Environment Systems and Decisions (2022) 42:234–250
https://doi.org/10.1007/s10669-022-09859-x

Using log analytics and process mining to enable self-healing in the Internet of Things

Prasannjeet Singh[1] · Mehdi Saman Azari[1] · Francesco Vitale[2] · Francesco Flammini[1,3] · Nicola Mazzocca[2] · Mauro Caporuscio[1] · Johan Thornadtsson[4]

Accepted: 22 April 2022 / Published online: 17 May 2022

Abstract
The Internet of Things (IoT) is rapidly developing in diverse and critical applications such as environmental sensing and industrial control systems. IoT devices can be very heterogeneous in terms of hardware and software architectures, communication protocols, and/or manufacturers. Therefore, when those devices are connected together to build a complex system, detecting and fixing any anomalies can be very challenging. In this paper, we explore a relatively novel technique known as Process Mining, which—in combination with log-file analytics and machine learning—can support early diagnosis, prognosis, and subsequent automated repair to improve the resilience of IoT devices within possibly complex cyber-physical systems. Issues addressed in this paper include generation of consistent Event Logs and definition of a roadmap toward effective Process Discovery and Conformance Checking to support Self-Healing in IoT.

Keywords Self-repair · Self-diagnostics · Resilience · Data driven · Anomaly detection · Cyber-physical systems

1 Introduction

The Internet of Things (IoT) is emerging as a paradigm capable of pushing seamless interconnection of personal and industrial electronic devices among themselves and to online cloud services. This phenomenon includes home automation, environmental sensing as well as most industries such as heavy machinery, automobiles, etc., where each device is connected to monitoring and actuation systems. The IoT can be hence described as a system (of systems) that includes the collection of both tangible and intangible (i.e., Cyber-Physical) components that are interconnected, either wireless or wired. The whole system is usually provided with a connection to the Internet, including cloud-based services. The components may include a range of sensors which can use various types of local area connections and wide area connectivity, and can record environmental, human or other machine-based events and activities, which can then be relayed to other components, such as data collection, information processing and event correlation systems. All those information can be used to support decisions in order to operate the system effectively, to gather data, to troubleshoot problems in the system, or to perform any other relevant tasks (Kramp et al. 2013). The deployment diagram of a typical IoT system is shown in Fig. 1. Multiple connectivity means between entities can be observed; some devices are connected wirelessly, whereas some have a wired connection. Moreover, the sub-systems belong to different manufacturers. However, all of them work together to form an orchestration and perform functions of different complexity and criticality, in which resilience properties can be essential (Flammini 2019).

Similar to Fig. 1, most of the components in an IoT system are manufactured by different vendors. Although many failures *within* an individual component may be taken care of by their respective manufacturers, it is unlikely that any other failure happening due to interactions *between* different components in an IoT system, such as a simple connectivity problem, will be considered by the manufacturers when

✉ Francesco Flammini
francesco.flammini@mdu.se

[1] Department of Computer Science and Media Technology, Linnaeus University, Växjö, Sweden

[2] Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Naples, Italy

[3] School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden

[4] Information, Sigma Technology, Gothenburg, Sweden

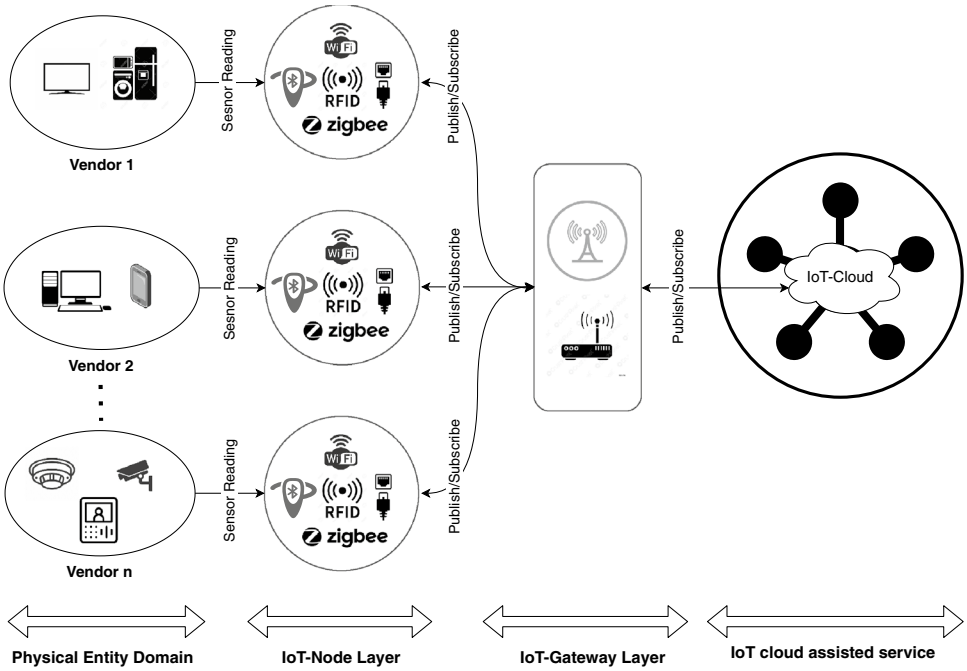


Fig. 1 A typical smart home deployment diagram

developing their own systems. This means that problems due to interactions between different devices are generally not covered under any warranty.

A popular report from Forrester (Pelino et al. 2018) predicts that there will be a huge growth in the IoT industry in the next years: enterprises will increase their efforts to introduce voice-based services to consumers and new European guidelines will allow commercializing the IoT data. In 2010, the number of objects connected to the Internet surpassed the earth's human population (Mohammadi et al. 2015). Reports from Gartner[1] and Juniper Research[2] suggest that the worldwide spending in IoT Security was 1.1 Billion USD and it skyrocketed more than 3.1 Billion USD in 2021. Furthermore, it is projected to shoot up to $6 Billion in 2023, which is a direct result of the increase in vulnerabilities in the systems. The above discussion leads us to believe that diagnosing and troubleshooting failures in an IoT system will be a huge problem in the near future, unless all the entities within the system belong to the same company.

IoT applications operate in complex distributed systems. One of the hardest challenges when such systems are deployed is their maintenance (Coulouris et al. 2011). While performing offline testing for verification and validation of these systems is possible using a wide range of unit and integration testing (Sommerville 2016), including some efficient approaches known as abstract testing (Flammini et al. 2009), run-time maintenance is possible only by continuously monitoring these systems. For instance, Caporuscio et al. (2020) introduce the concept of smart-troubleshooting in which system life-cycle is decomposed in four sequential activities, consisting of Prevention, Detection, Recovery and Adaptation. All of these activities can be performed for run-time maintenance of IoT systems.

When a system is capable of autonomously detect anomalies and recover from them, it is said it possesses the so-called Self-healing capabilities. Self-healing refers to the automatic recovery process by detection and diagnosis of faults and their subsequent correction in a temporary or permanent manner. Self-healing systems are of particular interest because Self-Healing directly impacts improvements in dependability. Self-adaptive systems are ones that monitor their execution environment and react to changes by modifying their behavior in order to maintain an appropriate quality of service. Therefore, there is a substantial intersection between Self-healing and Self-adaptiveness. Self-healing systems can be considered as a specific kind of Self-adaptive systems[3].

When performing run-time maintenance, the primary goal is to detect errors in the system. There are many ways

[1] W. Bamiduro, R. van der Meulen, Worldwide IoT security spending will reach $1.5 billion in 2018, Gartner (2018). https://gtnr.it/2Kd0hgl.

[2] Juniper Research, IOT SECURITY SPEND TO REACH $6 BILLION BY 2023, GROWING 300% FROM 2018, 2018. https://www.juniperresearch.com/press/iot-security-spend-to-reach-6-billion-by-2023.

[3] Artur Andrzejak, Kurt Geihs, Onn Shehory, John Wilkes - Self-Healing and Self-Adaptive Systems Dagstuhl Seminar 09201. https://www.dagstuhl.de/en/program/calendar/semhp/?semnr=13511.

to detect if an error has occurred in a system (Silva 2008). One of them is System-Level Monitoring; this technique is used by several commercial enterprise applications. Another notable method is to detect errors/failures at the application layer. However, one of the most common way to examine a system for detecting errors or failures is by analyzing the logs generated by the system. Logs may be produced either by collecting raw sensory data or by tracing activities performed during system operation. Considering this last option, there is an emerging research field that analyzes the traced activities in the log (which is also referred to as Event Log) to perform different kinds of activities: Process Mining. In this paper, we focus on the idea of analyzing Event Logs for Self-healing of IoT by automatically diagnosing and troubleshooting errors and failures in IoT devices. The choice of the approaches addressed in this paper is supported by a recent survey of the state-of-the-art addressing open issues, challenges and opportunities in this research field (Caporuscio et al. 2020).

The organization of this paper is as follows: Employable techniques for Prevention, Detection, and error/failure Diagnosis and Recovery in IoT systems will be briefly surveyed in Sect. 2, together with the inspection of the state-of-the-art for error/failure detection. Specific methods for Event Log analysis using Process Mining will be addressed in Sect. 3. Finally, Sect. 4 will draw conclusions and hints about future developments.

2 Prevention, detection and diagnosis in IoT

Prevention, detection and diagnosis of anomalies play main roles in run-time maintenance of IoT systems. These activities can be autonomously performed when the environment where the application operates in is smart and can exploit highly-capable computing and storage resources. This is the case for cyber-physical systems (Caporuscio et al. 2020) (CPS). CPS integrate a high number of sensors, actuators, highly-capable computing and storage nodes. This integration is possible due to advancements in interconnecting heterogeneous nodes. This aggregation and interconnection of heterogeneous Things is, as stated before, commonly referred to as the IoT (Baheti and Gill 2011). Currently, IoT mostly refers to network interfacing and communication of physical objects, devices and peripherals with a central processing unit based on a cloud computing network. However, this centralized cloud structure restricts the feasible applications due to confidentiality and latency reasons. To address these issues, data processing and communication should be served closer to sensors and actuators themselves or should be moved back to the edge of the network. This shifts the environment the application operates in from the scalable and homogeneous cloud environment to a restricted heterogeneous edge network (Al-Fuqaha et al. 2015). Despite the increasing popularity of IoT, security and reliability challenges impose a notable impediment to pervasive adoption and application of these devices (Sfar et al. 2018). By growing the number of involved dynamic heterogeneous resources in a cloud framework, the reliability and security concerns become more critical (Thamilarasu and Chawla 2019). In other words, the high number of heterogeneous, resource-restricted (e.g., energy, power and memory), usually cheap and often unreliable components in the environment threaten reliability and security. Moreover, when resources keep varying, the manual management of configuration, healing, protection, and maintenance of these networks of devices is difficult even for experienced personnel. For this reason, another main challenge is the autonomous management of resources. This concern can be addressed by Autonomic Computing. Autonomic Computing (Psaier and Dustdar 2011) introduces a Self-* concept to address all the concerns like Self-configuration, Self-healing, Self-protection, and Self-optimization.

- *Self-configuration* The ability of the system to readjust itself on the fly.
- *Self-healing* The ability of the system to discover, diagnose, and react to disruptions.
- *Self-optimization* The ability of the system to maximize resource utilization to meet end-user needs.
- *Self-protection* The ability of the system to anticipate, detect, identify, and protect itself from attacks.

Self-Healing enables the system to detect errors and failures, and possibly recover from failures and continue to operate smoothly. A Self-healing system must detect errors and failures and take required actions to make sure that the failure does not impact the correctness of the system. There can be several reasons that can induce errors, and possibly failures, in systems, such as hardware faults, software faults and network faults. A Self-protecting system helps detecting intrusive treatment and take autonomous actions to protect itself against abnormal behavior. A Self-protecting system will be able to detect and protect its resources from both internal and external attacks. The principal requirements needed to design a Self-Protected system are the following (Chopra and Singh 2014):

- By defining its own operation, a Self-protecting system should be capable of discriminating legal behaviors from illegal behaviors. In other words, a Self-protected system should be capable to detect intrusions.
- After detecting the attack, the system should be able to react by blocking the attack or logging a warning.

– Prevent the Self-protected components from being compromised.

2.1 Fault-tolerant techniques for preventing, detecting, and diagnose anomalies

According to the definition of Chandola et al. (2009), Anomaly Detection is "the problem of finding patterns in data that do not conform to expected behavior". The non-conforming patterns are the so-called anomalies. Chandola et al. (2009) systematically approached how an Anomaly Detection technique can be implemented considering (1) the type of anomalies, e.g., software errors due to human-introduced bugs or hardware crashes, (2) the attribute of anomalies (point, contextual and collective anomalies), and (3) the research field where suitable Anomaly Detection techniques can be extracted and applied for detecting anomalies. For the sake of consistency in this paper, we will follow the taxonomy for dependable and secure computing as proposed by Avizienis et al. (2004): this taxonomy allows us to reason with the familiar terms *fault*, *error* and *failure*. Therefore, instead of using the general term "anomaly", we will address inconsistent system states as *errors* and the deviation from correct service operation as *failures*. This helps in determining one of the points Chandola et al. (2009) highlight when determining how to build an Anomaly Detection technique, i.e., defining the type of anomalies. Introducing faults in a system is inevitable: since the expected number of bugs in a system is proportional to its lines of code (Lipow 1982), it can be safely assumed that it is impossible to exhaustively list down all possible faults in a system. Therefore, more emphasis is given in developing *fault-tolerant* systems, rather than building *fault-less* systems. A system could be termed as *fault-tolerant* if it is able to prevent an *error* from turning into a *failure*. Here follows a brief literature review on fault-tolerant techniques for Preventing, Detecting and Diagnose errors and failures in IoT systems.

Su et al. (2014) propose a fault-tolerant mechanism which is implemented on a WuKong[4] middleware. Gia et al. (2015) propose a fault-tolerant architecture for healthcare IoT systems consisting of wireless sensor networks (WSN). Misra et al. (2012) propose a method for fault-tolerant routing in IoT. A fault-tolerant routing protocol is suggested for IoT systems based on mixed cross-layer and learning automata (LA). van der Kouwe (2016) proposes *fault-injection*, in his thesis, where artificial faults should be injected into a system to learn the behavior of a system with activated faults, possibly preventing system failure when recovery actions can be performed.

Cyber-Physical Systems, such as SCADA (Supervisory Control And Data Acquisition), are an appropriate example for the current state-of-the-art in fault-tolerant Internet of Things systems (Sajid et al. 2016). These systems are competent in offering flexibility, stability and fault tolerance. These systems, exploiting cloud computing services integrated with the Internet of Things, can be judged as Smart Industrial Systems which are predominantly employed in smart grids, smart transportation, eHealthcare and smart medical systems.

Intentional attacks are common threats in IoT. Thus, it is important to employ an IoT intrusion prevention mechanism to prevent those threats. Various methods of intrusion detection are discussed and a taxonomy to classify them is provided by Zarpelão et al. (2017). Bertino and Islam (2017) propose various guidelines that can prevent an IoT system from being compromised. Kasinathan et al. (2013) propose a DoS Detection Architecture for 6LoWPAN IoT systems using *Suricata*, an open-source intrusion detection system that detects and eliminates the attack using appropriate countermeasures before network operations are disrupted.

Another efficient way of preventing intrusions is the installation of Honeypot in a system. Likewise, Honeynets are an aggregation of Honeypots that are intended to imitate usual servers and network services (Provos and Holz 2007). In general, honeypots are essentially a technique of deception, where the defender purposely hoodwinks the attacker into acting in one's favor (Daniel and Herbig 2013).

Yu et al. (2015) reject the idea of Honeypots in an IoT setup due to non-scalability and dependency issues, La et al. (2016) consider the possibility and use game theory to analyze the situation where both the attacker and the defender try to deceive each other.

While virtual patching is a type of firewall often mentioned as a web application firewall (WAF), an IoT system also needs a fully-fledged firewall, as most of the embedded systems that are part of the IoT system have little to no security. A recent improvement on the traditional firewalls for IoT is the *Smart Firewall*, which, unlike traditional software-based firewalls, is a hardware-based device[5]. Gupta et al. (2017) have proposed an implementation for a firewall for Internet of Things using Raspberry Pi as a gateway.

[4] K.-J. Lin, D. Shih, Y.-C. Wang, J. Hsu, N. Reijers, G. Chou, B. Tsai, Wukong - intelligent middleware for internet-of-things (2016). https://newslabntu.github.io/wukong4iox/.

[5] K. Colburn, Smart firewalls protect the internet of things - which are the best? | wtop (2017). https://wtop.com/tech/2017/10/smart-firewalls-protect-internet-things-best/.

2.1.1 Data-driven intrusion, error and failure detection techniques for IoT

In the previous section a brief review on the possible approaches for building fault-tolerant IoT systems has been done. All fault-tolerant approaches make use of data-driven error and failure detection techniques: data collected from the deployed environment get inspected and classified according to the detection technique used. How data are regarded to as being anomalous, i.e., how it is shown whether there is an error in the state of the system or the system itself has failed, depends on the detection technique. Most Anomaly Detection techniques, i.e., techniques for error and failure detection, make use of machine learning (ML). ML is defined by Witten et al. (2011) as "the acquisition of structural description from examples." This description can be used to infer classification rules (or other types of unsupervised rules, e.g., clustering rules). These rules can be applied to run-time data for error and failure detection, as the following brief literature review explores.

It is obvious from the recent research that the application of ML for intrusion and fault detection of the IoT devices is growing quickly. The most applicable and famous conventional ML algorithms for IoT errors and failures detection are Decision tree, support vector machine (SVM), K-nearest neighbor, Bayes Classifier, Neural Networks (Nisioti et al. 2018).

Silva and Schukat (2014) used a KNN classifier to design an intrusion detection system based on the Modbus/TCP protocol. Even though the developed mechanisms could obtain acceptable performance to some extent, they were dedicated to particular protocols with high false positive rates (FPRs).

Anthi et al. (2018) proposed an intrusion detection system for the IoT. To achieve this purpose, different ML classifiers have been developed for properly detecting network scanning probing and simple forms of denial of service (DoS) attacks.

Pajouh et al. (2016) proposed an intrusion detection model for detecting suspicious behaviors with the two-tier classification module based on the Naive Bayes and the Certainty Factor version of K-Nearest Neighbor algorithms. The designed model was also able to identify malicious activities like User to Root (U2R) and Remote to Local (R2L) attacks.

Stewart et al. (2017) developed an adaptive intrusion detection system for fitting the dynamic architectures of SCADA systems. For this purpose, the authors used different OCSVM models to choose the most proper one for effectively detecting various attacks. Nevertheless, the proposed model, in spite of producing a high false alarm rate for detection, required a high amount of computational resources.

With the evolution of the Industrial IoT (IIoT) the number of interconnected devices in a network will grow drastically, which given rise to notable amount of data. In contrast to traditional IoT networks, in the Industrial IoT, both the volume and characteristics of the produced data is important. Accordingly, the Big Data produced by Industrial IoT network needs intelligent real-time processing system (Liang et al. 2020). However, on large-scale of Industrial IoT networks, traditional ML methods show low accuracy and less scalability for attack and fault detection. Toward solve this problem, deep learning (DL) can play a major role in developing an intelligent processing systems to handle such huge amount of data from Industrial IoT networks to detect and diagnose the fault and attack. Currently, the DL algorithms are broadly used and obtained great performance in detecting cyberattacks (Meidan et al. 2018). For instance, to improve the accuracy in detection tasks, in Nicolau and McDermott (2018) authors used Auto-Encoder approach to project the original data to a new latent representation space. Nonetheless, the performance of the DL approaches highly depends on the amount of training data. Typically, to achieve a high accuracy DL model, needed to have a huge amount of labeled training data (Nisioti et al. 2018). Furthermore, require supposing that both test datasets and training datasets are subject to the same distribution. Therefore, the DL approaches only perform well under the two main assumption, i.e., large training datasets and same data distributions of training datasets and test datasets (Wen et al. 2017). However, in real applications, distribution discrepancy usually exists between training and testing data, which leads to significant reduction in the DL performance. Particularly, in network security, different kinds of attacks such as zero-day attacks, could be appeared on a daily basis (Nicolau and McDermott 2018). As a result, the practical industrial IoT test datasets are usually various from the training datasets. For solving this problem, one way is collecting a huge amount of labeled training data from multiple industrial IoT devices under the different possible working condition. However, manual Labeling of large amount of data is very expensive and time-consuming process (Buczak and Guven 2015). Consequently, it could restrict the real application of DL algorithms in error, failure and attacks detection of IoT devices under the different scenarios.

3 Log analysis and process mining

Recall that Sect. 2 introduced the systematic approach Chandola et al. (2009) used for defining an Anomaly Detection technique. The previous Sections showed how existing research narrows the scope for the types of anomalies to discover, what characterizes these anomalies, and which techniques, from the ML field, can be employed to discover anomalies. In this Section we will focus on detecting

behavioral anomalies, e.g., security, software or hardware errors/failures, by employing techniques extracted from Process Mining. We will first start by considering how Events can be logged from the operating IoT system. Then we will focus on the available Process Mining activities that enable discovering process models and behavioral anomalies from Event Logs.

3.1 Logging

When possible, the code of the application can be instrumented by introducing explicit logging rules. These logging rules get triggered when specific events happen. In IoT systems, however, most of the times the logged data is low-level sensory data collected by the deployed IoT devices. In this work, we are interested in analyzing the activities performed by the communicating IoT devices. We will, therefore, first provide a brief literature review that propose frameworks for extracting activities from low-level sensory data. Afterward, supposing it is possible to explicitly log Events from the application, we will provide an overview on available techniques for instrumenting the code for logging Events.

3.1.1 Low-level sensory data connection to high-level activities

Connecting low-level sensory data to activities performed in the system is not a trivial task. However, lots of research work has been done on detecting behavioral anomalies inspecting Event Logs, i.e., logs that explicitly collect the activities performed by the system at run-time. Therefore, it is worthwhile to address the problem of connecting low-level sensory data to high-level activities for further analyses.

Seiger et al. (2020) try to provide a framework for connecting low-level sensory data retrieved from a Smart Factory IoT application to high-level activities tied to high-level behavioral models.

Bakar et al. (2016) also provided, among other things, a framework for extracting activities from low-level sensory data. This work explores the pipeline raw data go through for extracting activity labels. It first starts with the collection of raw data as a collection of samples. These samples are preprocessed and segmented for scoping the research for activities in a limited time-window. Significant features are then extracted from each time-windowed sample data. Finally, activities are extracted. The authors of the work review a number of techniques for extracting activities from data. These techniques can extract activities either using supervised approaches, which require having pre-existing labeled sample data, or using unsupervised approaches.

Suryadevara and Mukhopadhyay (2012) consider sensor networks attached to monitors that determine whether a certain appliance is turned on or not. The system processes the information and labels the events considering some rules. For example, if pressure is detected on a bed, and the pressure is applied between 9 p.m. and 6 a.m., the system logs this event as the householder being asleep.

Hemmer et al. (2020) propose, among other things, a framework for extracting activities from low-level sensory data. This framework is a pipeline that makes use of Data Normalization, clustering and Time-Splitting techniques to identify what the authors regard to as the states the system is in during its operation. The states, together with their timestamps, configure the entries of the resulting Event Log.

3.1.2 Event logging instrumentation

Traditional logging techniques in major applications are not structured and many of the errors logged by them do not lead to failures. While these errors might be useful in some cases, most of the times they only decrease the readability of error logs by humans. Furthermore, there are some instances where errors, which were directly responsible for the failure of a system, could not be captured by these logging techniques. Considering that we are only focused on errors that lead to a failure, these techniques lead to a lot of false positives and false negatives. Here we focus only on the effective errors that may evolve into failures.

Most of the logging patterns try to detect errors and log them by placing a line of code at the end of a block of instructions. These techniques may not be able to detect errors such as infinite loops, etc. Therefore, instead of working on the resultant error logs, Cinque et al. (2012) aim to develop a new logging mechanism which takes care of these issues and other such issues, e.g., false positives and false negatives, by monitoring changes in the control flow of the program, placing the logging instructions strategically in the code. Moreover, the proposed logging mechanism aims to detect only those errors which cause failure.

The Rule-Based Logging Mechanism A set of error-modes are established, based on the widely accepted taxonomy in the dependability area (Avizienis et al. 2004), such as Service Error (SER: Prevents an invoked service from reaching the exit point.), etc. Different types of code injections, called Login Rules (LR) in the form of events, are then introduced in the source code of the system. This helps in logging events happening in the system, such as, e.g., LR-1 (Service Start), LR-2 (Service End), HTB (Heartbeat), etc.

In Rule-Based logging, these Login Rules don't write the log-file directly but are processed dynamically by a framework known as LogBus. The above logging mechanism can be extended to work within an IoT system by introducing a new entity within the system, which can be in the form of a framework. We name it as the *LogMonitor* (Fig. 2). Contrary to traditional logging mechanisms where lines of

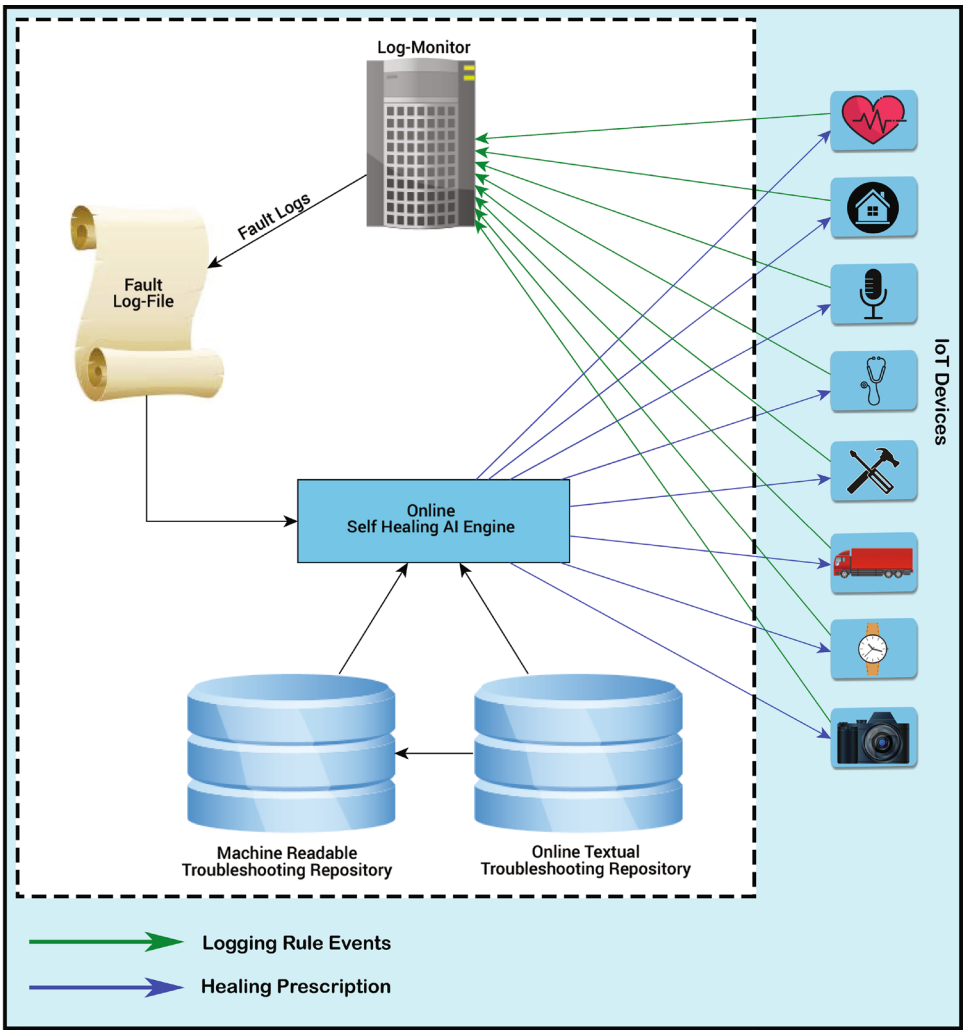


Fig. 2 Conceptual structure of a log-monitor

code written within the system update the error logs, the LogMonitor will monitor all the events spread over various IoT systems in an environment logged by the aforementioned logging rules, and eventually update the error log. This way a consistent error log can be generated for the entire system. However, this mechanism assumes that we have access to the source code of the software each entity within the IoT system hosts. Other tedious methods, such as parsing Event Logs (He et al. 2017; Du and Li 2016) from each entity, may have to be performed, in case internal access is unavailable.

3.2 Defining event logs

In the previous section, an approach was discussed to make an Event Log consistent in an IoT system. However, there still is a need to define the Event Logs and categorize them in such a way that appropriate tools can be developed, working on each category efficiently. Some Event Logs contain timestamps; some contain user-readable texts while others do not. Some contain just a few sets of information in the events, while some may contain hundreds of information in a single line of the Event Log. Although the format of the Event Logs depend on the manufacturer, in most of the standardized cases, an Event Log has an *Event ID*, a *Timestamp*, one or more primary attributes such as *Activity*, and other secondary attributes such as *Source*, *Server Name*, etc.[6] Thus, for a better understanding, let us assume a general Event Log example, that is readily comprehensible, as shown in Table 1. The given Event Log can represent most types of Event Log generated by a system, as it includes some identification (*Case ID and Event ID*), a *Timestamp* and, in this case, one primary attribute (*Activity*) and zero secondary attributes. The number of attributes for an Event Log is not constrained. However, these Event Logs are bound by some underlying assumptions (Van Der Aalst 2016). One

[6] IBM: Event Log Attributes. https://ibm.co/2yGbDE2.

Table 1 A sample event log

Case ID	Event ID	Time	Activity
1	85477	15-01-2019:14.05	Connection request received
1	85478	15-01-2019:14.06	SYN
1	85479	15-01-2019:14.06	SYN-ACK
1	85480	15-01-2019:14.08	Pin verification
1	85481	15-01-2019:14.08	Verification successful
1	85482	15-01-2019:14.08	ACK
1	85483	15-01-2019:14.09	Connection successful
2	92199	11-02-2019:09:11	Connection request received
2	92200	11-02-2019:09:12	SYN
2	92200	11-02-2019:09:12	SYN-ACK
2	92201	11-02-2019:09:13	Pin verification
2	92201	11-02-2019:09:15	Verification failed
2	92202	11-02-2019:09:16	Pin verification
2	92202	11-02-2019:09:17	Verification failed
2	92200	11-02-2019:09:18	Pin verification
2	92201	11-02-2019:09:18	Verification successful
2	92201	11-02-2019:09:20	ACK
2	92202	11-02-2019:09:21	Connection successful

important assumption among the aforesaid is that each event refers to some activity. For example, in Table 1, every event consists of an activity, such as pin verification, connection successful, etc. A system such as the *LogMonitor*, as discussed in the Sect. 3.1.2 above, could be programmed in such a way that it generates the Event Logs in the above format.

As can be observed, the Event Logs in Table 1 are initially divided into Cases. Each case is further divided into Event IDs. It is worth noting that events listed under a case should only relate to precisely one case. Moreover, all the events in a case should be chronologically ordered, and at least one attribute is mandatory for an Event Log to be valid.

3.3 Process mining

An IoT system can be expected to generate anywhere from a thousand to hundreds of thousands of Events Log instances in an hour, and as a result it is imperative to extract meaningful information from them that can help us troubleshoot any error. One of the ways through which this can be achieved is Process Mining. It is an excellent tool to capture meaningful information from Event Logs. It is mainly used to offer new medium to discover, monitor and improve processes in several application domains. However, it can also be used to analyze and rectify errors in a system. It is a comparatively new research discipline that tries to extract knowledge from Event Logs of real processes (i.e., not assumed processes) which are readily available in today's information systems. A report from Gartner estimated the market size of Process Mining to approach $160 Million in 2018 (Kerremans 2018). This is expected to grow to $1,421.7 million by 2023 with 50.3% as the Compound Annual Growth Rate.[7] The same report also predicts the growth of Process Mining usage in the IT industry (Fig. 3).

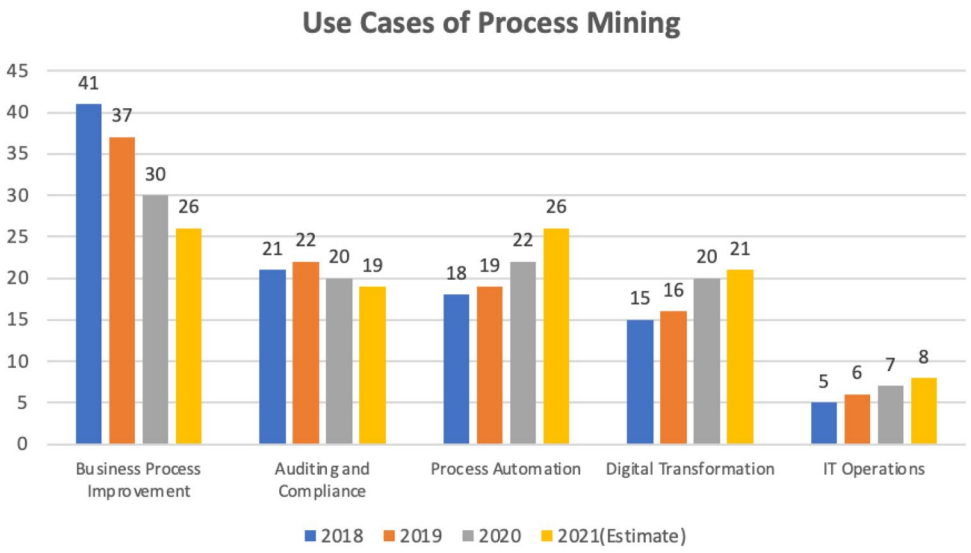


Fig. 3 Projected process mining use case in 2021

[7] Press-Release, Global process analytics market, 2023 by process mining type, deployment type, organization size, application, Tech. rep., MarketWatch (2018). https://on.mktw.net/2YafVlE

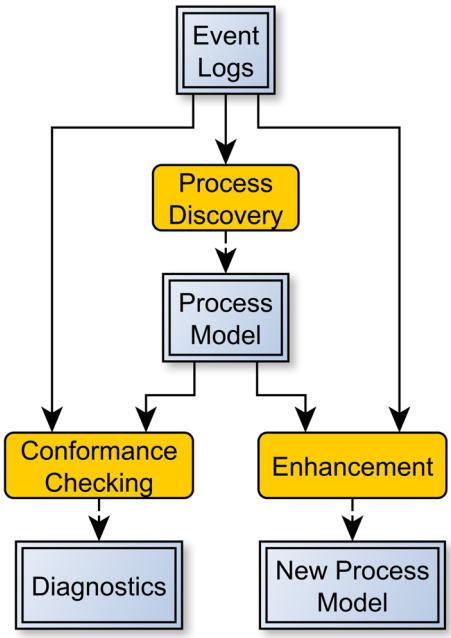


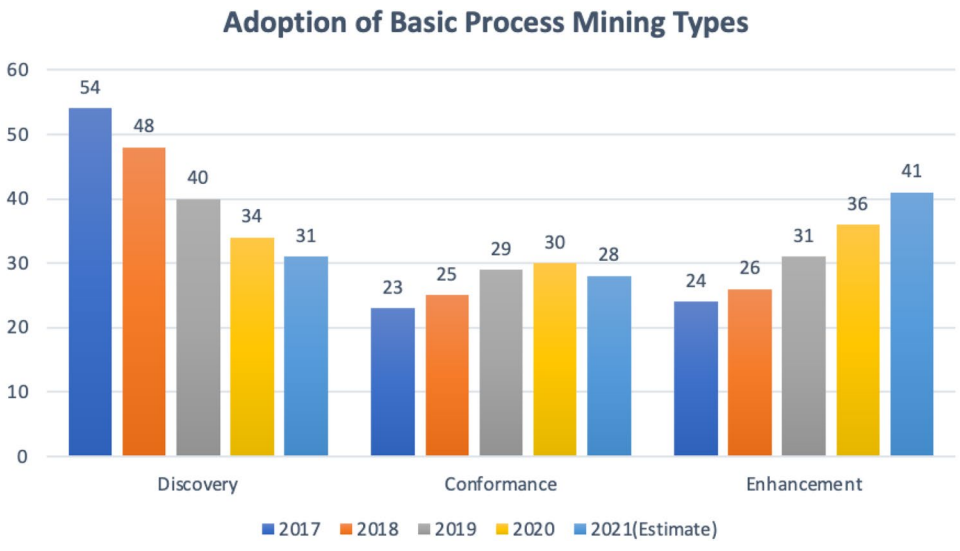
Fig. 4 Different types of process mining in terms of input and output

A process model generated by Process Mining using the Event Logs can be very effective in handling large amounts of data. Furthermore, many commercial and academic systems have implemented Process Mining algorithms. Vossen (2012) claims that an active group of researchers are working on Process Mining and it has become one of the "hot topics" in Business Process Management (BPM) research. Additionally, Process Mining functionality is added by more and more software vendors to their tools. Notable examples are: Discovery Analyst (StereoLOGIC), ARIS Process Performance Manager (Software AG), and Comprehend (Open Connect). Aforementioned reasons led us to believe that Process Mining can be an effective tool for solving the problem of Self-Healing in IoT systems. Besides this, as of July, 2015, there were 18 PhD students being funded by Philips who were working in analyzing the Philips X-ray machines and electronic shaving devices among other items to see how these machines are used in the field, when and why they fail, etc.[8]Moreover, most of the X-ray machines manufactured by Philips use Process Mining for fault diagnosis (Van Der Aalst 2016; Vossen 2012).

Since Process Mining primarily deals with Event Logs, it is assumed that the Event Logs used are in accordance with the guidelines mentioned in Sect. 3.2. The Event Log is the starting point and can be used to conduct three types of Process Mining activities. We will briefly describe these three classes of activities below (Fig. 4).

- *Process discovery* The activities related to this class are concerned with discovering process models from Event Logs through use of Process Discovery algorithms, such as the α-algorithm or the Inductive Miner (Van Der Aalst 2016).
- *Conformance checking* This class of activities are concerned with comparing the activities found in the Event Log with normative process models. These normative process models may have been developed at design-time or may have been discovered through Process Discovery algorithms.

Fig. 5 Projected adoption of basic Process Mining types



[8] W. van der Aalst, Process mining data science in action (2015). https://www.youtube.com/watch?v=kIeLaNzw9hI &t=704s.

– *Enhancement* The activities of this class are focused on improving existing process models by recovering insights from available Event Logs.

While Process Discovery still remains the most popular type of Process Mining, its usage is constantly declining, whereas the usage of the other two, i.e., Conformance Checking and Enhancement, is increasing and is expected to gain even more popularity in the future as well (Kerremans 2018) (Fig. 5).

3.4 Process discovery

For creating a process model in Process Mining, we start from a bunch of behavior (Event Logs, in our case) and automatically construct the model based on the logs. This is because commonly, there are no appropriate existing models, or the models are flawed or incomplete. Reference (Rolland 1998) describes a process model as, roughly, *an anticipation of what the process will look like*.

A process model can be represented using various notations, and the fact that there are many notations available demonstrates the significance of process modeling. A simple Event Log, as shown in the Table 1, may be represented using an intuitive model. However, in case of an IoT system, or any electronic system whatsoever, the Event Logs are far more complex than the one illustrated. As a result, a legitimate, well-defined notation should be used to represent a process model. Moreover, process models for complicated systems are prone to many errors (Van Der Aalst 2016), some of which are here outlined:

– *Oversimplification of the model* Most models have a propensity to focus only on the desirable behavior, and in a way overlook the events that are less likely to happen. There are chances that these "overlooked" events cause most of the errors and thus, the model may not be helpful.
– *Incompetence in satisfactorily capturing human behavior* Simple and mundane processes involving human engagement are prone to modifications, as human behavior is unpredictable. These changes, although minor, should nevertheless be incorporated in a model, as these nonconformities by and large result in an eventual error.
– *Restricted or redundant abstraction level* It is important that an appropriate abstraction level is chosen based on the objective and the input data. There is a plausibility that the model is too abstract to answer a detailed question, or conversely, a model is too detailed, and thus redundant, to answer a simple question.

Manually specified process models are susceptible to these (and similar) errors. Moreover, a poorly designed model may

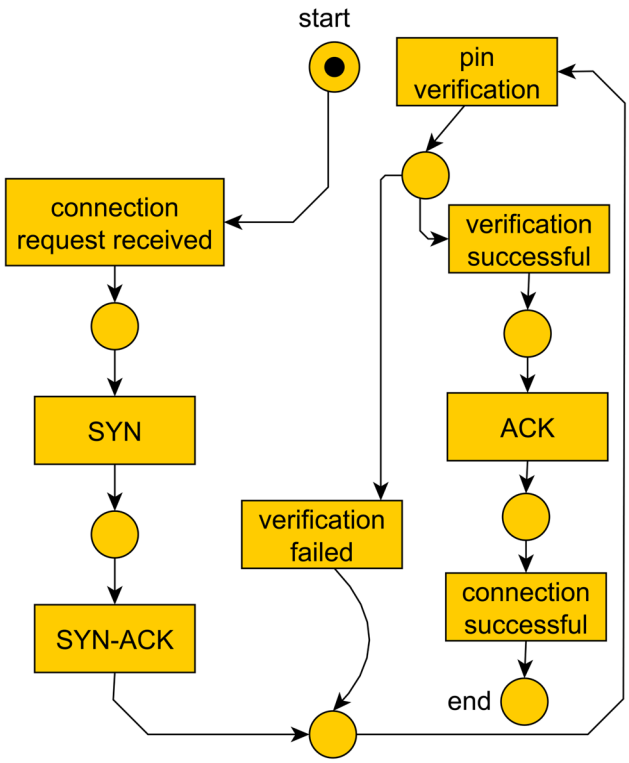


Fig. 6 A Petri net process model generated using α-algorithm for the sample event log displayed in Table 1

engender wrong conclusions. To eradicate these problems, Process Mining uses Event Logs to create a process model. An Event Log contains the exact steps that the system underwent to complete a task. Additionally, using a surfeit of Event Logs for modeling will result in the inclusion of all the events that might be less likely to happen, as discussed above. Finally, a process model is capable of providing different views using different abstraction levels for the same system.

Algorithms such as the α-algorithm (Van Der Aalst 2016) can automatically generate process models based on Event Logs, as will be discussed in future sections. Notations such as EPC's, BPMN, UML activity diagrams, Transition System, Workflow Nets, YAWL etc. can be used to describe process models. However, Petri nets are capable of describing concurrent processes without much effort like the transition systems. This is one of the oldest process modeling language. Due to its broad usage, it has also been extensively investigated; as a result, there exist many tools to analyze them[8] (Van Der Aalst 2016; Vossen 2012). A Petri net, as suitably described by authors in Manoj et al. (2012), is "a directed bipartite graph, in which the nodes represent transitions (i.e., events that may occur, signified by bars) and places (i.e., conditions, signified by circles)." Petri nets have a static network structure. *Tokens* flow through the network according to specific firing rules. The distribution of tokens

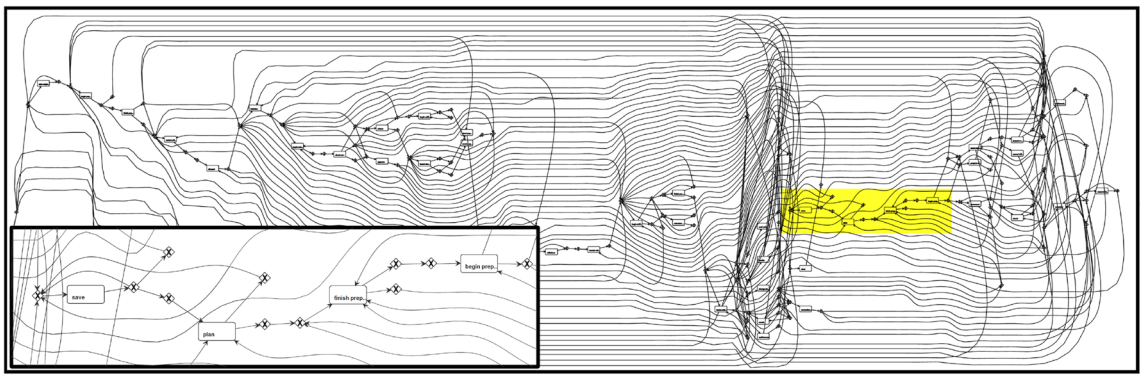


Fig. 7 A complex process model using Petri nets, and a zoomed-in version of the shaded region shown in the inset

over places (also referred to as *marking*) determines the state of the Petri net. Petri and Reisig (2008) provides a broad understanding of Petri nets. Furthermore, a *labeled Petri net* is an extension of the basic Petri nets, including a set of activity labels. Figure 6 is a Petri net model for the sample Event Log shown in Table 2.

There are various existing algorithms that convert Event Logs to a process model as above. One of the most prominent and rather naïve algorithm is the α-algorithm (Van Der Aalst 2016). Additionally, Fig. 7[9] is an example of a seemingly more complicated process model. It was created using ProM Tools. The inset in Fig. 7 shows a zoomed-in version of the shaded region of the process model in Fig. 7. Similarly, process models also contain detailed information, but relevant information may be available only in a particular section.

3.5 Conformance checking

Once a reference (normative) process model is in place, event data stored in an event log can be replayed on-top of the model for diagnosing whether reality aligns with the normative model's specifications. Therefore, conformance checking, i.e., event log replay, allows linking reality to process models. Specifically, among the opportunities conformance checking opens, we find (Van Der Aalst 2016):

- Assessment of the quality criteria of the normative model against the events found in the event log;
- Repair of normative process models exploiting insights provided by event data;
- Detection of control-flow anomalies due to, e.g., missed or wrongly-ordered activities executions.

Concerning the assessment of quality criteria, there are:

- Fitness
- Simplicity
- Generalization
- Precision

In the rest of the section, out of these criteria, we will focus on fitness, showing it provides useful insights on the process captured through event data.

3.5.1 Fitness assessment

Conformance checking techniques aim at highlighting whether event logs fit a normative process model. This diagnosis is provided through a quantitative metric, termed "fitness", whose value, a real number constrained between 0 (no fit) and 1 (best possible fit), depends on the specific technique used. Considering event data are organized in traces, where each trace refers to sets of event log entries with the same case ID, techniques can be classified according to their fitness analysis granularity. For instance, a trace-level technique may diagnose an event log does not fit even if only a single event in the whole trace does not conform to the expected control flow. Other techniques, such as token-based ones, work at a finer granularity, exploiting event-level information rather than whole traces.

In order to show the potential of finer-grained, fuzzier fitness evaluations, we will first introduce a popular conformance checking technique, Token Replay. Then, the framework used for performing experiments using this technique will be thoroughly described. Finally, a set of experiments will be designed and presented for evaluating

[9] This sample process model was created using the Event-Logs of BPI Challenge 2018 (https://data.4tu.nl/repository/uuid:3301445f-95e8-4ff0-98a4-901f1f204972). The log-file contains 2,514,266 instances of events that resulted in the process model as shown in the figure.

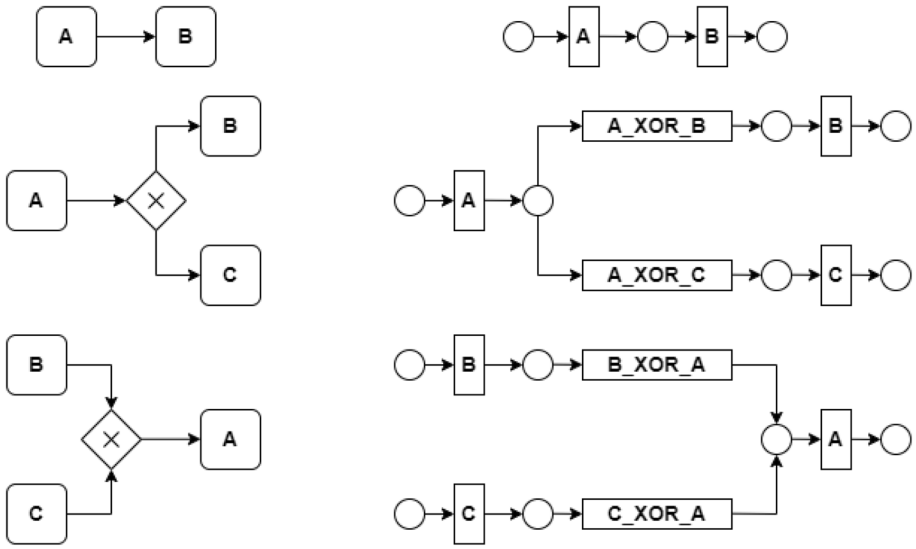


Fig. 8 Ruleset for precedence, XOR split, and XOR join BPMN activities relations

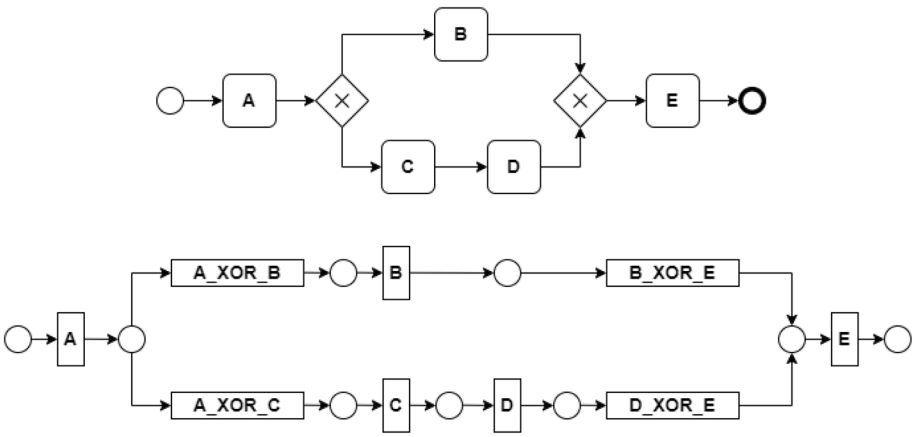


Fig. 9 BPMN model and its trace-equivalent Petri net counterpart

the usefulness of finer-grained diagnoses when different kinds of control-flow anomalies are found in event data.

Token Replay Token replay is a technique which replays event data on a normative process model. This technique is tailored for replaying traces on normative Petri nets, recording whether transitions found in event data are allowed to fire considering the state the process goes through as events are replayed on the model. When a transition, which is mapped to the activity the replayed event records, is fired without the needed requirements, this situation is marked as a deviation from the nominal behavior. As transitions are fired, four quantities are recorded and updated accordingly:

- p (produced tokens)
- c (consumed tokens)
- m (missing tokens)
- r (remaining tokens)

Once all traces in the event log get replayed, the event log fitness is computed with the following formula (Van Der Aalst 2016):

$$Fitness(L,N) = \frac{1}{2}\left(1 - \frac{\sum_{\sigma \in L} L(\sigma) \cdot m_{N,\sigma}}{\sum_{\sigma \in L} L(\sigma) \cdot c_{N,\sigma}}\right) + \frac{1}{2}\left(\frac{\sum_{\sigma \in L} L(\sigma) \cdot r_{N,\sigma}}{\sum_{\sigma \in L} L(\sigma) \cdot p_{N,\sigma}}\right) \quad (1)$$

where L is the event log, N is the normative Petri Net, and σ accounts for the traces that belong to L. When considering a single trace, the formula reduces to:

$$Fitness(\sigma,N) = \frac{1}{2}\left(1 - \frac{m_{N,\sigma}}{c_{N,\sigma}}\right) + \frac{1}{2}\left(1 - \frac{r_{N,\sigma}}{p_{N,\sigma}}\right) \quad (2)$$

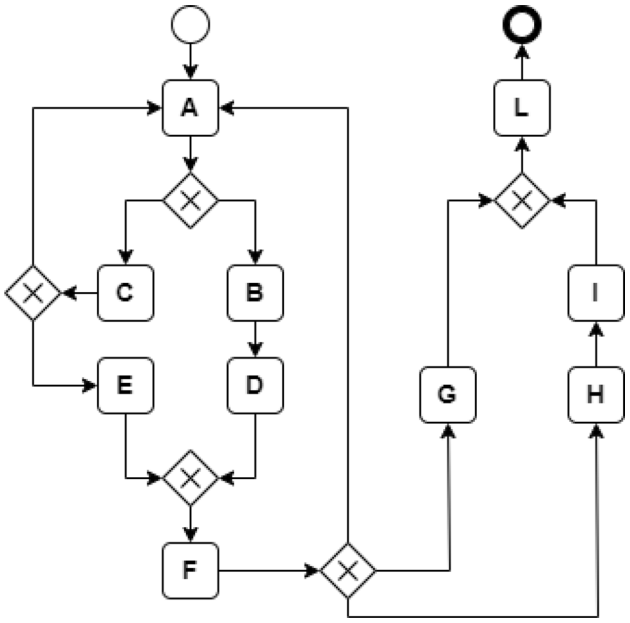


Fig. 10 BPMN reference model

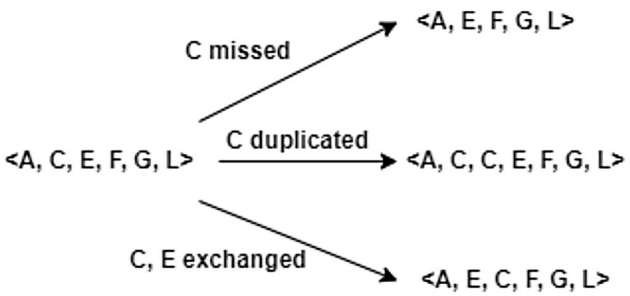


Fig. 11 Sample control-flow anomalies

Table 2 Flattened view showing part of the results of experiment 1

Batch n.	FMA	FDA	FEA
0	0.8664	0.8472	0.7694
1	0.8404	0.8244	0.8023
2	0.7941	0.8154	0.7813
3	0.7178	0.7951	0.7712
4	0.8584	0.8146	0.785
5	0.7838	0.7938	0.7268
6	0.6741	0.7756	0.726
7	0.8422	0.8166	0.7545
8	0.8039	0.799	0.79
9	0.7794	0.8104	0.7685

FMA fitness for missed activities, *FDA* fitness for duplicated activities, *FEA* fitness for exchanged activities

Table 3 Results of the statistical tests comparing fitness results with different injected anomaly types

Anomaly type 1	Anomaly type 2	$p > \|t\|$	Null hypothesis result
Missed activities	Duplicated activities	0.545	Accepted
Missed activities	Exchanged activities	< 0.001	Rejected
Duplicated activities	Exchanged activities	< 0.001	Rejected

4 Simulation and results

In the following, token replay will be shown to not only provide fitness measurements, but also other useful information for classifying control-flow anomalies.

4.1 Experiments' framework

As experiments will be performed comparing event data with normative process models through the token replay technique, we need the following:

(1) A normative process model described as a Petri net;
(2) Event data to be checked against the normative Petri net;
(3) A software implementing the token replay technique.

Regarding requirements 1. and 2., we re-used the open-source simulator PLG2 (Burattin 2016), which allows us simulating business process modeling notation (BPMN) models, and, therefore, generating event data. Considering two models are trace-equivalent if their set of execution sequences are equal (Van Der Aalst 2016), a BPMN model can be translated to a trace-equivalent Petri net if the correct translation rules for control-flow constructs of BPMN models are applied. BPMN-to-Petri net translation rules for precedence, XOR split, and XOR join BPMN activities relations are shown in Fig. 8; we will limit ourselves to normative BPMN models with only these activities relations, as this will allow us to generate a trace-equivalent Petri net for applying the token replay technique. Traces generated with PLG2 need to be translated from BPMN traces to Petri net traces. Consider, for instance, Fig. 9; here are a BPMN model (top) and its correspondent Petri net (bottom). The sample trace $< \texttt{A}, \texttt{C}, \texttt{D}, \texttt{E} >$ would be translated as $< \texttt{A}, \texttt{A\_XOR\_C}, \texttt{C}, \texttt{D}, \texttt{D\_XOR\_E}, \texttt{E} >$.

Regarding requirement 3., we will use an open-source implementation of the token replay technique developed by us and available at Vitale (2022). Each of the experiments

Table 4 Part of the results obtained when applying token replay to trace batches injected with anomaly types 0, 1, and 2 defined in experiment 2

Batch n.	Fitness	A	B	C	.	XOR_A_B	XOR_A_C	XOR_C_A	.	Anomaly
0	0.8577	0	0	0	.	15	15	0	.	0
1	0.7993	0	0	0	.	20	20	0	.	0
2	0.7197	0	0	0	.	22	22	0	.	0
3	0.603	2	0	0	.	24	24	15	.	1
4	0.5482	2	0	0	.	21	21	7	.	1
5	0.6082	3	0	0	.	27	27	13	.	1
6	0.8314	0	0	0	.	0	0	9	.	2
7	0.8416	0	0	0	.	0	0	5	.	2
8	0.77	0	0	0	.	0	0	6	.	2

that will be presented will use the normative BPMN model in Fig. 10.

4.2 Experiment 1

The goal of this experiment is evaluating whether different kinds of control-flow anomalies injected to traces simulated through PLG2 using the reference process model in Fig. 10 result in statistically significant fitness differences.

We will consider three types of control-flow anomalies: missed activities; duplicated activities; and exchanged activities. Figure 11 represents instances of such anomalies; considering the sample correct trace $< \mathtt{A}, \mathtt{C}, \mathtt{E}, \mathtt{F}, \mathtt{G}, \mathtt{L} >$, each of the three depicted cases show how such a trace changes when these anomalies are injected. Please note that multiple instances of such anomalies on a single trace can be injected. As a starting dataset, we have generated 1000 correct traces with PLG2. Out of these traces, we have split them into batches of 20 traces each, so as each batch is evaluated independently from one another, resulting in a total of 50 batches. The experiment is run three times, and, for each run, a certain number of control-flow anomalies instances of one of the types previously defined is injected to each batch. This process outputs three result tables of 50 rows each. The rows hold the batch evaluated, the fitness value computed using the token replay algorithm, and the number of injected instances of the type of control-flow anomaly (we have considered 5 for each experiment run). Table 2 shows a flattened view (limiting to 10 entries out of 50), collapsing fitness values of each of the three experiments; specifically, column FMA, FDA, and FEA record the fitness values computed when traces were injected with the missed activity anomaly, duplicated activity anomaly, and exchanged activity anomaly, respectively.

As each of the fitness values are computed for the same batches injected with different anomalies, a paired t-test for each possible pair of observations can be performed. Table 3 summarizes the tests results, determining the p-value for each of the tests performed and whether the null hypothesis was rejected or not. As it is clear from the table, the null hypothesis was rejected for both the comparisons against exchanged activities, meaning the results can discriminate with reasonable confidence whether the missed (duplicated) activities anomaly type or the exchanged activities anomaly type was injected.

4.3 Experiment 2

The goal of this experiment is showing process mining can be linked to classical machine learning techniques for classification tasks related to the traces handled through the token replay algorithm. Specifically, we will prove machine learning techniques are able to detect with satisfying accuracy figures whether a specific type of control-flow anomaly is present in the analyzed traces.

We will consider the same 1000 traces generated within experiment 1. However, we are going to split these traces in 200 batches of 5 traces each. Moreover, we are going to further split these batches in 3 parts, ending up with three sets of trace batches. To each of the set of batches, labeled 0, 1, and 2, a different anomaly type, using the same labels as the ones used for the sets, is injected. Specifically:

- To each trace of each batch of set 0 precedence anomalies are injected;
- To each trace of each batch of set 1 XOR split anomalies are injected;
- To each trace of each batch of set 2 XOR join anomalies are injected.

These types of anomalies slightly differ from the ones injected within experiment 1, as they are targeted at specific activities relations. Basically, for each trace, each time the trace has a precedence, XOR split, or XOR join relation between pairs of activities, these activities are exchanged in the trace, invalidating the relation and, therefore, causing a control-flow anomaly. Once anomalies are injected to the batches, they get processed using the token replay algorithm, which provides not only a fitness measurement, but also information about activities missing their input tokens

Table 5 Part of the results obtained when applying the K-Nearest Neighbor and C-Support Vector Classification machine learning algorithms to the copies of train and test datasets of experiment 2 *FFS_KNN* full feature space with K-nearest neighbor, *FFS_SVC* full feature space with support vector classification, *NXOR_KNN* no XOR information with K-nearest neighbor, *NXOR_SVC* no XOR information with support vector classification, *OF_KNN* only fitness information with K-nearest neighbor, *OF_SVC* only fitness information with support vector classification

FFS_KNN	FFS_SVC	NXOR_KNN	NXOR_SVC	OF_KNN	OF_SVC
1.0	1.0	0.8235	0.8235	0.6274	0.5686
1.0	1.0	0.8431	0.8627	0.5490	0.5882
1.0	1.0	0.8627	0.8039	0.6666	0.7254
1.0	1.0	0.7647	0.7647	0.6666	0.6470
1.0	1.0	0.9411	0.8823	0.5294	0.6666
1.0	1.0	0.7450	0.7843	0.6274	0.5686
0.9803	1.0	0.7843	0.7647	0.4705	0.6470

for firing when replaying traces out of the batches. Table 4 provides some of the results obtained when applying token replay to the anomalous trace batches.

Once the full dataset is available, the following routine is run 200 times:

(1) Split the full dataset into randomly sampled train and test sets (75% train, 25% test);
(2) Create two copies of the original sets, removing, in one case, all information about XOR transitions, and, in the other, all information about all transitions, leaving fitness values as the only information;
(3) Apply, for each copy of the (train, test) pairs, two machine learning algorithms for the classification of test instances: K-nearest neighbor (KNN) and C-support vector classification (C-SVC), both set with the default parameters provided by the implementations of these classifiers in the Python 3.8 scikit-learn 1.0.2 package;
(4) Evaluate the accuracy metric for each pair of sets and classifiers used.

Part of the results are shown in Table 5. Considering all the values for each of the columns of this table, we have that, on average, for the KNN machine learning algorithm, using the full feature space yields 99.96% accuracy, much higher than the averages in the case of no XOR information (81.34%) and only fitness information (59.56%). The same goes for C-SVC, with 100 % average accuracy using the full feature space (against the 82.23% and 62.92% accuracy figures with no XOR information and only fitness information, respectively).

4.4 Experiments' discussion

Out of the two experiments performed, it is clear that, though token replay represents one of the most basic finer-grained conformance checking techniques, the information provided by the application of such a technique provide useful insights about traces observed from a system. Specifically, experiment 1 showed that fitness values computed for trace batches injected with different types of control-flow anomalies may have statistically significant differences, whereas experiment 2 showed that fitness values, and related information about activities triggered without necessary conditions, provide useful results for the classification of different anomalies using widely-known machine learning algorithms, such as KNN and C-SVC.

5 Conclusion

Due to the huge interest and growth of IoT-based Cyber-Physical Systems in diverse smart-X applications where a certain degree of resilience is essential, in this paper we stressed the importance of methodologies, tools, technologies, procedures and frameworks supporting automatic threat (i.e., fault, error and failure) detection and—possibly—recovery. We have investigated the usage of Process Mining based on the analysis of IoT log files as a useful tool to support Self-diagnostics and Self-healing. We have shown the potential of those techniques and also highlighted some open issues to be addressed in future research. For instance, bisecting process models requires various protocols to be followed (Van Der Aalst 2016), which are not covered in this paper. However, once a robust process model is built, the following situations can be detected and managed through model analysis:

(1) Something did not happen although it was expected to happen.
(2) Something unexpected happened.
(3) The IoT system is operating according to its specification.
(4) Which is the most frequent path followed by the user, etc.

There are various tools available in the market to create a process model, e.g., ProM, Disco, Celonis, etc. Celik and Akçetin (2018) However, in order to create a process

model for a new system, a plugin may be needed so that the Event Logs can easily be comprehended by the tool. Note that Process Mining may also be capable of solving the problem of error predictability, where an end-user may be warned beforehand in case external events and/or user actions have a high probability of causing an error. Such analyses belongs to the categories known as prognostics, early warning and situation assessment. In conclusion, it can be said that generating consistent Event Logs throughout all the IoT devices in one environment by using a system such as the *LogMonitor* can be very effective for analyzing inter-device faults. Furthermore, generating process models using Process Mining, while adhering to the *Event Log Rules*, is an essential step toward Self-Healing in IoT-based CPS.

Funding Open access funding provided by Mälardalen University.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Al-Fuqaha A, Guizani M, Mohammadi M, Aledhari M, Ayyash M (2015) Internet of things: a survey on enabling technologies, protocols, and applications. IEEE Commun Surv Tutor 17(4):2347–2376

Anthi E, Williams L, Burnap P (2018) Pulse: an adaptive intrusion detection for the internet of things

Avizienis A, Laprie JC, Randell B, Landwehr C (2004) Basic concepts and taxonomy of dependable and secure computing. IEEE Trans Dependable Secure Comput 1(1):11–33

Baheti R, Gill H (2011) Cyber-physical systems. Impact Control Technol 12(1):161–166

Bakar U, Ghayvat H, Hasanm S, Mukhopadhyay SC (2016) Activity and anomaly detection in smart home: a survey. In: Mukhopadhyay S (ed) Next generation sensors and systems. Springer, Cham, pp 191–220

Bertino E, Islam N (2017) Botnets and internet of things security. Computer 50(2):76–79

Buczak AL, Guven E (2015) A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Commun Surv Tutor 18(2):1153–1176

Burattin A (2016) Plg2: Multiperspective process randomization with online and offline simulations. In: Online Proceedings of the BPM Demo Track 2016

Caporuscio M, Flammini F, Khakpour N, Singh P, Thornadtsson J (2020) Smart-troubleshooting connected devices: concept, challenges and opportunities. Futur Gener Comput Syst 111:681–697

Celik U, Akçetin E (2018) Process mining tools comparison. Online Acad J Inf Technol 9:97–104. https://doi.org/10.5824/1309-1581.2018.4.007.x

Chandola V, Banerjee A, Kumar V (2009) Anomaly detection: a survey. ACM Comput Surv (CSUR) 41(3):1–58

Chopra I, Singh M (2014) Shape–an approach for self-healing and self-protection in complex distributed networks. J Supercomput 67(2):585–613

Cinque M, Cotroneo D, Pecchia A (2012) Event logs for the analysis of software failures: a rule-based approach. IEEE Trans Softw Eng 39(6):806–821

Coulouris G, Dollimore J, Kindberg T, Blair G (2011) Distributed systems, concepts and design, 5th edn. Pearson, London

Daniel DC, Herbig KL (2013) Strategic military deception: Pergamon policy studies on security affairs. Elsevier, Amsterdam

Du M, Li F (2016) Spell: streaming parsing of system event logs. In: 2016 IEEE 16th International Conference on Data Mining (ICDM). IEEE, pp. 859–864

Flammini F (2019) Resilience of cyber-physical systems. Springer, Berlin

Flammini F, Mazzocca N, Orazzo A (2009) Automatic instantiation of abstract tests on specific configurations for large critical control systems. Softw Test Verif Reliab 19(2):91–110

Gia TN, Rahmani AM, Westerlund T, Liljeberg P, Tenhunen H (2015) Fault tolerant and scalable iot-based architecture for health monitoring. In: 2015 IEEE Sensors Applications Symposium (SAS). IEEE, pp. 1–6

Gupta N, Naik V, Sengupta S (2017) A firewall for internet of things. In: 2017 9th International Conference on Communication Systems and Networks (COMSNETS). IEEE, pp 411–412

He P, Zhu J, He S, Li J, Lyu MR (2017) Towards automated log parsing for large-scale log data analysis. IEEE Trans Dependable Secure Comput 15(6):931–944

Hemmer A, Badonnel R, Chrisment I (2020) A process mining approach for supporting iot predictive security. In: NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium. IEEE, pp 1–9

Kasinathan P, Pastrone C, Spirito MA, Vinkovits M (2013) Denial-of-service detection in 6lowpan based internet of things. In: 2013 IEEE 9th international conference on wireless and mobile computing, networking and communications (WiMob). IEEE, pp 600–607

Kerremans M (2018) Market guide for process mining. Gartner Inc, Stamford

Kramp T, Van Kranenburg R, Lange S (2013) Introduction to the internet of things. In: Bassi A, Bauer M (eds) Enabling things to talk. Springer, Berlin, pp 1–10

La QD, Quek TQ, Lee J, Jin S, Zhu H (2016) Deceptive attack and defense game in honeypot-enabled networks for the internet of things. IEEE Internet Things J 3(6):1025–1035

Liang F, Yu W, Liu X, Griffith D, Golmie N (2020) Toward edge-based deep learning in industrial internet of things. IEEE Internet Things J 7(5):4329–4341

Lipow M (1982) Number of faults per line of code. IEEE Trans Softw Eng 4:437–439

Manoj G, Immmanuel JS, Divya P, Haran A (2012) Modelling of system configuration and reconfiguration for ims. In: International Conference on Future Generation Communication and Networking. Springer, pp 285–292

Meidan Y, Bohadana M, Mathov Y, Mirsky Y, Shabtai A, Breitenbacher D, Elovici Y (2018) N-baiot-network-based detection of iot botnet attacks using deep autoencoders. IEEE Pervasive Comput 17(3):12–22

Misra S, Gupta A, Krishna PV, Agarwal H, Obaidat MS (2012) An adaptive learning approach for fault-tolerant routing in internet of things. In: 2012 IEEE Wireless Communications and Networking Conference (WCNC). IEEE, pp 815–819

Mohammadi M, Aledhari M, Al-Fuqaha A, Guizani M, Ayyash M (2015) Internet of things: a survey on enabling. IEEE, Piscataway

Nicolau M, McDermott J et al (2018) Learning neural representations for network anomaly detection. IEEE Trans Cybern 49(8):3074–3087

Nisioti A, Mylonas A, Yoo PD, Katos V (2018) From intrusion detection to attacker attribution: a comprehensive survey of unsupervised methods. IEEE Commun Surv Tutor 20(4):3369–3388

Pajouh HH, Javidan R, Khayami R, Dehghantanha A, Choo KKR (2016) A two-layer dimension reduction and two-tier classification model for anomaly-based intrusion detection in iot backbone networks. IEEE Trans Emerg Top Comput 7(2):314–323

Pelino M, Hammond J, Dai C, Miller P, Belissent J, Ask J, Fenwick N, Gillett F, Husson T, Maxim M, et al. (2018) Predictions 2018: Iot moves from experimentation to business scale

Petri CA, Reisig W (2008) Petri net. Scholarpedia 3(4):6477

Provos N, Holz T (2007) Virtual honeypots: from botnet tracking to intrusion detection. Pearson Education, London

Psaier H, Dustdar S (2011) A survey on self-healing systems: approaches and systems. Computing 91(1):43–73

Rolland C (1998) A comprehensive view of process engineering. In: International Conference on Advanced Information Systems Engineering. Springer, pp 1–24

Sajid A, Abbas H, Saleem K (2016) Cloud-assisted iot-based scada systems security: a review of the state of the art and future challenges. IEEE Access 4:1375–1384

Seiger R, Zerbato F, Burattin A, García-Bañuelos L, Weber B (2020) Towards iot-driven process event log generation for conformance checking in smart factories. In: 2020 IEEE 24th International Enterprise Distributed Object Computing Workshop (EDOCW). IEEE, pp 20–26

Sfar AR, Natalizio E, Challal Y, Chtourou Z (2018) A roadmap for security challenges in the internet of things. Digit Commun Netw 4(2):118–137

Silva LM (2008) Comparing error detection techniques for web applications: An experimental study. In: 2008 Seventh IEEE International Symposium on Network Computing and Applications. IEEE, pp 144–151

Silva P, Schukat M (2014) On the use of k-nn in intrusion detection for industrial control systems. In: Proceedings of The IT &T 13th International Conference on Information Technology and Telecommunication, Dublin, Ireland, pp 103–106

Sommerville I (2016) Software engineering, 10th edn. Pearson, London

Stewart B, Rosa L, Maglaras LA, Cruz TJ, Ferrag MA, Simoes P, Janicke H (2017) A novel intrusion detection mechanism for Scada systems which automatically adapts to network topology changes. EAI Endorsed Trans Ind Netw Intell Syst. https://doi.org/10.4108/eai.1-2-2017.152155

Su PH, Shih CS, Hsu JYJ, Lin KJ, Wang YC (2014) Decentralized fault tolerance mechanism for intelligent iot/m2m middleware. In: 2014 IEEE World Forum on Internet of Things (WF-IoT). IEEE, pp 45–50

Suryadevara NK, Mukhopadhyay SC (2012) Wireless sensor network based home monitoring system for wellness determination of elderly. IEEE Sens J 12(6):1965–1972

Thamilarasu G, Chawla S (2019) Towards deep-learning-driven intrusion detection for the internet of things. Sensors 19(9):1977

Van Der Aalst W (2016) Process mining: data science in action. Springer, Heidelberg

van der Kouwe E (2016) Improving software fault injection. Ph.D. thesis, Vrije Universiteit Amsterdam

Vitale F (2022) Tokenreplay. https://github.com/francescovitale/TokenReplay. Accessed 30 April 2022

Vossen G (2012) The process mining manifesto–an interview with wil Van Der Aalst. Inf Syst 37(3):288–290

Wen L, Gao L, Li X (2017) A new deep transfer learning based on sparse auto-encoder for fault diagnosis. IEEE Trans Syst Man Cybern Syst 49(1):136–144

Witten IH, Frank E, Hall M (2011) Data mining: practical machine learning tools and techniques, 3rd edn. Morgan Kaufmann, Amsterdam

Yu T, Sekar V, Seshan S, Agarwal Y, Xu C (2015) Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things. In: Proceedings of the 14th ACM Workshop on Hot Topics in Networks, pp 1–7

Zarpelão BB, Miani RS, Kawakani CT, de Alvarenga SC (2017) A survey of intrusion detection in internet of things. J Netw Comput Appl 84:25–37