{P}_{\mathcal{A}'_p}(w)\right)^2} \tag{3}$$

Intuitively, the Euclid distance sums the differences of probabilities assigned to strings by $\mathcal{A}_p$ and $\mathcal{A}'_p$. We use a parameter θ to control if these two automata are different enough to mark anomaly, i.e., $L_2(\mathcal{A}_p, \mathcal{A}'_p) > \theta$. The value of θ expresses sensitivity of detection in interval $[0, 1]$. Lower value means higher possibility of false alarms, higher values can cause that some anomalies would not be discovered. Based on our experiments we recommend values from 0.1 to 0.25.

A good news is that even though the sum in the definition of the Euclid distance ranges over all strings, distance L is computed in a polynomial time. The algorithm uses a matrix representation of probabilistic automata and on expressing the infinite sum in a closed form (see [29] or the following paragraphs for more details).

Euclid Distance Computation Before we focus on a computation of the Euclid distance we first provide necessary definitions. In the following text we use notion of *subprobabilistic automaton* (SPA). The subprobabilistic automaton is defined as probabilistic automaton except in the consistency condition, the equality relation $=$ is replaced with $\leq$. For two PAs $\mathcal{A}_1 = (\Sigma, Q_1, \delta_1, i_1, \mathbb{F}_1)$ and $\mathcal{A}_2 = (\Sigma, Q_2, \delta_2, i_2, \mathbb{F}_2)$ we define the product subprobabilistic automaton $\mathcal{A}_1 \odot \mathcal{A}_2 = (\Sigma, Q_1 \times Q_2, \delta_3, (i1, i2), \mathbb{F}_3)$ where $\delta_3((q_1, p_1), a, (q_2, p_2)) = \delta_1(q_1, a, q_2) \cdot$

$\delta_2(p_1, a, p_2)$, and $\mathbb{F}_3((q,p)) = \mathbb{F}_1(q) \cdot \mathbb{F}_2(p)$. Every SPA $\mathcal{A} = (\Sigma, Q, \delta, i, \mathbb{F})$ can be represented in matrix notation, denoted as $\mathcal{M}(\mathcal{A})$ and defined as $\mathcal{M}(\mathcal{A}) = (\boldsymbol{\alpha}, \{\mathbf{A}_a\}_{a\in\Sigma}, \boldsymbol{\beta})$, where $\boldsymbol{\alpha}$ is the vector of initial probabilities defined as $\boldsymbol{\alpha}[q] = 1.0$ if $q = i$ and $\boldsymbol{\alpha}[q] = 0$ otherwise. Vector $\boldsymbol{\beta}$ is the vector of acceptance probabilities defined as $\boldsymbol{\beta}[q] = \mathbb{F}(q)$, and $\mathbf{A}_a$ is the transition matrix for symbol a defined as $\mathbf{A}_a[q, q'] = \delta(q, a, q')$ for each $q, q' \in Q$ and $a \in \Sigma$. Further, we define $\mathbf{A}_\Sigma = \sum_{a\in\Sigma} \mathbf{A}_a$.

Using this matrix representation of SPAs the Euclid distance can be computed using Alg. 5.

Algorithm 5: Computation of L_2

Input: DPAs $\mathcal{A}$ and $\mathcal{A}'$
Result: $L_2(\mathcal{A}, \mathcal{A}')$
1 $(\boldsymbol{\alpha}_1, \{\mathbf{A}^1_a\}_{a\in\Sigma}, \boldsymbol{\beta}_1) \leftarrow \mathcal{M}(\mathcal{A} \odot \mathcal{A})$;
2 $(\boldsymbol{\alpha}_2, \{\mathbf{A}^2_a\}_{a\in\Sigma}, \boldsymbol{\beta}_2) \leftarrow \mathcal{M}(\mathcal{A} \odot \mathcal{A}')$;
3 $(\boldsymbol{\alpha}_3, \{\mathbf{A}^3_a\}_{a\in\Sigma}, \boldsymbol{\beta}_3) \leftarrow \mathcal{M}(\mathcal{A}' \odot \mathcal{A}')$;
4 $r_i \leftarrow \boldsymbol{\alpha}_i^\top (\mathbf{I} - \mathbf{A}^i_\Sigma)^{-1} \boldsymbol{\beta}_i$ for $i \in \{1, 2, 3\}$;
5 **return** $\sqrt{r_1 - 2r_2 + r_3}$;

The intuition behind Alg. 5 is the following reasoning:

$$
\begin{aligned}
L_2^2(\mathcal{A}, \mathcal{A}') &= \sum_{w\in\Sigma^*} \left(\mathcal{P}_\mathcal{A}(w) - \mathcal{P}_{\mathcal{A}'}(w)\right)^2 \\
&= \sum_{w\in\Sigma^*} \Big((\mathcal{P}_\mathcal{A}(w))^2 - 2\mathcal{P}_{\mathcal{A}'}(w)\mathcal{P}_\mathcal{A}(w) + (\mathcal{P}_{\mathcal{A}'}(w))^2 \Big) \\
&= \sum_{w\in\Sigma^*} \mathcal{P}_{\mathcal{A}\odot\mathcal{A}}(w) - 2 \sum_{w\in\Sigma^*} \mathcal{P}_{\mathcal{A}'\odot\mathcal{A}}(w) \sum_{w\in\Sigma^*} \mathcal{P}_{\mathcal{A}'\odot\mathcal{A}'}(w)
\end{aligned}
\tag{4}
$$

Further, we use the following lemma ensuring that the product of two probabilistic automata is a subprobabilistic automaton (i.e, for a given state, the sum of outgoing probabilities and the accepting probability is less or equal to 1).

Lemma 1. *Let $\mathcal{A}, \mathcal{A}'$ be PAs. Then $\mathcal{A} \odot \mathcal{A}'$ is a SPA.*

The last tile into the puzzle is the following lemma showing how to compute sum of probabilities of all strings of a given SPA using the matrix representation of the SPA. Note that for a PA this value is 1, but for general SPA it is a value ≤ 1.

Lemma 2. *Let $\mathcal{A}$ be a SPA. Then, $\sum_{w\in\Sigma^*} \mathcal{P}_\mathcal{A}(w) = \boldsymbol{\alpha}^\top (\mathbf{I} - \mathbf{A}_\Sigma)^{-1} \boldsymbol{\beta}$ where $\mathcal{M}(\mathcal{A}) = (\boldsymbol{\alpha}, \{\mathbf{A}_a\}_{a\in\Sigma}, \boldsymbol{\beta})$.*

Finally, using the reasoning about the Euclid distance and Lemmas 1 and 2 we get directly Alg. 5.

Table 1: Datasets used for experimental evaluation

Benchmark	IEC 104 flows	i-messages		Conv.	Devices
`iec104`	115	91	31	2	
`10122018-104Mega`	104,533	94,040	6,927	4	
`10122018-104Mega` (part 0)	9,905	8,876	503	2	
`13122018-mega104`	1,460,829	1,313,997	91,957	14	
`13122018-mega104` (part 1)	62,040	55,772	3,603	2	
`mega104-14-12-18`	14,597	9,657	9,125	2	
`mega104-17-12-18`	58,930	37,661	37,661	2	
`KTH-RTU1`	6,234,474	3,117,251	2,088,540	6	
`KTH-RTU1` (part 1)	184	96	59	2	
`KTH-RTU1` (part 2)	168	87	55	2	
`KTH-RTU4`	3,306,086	1,653,046	1,107,537	2	
`RICS`	1,550,304	775,152	519,352	2	

6 Experiments

We evaluate our learning and detection methods on a set of flow records of the IEC 104 traffic. In the first part of the evaluation, we focus on learning (discussed in Sec. 4). The second part is then describes anomaly detection based on the learnt automata models (discussed in Sec. 5).

6.1 Learning the Model using IEC 104 Flows

We have implemented the algorithm Alergia presented in Sec. 4 and used it with the values of the parameters α and t_0 set mostly according to our empirical experience (for more details, how to set parameter values, see [30]):

- The parameter α is set to 0.05 which gives a good balance between the merging (the strength of generalization and compactness) and classification error.
- The threshold parameter t_0 is set as $t_0 = \lfloor \log_2 |S| \rfloor$. The logarithmic function was chosen to obtain a small increase with the growing number of samples.

We evaluate the algorithm on the real IEC 104 traffic[4]. The characteristics of the benchmarks (name, the number of flows, i-messages, conversations, and communicating devices) are summarised in Tab. 1. The benchmarks contain from 31 to millions of conversations. The number of devices occurring in the traffic varies between 2 and 14. The benchmarks containing more than two devices are partitioned by a conversation pair and one of the partitions is selected (the parts are annotated with the partition number, e.g., 0, 1, 2). We also include the full unpartitioned version into this experiment even though the actual anomaly detection uses partitioned data only.

[4] All tested IEC 104 flows are available in CSV format at `https://github.com/matousp/datasets/tree/master/scada-iec104` [Sept 2020]. Datasets KTH-RTU1, KTH-RTU4, and RICS were provided by the RTSLab in Linköping [19].

Table 2: Results of the Alergia and the (reduced) Prefix tree learning.

Benchmark	**Alergia**			**Prefix tree**		
	Est. parameters	*States*	*Accuracy*	*States*	*Accuracy*	*Reduced states*
`iec104`	$\alpha = 0.05$, $t_0 = 3$	44	0% (0/21)	44	0% (0/21)	22
`10122018-104Mega`	$\alpha = 0.05$, $t_0 = 11$	8	100% (4642/4642)	49	99.8% (4636/4642)	35
`10122018-104Mega` (part 0)	$\alpha = 0.05$, $t_0 = 7$	8	99.7% (337/338)	48	99.7% (337/338)	35
`13122018-mega104`	$\alpha = 0.05$, $t_0 = 14$	8	99.9% (61606/61612)	38	99.9% (61606/61612)	28
`13122018-mega104` (part 1)	$\alpha = 0.05$, $t_0 = 10$	8	99.9% (2414/2415)	28	99.8% (2412/2415)	27
`mega104-14-12-18`	$\alpha = 0.05$, $t_0 = 11$	8	100% (6114/6114)	39	100% (6114/6114)	34
`mega104-17-12-18`	$\alpha = 0.05$, $t_0 = 13$	3	100% (25233/25233)	3	100% (25233/25233)	2
`KTH-RTU1`	$\alpha = 0.05$, $t_0 = 19$	12	100% (2088540/2088540)	12	100% (2088540/2088540)	–
`KTH-RTU1` (part 1)	$\alpha = 0.05$, $t_0 = 4$	9	98.3% (58/59)	9	98.3% (58/59)	–
`KTH-RTU1` (part 2)	$\alpha = 0.05$, $t_0 = 4$	9	100% (55/55)	9	100% (55/55)	–
`KTH-RTU4`	$\alpha = 0.05$, $t_0 = 19$	10	100% (1107537/1107537)	10	100% (1107537/1107537)	–
`RICS`	$\alpha = 0.05$, $t_0 = 17$	2	100% (519352/519352)	2	100% (519352/519352)	–

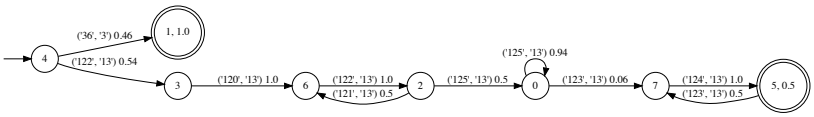


Fig. 4: A probabilistic automaton learnt using Alergia algorithm applied on benchmark `13122018-mega104` (part 1). The transitions are labeled with pairs (AsduType, CoT).

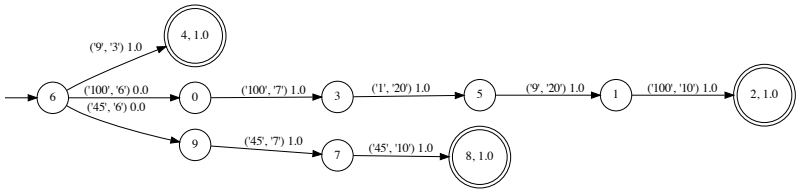


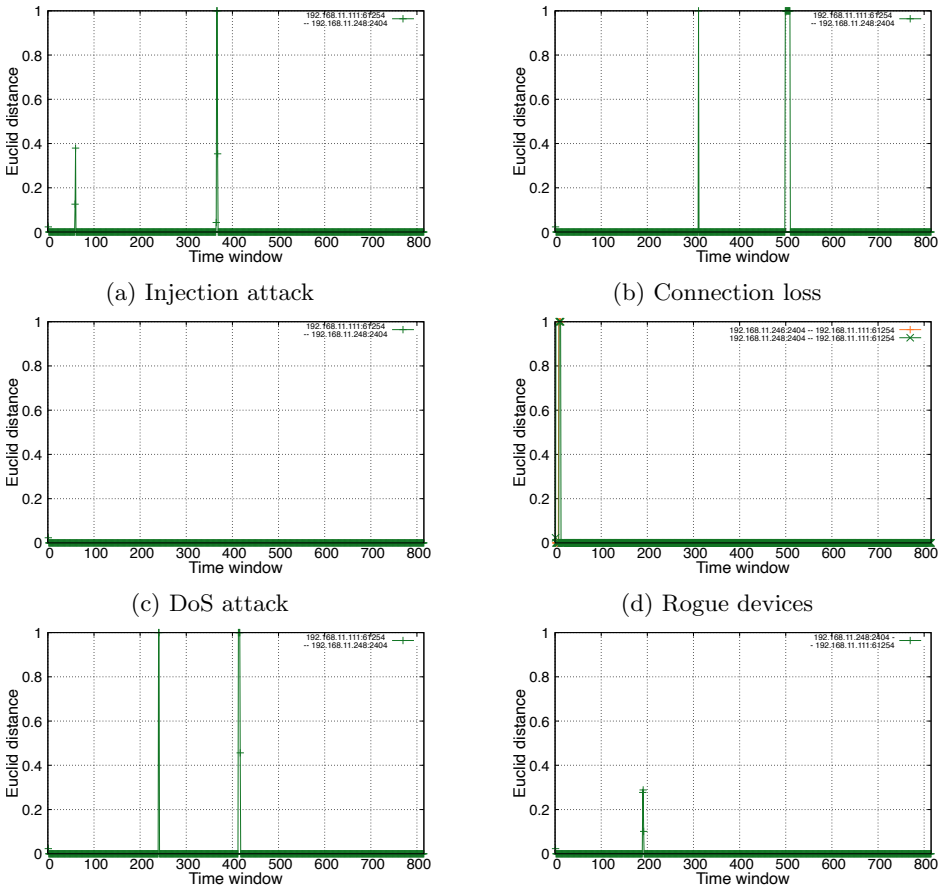
Fig. 5: A prefix tree learnt using Alergia algorithm applied on `KTH-RTU4`. Note that the probabilities contains rounded values, therefore the probability denoted as 0.0 means a very small value (e.g., $1.8 \cdot 10^{-6}$ for transitions from 6 to 0 and 6 to 9).

We applied the learning algorithms Alergia and Prefix tree (also with reduction) on each benchmark dataset. One third of each dataset was used for learning, the other two thirds were used for testing, i.e., evaluating the accuracy of the learnt model. The accuracy was computed as the ratio of the accepted conversations (with non-zero probability) to all conversations in the testing data. The results are shown in Tab. 2. Examples of a DPA learnt by Alergia and a PTA are shown in Fig. 4 and Fig. 5 respectively.

Discussion Tab. 2 shows a high accuracy of both Alergia and Prefix tree (about 99%) in all cases except `iec104`. The case of `iec104` illustrates a scenario with an insufficient learning data (the learning sample contains only one third of the 115 messages and 31 conversations, which does not cover the complexity of the communication enough). The learnt model then has a very little chance to recognise the testing communication. Notice also that Alergia was not able to generalize (it returned an automaton of the same size as Prefix tree).

In some cases (namely `13122018-mega104` and `10122018-104Mega`), a usage of Alergia leads to a slightly smaller number of false positives (i.e., messages that were wrongly classified as anomalies). In particular 100% (Alergia) vs. 99.8% (Prefix tree) in the case of `10122018-104Mega`. It is caused by the fact that Alergia uses merging of the prefix tree to generalise the sample and derive general regularities. This way it can recognise even valid conversations which do not precisely appear in the learning sample. In this particular case, Alergia learnt that the file transfer may contain any number of data segments

(messages with AsduType=125 and CoT=13, see Fig. 4), and thus classify as normal also conversations which contain different numbers of data segments not seen in the learning sample. Prefix tree, however, classifies as anomalies everything that does not appear in the learning sample, as it skips the generalization phase. The number of false positives generated by the prefix tree is, nevertheless, small (below 2%). This can be explained by the fact that we are dealing with a highly regular and relatively simple traffic which is almost entirely covered by the learning sample. It is also worth noticing that the reduction procedure for PTAs can significantly reduce the number of states. Since the reduction preserves the distribution, accuracy is the same as for unreduced version.



(a) Injection attack

(b) Connection loss

(c) DoS attack

(d) Rogue devices

(e) Scanning attack

(f) Switching attack

Fig. 6: Detection of the anomaly scenarios using $Distr_{aler}$. Each time window represents a five-minute snapshot of the traffic.

Alergia creates more compact automata then Prefix tree, again thanks to the merging in the generalization and compaction phase. The number of states created by Prefix tree is, however, still small, despite the large size of the learning set, thanks to relative simplicity of the communication. In a couple of benchmarks, in particular `KTH-RTU*`, the Prefix tree has the same number of states as the automaton obtained by Alergia. This is caused by a nature of the benchmarks containing not enough various traffic to apply the state merging.

The advantage of Prefix tree over Alergia is its simplicity and transparency. In a our scenario (simple highly regular communication and large learning sets), it is a viable option.

6.2 Experiments with Anomaly Detection

In this part, we focus on evaluation of our anomaly detection mechanisms (mentioned in Sec 5). In the experiment we use IEC 104 dataset `mega104-17-12-18` created at Brno University of Technology[4]. The benchmark consists of 58,930 messages of IEC 104 communication that were captured within 3 days of a real network traffic. We experimented with six types of anomalies discussed below in detail. Each type of anomaly was simulated by injecting into or removing communication from our traffic sample while keeping the original features of IEC 104 sessions. The DPA model of the normal traffic was trained on the original traffic. The results of the anomaly detection when using Alergia were indistinguishable from results when using Prefix tree for learning, therefore we give only one common summary of the results. Our anomaly detection was used to analyze input data within five minutes-long windows. The outputs are visualised in Fig. 6.

Injection attack In this scenario (see Fig. 6 a), an attacker compromised a host on the ICS network and started sending unusual requests. First, the attacker sent activation messages with AsduType=45 and CoT=6, which requested the execution of the command on the target host. The host correctly confirmed with CoT=7. The first attack took 5 minutes and included 83 packets. During the second injection attack the attacker tried to transfer a file from the target to the compromised host. The attacker sent messages with $\text{AsduType} \in \{122, 120, 121, 124, 125\}$ which represented a file transfer. The attack included 221 messages and took 15 minutes.

Connection loss This scenario (see Fig. 6 b) represents a short blackout of a device when connection was lost. The first connection failure took 10 and 146 messages were lost. The second failure lasted for about one hour and 921 messages were lost.

DoS attack This denial of service (DoS) attack (see Fig. 6 c) was directed against a control station. The attacker sent hundreds of legitimate packets to the destination. He used a spoofed IP address, which was sending spontaneous messages with AsduType=36 and CoT=3. The attack lasted for half an hour and contained about 1049 spoofed messages. As seen in Fig. 6 c), the attack was not

detected. It is because the DoS attack scenario contained additional conversations of the same type A that was present in the training dataset. The time windows of the valid communication corresponding to the windows where the attack occurs, also contained many conversations of the type A so that the constructed probabilistic automata could not capture the change. To make it clear, consider for instance a time window containing 10 messages of the type A and another time window containing 1000 messages of the type A. Then probabilistic automata obtained by Prefix tree corresponding to these windows are equal (the same is true for the Alergia algorithm). However, this limit of probabilistic automata approach can be removed by a combination of the detection procedure with a simple statistical analysis.

Rogue devices A rogue devices was connected to the ICS network and started communicating with an IEC 104 host using legitimate IEC 104 packets. The attacker used a sequence of spontaneous messages with AsduType=36 and CoT=3. The station correctly responded with supervisory APDUs. The attack lasted about 30 min. and included 417 packets, see Fig. 6 d).

Scanning attack This scenario includes the horizontal scanning (enumerating IP addresses of the network segment) and the vertical scanning (IOA addresses on the selected host), see Fig. 6 e). First, the attacker sent IEC 104 Test Frame messages on port 2404 (used by IEC 104) and observed responses. If a station responded, the attacker started the vertical scanning of the host using General Interrogation ASDUs sent to IOA addresses 1 to 127. Each attack took about 15–20 minutes.

Switching attack The switching attack implemented the similar scenario as used in the attack against Ukrainian power plant using CrashOverride malware [31]. During this attack a series of IEC 104 packets with AsduType=46 and a sequence of CoT numbers $(6, 7, 10)$ were sent to the target that caused switching the device on and off, see Fig. 6 f). The attack lasted for 10 minutes and transferred 72 packets.

Results We evaluated our detection methods described in Sec. 5 using above scenarios. For the detection via single conversation reasoning we set threshold $\mu = 0$ and for the case of the detection via distribution comparison we set $\theta = 0.25$.

The length of a time window was 5 minutes. The results comparing the proposed methods are shown in Tab. 3. We have compared the detection via single conversation reasoning (*Single*), detection via distribution comparison based on learning DPAs using Alergia ($Distr_{aler}$), and detection via distribution comparison based on learning DPAs using the Prefix tree ($Distr_{pref}$). The detection results for $Distr_{aler}$ of the considered scenarios are shown in Fig. 6. The graphs show Euclid distance of the valid traffic and the traffic under inspection for each time window (see Eq. 3, Sec. 5.2).

Table 3: Comparison of the detection methods

Anomaly	*Single*	$Distr_{pref}$	$Distr_{aler}$
Communication loss	✗	✓	✓
Switching attack	✓	✓	✓
Scanning attack	✓	✓	✓
DoS attack	✗	✗	✗
Rogue devices	✓	✓	✓
Injection attack	✓	✓	✓

Discussion From Tab. 3 we can see that the $Distr_{aler}$ and $Distr_{pref}$ detection methods are equally successful in all cases except the DoS attack scenario as discussed above.

The *Single* detection method does not find anomalies in DoS attack and the Communication loss scenario. In case of communication loss, *Single* is not able to detect an anomaly because it only analyses existing individual conversations (unlike the distribution comparison method).

From graphs in Fig. 6 we can see that in the case of $Distr_{aler}$, we are able to detect all anomalies, including multiple occurrences within the scenario (except the discussed DoS attack scenario) with no false positives. The same is true also for $Distr_{pref}$ (the graphs look the same, so we do not present them here). For the case of the *Single* detection approach, the situation is also encouraging. This detection approach is able to detect all anomalies including their multiple occurrences. Our detection methods do not report any false positives (no other windows in the traffic are evaluated as anomalous). They give alerts exactly on the ongoing anomalies, except the two missed anomalies discussed above.

7 Conclusion

We have introduced a new technique for efficient modelling of ICS/SCADA communication using probabilistic automata. Since the ICS communication is stable and regular, the automata capture the normal communication rather precisely using small number of states and edges. The automata are automatically generated from samples of ICS communication obtained from ICS flow records. The automata model the semantics of ICS communication exchanged between two ICS devices. The semantics is extracted from the protocol headers based on the expert knowledge. We showed that for IEC 104 communication, it is enough to consider only AsduType and Cause of Transmission (CoT) extracted from *i*-messages. We also make experiments with other ICS protocols (Goose, MMS, DLMS). Recommended header values of these protocols are listed in [10].

We experimented with two modes of anomaly detection. In *Single* mode, a single conversation could be marked as anomalous if it was not recognised by the learnt automaton. In *Distribution* mode, probabilistic distributions of entire five minutes long windows were compared against the distribution of the learnt

normal traffic. We demonstrated that these detection methods were able to detect common classes of cyber attacks on ICS/SCADA systems, i.e., the switching attack, command injection, connection of a rogue device, or the scanning. The automata were not suitable for detecting denial of service attacks if they used communication sequences that were present in the training dataset. However, a DoS could be easily detected by statistical methods.

Our choice of probabilistic automata as a modeling mechanism for the network traffic is based on the idea that DPAs can be efficiently learnt from positive examples and that besides the regular structure of the communication, they capture also its probability distribution (which proved beneficial for instance for the detection of connection loss).

In the future, we would like to apply this technique on other types of SCADA protocols, e.g., Modbus or Goose, that are built on the publish–subscribe model rather than the client–server data exchange as in case of IEC 104. Additionally, we plan to enhance our method with a statistical reasoning that can detect attacks like denial-of-service, and to investigate possible merits and feasibility of modeling time of the communication.

References

1. Stouffer, K., Pillitteri, V., Abrams, M., Hahn, A.: Guide to Industrial Control Systems (ICS) Security. Technical Report NIST-SP-800-82r2, National Institute of Standards and Technology (2015)
2. Committee, S.G.C.: Guidelines for Smart Grid Cybersecurity. Technical Report NISTIR-7628r1, National Institute of Standards and Technology (2014)
3. Assante, M.J., Lee, R.M.: The Industrial Control System Cyber Kill Chain. Technical report, SANS Institute (October 2015)
4. Miller, B., Rowe, D.C.: A survey of SCADA and critical infrastructure incidents. In: In Proceedings of the 1st Annual conference on Research in information technology, RIIT '12, ACM (2012) 51–56
5. Assante, M.J., Lee, R.M., Conway, T.: Modular ICS Malware. Technical report, Electricity Information Sharing and Analysis Center (E-ISAC) (August 2017)
6. O'Leary, J., Kimble, J., Vanderlee, K., Fraser, N.: Insights into Iranian Cyber Espionage: APT33 Targets Aerospace and Energy Sectors and has Ties to Destructive Malware (2017)
7. McCarthy, J., Powell, M., Stouffer, K., Tang, C., Zimmerman, T., Barker, W., Ogunyale, T., Wynne, D., Wiltberger, J.: Securing Manufacturing Industrial Control Systems: Behavior Anomaly Detection. Technical Report NISTIR-8219, National Institute of Standards and Technology (2018)
8. ENISA: Communication network dependencies for ICS/SCADA Systems. Technical report, European Union Agency for Network and Information Security (ENISA) (December 2016)
9. Matoušek, P., Ryšavý, O., Grégr, M.: Increasing Visibility of IEC 104 Communication in the Smart Grid. In: The 6th International Symposium for ICS & SCADA Cyber Security Research 2019, BCS Learning and Development Ltd (2019) 21–30
10. Matoušek, P., Ryšavý, O., Grégr, M., Havlena, V.: Flow based monitoring of ICS communication in the smart grid. Journal of Information Security and Applications **54** (2020) 102535

11. Wagner, C., François, J., State, R., Engel, T.: Machine learning approach for ip-flow record anomaly detection. In Domingo-Pascual, J., Manzoni, P., Palazzo, S., Pont, A., Scoglio, C., eds.: NETWORKING 2011, Berlin, Heidelberg, Springer Berlin Heidelberg (2011) 28–39
12. Hofstede, R., Bartoš, V., Sperotto, A., Pras, A.: Towards real-time intrusion detection for NetFlow and IPFIX. In: Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013). (Oct 2013) 227–234
13. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, New York, NY, USA (2010)
14. Barbosa, R.R.R.: Anomaly detection in SCADA systems: a network based approach. PhD thesis, University of Twente (4 2014)
15. Barbosa, R.R.R., Sadre, R., Pras, A.: Towards periodicity based anomaly detection in SCADA networks. In: Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012). (Sept 2012) 1–4
16. Rakas, S.V.B., Stojanović, M.D., Marković-Petrović, J.D.: A review of research work on network-based scada intrusion detection systems. IEEE Access **8** (2020) 93083–93108
17. Eder-Neuhauser, P., Zseby, T., Fabini, J., Vormayr, G.: Cyber attack models for smart grid environments. Sustainable Energy, Grids and Networks **12** (2017) 10 – 29
18. Matoušek, P., Havlena, V., Holík, L.: Efficient modelling of ics communication for anomaly detection using probabilistic automata. In: Proceedings of IFIP/IEEE International Symposium on Integrated Network Management. (january 2021) 1–9
19. Lin, C.Y., Nadjm-Tehrani, S.: Understanding IEC-60870-5-104 Traffic Patterns in SCADA Networks. In: Proceedings of the 4th ACM Workshop on Cyber-Physical System Security. CPSS '18, New York, NY, USA, ACM (2018) 51–60
20. Lin, C.Y., Nadjm-Tehrani, S., Asplund, M.: Timing-based anomaly detection in SCADA networks. In: International Conference on Critical Information Infrastructures Security, Springer (2017) 48–59
21. Martinelli, F., Mercaldo, F., Santone, A.: Real-Time SCADA Attack Detection by Means of Formal Methods. In: 2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE). (June 2019) 231–236
22. Goldenberg, N., Wool, A.: Accurate modeling of modbus/tcp for intrusion detection in scada systems. International Journal of Critical Infrastructure Protection **6**(2) (2013) 63 – 75
23. Caselli, M., Zambon, E., Kargl, F.: Sequence-aware Intrusion Detection in Industrial Control Systems. In: Proceedings of the 1st ACM Workshop on Cyber-Physical System Security. CPSS '15, New York, NY, USA, ACM (2015) 13–24
24. Caselli, M., Zambon, E., Petit, J., Kargl, F.: Modeling message sequences for intrusion detection in industrial control systems. In Rice, M., Shenoi, S., eds.: Critical Infrastructure Protection IX, Cham, Springer International Publishing (2015) 49–71
25. Claise, B., Trammel, B., Aitken, P.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011, IETF (September 2013)
26. IEC: Telecontrol equipment and systems - Part 5-104: Transmission protocols - Network access for IEC 60870-5-101 using standard transport profiles. Standard IEC 60870-5-104:2006, International Electrotechnical Commission, Geneva (June 2006)

27. Matoušek, P.: Description and analysis of IEC 104 Protocol. Technical Report FIT-TR-2017-12, Brno University of Technology (2017)
28. Matoušek, P., Ryšavý, O., Grégr, M.: Security Monitoring of IoT Communication Using Flows. In: Proceedings of the 6th Conference on the Engineering of Computer Based Systems. ECBS '19, Association for Computing Machinery (2019) 1–9
29. Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., Carrasco, R.C.: Probabilistic finite-state machines-part i. IEEE Trans. Pattern Anal. Mach. Intell. **27**(7) (July 2005) 1013–1025
30. Havlena, V., Holík, L., Matoušek, P.: Learning Probabilistic Automata in the Context of IEC 104. Technical report, Brno University of Technology (2020)
31. Dragos: CrashOverride. Analysis of the Threat of Electric Grid Operations. Technical report, Dragos Inc. (June 2017)