Communication Pattern Monitoring: Improving the Utility of Anomaly Detection for Industrial Control Systems

Man-Ki Yoon* and Gabriela F. Ciocarlie†
*Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801-2302
†Computer Science Laboratory, SRI International, Menlo Park, CA 94025-3493
Email: mkyoon@illinois.edu, gabriela.ciocarlie@sri.com

Abstract—**Attacks on Industrial Control Systems (ICS) continue to grow in number and complexity, and well-crafted cyber attacks are aimed at both commodity and ICS-specific contexts. It has become imperative to create efficient ICS-specific defense mechanisms that complement traditional enterprise solutions. Most commercial solutions are not designed for ICS environments, rely only on pre-defined signatures and do not handle zero-day attacks. We propose a threat detection framework that aims to detect zero-day attacks by creating models of legitimate, rather than malicious ICS traffic. Our approach employs a content-based analysis that characterizes normal command and data sequences applied at the network level, while proposing mechanisms for achieving a low false positive rate. Our preliminary results show that we can reliably model normal behavior, while reducing the false positive rate, increasing confidence in the anomaly detection alerts.**

I. Introduction

It is well known that Industrial Control Systems (ICS) (some of which are also known as Supervisory Control and Data Acquisition or SCADA systems) supporting the critical infrastructure of our nation are vulnerable and have even become the targets of well-crafted cyber attacks aimed at both commodity and ICS-specific contexts (e.g. Stuxnet [7]).

With the emergence of such cyber threats, it becomes imperative to create efficient ICS-specific defense mechanisms that complement traditional enterprise security solutions. Most existing context-specific solutions use a traditional approach to network intrusion detection systems (IDS), which rely on a pre-defined set of signatures to identify potential ICS attacks [6]. However, recent research has underlined the limitations of such an approach when faced with zero-day attacks [13], [24], [27], which have become prevalent in ICS. One alternative, known as Anomaly Detection (AD), has shown promise in detecting zero-day intrusions by creating models of legitimate, rather than malicious, traffic for enterprise networks [14], [15], but has not been fully explored in the ICS context [21].

ICS systems use well established, yet insecure, communication protocols such as Modbus [5], DNP3 [2] and CIP [1], which can exhibit unauthorized requests or commands, malformed messages, *etc.* as a consequence of an attack. We propose to use the intrinsic characteristics of these protocols to determine the normal behavior of an ICS device. We aim to address this problem by creating and maintaining accurate normality models by identifying the relevant features encompassed by the models. We propose to employ a content-based analysis that characterizes normal command and data sequences applied to communication patterns while reducing the false positive rate.

A. Assumptions

Compared to enterprise systems, industrial control systems exhibit a more constrained behavior, which is *predictable* [28]. Current ICS systems have *fixed topology* and their specialized functionality results in regular communication patterns. Moreover, ICS communication protocols can be *simple* [5] and *not* very *diverse*; each industry sector uses a few standard or recognized communication protocols (*e.g.*, in the electric power systems, Modbus [5], IEC 60870-5-104 [3], IEC 61850 [4] and DNP3 [2] are prevalent). Consequently, we conjecture that attacks in such environments would exhibit different communication patterns than those observed during normal operation.

B. Contributions

The key challenge for monitoring and defending ICS systems is creating a robust method for modeling normal ICS device behavior. This paper proposes a novel method for modeling ICS device communication behavior by using sequences of requests and replies, which are generated by regular communication patterns, to determine the state of a device. Our implementation and experiments focus on the problem of creating adaptive models, leveraging the intrinsic characteristics of the environment where the models are created. The work described here provides several contributions by:

- proposing a new probabilistic-suffix-tree-based approach for ICS anomaly detection, which extracts the normal patterns of command and data sequences from ICS communications;

SENT '14, 23 February 2014, San Diego, CA, USA
Copyright 2014 Internet Society, ISBN 1-891562-36-3
http://dx.doi.org/10.14722/sent.2014.23012

- identifying the imperative need for a low false positive rate and proposing a mechanism that reduces it; and
- building a system to implement the algorithms, applying the system to both real and simulated datasets, and analyzing the performance of the proposed framework.

II. Background

A. System Model

We consider a control network that comprises of a set of *master* devices $\mathcal{M} = \{m_1, m_2, \ldots, m_{N_m}\}$ and a set of *slave* devices $\mathcal{S} = \{s_1, s_2, \ldots, s_{N_s}\}$. A master m_i controls, or observes the state of a slave s_j, by sending a *request* that is composed of a *command* and a *command-specific data*. A slave may send a *response* back to the master after the requested execution. Similar to a request, a response is also composed of a command and data that represents the result of the execution. Because of the indistinguishability in the formats, we denote by $r_{i,j} = (c, d)$ a request/response from device i to j with command c and the corresponding data d, and call it a $message$. The message format is protocol-dependent. We assume fixed-length commands and variable-length data. We specifically denote the entire set of commands by Σ_c. Lastly, the *connection* established between master m_i and slave s_j is uniquely represented by $\omega_{i,j}$.

The network may implement multiple protocols; a device may interact with others with different protocols. Consequently, the unique connection identifier $\omega_{i,j}$ defined above can be extended to $\omega_{i,j,p}$ where p represents a protocol. We perform anomaly detection at the connection level, in a multi-protocol environment. However, for notation and explanation simplicity, we consider a single-protocol network.

B. Problem Description

Given the control network described above, a protocol analyzer monitors the messages transferred along all the connections $\Omega = \{\omega_{i,j}\}$ and aims to detect anomalies. The t^{th} message of a connection $\omega_{i,j}$, denoted by $r_{i,j}^{(t)} = (c^{(t)}, d^{(t)})$, is considered anomalous if the probability of observing the message, given a history of previous messages, is below a threshold. That is, if

$$P\left(r_{i,j}^{(t)} \middle| r_{i,j}^{(1)}, \ldots, r_{i,j}^{(t-2)}, r_{i,j}^{(t-1)}\right) < \theta, \tag{1}$$

then $r_{i,j}^{(t)}$ is deemed anomalous. In this paper, we assume that the messages in different connections are independent. That is, the message $r_{i,j}$ does not affect the command and/or data in $r_{i,k}$ or $r_{k,i}$. Depending on the network configuration and also on the protocol used, however, this assumption may not hold for some systems. Nevertheless, we claim that ICS environments use simple protocols with regular communication patterns, where our assumptions apply (*e.g.,* a message sent by a Modbus server to a slave will not affect the command and/or data sent to a different slave).

III. Content-based Anomaly Detection for ICS Devices

In this section, we present a method for calculating the probability in (1). We model message sequences as a Dynamic Bayesian Network (DBN) [20], [23] and use Probabilistic Suffix Tree (PST) [25] as the underlying predictive model. In what follows, for the simplicity of explanation, we limit ourselves to the analysis of one connection $\omega_{i,j}$ and thus we omit the subscript i, j.

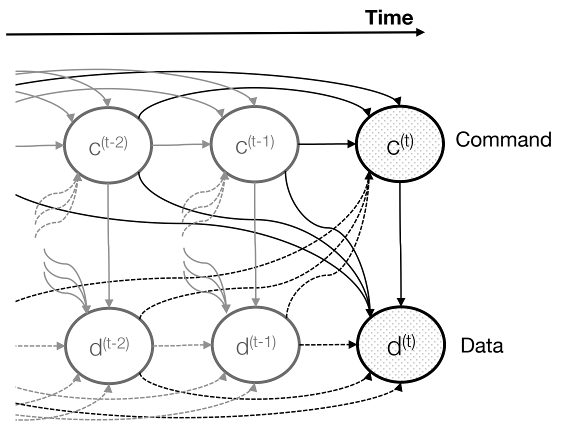


Fig. 1. Dynamic Bayesian network structure.

A. Dynamic Bayesian Network of Command and Data Sequence

As explained in the previous section, a message is composed of a command and a command-specific data. Data can vary in terms of the content and length, but the space that it can be generated from is limited (mainly due to protocol specification), once the command is given. Thus, the *unconditional* probability of message $r^{(t)} = (c^{(t)}, d^{(t)})$ can be modeled as

$$P\big(r^{(t)}\big) = P\big(c^{(t)}\big) P\big(d^{(t)} | c^{(t)}\big). \tag{2}$$

However, we should consider message history, since otherwise (2) would be proportional to the *frequency* of command (or data). For example, an otherwise normal message with rarely-observed command would look abnormal because of small $P(c^{(t)})$. Moreover, a particular task for a device can be carried out by a chain of messages. For instance, a master device may retrieve the state of a slave, and then, depending on it, the master may issue appropriate command(s). Hence, the message history helps detect *out-of-context* messages, and thus we calculate the following conditional probability as shown in the dynamic Bayesian network (Figure 1):

$$\begin{aligned} P\Big(r^{(t)} \Big| r^{(1)}, \ldots, r^{(t-2)}, r^{(t-1)}\Big) = \\ P\Big(c^{(t)} \Big| r^{(1)}, \ldots, r^{(t-2)}, r^{(t-1)}\Big) \times \quad (3) \\ P\Big(d^{(t)} \Big| c^{(t)}; r^{(1)}, \ldots, r^{(t-2)}, r^{(t-1)}\Big). \quad (4) \end{aligned}$$

The problem now reduces to finding the conditional probability distributions of (3) and (4). Note that each is conditioned on the previous *messages*. Although this is more general, as described above, we reduce the conditions to command sequence and data sequence (plus the current command),

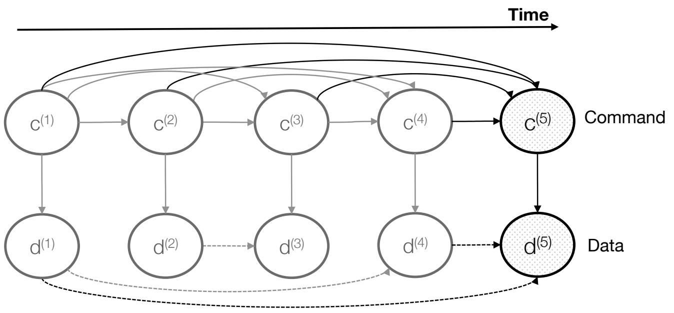


Fig. 2. DBN for an example sequence of length 5.

respectively, as follows:

$$P\Big(r^{(t)}\Big|r^{(1)},\ldots,r^{(t-2)},r^{(t-1)}\Big) =$$
$$P\big(c^{(t)}\big|c^{(1)},\ldots,c^{(t-2)},c^{(t-1)}\big)\times \quad (5)$$
$$P\big(d^{(t)}\big|c^{(t)};d^{(1)},\ldots,d^{(t-2)},d^{(t-1)}\big). \quad (6)$$

(5) assumes that previous data does not affect the current command. Again, this may or may not hold depending on the task that is carried out by the command. However, in general, the command flow is not easily influenced by the data associated with each command. Such a deviation rarely happens when, for example, an exception or a fault must be handled (which can also be considered an anomaly). Similarly, (6) considers the current data as independent from the previous commands. In fact, it rather depends on the previous data that was sent along with the same command type as the current command. Hence, the conditional probability of a message can then be represented as follows:

$$P\big(r^{(t)}\Big|r^{(1)},\ldots,r^{(t-2)},r^{(t-1)}\big)$$
$$= P\big(c^{(t)}\big|c^{(1)},\ldots,c^{(t-2)},c^{(t-1)}\big)P\big(d^{(t)}\big|c^{(t)};D(c^{(t)})\big), \quad (7)$$

where $D(c^{(t)}) = \{d^{(s)}|c^{(s)} = c^{(t)}, 1 \leq s \leq t-1\}$, *i.e.*, the data that has been sent along with the same command type as $c^{(t)}$. Figure 2 shows the DBN for an example sequence of length 5, where $c^{(1)} = c^{(4)} = c^{(5)}$ and $c^{(2)} = c^{(3)}$. Note that there is no edge from $c^{(x)}$ to $d^{(y)}$ such that $x < y$, from $d^{(\cdot)}$ to any $c^{(\cdot)}$, and from $d^{(x)}$ to $d^{(y)}$ such that $c^{(x)} \neq c^{(y)}$. Once we observe the fifth message, its probability is calculated by $P\big(c^{(5)}\big|c^{(1)},\ldots,c^{(4)}\big)P\big(d^{(5)}\big|c^{(5)};d^{(1)},d^{(4)}\big)$.

Although the dimensions of the conditional probability distributions (3) and (4) are reduced by the independence assumption, calculation of (7) as $t \to \infty$ is still computationally unfeasible, because all the possible joint assignments would hardly be seen. Hence, we need a way to approximate it.

B. Pattern of Command (and Data) Learning Using Probabilistic Suffix Tree

As described in Section I-A, ICS networks behave in a fairly deterministic way: devices repeat certain sequences of tasks unless otherwise interrupted. If we look at each connection level, a set of periodic operations forms a *pattern* of command sequence that hardly varies in a normal condition.[1] The degree of data variation can be higher; however, data usually also exhibits a pattern in the content or in the structure. Due to this presence of patterns in message sequences, we can further reduce the dimensionality in (7) considering the consistency of a given command (or data) with the recent history of commands (or data) as a part of the underlying pattern. In what follows, we present a method to learn the hidden pattern from a sequence of commands (or data). Note that the same learning method presented here is applied to both command sequences and data sequences.[2] Thus, for the rest of this section we use 'element' to refer to command or data.

Now, suppose we are given the following sequence of elements:

$$\sigma_1, \sigma_2, \sigma_1, \sigma_2, \sigma_3, \sigma_3, \sigma_1, \sigma_2, \sigma_1, \sigma_2, \sigma_3, \sigma_3$$

One may model the sequence as a first-order Markov chain, *i.e.*, $P(\sigma^{(t)}|\sigma^{(1)},\ldots,\sigma^{(t-1)}) = P(\sigma^{(t)}|\sigma^{(t-1)})$. The first-order Markov chain model can detect certain abnormal subsequences; for instance, 'σ_1, σ_3' or 'σ_2, σ_2', *etc.*, since those have never appeared in the sequence. However, it does not fit well for modeling the *normal* subsequences because, for example, $P(\sigma_1|\sigma_2) = P(\sigma_3|\sigma_2) = 0.5$ although 'σ_2, σ_1' and 'σ_2, σ_3' are still legitimate subsequences. However, a Markov chain of order 3 (or more) learns the above normal subsequences without any *ambiguity*. If a sequence of elements is generated by an underlying pattern and exhibits no noise, there always exists a minimum order, m, for a Markov chain allows us to *predict* the probability of an element by just looking at the m most recent elements.

However, finding such an m^{th}-order Markov chain is challenging since m is unknown especially during an online-learning phase. Even if it is known *a priori*, the construction of the conditional probability distribution remains computationally expensive if m is large, due to the fixed-order model. In fact, the minimum required length of the recent history of elements can vary. For example, knowing that the most recent one was σ_1, we can always expect to see σ_2 next. Similarly, 'σ_2, σ_3' is always followed by σ_3. This *variable-order* Markov chain not only can reduce the joint assignments of elements that need to be learned, but also enables us to learn the underlying pattern without any prior knowledge.

The challenge is to build such a model even in the presences of noise, *i.e.*, legitimate variations from the base pattern due to missing, out-of-order messages and/or sporadic tasks. To address this challenge, we use the *Probabilistic Suffix Tree* (PST, or Prediction Suffix Tree) [25], which uses a variable-order Markov model representation. Intuitively speaking, a PST learns a set of subsequences of different lengths, *e.g.*, 'σ_1', 'σ_2, σ_3', each of which can be a significant indicative of the next element. This enables us to efficiently calculate the probability of the 'next' element without having to look back all or a pre-defined length of the history. That is,

$$P\big(\sigma^{(t)}|\sigma^{(1)},\ldots,\sigma^{(t-1)}\big) \sim P\big(\sigma^{(t)}|\sigma^{(t-k)},\cdots,\sigma^{(t-1)}\big)$$

[1]In certain protocols such as Modbus [5], a slave responses back with the command originally specified by the master. Thus, the sequence of commands in the slave's messages is identical to the one issued by the master unless an exception occurred.

[2]We try to find the pattern of the command sequences for each connection and the pattern of the data sequence for each *command type* in each connection.

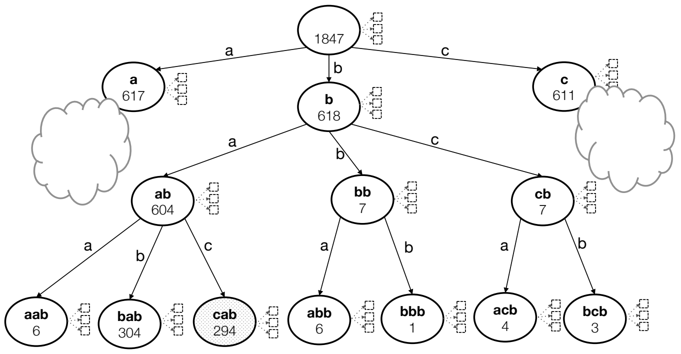


Fig. 3. A probability suffix tree for a sequence of length 1847 generated with the base pattern of ababcc. The maximum depth is set to 3.

for some k that varies depending on $\sigma^{(t-1)}$, $\sigma^{(t-2)}$, and so on. To explain how a PST is built and used, let us consider an example shown in Figure 3. The PST is learned from a sequence of length 1847 that is generated with the base pattern of ababcc where $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ is the set of all possible elements.[3] Each node is associated with string $s \in \Sigma^*$, e.g., cb, cab, and the value associated with each node represents the number of occurrences of $s\sigma$, $\forall \sigma \in \Sigma$, in the whole sequence. We observe that the frequencies of the substrings from the base pattern, ababcc, are dominantly higher than the ones caused by noise.

Now, suppose we have observed [$\cdots$bccab] and a is given as the new observation. To calculate $P(\mathsf{a}|\cdots\mathsf{bccab})$, we traverse the tree from the root by reading backward the elements of the history until we cannot go further down. Each node (including inner nodes) models the conditional probability distribution $P(\sigma|s)$ for $\sigma \in \Sigma$, where s is the string associated with the node. Figure 4 shows the node with $s = \mathsf{cab}$, where the traverse for [$\cdots$bccab] would stop. In each node, the frequency of $s\sigma$ for different σ is counted. For instance, caba appeared 261 times in the entire sequence. The probability of an element is then calculated by

$$P(\sigma|history) \approx P(\sigma|s) = \frac{n(s\sigma)}{\sum_{\sigma' \in \Sigma} n(s\sigma')}, \tag{8}$$

where $n(x)$ is the frequency of the substring x in the whole sequence. Thus,

$$P(\mathsf{a}|\cdots\mathsf{bccab}) \approx P(\mathsf{a}|\mathsf{cab}) = \frac{261}{294} \approx 0.8877.$$

The probability distribution would change if the traverse stopped at the node with $s = \mathsf{ab}$ instead of cab. A traverse could go down to the maximum depth D that is given as a parameter ($D = 3$ for the PST in Figure 3). However, a pruning technique is used to cut some branches for a better approximation of the conditional probability [25]. To explain this, let us assume that we have seen [$\cdots$ccaab] followed by b. The traverse in the PST in Figure 3 would stop at the node aab. $P(\mathsf{b}|\mathsf{aab})$ is an inaccurate approximation of $P(\mathsf{b}|\cdots\mathsf{ccaab})$ because aab is not *significant*; it can be present due to noise. In this case, calculation of $P(\mathsf{b}|\mathsf{ab})$, which is probabilistically more significant, is advisable. The base pattern, however, is

[3]The sequence is obtained from 1847 Modbus commands communicated over a connection between a master and a slave deployed in a testbed.

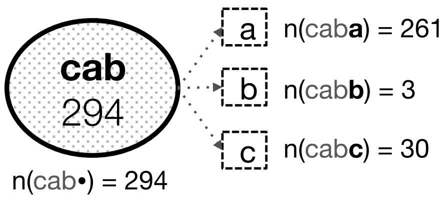


Fig. 4. Example of a PST node with $s = \mathsf{cab}$.

unknown; thus, we examine the significance of a node by calculating $P(s)$ as follows:

$$P(s) = \frac{\sum_{\sigma' \in \Sigma} n(s\sigma')}{L - l(s)}, \tag{9}$$

where L is the length of the whole sequence and $l(s)$ is the length of string s. The denominator represents the number of all possible substrings of length equal to $l(s\sigma')$ for any $\sigma' \in \Sigma$ in the whole sequence. For example, $P(\mathsf{aab}) = 6/(1847-3) \approx 0.0033$. When we traverse the PST, the branches to such nodes that have low $P(s)$ compared to a threshold are pruned, so that the conditional probability can be obtained with a high significance.[4]

C. Incremental Learning of PST

The original PST is suited for an offline-learning; a PST is learned from a long sequence of elements (command or data in our case), given in advance, and by counting the frequencies of substrings of different lengths (up to D). Such a batch learning is not applicable to network-level anomaly detection, given that timely action must be taken in case of an anomaly.

Consequently, we implement an *incremental* PST learning, which starts from an empty tree and, as a new element is being observed, updates it using recently-read elements. Since the tree is bounded by the maximum depth parameter D, we only need to keep the D most recent elements, $\sigma^{(t-D)}, \sigma^{(t-D+1)}, \cdots, \sigma^{(t-1)}$. When a new element $\sigma^{(t)}$ is observed, we traverse down the tree from the root node until reaching a leaf node (create any nodes when needed) and update the counter (for the $\sigma^{(t)}$) at each node. In terms of computational efficiency, this takes only $\mathcal{O}(D)$.[5] Note that no pruning is applied: since the PST is incrementally built, some nodes that are initially not significant can later turn into significant ones. Thus, pruning is used only when calculating the probabilities as described in the previous subsection.

The analyzer attached to the network under protection builds one PST for the commands sent along each connection $\omega_{i,j}$ and up to Σ PSTs for the data (i.e., one for each command type) on the connection, which can total a maximum of $|\Omega|(1 + \Sigma)$ PSTs. Each PST is associated with a buffer of commands and data, respectively, whose length is D_c and D_d, respectively.

[4]Pruning can be also used to reduce the memory required to store the tree. This can be done by comparing the distributions $P(\sigma|s)$ and $P(\sigma|\sigma's)$ for some σ'. If $P(\sigma|s) \sim P(\sigma|\sigma's)$ for every $\sigma \in \Sigma$, then the branch from s to $\sigma's$ is pruned.

[5]This is based on the assumption that searching the counter for $\sigma^{(t)}$ at each node takes $\mathcal{O}(1)$. If a linear search is used, the time complexity can be $\mathcal{O}(D\Sigma)$.

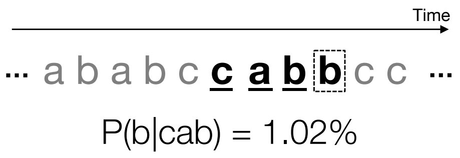


Fig. 5. Example of a false positive due to a missing element.

D. False Positive Reduction using Missing Element Inference

So far we have seen how to approximate the probability of a command (or a data) given each history sequence. The calculation of message probability is then straightforward: the analyzer finds the two PSTs, the command and the data PST, then traverses each tree using the associated buffers and the newly observed message, each of which will find the approximation of each term in (7).

Due to the critical nature of ICS systems, achievement of a very low false positive rate is imperative; otherwise, such systems' availability can degrade in the presence of remediation. This is a very challenging problem. However, considering that the messages communicated over a connection follow a pattern, certain sources of false positives can be reduced. Among them, in this paper, we reduce the false positives due to a *missing element* (*e.g.*, a missing packet). To illustrate our approach, let us consider a sequence of commands, shown in Figure 5, which is a part of the sequence generated from the base pattern of ababcc as in Figure 3. If we calculate the probability of the last b (in the box), according to Figure 4, $P(\mathsf{b}|\mathsf{cab}) \approx 1.02\%$, which is very low since such a sequence has rarely been seen. By looking at the sequence carefully, we can see that a command of 'a' *might* be missing between the two bs. The missing element, 'a', also causes subsequent false alarms since the next D elements would be expecting the presence of the missing one. Accordingly, $P(\mathsf{c}|\mathsf{abb})$ and $P(\mathsf{c}|\mathsf{bbc})$ will be very low.[6]

Conversely, however, *restoring* such a missing element can reduce multiple false positives that could possibly occur subsequently. Consider an element, which is supposed to be seen between $\sigma^{(t-1)}$ and $\sigma^{(t)}$, *i.e.*, the most recent and the current elements, respectively, is missing (although its actual absence or presence is unknown). In general, as soon as that happens, the probability of the following element, $\sigma^{(t)}$, will be affected by the missing one. Thus, whenever the probability of the t^{th} element (i.e., command or data) is significantly low, *i.e.*, less than the threshold used in (1), we try to recover the element between $(t-1)^{th}$ and t^{th} elements, assuming that it could be missing. This can be done by inferring the *most probably missing* element given the current history $\sigma^{(t-D)} \cdots \sigma^{(t-1)}$ as follows:

$$\sigma^* = \arg\max_{\sigma \in \Sigma} P\big(\sigma | \sigma^{(t-D)} \cdots \sigma^{(t-1)}\big).$$

Then, assuming that σ^* is missing between $\sigma^{(t-1)}$ and $\sigma^{(t)}$, we recalculate the probability of $\sigma^{(t)}$ as

$$P\big(\sigma^{(t)} | \sigma^{(t-D)} \cdots \sigma^{(t-1)}\big) \sim P\big(\sigma^* | \sigma^{(t-D)} \cdots \sigma^{(t-1)}\big) P\big(\sigma^{(t)} | \sigma^{(t-D+1)} \cdots \sigma^{(t-1)} \sigma^*\big).$$

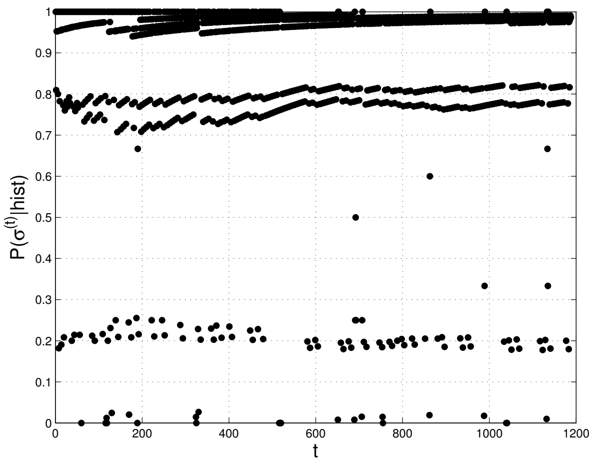
Our approach takes into account not only the probability of σ^* being actually the $(t-1)^{th}$ element (the second term), but also the likelihood of σ^* being positioned there (the first term). In the example above, $\sigma^* = \arg\max_{\sigma \in \{\mathsf{a},\mathsf{b},\mathsf{c}\}} P(\sigma|\mathsf{cab}) = \mathsf{a}$ according to Figure 4, and thus, $P(\mathsf{b}|\mathsf{cab}) \sim P(\mathsf{a}|\mathsf{cab})P(\mathsf{b}|\mathsf{aba})$, which is high enough to be considered normal.

As explained above, one missing element can cause subsequent false positives. This can be alleviated by inserting σ^* into the history; *e.g.*, by inserting 'a' before the boxed 'b' in the sequence presented by Figure 5. Moreover, $P(\mathsf{c}|\mathsf{abb})$ and $P(\mathsf{c}|\mathsf{bbc})$ can be recalculated by $P(\mathsf{c}|\mathsf{bab})$ and $P(\mathsf{c}|\mathsf{abc})$. However, we do not update the PST (in Figure 3) with the inferred missing element, since that can lead to biased probability distributions. Suppose that, after we have seen the dominant pattern ababcc as in the example above, abbcc starts appearing more often than before. For the first few times we will try to restore an a, believing it is missing. However, if we observe this pattern a non-negligible number of times, learning the new pattern is a more reasonable approach than to keep the initially dominant one. Use of the inferred missing element to update the PST makes the initially dominant pattern unchanged, which can cause more false positives (and also low detection rate), when the base pattern actually changes. Hence, we keep two buffers of elements (in each PST): one for building PST as described in Section III-B and another for calculating the probability of an element as in Section III-D. This means that the PST is built purely with what we have actually seen and the probability of a newly observed element is calculated mainly based on the learned PST, but possibly also with some inferred elements. By separating the buffers, we can limit locally the effect of the restored element (up to the next D elements) while preserving the global pattern.

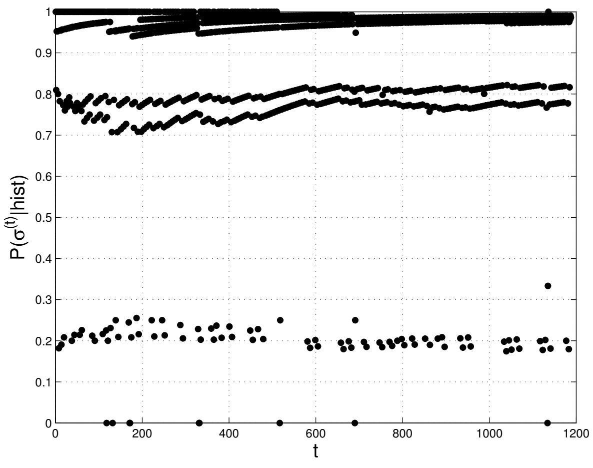
Figure 6 shows an example of how false positives are reduced. The sequence is from a master-to-slave connection observed in a testbed and contains 1321 Modbus commands. The sequence is a mixture of ababcc, abcc and abababcc (because of these multiple base patterns, the points are somewhat scattered). The probabilities of the first 10% commands were not calculated, but, as described above, the tree is incrementally learned with all of the 1321 commands. The plots are the probabilities of the remaining commands: (a) without any recovery of missing element and (b) with recovery for $\theta = 10\%$. The maximum PST depth is set to $D = 3$. Many false positives (see points below 0.1) were reduced from 25 to 9 false positives out of 1189 commands by the inference of missing elements. Most of the remaining false positives were due to out-of-order commands. For example, in '$\cdots$ abcacba', the last four commands caused four false positives in (a). With the recovery method, only one false positive at the last b was made. Looking at the sequence carefully, there is a high possibility that the order of ac was actually flipped, an event which cannot be corrected well by the proposed method.

Note again that the procedure explained above is carried out whenever an element seems to be abnormal, i.e., $P(\sigma^{(t)}|\sigma^{(t-D)} \cdots \sigma^{(t-1)}) < \theta$. This does not always improve

[6]If the pruning technique is used, we would end up calculating $P(\mathsf{c}|\mathsf{b})$ instead of $P(\mathsf{c}|\mathsf{abb})$ because $P(\mathsf{b})$ and $P(\mathsf{bb})$ are too low. In this case, $P(\mathsf{c}|\mathsf{b})$ will be close to 0.5 because of the base pattern ababcc. This shows that the pruning technique may also help reduce false positives.



(a) Without any recovery of missing element



(b) With recovery for $\theta = 10\%$

Fig. 6. Probabilities of commands observed from a Modbus testbed connection.

the probability, of course, since the low probability could be simply due to a rare observation, *i.e.*, a legitimate noise. Hence, the probability can even be lowered. In general, our approach does not increase the number of false positives, since without such a recovery, they would have already been considered abnormal. Furthermore, it could be possible to recover the wrong σ^* in the absence of a single, dominant pattern. In this case, the inferred σ^* can mislead the elements following it, and thus, can lead to low probabilities of them. However, as will be seen in Section IV, the proposed method can substantially reduce the false positive rate when the sequence under monitoring has a dominant pattern, which, in general, is the case for the ICS environment. Finally, we are aware that this technique could allow an attacker to insert additional specially crafted commands or data in legitimate sub-sequences. For the future, we plan to address this limitation by exploring randomization and corroboration capabilities that detect attacks against the architecture itself.

IV. EVALUATION

In this section, we quantify the performance of the proposed framework. We first evaluate it using a dataset obtained from a Modbus network testbed. Due to the lack of attack data, we evaluate how well the probabilistic suffix tree can model *normal* sequences of command and data. Then, we generate and use a synthetic dataset to see the performance of our framework in the presence of anomalies.

A. Evaluation with Modbus Dataset

The dataset is obtained from a Modbus network testbed that consists of 43 connections established among 2 masters and 25 slaves. The number of commands used by the connections is 4, *i.e.*, $\Sigma_c = \{a, b, c, d\}$. To the best of our knowledge, there are no attack/abnormal scenarios in this dataset, and thus all sequences are considered normal. The threshold, θ, and the maximum depth of PST trees, D, are fixed to 10% and 5, respectively, regardless of connections. With these settings, we compare the following three methods:

1) BATCH: Batch learning where each PST tree is learned with the entire sequence, *i.e.*, offline learning. Once it is built, the probability of each element (either command or data) of the sequence is calculated.
2) INC: The incremental learning presented in Section III-C, *i.e.*, *without* missing element inference.
3) INC_INF: The incremental learning *with* missing element inference presented in Section III-D.

Because we consider an online learning approach for INC and INC_INF, the first 10% of commands in each sequence are only used to build each PST (no probability calculation is performed) for all three methods.

We compute the false positive rate (FPR) of each sequence (of commands, data, or both) as follows:

$$FPR = \frac{\sum_{t=1}^{N} I\Big(P(\sigma^{(t)}|history) < \theta\Big)}{N},$$

where N is the length of the sequence (excluding the first 10% of elements) and $I(expr)$ is 1 if $expr$ is true or 0 otherwise.

Figure 7 presents the false positive rate of each method for the *command* sequence corresponding to each connection (Table I shows the mean and the standard deviation of each method). We can observe that INC_INF substantially reduces the number of false alarms for most of the connections as compared to INC. The reduction is significant especially in Connection 1–18, in which the base pattern is very simple, *e.g.*, bbc, and the commands follow the pattern quite well. As previously explained, the recovery of a missing command in such a case is fairly accurate, since the possible variations of the base pattern are limited. However, the reductions for the other connections are not that significant, since the dominant pattern is somewhat vague (in varying degrees) because of multiple base patterns or more noise (variations). In such cases, a recovery is more likely to fail, simply because such irregularity would set an incorrect context (*i.e.*, recent history), which would naturally lead to incorrect inference. Thus, in some particular cases, performance of the recovery process is not recommended, given the results for Connection 28 and 40.

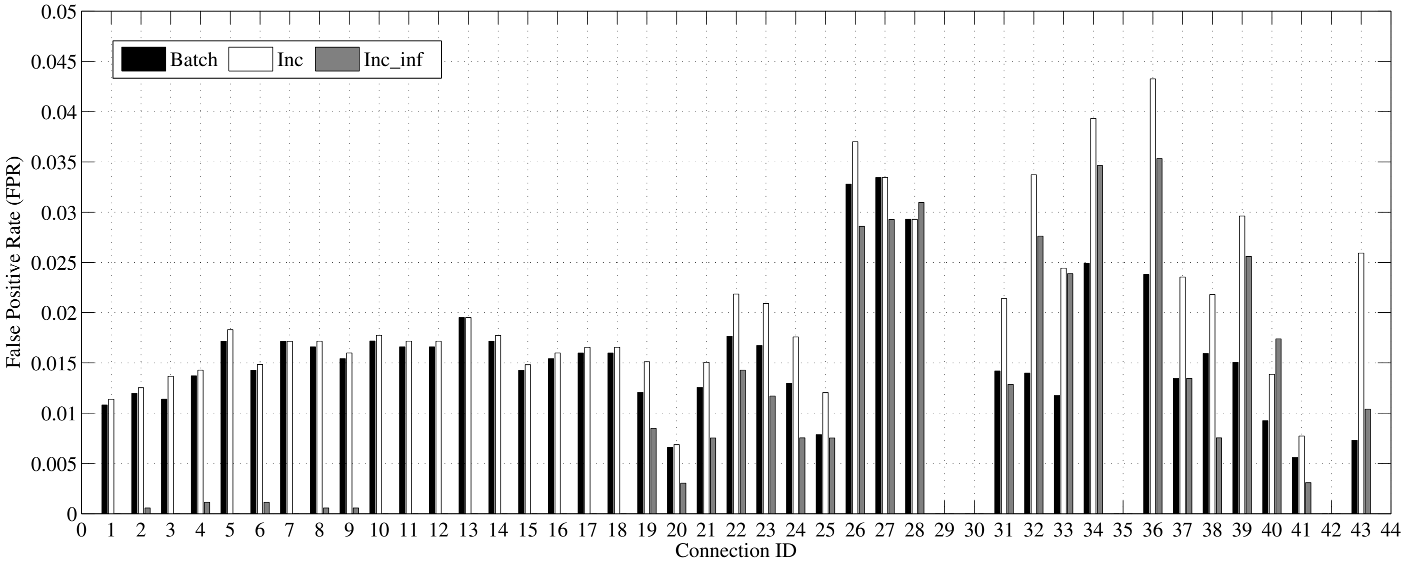


Fig. 7. False positive rates of command sequences on the Modbus testbed's connections.

TABLE I. THE AVERAGE FPR PER CONNECTION AND STD. DEVIATION.

	BATCH	INC	INC_INF
mean	1.429%	1.819%	0.848%
stdev	0.007	0.010	0.011

From these results, we can also observe that BATCH's FPR is always lower than or equal to the FPR of INC. This should be true since in INC, the probability of the current element is calculated using the PST learned so far.[7] Thus, if some noisy pattern appears in the early phase and starts appearing more and more later, INC would raise false alarms for them in the early phase, although they would be considered normal in the later phase. In BATCH, on the other hand, any sub-patterns that appear in the entire sequence are *smoothed out* and are learned beforehand, and thus would cause less false positives even if the base pattern changes over time. Even though BATCH performs better than INC (in terms of FPR), in many cases INC_INF achieves lower false positive rate than BATCH. Moreover, as we will illustrate later, BATCH does not detect anomalies well for the same reason it generates fewer false positives.

Next, using the dynamic Bayesian network model presented in Section III-A, we analyze the same dataset by calculating the *message-level* probabilities (i.e., Equation (7)). Figure 8(a) shows an example, where each point represents

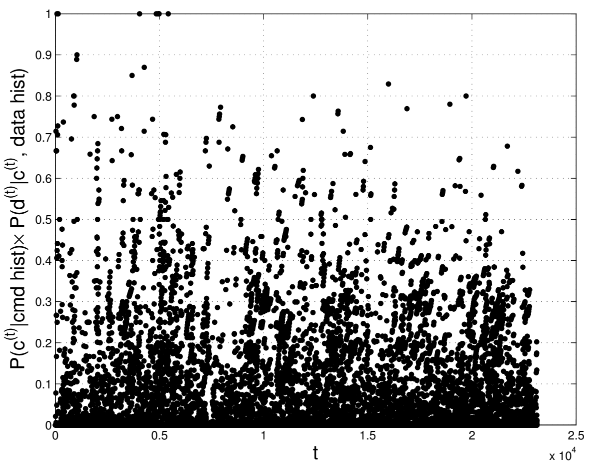
$$P\big(c^{(t)}\big|\text{cmd history}\big)P\big(d^{(t)}\big|c^{(t)};\text{data history})\big)$$

of each message that was captured in Connection 43. Unlike the false positive rate of the command sequence in Figure 7 (Connection 43, INC_INF), the FPR at the message-level is approximately 87% (20111 FPs out of 23123 messages), which is unacceptably high. This result is attributed to the very high FPR of the data sequence. This situation occurs often in data sequences since data does not exhibit a significant pattern. However, we observe that, for a fixed command type, data follows certain *templates* in its format. This is due to the nature of protocol-based communications, for which data is not arbitrarily generated. Instead, some of the *data fields* may vary, with others dependent on the rest of the fields, and still others simply fixed.

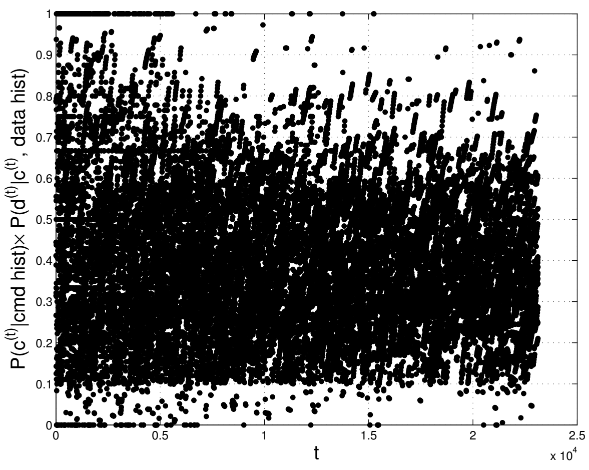
Consequently, we implemented a template-based online data clustering that incrementally extracts data templates and maps each data to one of them. Due to lack of space, we do not detail the procedure, but the main idea is to represent each cluster as a data template such as `75**0001`, `27**0001`, *etc.*, where $*$ is a wild-card. Each data d is labeled with the ID of the most similar template, determined by computing the Hamming distance [22] between d and all templates and choosing the minimum distance one, with the distance above a given threshold.[8] If there exists no such template, the data itself becomes a template of a new cluster and can be updated later as more data is available (*i.e.*, some of the fields are designated as wild-card). Accordingly, the PST for data sequence is built with the new labels, *i.e.*, template IDs. Consequently, data observed in Connection 43 was clustered into 3 templates (for the similarity threshold of 66%) – `75**0001`, `27**0001` and `00**0001`, and resulted in probabilities close to 1 due to a significant pattern in data sequence in the form of templates. This led to a considerable reduction in false positives as Figure 8(b) shows; with data clustering, most of the points (the message probabilities) moved towards the higher probabilities. As a result, only 362 messages (1.57%) were classified as abnormal. It should be noted that the data clustering method does not decrease the FPR since the clustering naturally reduces variations and thus increases the chance of exhibiting

[7]Note that if there is a clearly-dominant pattern, then the two methods will converge to the same PST when t is large enough.

[8]The similarity between two data d_1 and d_2 is calculated as $\frac{L-HD}{L} \in [0,1]$ where $L = l(d_1) = l(d_2)$ is the data length and $HD \in [0, L]$ is the Hamming distance between them. If $l(d_1) \neq l(d_2)$, we define the similarity as 0. Wild-cards do not count toward L and HD.



(a) Without data clustering. FPR=87%.



(b) With data clustering. FPR=1.57%.

Fig. 8. Message-level probabilities for Connection 43.

a significant pattern. In fact, although not presented here, the FPRs at the message-level of all connections decreased after applying the data clustering.

B. Evaluation with Synthetic Dataset

We now evaluate the PST-based pattern modeling with a set of synthetic data to determine how well it can detect anomalies while keeping the false alarms rates low. In what follows, we assume that we monitor command sequences only.

1) Input Generation: A set of random sequences is generated with the following parameters. For each sequence, a base pattern is first generated. Its length can range from 2 to 10 and it is randomly generated with 5 command types. Then, to build a command sequence, the base pattern is simply repeated. Each repetition, however, can vary according to the *missing probability*, P_M, which is given as an input parameter. This parameter controls the *drop rate* of a command. If $P_M = 0$, the entire sequence is simply a perfect repetition of the base pattern, while if $P_M > 0$, each command can be dropped with probability P_M. A higher P_M makes the resulting sequence look more random. To mimic attack scenarios, some random, abnormal subsequences are embedded into the dataset under test. At any position, a short sequence (that is randomly generated with the same command type pool) whose length ranges from 1 to the base pattern's length is inserted with the *attack* probability, $P_A \in [0, 40\%]$. With these rules, a sequence of length 30000 is generated. For the missing probability, we used 10% and 50%. For each, we generated 4000 sequences. Each marker in the following plots represents the average of the 4000 sequences for a given P_M.

2) Detection and False Positive: Given a sequence and the maximum PST depth, D, we calculate the probability of commands $c^{(1)}, c^{(2)}, \ldots$, as learning the PST incrementally. If $c^{(t)}$ is deemed abnormal, and there was indeed an attack sequence (or a part of it) within the recent D commands, the attack is *detected*. If there was no attack sequence, then this decision is a *false positive*. If $c^{(t)}$ is deemed normal, but there was an attack sequence (or a part of it) within the recent D commands, we *missed* the attack.

3) Result: Figure 9 presents the receiver operating characteristic (ROC) curve of the sequences generated with 10% missing probability. For each method, six values of thresholds are tested: $\theta = 0.01, 0.1, 0.2, 0.3, 0.4, 0.5$ (markers in the plot, left to right). First, similar to the result obtained with the real data in Section IV-A, the false positive rate of INC_INF is considerably lower than the others. However, there is no performance hit in terms of the detection rate; for a similar level of detection accuracy, BATCH and INC generate more false alarms. In other words, INC_INF outperformed the other methods in terms of the ability to distinguish attacks from legitimate noises.

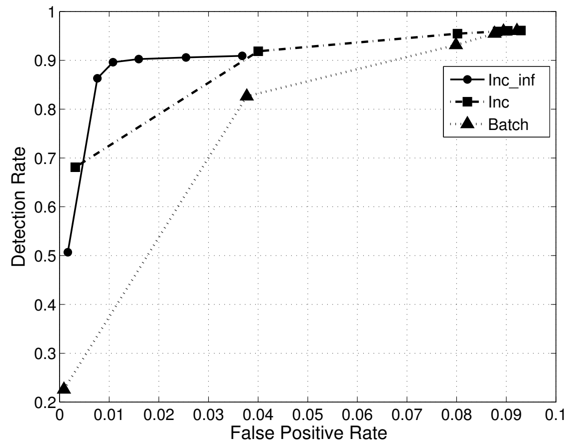


Fig. 9. Receiver operating characteristic (ROC) curve when $P_M = 10\%$.

This can be supported also by the signal-to-noise ratio (SNR) plot in Figure 10, computed as $DetectionRate/FalsePositiveRate$ (higher values mean better results), where INC_INF's SNR is higher than the others for every threshold. Another important result that

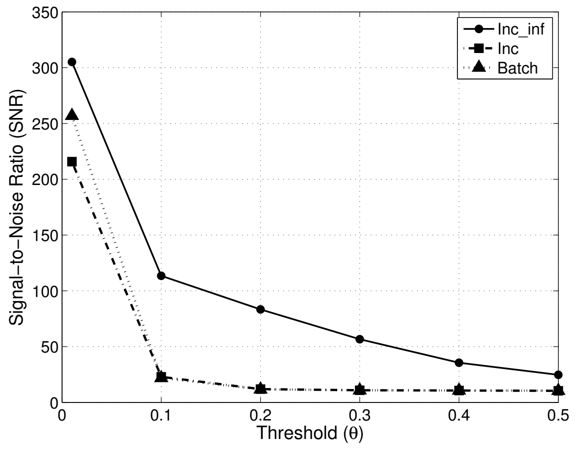


Fig. 10. Signal-to-noise ratio (SNR) when $P_M = 10\%$

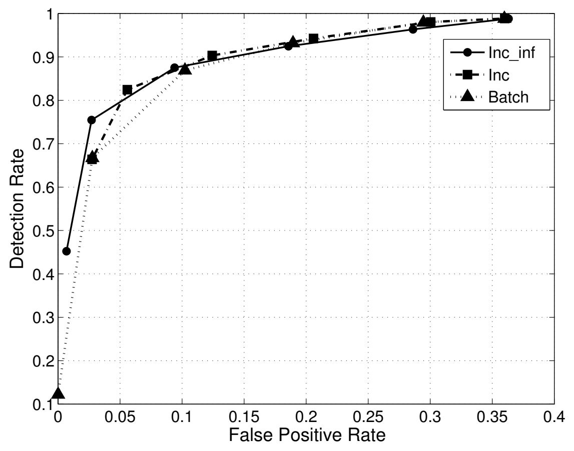


Fig. 11. Receiver operating characteristic (ROC) curve when $P_M = 50\%$

should be pointed out in Figure 9 is the low detection rate of BATCH, especially compared to INC. As mentioned earlier in Section IV-A, this is due to the fact that attack sequences and their subsequences are smoothed out and thus are deemed normal (probabilities are low but still above the threshold).[9] This also increases the chance of missing attacks. The performance gaps among the methods, however, are dependent on whether or not there exists a dominant pattern in the sequence under monitoring. As the ROC curve in Figure 11 shows, INC_INF's ability to sort out attack sequences has degraded with $P_M = 50\%$, i.e., more randomness in the sequence under monitoring. In such a case, employing a recovery of missing element may rather lead to more false positives by setting up a wrong context for near-future elements. Hence, it is recommended to apply the INC_INF method for monitoring sequences that exhibit a dominant pattern.

[9]However, as explained earlier, the two methods will be converged as time goes by.

V. RELATED WORK

So far, anomaly detection approaches have not been fully explored in the ICS context. Bigham *et al.* [10] analyzed n-grams [16] of power readings in an electricity network to detect anomalous events. The authors also proposed a technique that learns invariants, *i.e.* relationships between different power readings to model the normal behavior of the system. Düssel *et al.* [18] evaluated different combinations of feature extraction and similarity measurement techniques for SCADA network payload. These techniques were used to build the center of mass of the normal data, performing n-gram analysis on payloads. Abnormalities were determined by computing the distance from the center. Moreover, Cheung *et al.* [12] demonstrated the regularity in communications among SCADA network devices and developed a communication-pattern-based detector that triggers an alert when the availability of (Modbus) server or service is changed due to, for example, a denial-of-service attack. Valdes *et al.* [28] proposed a flow-based anomaly detection approach, which keeps a library of flows and, using simple statistics, such as mean and variance, detects flows that are unexpected or exhibit significant change in parameters such as packet inter-arrival time, volume, *etc.* Hadziosmanovic *et al.* [21] analyzed network- and host-based data traces from a real-world industrial control system network. The traces included communication topology and patterns, message type and content, and also host-level information such as system log and memory traces. The authors evaluated each of these data-trace types in terms of various criteria including threat scope, approach validation, analysis granularity and so on. Good summaries of SCADA-specific intrusion/anomaly detection are provided by Zhu and Sastry [29] and Garitano *et al.* [19].

To the best of our knowledge, probabilistic suffix trees have been mainly used for modeling and predicting discrete sequences, with applications including biological sequence modeling [9], corrupted text correction [25], part-of-speech disambiguation resolving [26], music representation [17], *etc.*, but not for the ICS context. Begleiter *et al.* [8] compares PSTs against other types of variable-order Markov models (VMMs) such as context tree weighting (CTW) and prediction by partial match (PPM). Chandola *et al.* [11] provides an extensive survey on various anomaly detection techniques for discrete sequences, considering among different applications, sequences of system calls or user commands.

VI. CONCLUSION

This paper proposed a novel anomaly detection method for ICS devices. Our approach is to build adaptive behavior models and use the intrinsic characteristics of the environment where the models are created to improve performance and reduce the false positive rate. The proposed method has been implemented and applied to a Modbus network testbed and a synthetic dataset. The experimental results showed that our framework exhibits a high detection rate for the synthetic dataset while successfully keeping the false positive rate in check.

A complete evaluation on real operational datasets will be a critical next step. However, due to the sensitivity of the data, obtaining real ICS traffic from the research or industry community is a difficult process, and most research organizations are prevented from sharing data by active non-disclosure agreements. We are currently pursuing different paths for getting access to real ICS data to further evaluate our framework. Moreover, we plan to extend the set of protocols that we investigate and to target different industry sectors. Last but not least, we plan to also extend the ICS-specific anomaly detection techniques within a more flexible and general framework, that can cope with long lasting attacks targeting our architecture.

Acknowledgments

The authors would like to thank Jaesik Choi at UNIST for his valuable comments and suggestions on improving the PST learning method and Martin Fong at SRI International for his precious help with the implementation of the anomaly detection system. This work was funded by the US Department of Homeland Security (DHS) Science and Technology (S&T) Directorate under contract no. HSHQDC-10-C-00144. The views and conclusions contained herein are the authors' and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DHS or the US government.

References

[1] "Common Industrial Protocol (CIP)," http://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00123R0_Common%20Industrial_Protocol_and_Family_of_CIP_Netw.pdf.

[2] "Distributed Network Protocol (DNP3)," http://www.dnp.org/pages/aboutdefault.aspx.

[3] "IEC 60870-5-104," http://webstore.iec.ch/preview/info_iec60870-5-104%7Bed2.0%7Den_d.pdf.

[4] "IEC 61850," http://www.iec.ch/smartgrid/standards/.

[5] "Modbus," http://www.modbus.org/specs.php.

[6] "Quickdraw SCADA IDS," https://www.digitalbond.com/tools/quickdraw/.

[7] "Stuxnet," http://www.symantec.com/security_response/writeup.jsp?docid=2010-071400-3123-99.

[8] R. Begleiter, R. El-yaniv, and G. Yona, "On prediction using variable order markov models," *Journal of Artificial Intelligence Research*, vol. 22, pp. 385–421, 2004.

[9] G. Bejerano and G. Yona, "Variations on probabilistic suffix trees: statistical modeling and prediction of protein families." *Bioinformatics*, vol. 17, no. 1, pp. 23–43, 2001.

[10] J. Bigham, D. Gamez, and N. Lu, "Safeguarding SCADA Systems with Anomaly Detection," in *MMM-ACNS*, ser. Lecture Notes in Computer Science, V. Gorodetsky, L. J. Popyack, and V. A. Skormin, Eds., vol. 2776, 2003, pp. 171–182.

[11] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection for discrete sequences: A survey," *IEEE Trans. on Knowl. and Data Eng.*, vol. 24, no. 5, pp. 823–839, May 2012.

[12] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes, "Using Model-based Intrusion Detection for SCADA Networks," in *Proceedings of the SCADA Security Scientific Symposium*, 2007.

[13] J. R. Crandall, Z. Su, S. F. Wu, and F. T. Chong, "On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits," in *Proceedings of the 12th ACM Conference on Computer and Communications Security*, ser. CCS '05, 2005, pp. 235–248.

[14] G. F. Cretu, A. Stavrou, M. E. Locasto, S. J. Stolfo, and A. D. Keromytis, "Casting out demons: Sanitizing training data for anomaly sensors," in *IEEE Symposium on Security and Privacy*, 2008, pp. 81–95.

[15] G. F. Cretu-Ciocarlie, A. Stavrou, M. E. Locasto, and S. J. Stolfo, "Adaptive anomaly detection via self-calibration and dynamic updating," in *RAID*, 2009, pp. 41–60.

[16] M. Damashek, "Gauging similarity with n-grams: Language-independent categorization of text," *Science*, vol. 267, no. 5199, pp. 843–849, 1995.

[17] S. Dubnov, G. Assayag, O. Lartillot, and G. Bejerano, "Using machine-learning methods for musical style modeling," *Computer*, vol. 36, no. 10, pp. 73–80, Oct. 2003.

[18] P. Düssel, C. Gehl, P. Laskov, J.-U. Buß er, C. Störmann, and J. Kästner, "Cyber-critical infrastructure protection using real-time payload-based anomaly detection," in *Proceedings of the 4th International Conference on Critical Information Infrastructures Security*, ser. CRITIS'09, 2010, pp. 85–97.

[19] I. Garitano, R. Uribeetxeberria, and U. Zurutuza, "A review of scada anomaly detection systems." in *SOCO*, ser. Advances in Soft Computing, E. Corchado, V. Snsel, J. Sedano, A. E. Hassanien, J. L. Calvo-Rolle, and D. Slezak, Eds., vol. 87, 2011, pp. 357–366.

[20] Z. Ghahramani, "Learning dynamic bayesian networks," in *Adaptive Processing of Sequences and Data Structures*, ser. Lecture Notes in Computer Science, vol. 1387, 1998, pp. 168–197.

[21] D. Hadiosmanović, D. Bolzoni, S. Etalle, and P. H. Hartel, "Challenges and opportunities in securing industrial control systems," in *Proceedings of the IEEE Workshop on Complexity in Engineering, COMPENG 2012, Aachen, Germany*, June 2012, pp. 1–6.

[22] R. W. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[23] K. Murphy, "Dynamic bayesian networks: Representation, inference and learning," Ph.D. dissertation, UC Berkeley, 2002.

[24] J. Newsome, B. Karp, and D. Song, "Polygraph: automatically generating signatures for polymorphic worms," in *Security and Privacy, 2005 IEEE Symposium on*, 2005, pp. 226–241.

[25] D. Ron, Y. Singer, and N. Tishby, "The power of amnesia: Learning probabilistic automata with variable memory length." *Machine Learning*, vol. 25, no. 2-3, pp. 117–149, 1996.

[26] H. Schütze and Y. Singer, "Part-of-speech tagging using a variable memory markov model," in *Proceedings of the 32Nd Annual Meeting on Association for Computational Linguistics*, ser. ACL '94, 1994, pp. 181–187.

[27] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis, and S. J. Stolfo, "On the infeasibility of modeling polymorphic shellcode," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, ser. CCS '07, 2007, pp. 541–551.

[28] A. Valdes and S. Cheung, "Communication pattern anomaly detection in process control system," in *IEEE International Conference on Technologies for Homeland Security*, 2009.

[29] B. Zhu and S. Sastry, "SCADA-specific Intrusion Detection/Prevention Systems: A Survey and Taxonomy," in *Proceedings of the 1st Workshop on Secure Control Systems*, 2010.