information

MDPI

Article

Verification Method for Accumulative Event Relation of Message Passing Behavior with Process Tree for IoT Systems †

Mohd Anuaruddin Bin Ahmadon * and Shingo Yamaguchi *

Graduate School of Sciences and Technology for Innovation, Yamaguchi University, 2-16-1 Tokiwadai, Ube 755-8611, Japan

* Correspondence: anuar@yamaguchi-u.ac.jp (M.A.B.A.); shingo@yamaguchi-u.ac.jp (S.Y.)

† This paper is an extended version of our paper published in Mohd Anuaruddin Bin Ahmadon and Shingo Yamaguchi, "Process-Based Anomaly Detection and Analysis for Cyber-Physical System with MQTT Protocol," Proceedings of the IEEE ICCE 2020, Las Vegas, NV, USA, 4–6 July 2020.

Received: 19 March 2020; Accepted: 20 April 2020; Published: 23 April 2020

Abstract: In this paper, we proposed a verification method for the message passing behavior of IoT systems by checking the accumulative event relation of process models. In an IoT system, it is hard to verify the behavior of message passing by only looking at the sequence of packet transmissions recorded in the system log. We proposed a method to extract event relations from the log and check for any minor deviations that exist in the system. Using process mining, we extracted the variation of a normal process model from the log. We checked for any deviation that is hard to be detected unless the model is accumulated and stacked over time. Message passing behavior can be verified by comparing the similarity of the process tree model, which represents the execution relation between each message passing event. As a result, we can detect minor deviations such as missing events and perturbed event order with occurrence probability as low as 3%.

Keywords: message-passing; event execution verification; process mining; closed-loop IoT

1. Introduction

IoT is a network that consists of uniquely addressable objects such as sensors and actuators. These objects interconnect with each other, and IoT allows sensing and controls tasks remotely. In the real world, IoT objects can consist of heterogeneous cyber and physical objects that communicate with specific protocols. Each IoT object must operate normally to follow this communication protocol. However, cyber-physical objects are prone to hardware or software error, signal interference, power instability, and also cyber-attacks. Therefore, they may not be able to communicate based on the specification of the IoT networks. IoT system is widely implemented in monitoring and control systems such as Network Control Systems (NCS) [1], Industrial Control Systems (ICS), Supervisory Control and Data Acquisition (SCADA) Systems [2], and Distributed Control Systems (DCS) [3]. These systems play a vital part in industrial infrastructures, energy management, machine automation, and healthcare support. The stability of an IoT network is essential in ensuring the quality of monitoring and control of the network infrastructures.

One of the significant challenges in IoT is scalability. Due to the nature of the distributed and scalable network, IoT objects inter-communicate more frequently, and a large amount of message transmission happens. Most large-scale IoT systems operate based on the feedback-control loop model [4–6]. In this feedback-control loop model, once a breakdown occurs, it will cause a chain of breakdown to the cyber and physical parts and eventually will risk the safety of the user [7]. Therefore,

before an accident happens, it is important to analyze and detect minor incidents as early as possible. From the viewpoint of incident management, it is important to detect minor abnormality so that proper preventive measures can be taken accordingly. However, incidents caused by minor abnormalities do not always occur in every cycle. Figure 1 illustrates incidents caused by minor abnormalities that occur at certain system cycles. IoT systems are prone to minor abnormalities that may be caused by broken sensors [8], low-battery power [9], and weak signals due to interference [10]. These factors do not directly cause abnormal behavior but show the signs of a major breakdown in the future.

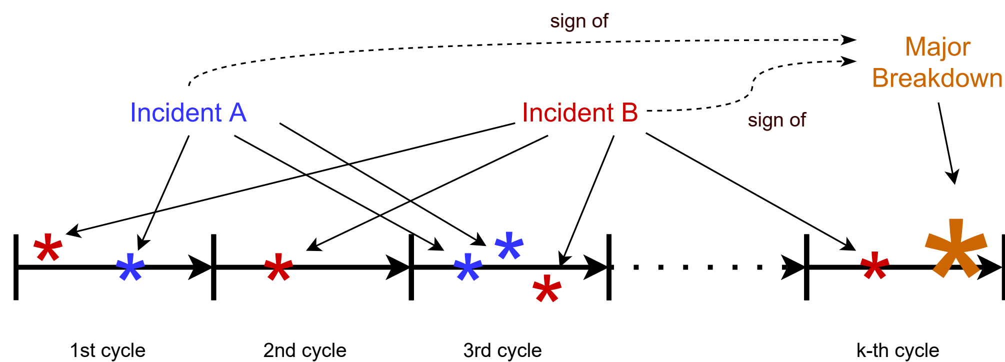


Figure 1. An illustration of minor and infrequent incidents in closed-loop IoT systems. Minor incidents show the sign of major breakdown. However, incidents caused by a minor abnormality are hard to detect unless with accumulative behavior monitoring.

It is crucial to implement a system that monitor NCS, ICS, SCADA, or DCS to ensure continuous normal operation. However, monitoring a system is still a challenging task due to the infrequent occurrences of abnormality at each system cycle. In order to verify the stability and correctness of the IoT systems, a constant monitoring and verification technique is required. The most practical way of monitoring complex and scalable closed-loop IoT systems is by modeling its normal behavior and look for any deviations during operation. However, for systems with high parallel message transmission, they will have minor and infrequent random changes in the system behavior. Therefore, it is hard to detect any deviations by only looking at one point of an event such as packet header or data value. The IoT Developer Survey 2019 by Eclipse Foundation [11] showed that IoT protocol such as MQTT [12], XMPP [13], REST [14], and CoAP [15] is gaining attention from system developers. These protocols can ensure the quality of services and stable message handling. Figure 2 shows an example of the message passing of a heating, ventilation, and air conditioning (HVAC) system via MQTT protocol. The HVAC system consists of a message passing server called broker, heat sensors, valves, and cooler fans. It shows a simple message passing between the broker and the nodes with CONNECT, CONNACK, PUBLISH, PUBACK, SUBSCRIBE, and SUBACK messages executed in sequence within two parallel (*PAR*) blocks. The execution sequence between *PAR* blocks has no sequence order. However, the sequence order of MQTT's handshaking inside the *PAR* blocks must be strictly followed at all times. Due to factors such as low battery power or signal interference, messages might be dropped or delayed. It is hard to detect dropped or delayed messages by only looking at one point of message passing event.

In this paper, we propose a process-based message passing verification method for IoT systems. While conventional detection methods focus on monitoring data packets at one point, we focus on the accumulative events of message passing. Concretely, we verify the relationship of the execution order of message passing between the devices over a certain period of system cycle. As shown in Figure 3, we performed a 'bird's eye' monitoring technique by discovering the variation of the process model of the message passing behavior with process mining. Then, we grasp the structure of the process model with a tree model called process tree to produce an accumulative normal model. Using the process

tree, we check for any deviations from the normal behavior by comparing the observed behavior with the accumulative normal model.

After the introduction in Section 1, Section 2 gives the problem statement of message passing behavior verification. Then, we give the method of verification using process tree similarity check. In Section 3 we give the results of the evaluation experiments and shows the reliability of normal process models and detection capability of the proposed method. Next, in Section 4 we discuss the evaluation result. In Section 5 we discuss several related works and compare the proposed method with the previously proposed methods. Finally, we give the conclusion in Section 6.

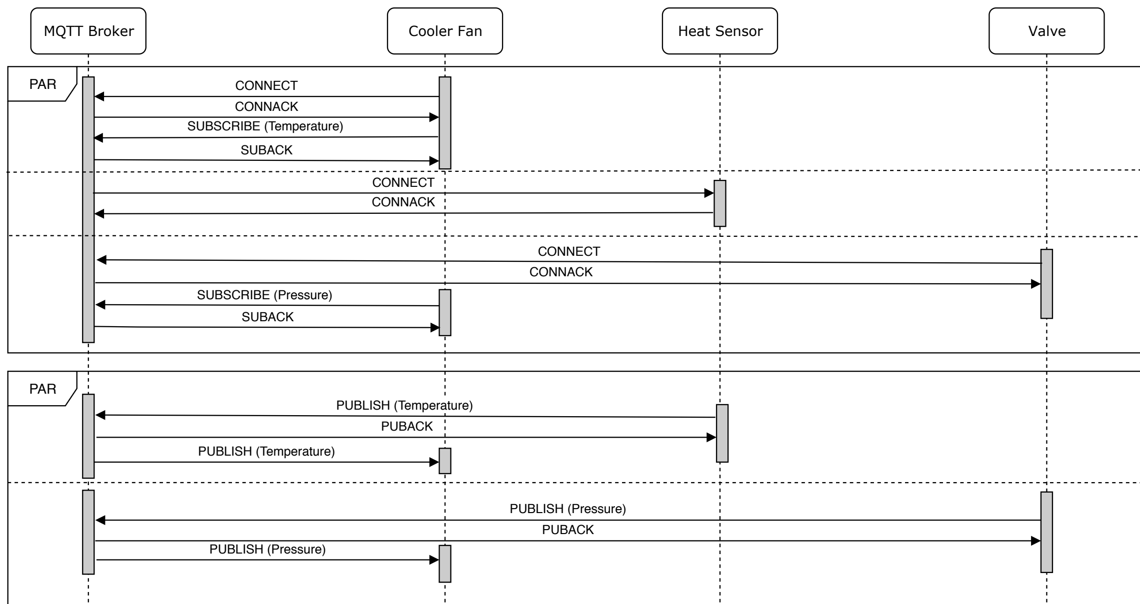


Figure 2. An example of a simplified heating, ventilation, and air conditioning (HVAC) system's message passing using MQTT protocol.

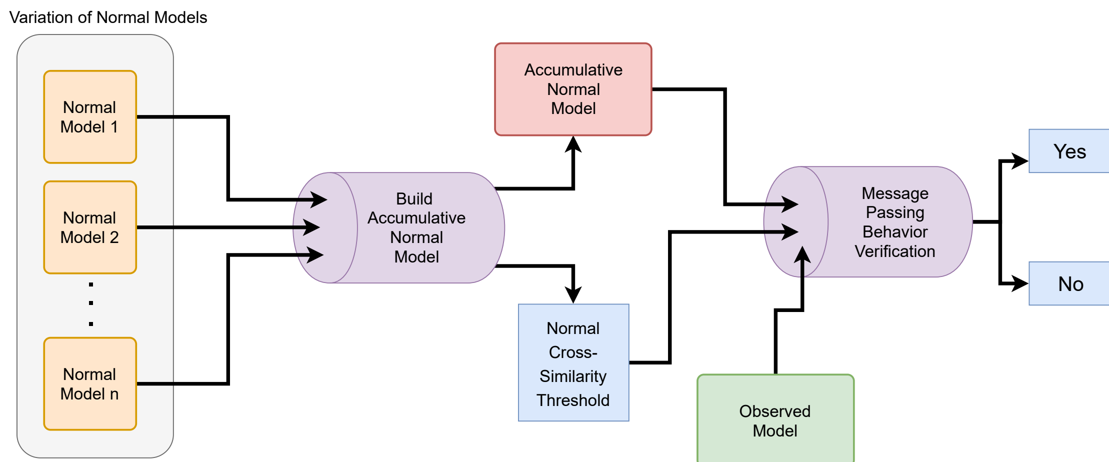


Figure 3. Overview of our approach.

2. Verification Method of Message Passing Behavior in an IoT System

In this section, we give the problem of message passing behavior verification and introduce the proposed method.

2.1. Message Passing Behavior Verification Problem

We take a wireless sensor network (WSN) as an example of an IoT system with many frequent and parallel message-passing events. In WSN, lossy transmission and battery reduction is a usual phenomenon. This phenomenon will cause the system to behave differently than expected. Some examples that may cause different behavior are delayed messages and dropped messages due to low battery and weak signals. The transmission and receiving ratio decrease over time, and it is hard to grasp the changes in behavior, especially when they are too minor. We can consider this phenomenon to be the cause of fluctuations and time delay. As stated in the introduction, minor deviation from normal behavior can be considered as incidents that show the signs of a possible major breakdown. It is important to collect and analyze the incidents in order to prevent possible accidents or re-occurrences in the future.

Figure 4 shows an example of an anomaly detection system positioned in an IoT system. The message passing log is collected from the centralized log collection server, such as the broker. Our objective is to detect if there is any deviation of message passing behavior from normal behavior. If the HVAC system is a closed-loop system that only operates at pre-specified execution order at all times, we should expect the same behavior for normal operation. Several works have been proposed in Ref. [16–22] regarding behavior verification. However, as stated in the previous sections, we cannot ignore fluctuations [23] and delays in a system. Most IoT systems such as WSN run in low-power and lossy networks where delayed messages and dropped messages are expected.

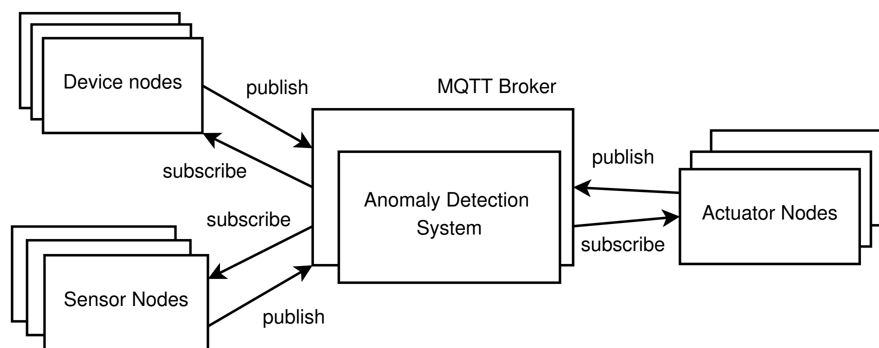


Figure 4. Overview of an IoT system with anomaly detection system.

Instead of detecting major deviations, we focus on minor deviations caused by fluctuations and delays that exist in message-passing as follows:

- Fluctuation: Infrequent dropped message due to broken sensor, low sensor battery, or weak network signal in IoT devices.
- Delay: Invalid order of message passing due to delay in sending and receiving messages.

Figure 5 shows the illustration of a publish-subscribe message of the HVAC system in a building automation system. The message passing protocol was described in Section 1. The HVAC system operates to control the temperature by controlling the level of heat by circulating steam and air using a valve and cooler fan. Steam is circulated through the valve, depending on the level of heat sensed by the heat sensor to various rooms in the building. Then, the cooler fan controls the air ventilation to minimize the temperature difference between outdoor and indoor air temperature. The sequence of message passing is important for temperature control. A heat sensor must be able to continuously send messages containing the heat status i.e., at each 1 s interval. Then, the valve and cooler fan must receive the data at every 1 s simultaneously so that the difference between the temperature is controlled properly. In case of low battery, a heat sensor might infrequently lose power and drop the message at around 2 or 3 s intervals. Fluctuation in sensor reading can cause unstable circulation of steam and air inside the rooms. Furthermore, if signal interference occurs due to a large area or dense sensor

network, the cooling fan might not be able to send or receive the signal for a while. Communication protocols such as MQTT can guarantee the quality of service by repeating to send the same message again. However, even though the message is delivered, the message is considered a late message because of the delay. Delay in sending messages can cause an asynchronous response between the cooler fan and the valve. Therefore, the cooler fan and valve will have to work harder than usual to circulate the steam and ventilate the air. Eventually, it will cause a breakdown of the steam and air circulation system. See Figure 6 for an example of a dropped message and a delayed message in an event log. In conventional ways, the event log is only used when looking at a point of events. However, when the above fluctuations and delays occur, it is hard to grasp any unusual behavior by only looking at the event log. Some messages might be missing or recorded in disoriented order.

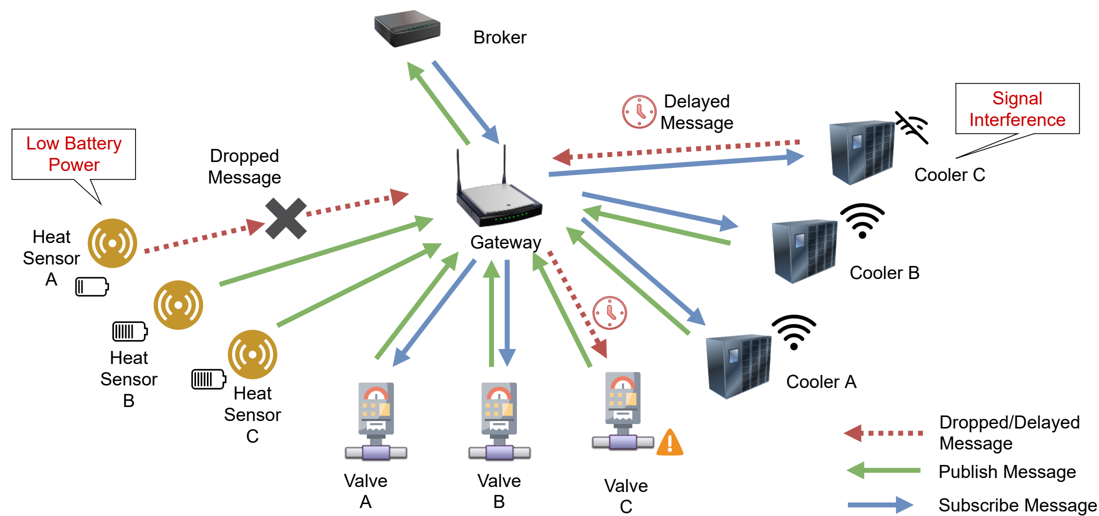


Figure 5. An illustration of message passing in the HVAC system.

Event Log of Normal Behavior

507848,mosquitto version 1.5.7 starting
507848,Using default config.
507848,Opening ipv4 listen socket on port 1883.
507848,Opening ipv6 listen socket on port 1883.
...
507855,New client connected from 192.168.12 as CoolerFan (c1, k60).
507855,Sending CONNACK to CoolerFan (0, 0) }
507855,Received PUBLISH from HeaterSensor (d0, q0, r0, m0, 'Temperature')
507855,Sending PUBLISH to CoolerFan (d0, q0, r0, m0, 'Temperature')
507855,Received SUBSCRIBE from CoolerFa n
...
507869,New connection from 192.168.1.12 on port 1883.
507869,New client connected from 192.168.1.12 as CoolerFan (c1, k60).
507869,Sending CONNACK to CoolerFan (0, 0)
507869,Received SUBSCRIBE from CoolerFan
507869,Pressure (QoS 2)
507869,CoolerFan 0 Pressure
507869,Sending SUBACK to CoolerFan
507869,Received DISCONNECT from CoolerFan
...

Delayed Message

Dropped Message

Event Log of Monitored Behavior

507848,mosquitto version 1.5.7 starting
507848,Using default config.
507848,Opening ipv4 listen socket on port 1883.
507848,Opening ipv6 listen socket on port 1883.
...
507855,New client connected from 192.168.12 as CoolerFan (c1, k60).
507855,Sending CONNACK to CoolerFan (0, 0) }
507855,Sending PUBLISH to CoolerFan (d0, q0, r0, m0, 'Temperature')
507855,Received PUBLISH from HeaterSensor (d0, q0, r0, m0, 'Temperature')
507855,Received SUBSCRIBE from CoolerFan
...
507869,New connection from 192.168.1.12 on port 1883.
507869,New client connected from 192.168.1.12 as CoolerFan (c1, k60).
507869,Sending CONNACK to CoolerFan (0, 0)
507869,Received SUBSCRIBE from CoolerFan
507869,Pressure (QoS 2)

507869,Sending SUBACK to CoolerFan
507869,Received DISCONNECT from CoolerFan
...

Figure 6. Event log of a message passing.

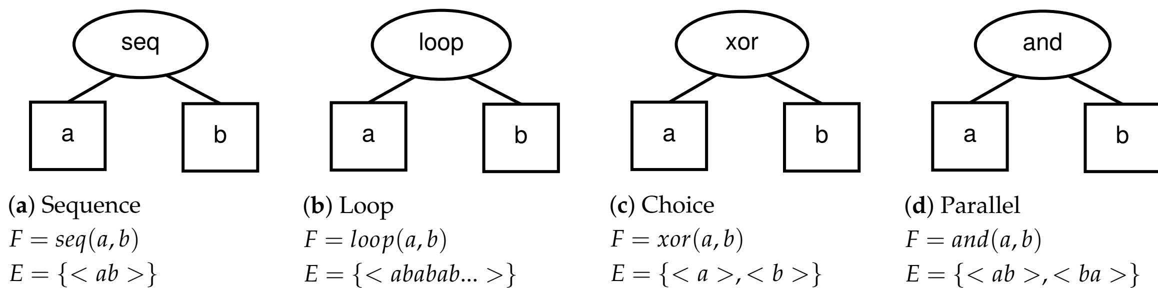
2.2. *Message Passing Behavior Verification Method*

In this section, we propose our method of message passing behavior verification. As the first step, we need to grasp the relation between message-passing events. Therefore, we need to build a process model that can represent relations between events. Therefore, we applied a tree-based model called process tree as our process model. The importance of visualizing and grasping the event relation with process tree is that fluctuations and delays can be represented with the process tree operator.

Process tree [24,25] represents the structure of a process. Each leaf node and each internal node respectively represent tasks and operators in the process. Process trees can be obtained from the event log using a process mining algorithm called inductive miner (IM) [26]. IM is available in the process

mining tool ProM [27] as a plugin. Figure 7 shows the most basic process trees and its formula F. It has four operators Θ; sequence (*seq*), selection (*xor*), parallel (*and*), and loop (*loop*) operators. An ordered operator has child nodes with an ordered relationship, such as *seq* and *loop*. An unordered operator has child nodes with no ordered relationship between them, such as *xor* and *and*. The process tree is represented by π_N. The definition of a process tree is given as follows:

Definition 1. *A process tree is a rooted tree* (V, E, Θ, r) *with root* $r \in \Theta$ *where* Θ *is a set of operators such that* $\Theta = \{\theta_1, \theta_2, \cdots, \theta_i\}$*,* V *is the set of vertices such that* $V = \{v_1, v_2, \cdots, v_j\}$*, and* E *is the set of edges such that* $E = \Theta \times V$*.*



(**a**) Sequence
$F = seq(a, b)$
$E = \{< ab >\}$

(**b**) Loop
$F = loop(a, b)$
$E = \{< ababab... >\}$

(**c**) Choice
$F = xor(a, b)$
$E = \{< a >, < b >\}$

(**d**) Parallel
$F = and(a, b)$
$E = \{< ab >, < ba >\}$

Figure 7. Process tree Π with the equivalent process tree formula F and trace E. *seq* and *loop* are ordered operators in which the trace E must follow the sequence of the action from left to right, as shown in the process tree. *xor* and *and* are non-ordered operators where the actions do not need to follow sequence order.

Figure 8 shows an example of a process tree that represents an event log of message passing in the HVAC system. As a comparison example, Figure 9 shows the comparison of process trees based on operator type and subtrees. Based on Levenshtein distance, we can check the distance between the trees. Next, we show the conditions of equivalence for process trees, which are decided by the type of operators. A process tree consists of operator nodes and action nodes. Operator nodes represent the relations between the actions. We can calculate the equivalence by calculating the edit distance between the process tree formulas. A method to calculate the distance is by using dynamic-programming based Levenshtein distance. We extended Levenshtein distance [28] to calculate the similarity of two process tree. The modified Levenshtein distance set the editing cost when two elements are not equal, such that $X_i \neq Y_j$ depends on the type of operator, i.e., ordered or non-ordered.

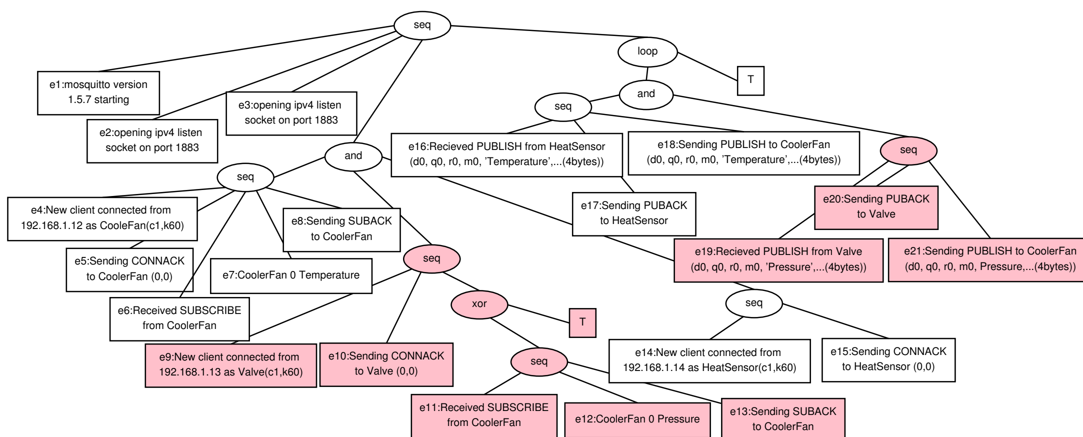


Figure 8. A process tree model L_α representing the normal behavior. The nodes colored in red show events with fluctuations, such as dropped and delayed messages. The normal model tolerates missing nodes and perturbed order.

Figure 9 shows the cases that determine the editing costs of process tree; (i) process trees with both ordered operator; (ii) process trees with both non-ordered operator; and (iii) process trees with different operators. Here, we apply the concept of equivalence to show that a process tree L_α is the same at L_β. Basically, the process tree L_α is equivalent to L_β if the root operator and the subtrees are the same. Then, if the root is an ordered operator, i.e., sequence or loop, then its subtrees must be equivalent in order from left to right. If the root is a non-ordered operator, i.e., parallel or exclusive-choice, then the sets of the subtrees must be the same. The order of subtrees is ignored for non-ordered operators. Next, we check the equivalence of subtrees recursively and return the equivalence value of each subtree. For ordered operators, we check the subtrees in order, and for an unordered operator, we perform an element subset cross-check between all of the subtrees.

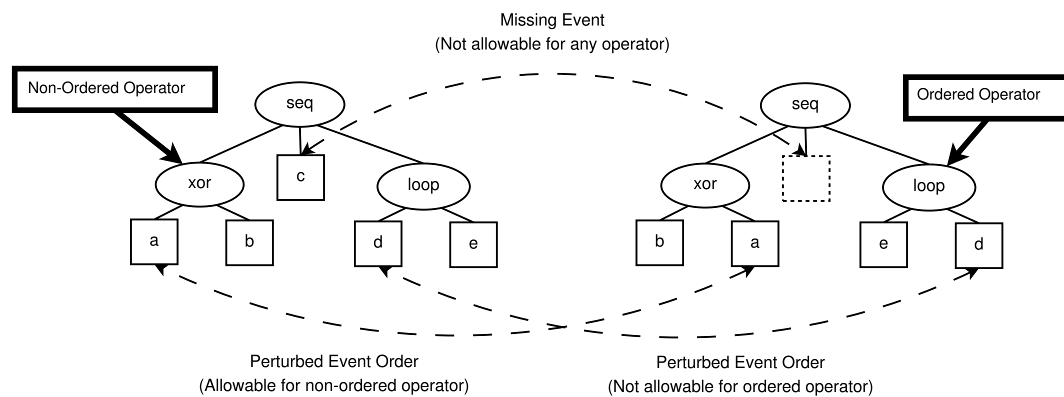


Figure 9. An illustration of execution relation comparison between two process trees for behavior verification. The process tree on the left is a process tree representing normal behavior, and the process tree on the right represents observed behavior.

Equation (1) shows the equation for calculating the Levenshtein distance. Note that the difference with original Levenshtein distance is that function $lev_{X,Y}$ calculates the distance based on the type of operator. Figure 10 shows the illustration of similarity calculation using the modified Levenshtein distance using matrix iteration and function $lev_{X,Y}$ is recursively called at each operator.

$$lev_{X,Y}(i,j) = \begin{cases} \max(i,j) & if\ min(i,j) = 0, \\ \min \begin{cases} lev_{X,Y}(i-1,j)+1 \\ lev_{X,Y}(i,j-1)+1 \\ lev_{X,Y}(i-1,j-1)+1_{(X_i \neq Y_j)} \end{cases} & otherwise \end{cases} \tag{1}$$

Depending on whether the respective symbols are the same or not, the cost corresponds to delete, insert, match, or mismatch. For example, we remove c and e from $seq(a,b,c,e)$ and add c and d to the order to make $seq(a,c,b,d)$. The cost of editing $X = seq(a,b,c,e)$ to $Y = seq(a,c,b,d)$ is 4. $dist(X,Y)$ is a function that finds the edit distance of the strings X and Y. $sim(X,Y)$ has a value between 0 to 1. The similarity calculation is given as follows:

$$sim(X,Y) = 1 - \frac{dist(X,Y)}{max(|X|,|Y|)} \tag{2}$$

Concretely, we give our procedure of verification in this section. To detect any anomalies, we perform a partial similarity check on all the subtrees of the mined process trees. The verification procedure is as follows:

≪ Process Tree-Based Message Passing Behavior Verification ≫
Input: Normal model $L_{\alpha_1}, L_{\alpha_2}, \cdots, L_{\alpha_i}$, Event log E_β
Output: Does event log E_β contain abnormal behavior?

1 Extract process tree L_β from E_β using inductive miner [2].
2 Build accumulative normal model L_α by marking and adding nodes with Levenshtein distance more than 1 between $L_{\alpha_1}, L_{\alpha_2}, \cdots, L_{\alpha_i}$.
3 Calculate the cross-similarity $sim(L_{\alpha_n}, L_{\alpha_m})$ where $n \neq m$. Obtain the lower bound Σ normal threshold from the standard deviation of cross-similarity.
4 Calculate the similarity between L_α and L_β. If $\text{sim}(L_\alpha, L_\beta) \leq \Sigma$, then output "No" and stop.
5 Output "Yes" if $sim(\ell_n, \ell_m) \leq \Sigma$ and stop.

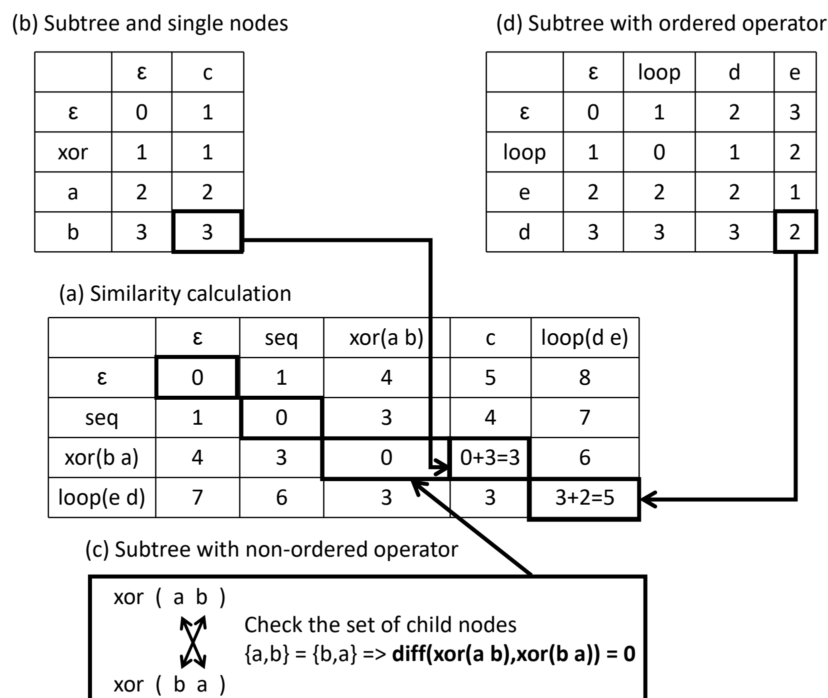


Figure 10. An illustration of the similarity calculation of two process tree formulas. (**a**) Distance calculation at main formula; (**b**) when comparing subtree and single nodes, we check if the single node is a child of the subtree's operator; (**c**) for a subtree with a non-ordered operator, we check the set of child nodes. For the same subset, the distance is 0; (**d**) for a subtree with an ordered operator, we check the sequence by recursively calling the modified Levenshtein function $lev_{X,Y}$.

By applying the given ≪ Process Tree-Based Message Passing Behavior Verification ≫ procedure, we can check for nodes that deviate from the normal process tree. Figure 11 shows the illustration of observed model L_β. By comparing to L_α, we can verify the deviation in the nodes of the process tree depending on the type of operators. The procedure returns 'yes' because an anomaly was found at node e_{18}. The purple nodes show nodes that are different from the nodes in L_α.

In the experiments in Section 3, we performed the given procedure to around 100 to 240 events in message passing. We consider the following cost for implementation:

1. Process Mining: The process mining takes less than 2 s to be computed using IM [26] algorithm for event log with around 3000 traces. The complexity of IM is discussed in Ref. [29,30], which shows that compared to other process discovery methods, IM is simpler and can be performed in real-time for large event logs.

2. Similarity Check: The similarity check is based on the recursive version of Levenshtein distance, which uses an iterative matrix to hold the distance of each subtree. Distance is usually computed with dynamic-programming procedure. Computation of two process tree formula $sim(X,Y)$ takes $O(|X| \times |Y|)$ time and $O(|X| + |Y|)$ space. However, since the process tree formula can be decomposed into subtree formulas, we can obtain shorter strings and it can be computed in parallel [31] to achieve shorter computation time. In the implementation, we computed 200 events at most in 24 s using parallel processing technique.
3. Storage: The storage for 1000 traces took around 152 kb. If we take 100 cycles that produce 10,000 traces, it will take around 1.5 MB per log file. However, we recorded the log in the broker who has hardware performance more than enough to capture the message log and execute the verification procedure.

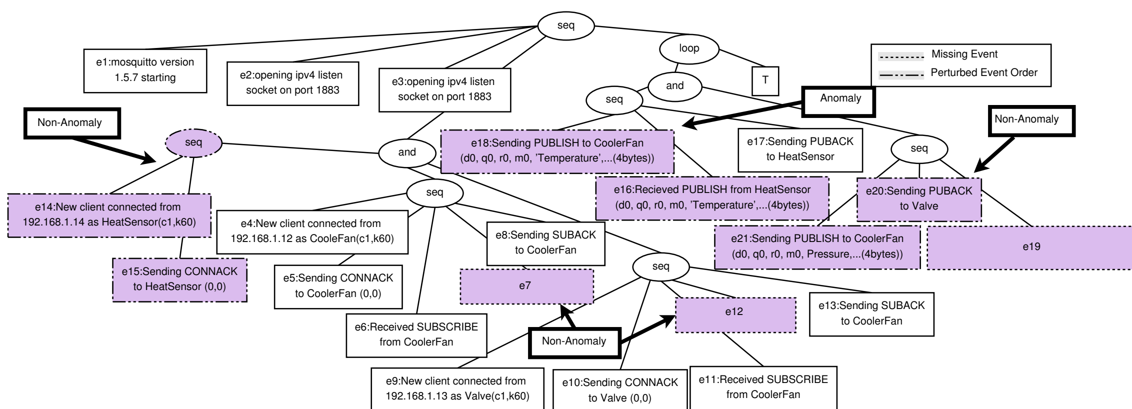


Figure 11. A process tree model L_β representing observed behavior. Some missing events and perturbed event order are considered non-anomaly because the normal model L_α marks them as tolerable nodes.

From the consideration of the cost to implement the verification method, we found that the method is implementable for real-world usage. We can also expect near real-time detection based on the evaluation shown in Section 3.

3. Results

In this section, we show the feasibility of verification applied to anomaly detection and some analysis regarding the normal models. We give an evaluation to the (i) analysis reliability of normal models; and (ii) effectiveness of verification capability for detecting anomalies with low occurrences in message passing. First, we explain the experimental setup then we show the evaluation results.

3.1. Experimental Setup

We conducted the experiments using three types of nodes that represent heat sensors, valves, and cooler fans, as shown in Figure 6. The data are generated using 1 broker node, 1 wireless network router, and 44 nodes that simulate 24 heat sensors, 10 valves, and 10 cooler fan nodes that run MQTT client. Once the MQTT clients started, each node will send a CONNECT message, and when the broker acknowledges the connection, a CONNACK message will be sent. The MQTT clients perform message transmission where the 24 heat sensors send publish messages that include PUBLISH and PUBACK every 1-s interval. Then, valve and cooler fan nodes subscribe to the temperature data from heat sensors using the SUBSCRIBE and SUBACK messages at each 1-s interval. Valve and cooler fan nodes will also send publish messages at 1 s interval to report their running status. The message transmission log is collected for 100 min. The message log for normal models is collected for 5 cycles with 100 min of running time each. Then, we collected message logs for 3 types of observed models where each has a failed sensor reading and delay occurring probability at each 10%, 5%, and 3%

embedded into the sensor and actuator program. Failed sensor readings are set to skip the publish and subscribe message based on the given probability. Delay is simulated by setting the increasing of message intervals for publishing and subscribe between 1 to 5 s randomly. Based on the given probability value, all the sensors and network have the same probability of error occurrences for each observed model. The network setting of the experiment is given in Table 1. We implemented the communication protocol using MQTT 5.0. The broker runs on Linux, which runs open IoT platform Contiki Cooja 3.0 on a workstation with 4-core CPU with 12 threads and 16GB RAM. The broker runs on Eclipse Mosquitto 1.6.9. Once the experiment started, the message transmission begins after 1 s in order to synchronize the cycle time for each node.

We perform process mining and anomaly detection for the evaluation on a workstation with 6-core (12-threads) 3.6 GHz processor and 16 GB RAM. Process mining is performed with process mining tool ProM 6 [27] using the IM plugin. The anomaly is detected using the implemented modified Levenshtein algorithm. The source code written in Python is available at Github [32]. The process tree similarity computation program utilizes multi-processing library Ray [33] in order to reduce the computation time. The computation time for each observed model to be computed took around 10 s to 12 s of process tree between 100 to 200 nodes.

Table 1. Network setting for the experiment.

Network Protocol	**MQTT vs. 5.0 (TCP/IP)**
Broker	HP ProLiant ML150 G6 4-core 2.5 GHz and 16 GB RAM
Operating System	Linux Ubuntu Desktop 19.10 with Contiki Cooja 3.0
MQTT Broker	Eclipse Mosquitto 1.6.9
MQTT Client	MQTT Clients 1.4.12
Node Startup Delay	1000 ms

We performed process mining of normal models obtained from the experiment with a large number of parallel events (110 events with 40 AND and XOR constructs). We extracted 5 batches of normal models for the event stream duration of 100 min and performed 1000 cross-checks (10×100) between the models at each 1 min of system cycle. We extracted 5 batches of normal models for the event stream duration of 100 min and performed a cross-check between the models.

3.2. Reliability of Normal Process Tree Models

Figure 12 shows the cross similarity check between 5 normal models $A1$, $A2$, $A3$, $A4$, and $A5$. The process mining result shows that no completely the same models can be extracted every time. Therefore, we cannot set a fixed threshold for detecting anomalies. To set the threshold, we standardize the similarity value by calculating the mean and 95% confidence interval to set the lower bound and upper bound of normal models' similarity. The standardized values are shown in Figure 13. From the sample of normal models between 1 min and 8 min, we found that the models become more similar over time (if they contain no anomaly). This is because it accumulates more events. The longer the time taken to accumulate event log, we can obtain more stable models depending on the scalability. However, we cannot ignore the noise. For larger systems with a lot of parallel events, the process mining will produce more normal process models that are less similar.

The standard deviation using the lower bound and upper bound of the cross-similarity between 1 min and 100 min is shown in Figure 13. The standard deviation of less than 0.01 after 3 min shows that the models became more stable over time. However, no perfect similarity is achieved between the models. It shows that every time there are minor parts that cause the process tree to be different due to a large number of parallel events. After 3 min, the amount of noise only changed between 27% and 28%. Therefore, by considering the similarity that is less than 73% to 74%, we can detect the anomaly.

We embedded the continuous anomalies into the event stream from minute 1 to 100. The strength of the abnormality is decided by the probability of occurrences between 10%, 5%, and 3% at each trace. The observed model is compared with the 5 normal models. We cannot decide a specific threshold value since the model changes every time. Therefore, we standardize the cross similarity value to decide a dynamic threshold. We calculated the Z-score of each interval and obtained the normal threshold of $-1.2 < h < 1.2$.

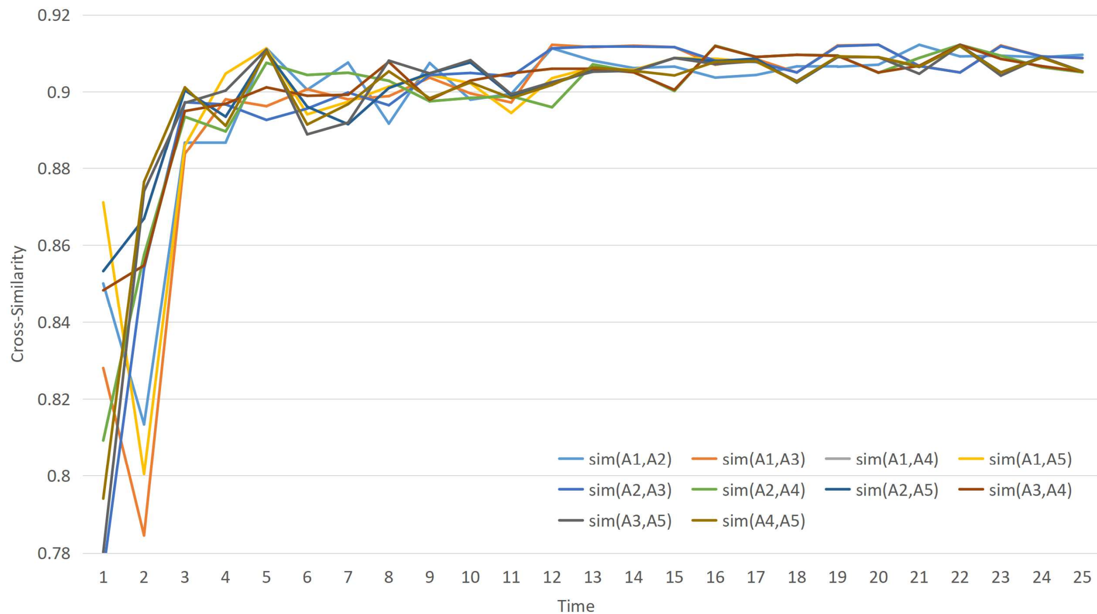


Figure 12. Cross Similarity of all normal process trees.

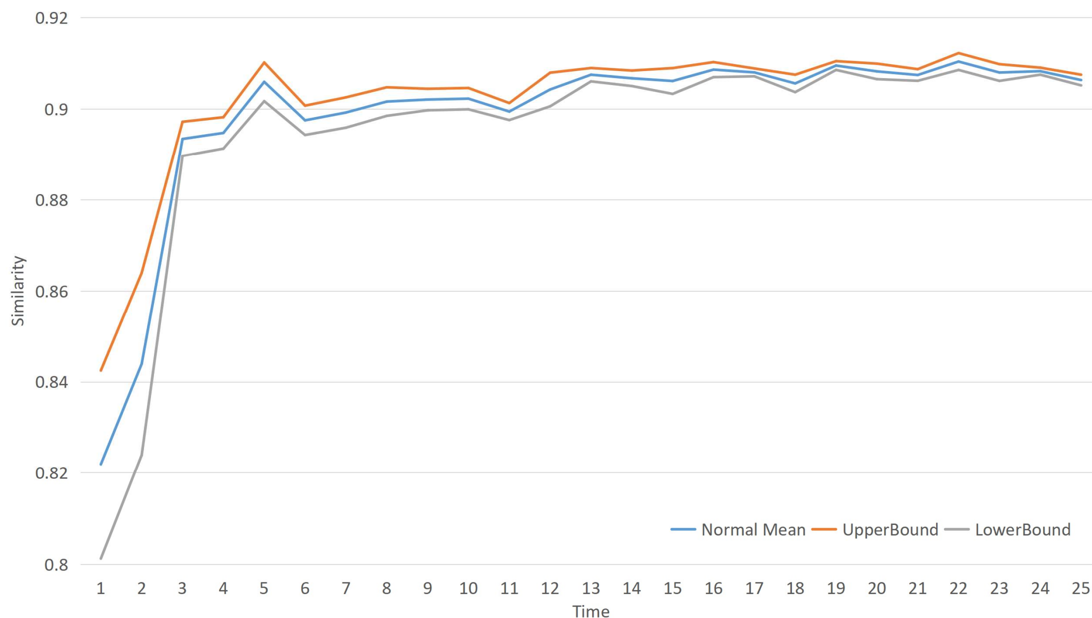


Figure 13. Mean of similarity of all normal process trees with lower bound and upper bound at a 95% confidence interval.

3.3. Detection Capability

For the detection, we perform a test to the following anomalies:

1. Missing Event Episode: Events that are missing from traces, i.e., change of $< abcb >$ to $< a_c_ >$. In real situations, the anomaly is due to a broken sensor or dropped message packet.

2. Perturbed Event Order: Event order are not in normal order i.e., change of $< abc >$ to $< acb >$. This anomaly might be caused by delay in MQTT message passing or invalid control-flow execution.
3. Mixed: Anomaly that contains both of the above.

4. Discussion

Figure 14 shows the detection result of anomaly with occurrence probability at 10%. We can see major deviations from the threshold value at first detection (3 min). All anomalies deviate from normal behavior. Even at 10%, anomaly probability has a strong influence on system behavior. Therefore, if an anomaly occurs more than 10%, we can easily detect the changes in behavior. Figure 15 shows the detection result of the anomaly with occurrence probability at 5%. We can see minor deviations from the normalized threshold value at first detection (3 min). The perturbed event order shows an obvious deviation. A mixed anomaly is mostly near the lower bound and was detected after 8 min. Figure 16 shows the detection result of the anomaly with occurrence probability at 3%. We can see minor deviations from normal threshold value at first detection (3 min) of system operation (for perturbed event and mixed anomaly). The first detection of a missing event is slower (15 min) compared to other anomalies. Overall, we successfully detected the three types of anomalies at a different time interval.

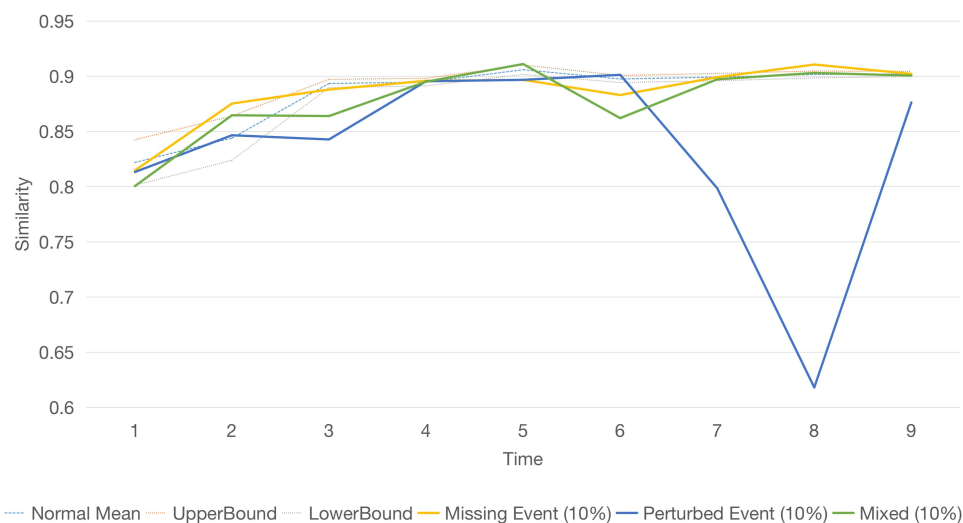


Figure 14. Detection of an anomaly with 10% occurrence probability. Anomaly occurrence probability more than 10% will show obvious deviation in the Z-score, especially for mixed anomaly.

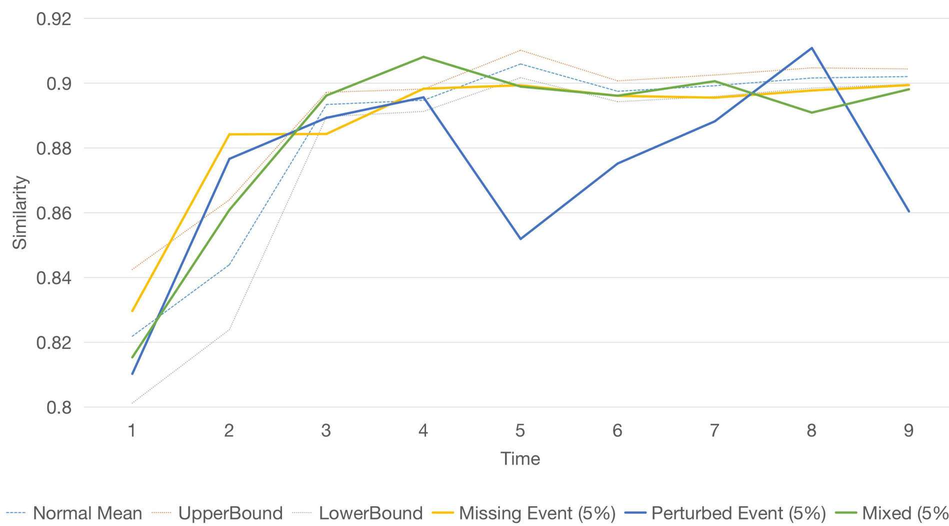


Figure 15. Detection of an anomaly with 5% occurrence probability. Perturbed event order shows obvious deviation compared to other anomalies. However, mixed anomaly was detected a bit slower compared to other anomalies.

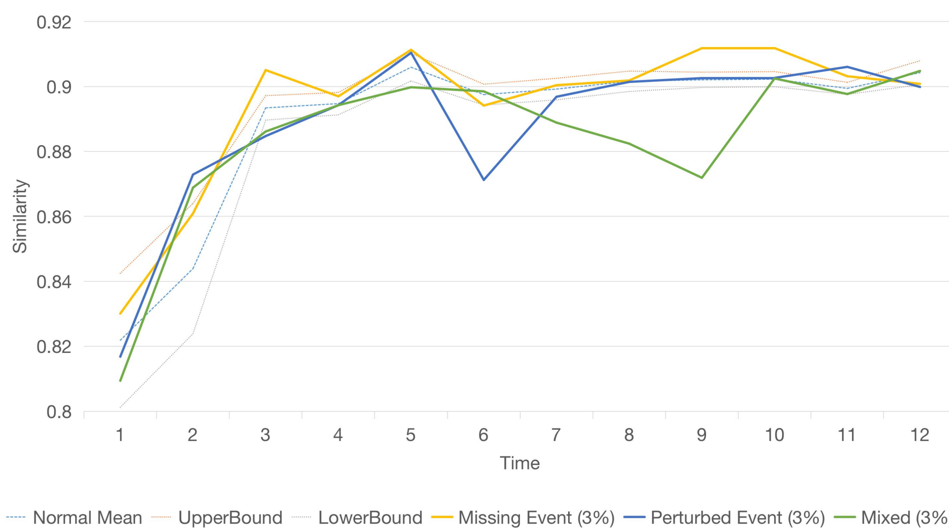


Figure 16. Detection of an anomaly with 3% occurrence probability. First detection of missing event is slow as it shows less deviation compared to other anomalies. However, it was successfully detected after 15 min.

We concluded our result in Table 2. In the case of minor anomaly, the proposed approach performs better at detecting perturbed events compared to the missing event. A mixed event is between those two because two different anomalies interfere with each other. This is because process mining extracts the relations based on the existence of the events. We found that missing events that occur in exclusive-choice relations are harder to detect. Meanwhile, perturbed events are events with order disorientation. It is easier to detect if the anomaly exists under the ordered operator. For mixed anomaly, if the occurrence probability is higher, we can see obvious deviations. Sometimes, mixed anomaly might cause false positive because of the interference between the execution relations.

Table 2. Summary of detection capability.

Anomaly Type	Deviation Range	First Detection (3%)	First Detection (5%)	First Detection (10%)
Missing Event	Low (10–20%)	15 min	3 min	3 min
Perturbed Order	High (>60%)	3 min	3 min	3 min
Mixed	Medium (20–60%)	3 min	8 min	3 min

5. Related Work

Several intrusion detection techniques were surveyed by Mitchell et al. [34]. DDoS and Malware attacks can be detected with behavior-based methods by monitoring the bandwidth and packets sent on the network. Other recent methods are monitoring the n-grams models [16], training the anomaly-model with machine learning such as Recurrent Neural Networks (RNN) [17] and Logistic Regression Analysis (LRA) [18]. However, the proposed behavior-based methods by previous research also have some drawbacks that can result in an increase in false-positive detection when dealing with continuous and multivariate data. Moreover, machine learning methods are black-boxed. Therefore, we cannot perform a direct analysis to the detected anomaly.

Haripriya et al. [19] proposed a dynamic fuzzy-based intrusion detection scheme to detect DoS attacks in the MQTT-based network. However, the machine learning technique is black-boxed, and it is hard to find the cause of detection. Myers et al. [20] proposed a process-based detection method using process mining and conformance checking. Kiyoshi et al. [21] from NEC Corporation introduced an invariant analysis that takes normal behavior and performs a detection that will produce alerts when abnormal behavior is found. The method checks for the abnormal spike of data values and produces alerts than exceeds a certain threshold. The methods proposed by Myers et al. and Kiyoshi et al. use the expected (normal) model, which are assumed to have no unpredicted fluctuation all the time. If a normal model is unstable (which contains some unpredicted behavior due to high parallelism of message passing), the detection will output false-positive results. Ushida [23] stated that two properties, such as fluctuations in sensor readings and time delays, are common random components that affect the stability of a control system. Even in a closed-loop control system, these properties exist.

Myers et al. used process mining to extract process model called Petri net from an event log. Petri net can represent the execution order of actions. It can also be extracted with IM. IM can ensure the perfect fitness of the discovered Petri net where the structure of Petri net is equivalent to a process tree. The method proposed by Myers et al. used expected models that required detailed knowledge of how the system operates. Then, assuming the expected model is perfectly designed, it is used as a standard model to be compared with the observed models. As stated in Ref. [20], it is hard to build the expected model without expert knowledge regarding the system. Moreover, systems with fluctuations and time delays will produce unpredictable message passing patterns. Even for a normal situation, we can expect some false positive alerts if fluctuations in sensor readings and time delays occur. Therefore, it is important to not only create a normal static model but a normal model that can handle changes in IoT systems. The proposed method applies the bird's eye monitoring technique that can grasp the whole execution sequence of message passing in an unstable closed-loop IoT system.

The literature in Ref. [35] proposed outlier detection in the multi-dimensional sensor in WSN. A Hidden Markov Model was used for model training and model-based likelihood estimation. The method can provide detection in real-time where if a data value from multiple sensors is detected, the outlier can be detected. This method is similar to Kiyoshi et al. [21] method, where the correlation between multivariate data is considered. The detection result does well in the outlier detection of multivariate sensor data. On the contrary, our proposed method focuses on the process where the sequence of the message transmission is taken into consideration. In our process-based detection, we need to accumulate the message passing log to detect if there is any deviation from the normal process model. We model the normal model by stacking the variations of normal models, which accumulates minor abnormality so that incidents can be detected. Our method is near real-time

because in the evaluation of the feasibility of detection, the first detection is made at least once the first cycle is completed. The detection and deviations of 3 types of anomalies were evaluated, where we found that delayed messages and dropped messages have a different level of deviation range.

6. Conclusions

In this paper, we proposed a process-based message-passing behavior verification method using process tree and process mining for lossy networks. While conventional verification methods focus on the data and a single point of event, the proposed method focuses on the accumulative event relation. Concretely, we focus on the relationship of the execution order of message passing between the devices. We performed a bird's eye monitoring technique by discovering the process model of the IoT system with process mining. Then, we grasp the structure of the process model with a tree model called a process tree. Using the process tree, we check for any deviations of normal behavior's process tree. We showed the feasibility of our verification method, which can detect minor deviations caused by infrequent dropped and delayed messages in the message passing events. As a result, we showed the feasibility of detecting abnormalities with the proposed verification method as low as 3% for an IoT system with low-power and lossy networks.

Author Contributions: Conceptualization, M.A.B.A. and S.Y.; methodology, M.A.B.A.; software, M.A.B.A.; validation, M.A.B.A.; formal analysis, M.A.B.A.; investigation, M.A.B.A.; resources, M.A.B.A.; data curation, M.A.B.A.; writing—original draft preparation, M.A.B.A.; writing—review and editing, M.A.B.A. and S.Y.; visualization, M.A.B.A.; supervision, S.Y.; project administration, S.Y.; funding acquisition, S.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Yamaguchi University Fund Research Grant Program for Young Scientists 2019 and partially supported by Interface Corporation, Japan.

Acknowledgments: The authors would like to thanks Graduate School of Sciences and Technology for Innovation, Yamaguchi University for providing facilities during the research. Thank you to Interface Corporation, Japan for providing the funding to this research since 2017.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

NCS	Network Control System
ICS	Industrial Control System
SCADA	Supervisory Control and Data Acquisition
DCS	Distributed Control Systems
IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
XMPP	Extensible Messaging and Presence Protocol
RESTFUL	Representational State Transfer
CoAP	Constrained Application Protocol
DDoS	Distributed Denial-of-Service
RNN	Recurrent Neural Network
LRA	Logistic Regression Analysis
IM	Inductive Miner
WSN	Wireless Sensor Network

References

1. Casado-Vara, R.; Vale, Z.; Prieto, J.; Corchado, J.M. Fault-Tolerant Temperature Control Algorithm for IoT Networks in Smart Buildings. *Energies* **2018**, *11*, 3430. [CrossRef]
2. Wain, A.; Reiff-Marganiec, S.; Jones, K.; Janicke, H. Towards a distributed runtime monitor for ICS/SCADA systems. In Proceedings of the 4th International Symposium for ICS & SCADA Cyber Security Research 2016 (ICS-CSR), Belfast, UK, 23–25 August 2016; pp. 1–10.

3. Delgado, R.; Park, J.; Choi, B.W. Open Embedded Real-time Controllers for Industrial Distributed Control Systems. *Electronics* **2019**, *8*, 223. [CrossRef]
4. Carli, R.; Cavone, G.; Ben Othman, S.; Dotoli, M. IoT Based Architecture for Model Predictive Control of HVAC Systems in Smart Buildings. *Sensors* **2020**, *20*, 781. [CrossRef] [PubMed]
5. Lee, E.; Seo, Y.-D.; Kim, Y.-G. Self-Adaptive Framework Based on MAPE Loop for Internet of Things. *Sensors* **2019**, *19*, 2996. [CrossRef] [PubMed]
6. Mallikarjuna, B. Feedback-Based Fuzzy Resource Management in IoT-Based-Cloud. *Int. J. Fog Comput.* **2020**, *3*, 1–21. [CrossRef]
7. Sato, K.; Kawamoto, Y.; Nishiyama, H.; Kato, N.; Shimizu, Y. A modeling technique utilizing feedback control theory for performance evaluation of IoT system in real-time. In Proceedings of the 2015 International Conference on Wireless Communications & Signal Processing (WCSP), Nanjing, China, 15–17 October 2015; pp. 1–5.
8. Lin, Y.-B.; Lin, Y.-W.; Lin, J.-Y.; Hung, H.-N. SensorTalk: An IoT Device Failure Detection and Calibration Mechanism for Smart Farming. *Sensors* **2019**, *19*, 4788. [CrossRef] [PubMed]
9. Eriş, Ç.; Güngör, V.Ç.; Bölük, P.S. Analysis of battery-powered sensor node lifetime for smart grid applications. In Proceedings of the 2016 24th Signal Processing and Communication Application Conference (SIU), Zonguldak, Turkey, 16–19 May 2016; pp. 2117–2120. [CrossRef]
10. Celaya-Echarri, M.; Azpilicueta, L.; López-Iturri, P.; Aguirre, E.; Falcone, F. Performance Evaluation and Interference Characterization of Wireless Sensor Networks for Complex High-Node Density Scenarios. *Sensors* **2019**, *19*, 3516. [CrossRef] [PubMed]
11. IoT developer Survey 2019 Results. Available online: https://iot.eclipse.org/community/iot-surveys/ (accessed on 21 April 2020).
12. MQTT Communication Protocol. Available online: https://mqtt.org/ (accessed on 21 April 2020).
13. XMPP Protocol. Available online: https://xmpp.org/ (accessed on 21 April 2020).
14. REST. Available online: https://restfulapi.net/ (accessed on 21 April 2020).
15. CoAP. Available online: https://coap.technology/ (accessed on 21 April 2020).
16. Wressnegger, C.; Schwenk, G.; Arp, D.; Rieck, K. A Close Look on n-Grams in Intrusion Detection: Anomaly Detection vs. Classification. In Proceedings of the ACM Conference on Computer and Communications Security, Berlin, Germany, 4–8 November 2013; pp. 67–76.
17. Goh, J.; Adepu, S.; Tan, M.; Lee, Z.S. Anomaly Detection in Cyber Physical Systems Using Recurrent Neural Networks. In Proceedings of the IEEE 18th International Symposium on High Assurance Systems Engineering (HASE), Singapore, 12–14 January 2017; pp. 140–145.
18. Noureen, S.S.; Bayne, S.B.; Shaffer, E.; Porschet, D.; Berman, M. Anomaly Detection in IoT System using Logistic Regression Analysis. In Proceedings of the 2019 IEEE TPEC, Anaheim, CA, USA, 17–21 March 2019; pp. 1–6.
19. Haripriya, A.P.; Kulothungan, K. Secure-MQTT: An efficient fuzzy logic-based approach to detect DoS attack in MQTT protocol for internet of things. *EURASIP J. Wirel. Comm. Netw.* **2019**, *2019*, 90.
20. Myers, D.; Suriadi, S.; Radke, K.; Foo, E. Anomaly Detection for Industrial Control Systems using Process Mining. *Comput. Secur.* **2018**, *78*, 103–125. [CrossRef]
21. Kiyoshi, K.; Hiroyuki, M.; Tomoya, S.; Mayumi, T. Big Data Analytics in the Cloud-System Invariant Analysis Technology Pierces the Anomaly. *NEC Tech. J.* **2015**, *9*, 85–89.
22. Gou, Z.; Ahmadon, M.A.B.; Yamaguchi, S.; Gupta, B.B. A Petri net-based framework of intrusion detection systems. In Proceedings of the IEEE 4th Global Conference on Consumer Electronics (GCCE), Osaka, Japan, 27–30 October 2015; pp. 579–583.
23. Ushida, S. Control performance improvements due to fluctuations in dynamics of stochastic control systems. In Proceedings of the 2011 50th IEEE Conference on Decision and Control and European Control Conference, Orlando, FL, USA, 12–15 December 2011; pp. 1430–1436.
24. Ahmadon, M.A.B.; Yamaguchi, S. State Number Calculation Problem of Workflow Nets. *IEICE Trans. Inf. Syst.* **2015**, *98*, 1128–1136. [CrossRef]
25. van der Aalst, W.M.P. On the Representational Bias in Process Mining. In Proceedings of the 2011 IEEE 20th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Paris, France, 27–29 June 2011; pp. 2–7.

26. Leemans, S.J.J.; Fahland, D.; van der Aalst, W.M.P. Discovering Block-Structured Process Models from Incomplete Event Logs. In *International Conference on Applications and Theory of Petri Nets and Concurrency*; Springer: Cham, Switzerland, 2014; pp. 91–110.
27. Process Mining Tool (ProM). Available online: http://promtools.org/ (accessed on 21 April 2020).
28. Gusfield, D. *Algorithms on Strings, Trees, and Sequences*; Cambridge University Press: Cambridge, UK, 1997; ISBN 0-521-58519-8.
29. Ghawi, R. Process Discovery using Inductive Miner and Decomposition. *arXiv* **2016**, arXiv:1610.07989.
30. Dijkman, R.; Gao, J.; Syamsiyah, A.; van Dongen, B.; Grefen, P.; ter Hofstede, A. Enabling efficient process mining on large data sets: Realizing an in-database process mining operator. *Distrib. Parallel Databases* **2020**, *38*, 227–253. [CrossRef]
31. Boroujeni, M.; Ehsani, S.; Ghodsi, M.; HajiAghayi, M.T.; Seddighin, S. Approximating Edit Distance in Truly Subquadratic Time: Quantum and MapReduce. In Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, USA, 7–10 January 2018.
32. Process Tree Based Anomaly Detection Program. Available online: https://github.com/anuaruddin/processs-tree-based-anomaly-detection (accessed on 4 April 2020).
33. Ray Multi-Processing Library. Available online: https://ray.readthedocs.io/en/latest/multiprocessing.html (accessed on 4 April 2020).
34. Mitchell, R.; Chen, I.R. A survey of intrusion detection techniques for cyber-physical Systems. *ACM Comput. Surv.* **2014**, *46*, 1–29. [CrossRef]
35. Wang, C.; Lin, H.; Jiang, H. Trajectory-based multi-dimensional outlier detection in wireless sensor networks using Hidden Markov Models. *Wirel. Netw.* **2014**, *20*, 2409–2418. [CrossRef]

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).