Exploring Anomaly Detection in Systems of Systems

Author 1
Affiliation
Address
Address
mail

Author 2
Affiliation
Address
Address
mail

Author 3
Affiliation
Address
Address
mail

ABSTRACT

The loosely coupled integration of heterogeneous existing systems, together with the ongoing replacement of monolithic systems design with Off-The-Shelf (OTS) approaches, promotes a new architectural paradigm that is called System of Systems (SoS). In SoSs, independent and autonomous constituent systems (CSs) cooperate to achieve higher-level goals. Some inherent challenges are that boundaries of the SoS may be partially unknown and the components may be governed by different authorities, affecting the ability to observe the system as a whole. Further, novel challenges related to dependability and security are introduced, such as the detection of emerging and possibly unexpected behaviors resulting from the interconnection of previous disconnected CSs. In this paper we explore these challenges questioning if a novel mindset to error, malware or intrusion detection is needed when dealing with SoSs. With the support of a state of the art review, we first identify the design principles and the performance targets of a monitoring and anomaly detection framework. Then we discuss these principles at the light of SoS fundamentals. Ultimately, we propose an approach to design a monitoring and anomaly detection framework for SoSs aggregating i) monitoring approaches ii) SoS properties, and iii) anomaly detection techniques.

CCS Concepts

• **Security and privacy → Intrusion/anomaly detection and malware** • **Security and privacy → Distributed systems security** • Computer systems organization → Peer-to-peer architectures • Computer systems organization → Reliability

Keywords

Systems-of-Systems; Anomaly Detection; Monitoring;

1. INTRODUCTION

In recent years, the architectural paradigm of System of Systems (SoS) [28], [24], [31] have been continuously growing in popularity. Systems of Systems are built through the composition of both new and already existing Constituent Systems (CSs), which are independent and operable. The purpose of integrating such CSs is to provide new and enhanced services, not achievable by the single CS in isolation. This introduces a macro-level, at which the SoS operates, and micro-levels which distinguish the operation of the individual CS. *Emerging phenomena*, that are not visible at the micro-level, can happen at the macro-level: such phenomena may be unexpected and potentially detrimental for the SoS [25]. To explain this concept, we consider the interaction between two processes that ends in a deadlock i.e., a complete halt of the system that holds forever. The risk of a deadlock should not be considered at the *micro-level* (i.e., the level of the individual processes). Nevertheless, a causal dependency between the totality of the processes – the *macro-level* - and that lead to a deadlock can be observed. Generalizing these concepts, the combination of different and independent CSs may result in an SoS with emergent phenomena, that the individual CSs may ignore or may not be ready to manage [25]. Further, SoSs are characterized by properties as dynamicity, interoperability, evolvability.

To satisfy dependability and security requirements, it is thus evident that SoSs require solutions to perform error or attack detection despite the SoS properties discussed above. In other words, it is required to infer the status of the SoS at the macro-level through observing (part of) the CSs at the micro-level. Such ability should also cope with governance aspects involving CSs, which may be owned by third parties or may be OTS components. To achieve such goals, anomaly-based detection techniques [1], [17], [18] are a candidate solution. The main advantages of adopting anomaly-based detection techniques lie in their suitability for dynamic and complex systems. In fact, online anomaly detection techniques are able to adapt their behaviour depending on the current context of the system, without requiring huge periods of training [4], [17], [3]. Further, alternative solutions as fingerprint-based detection techniques are not suitable to identify unexpected behaviours that were not described in advance and that can result from the interoperation of CSs as explained above. This means that anomaly detection is very suitable for SoS, where dynamic sets of CSs collaborate to achieve various targets through time.

Looking at available solutions, it is noticeable that enterprise frameworks, which allow checking if the observed behaviour is normal or anomalous, exist [20]. In particular, enterprise solutions such as *Nagios* [21], *Ganglia* [22] or *Zenoss* [23] allow the user to setup both monitoring and data analysis strategies. However, these enterprise frameworks have common lacks that can impact their suitability for SoS. In particular, they i) do not allow executing sophisticated data analysis (e.g., anomaly detection techniques), while they always allow to setup static thresholds, ii) report the anomaly alerts as they happen without trying to correlate them, and iii) use a monitoring strategy that is not always suitable for the micro-macro level distinction we have in SoSs. Moreover, changes or updates at the application level call for a manual reconfiguration of such a monitoring system that is consequently not suitable for dynamic contexts.

Summarizing, the findings of the paper are the followings: i) identify the main design aspects behind a monitoring and anomaly detection framework, ii) explore frameworks for anomaly detection that tackle SoS-related challenges and iii) propose high-level guidelines for performing anomaly detection in SoSs.

The paper is organized as follows. Section 2 lists the state-of-the-art contributions regarding SoSs, while Section 3 motivates the study and traces the research direction that is expanded in the rest of the paper. Afterwards, we define the architectural (Section 4) and performance targets (Section 5). Section 6 tackles together SoSs and anomaly detection, ultimately defining design directions for a monitoring and anomaly detection framework for SoSs. Section 7 concludes the paper and explores future works.

2. BASICS ON SYSTEMS-OF-SYSTEMS

2.1 Definition and Classification

As remarked in [28], several definitions of SoS have been proposed in the literature according to real-world applications in different areas, including dependability [6] and security [10]. According to [31], we consider that "*an SoS is an integration of a finite number of Constituent Systems which are independent and operable, and which are networked together for a period of time to achieve a certain higher goal."* Constituent Systems (CSs) can be existing legacy systems or newly developed components, and they may include physical objects and humans: a CS is *an autonomous subsystem of an SoS, consisting of computer systems and possibly of controlled objects and/or human role players that interact to provide a given service* [41].

An SoS may have different degrees of control and coordination [32] identifying four categories, namely directed, acknowledged, collaborative and virtual. A *directed* SoS is managed by a central authority providing a clear objective to which each CS is subordinate; the CSs that form the SoS may operate independently, but they are subordinated to the central purpose. An *acknowledged* SoS has a clear objective but the CSs might be under their own control thus funding an authority in parallel with the SoS. In a *collaborative* SoS, the central management organization does not have coercive power and CSs act together to address shared common interests. Finally, a *virtual* SoS has no clear objective and its CSs do not even know one another.

The degree of control and coordinated management of the CSs that form the SoS is relatively tight in a directed SoS, but it gets looser as we move to the acknowledged, collaborative and finally virtual category. This will affect the monitoring approaches that we will discuss in Section 4.2.

2.2 Viewpoints for Dependable and Secure SoSs

The challenges posed to design, develop and maintain dependable and/or secure SoSs can be summarized as viewpoints [28], [29], [30] i.e., dimensions of analysis for such SoS. In particular, we will expand and focus on the viewpoints *architecture*, *dynamicity and evolution, emergence, governance*, *time*, *dependability and security*.

Architecture. The architecture of an SoS can be defined in terms of heterogeneous CSs interacting each other through cyber or physical channels. *Relied Upon Message Interfaces* (RUMIs) and *Relied Upon Physical Interfaces* (RUPIs) [37] establish the boundaries between interacting CSs and the roles for their interactions. RUMIs establish the cyber data that are exchanged and the timing of message exchange, while RUPIs enable the physical exchange of things or energy among CSs.

Architectures of dependable and secure applications can be characterized as mixed-criticality architectures, where different parts of the system have different dependability and security requirements. To cope with this issue, in [28] authors propose *architectural hybridization* [24], where different subsets of requirements are satisfied in different parts of the target system.

Evolution and Dynamicity. Dynamicity and evolution are two important challenges of SoS and they have effects on security and dependability requirements. *Dynamicity* refers to short-term changes of the SoS e.g., in response to environmental variations or components failures. *Evolution*, instead, refers to long-term changes that are required to accomplish variation to the requirements in face of an ever-changing environment [31], [28].

Emergence. An emergent phenomenon manifests when CSs act together, and the emergent phenomenon is not observable by looking at single CSs separately. For instance, if a crowd enters a narrow alley then it alters its movements, individuals reduce their pace in order to avoid hitting or getting to close to others in front. This collaborative behavior does not emerge if we consider individuals separately: this means that an SoS is not just the sum of its CSs. *Emergence* can be expected or unexpected, detrimental or non-detrimental [25]. Beneficial are for example self-organization and evolution of biological systems, while detrimental are for example traffic jams due to the interaction of single cars. Moreover, emergence can be expected or unexpected. In particular, detrimental unexpected emergent phenomena may expose vulnerabilities or lead to novel faults that are consequently difficult to tolerate [33].

Governance. Distributed ownership of individual components is a challenge for any complex system [27], which is usually an ensemble of existing systems, including third-party, OTS or more in general non-proprietary components. SoS governance is significantly more complicated and must change to accommodate the business requirements of an SoS.

Time. In a recent report from the GAO to the US Congress [39] it is noted that *a global notion of time is required in nearly all infrastructure SoSs,* such as telecommunication, transportation, energy, etc. In large cyber-physical SoSs the availability of a global sparse time is fundamental to reduce the cognitive complexity to understand, design and implement SoS [25]. However, CSs typically use unreliable clocks. With respect to monitoring, this may result in inconsistent timestamps in observed data, leading to misunderstandings or wrong interpretations. It is thus relevant that CSs shares a global view of time.

Dependability and Security. SoSs are composable systems, with a high degree of uncertainty on their boundaries. Since the environment may unpredictably change, or it may be so various becoming really hard to model, the whole monitoring and assessment process can be negatively affected. Monitoring an SoS means to devise adaptive monitors that are able to cope with several environments and a variable number of interacting CSs.

3. MOTIVATIONS AND RELATED WORKS

Summarizing, an SoS is not simply an ensemble of CSs: instead, CSs individually operating at a micro-level cooperate to provide new functionalities that *emerge* at a macro-level [25]. Critical SoS should avoid or mitigate detrimental emerging phenomena which can damage the whole system and the connected critical components. However, if unexpected, emerging phenomena

cannot be easily avoided or mitigated through the rules that we set using our knowledge of the SoS.

Considering the structure of the CSs, which includes physical objects and humans, it appears that observing all the internals of CSs to check their behavior may be not possible. Thus, the monitoring effort should be directed to Relied Upon Message Interfaces (RUMIs) and Relied Upon Physical Interfaces (RUPIs).

3.1 Novelty

All the issues above call for a monitoring solution that i) continuously observes the SoS to avoid or mitigate detrimental phenomena, ii) gathers data of RUMIs and RUPIs or internal data of CSs where possible, and iii) is able to infer the status of the properties of the macro-level looking *only* at data collected at micro-level. It follows that detection algorithms based on fingerprints e.g., antiviruses [38], intrusion detectors [36] or failure predictors [2], may result not adequate for detecting unexpected phenomena and in general for SoSs due to SoSs dynamicity.

In such a context, anomaly detection seems one of the most suitable approaches in detecting unexpected behaviors in dynamic and complex SoSs. In the security domain, this technique was proven effective [38] in detecting zero-day attacks, which exploit unknown vulnerabilities to get into the targeted system. The same approach is commonly used to detect threats to dependability in complex systems, also when the system is composed by OTS components [3]. To the authors' knowledge, the topic of bringing anomaly detection into the paradigm of SoS was not explored in the recent years. Consequently, after expanding the topic of *anomaly detection*, in the rest of the paper we will investigate and explore the characteristics of a monitoring system for SoS, which runs data analysis features based on anomaly detection. The aim is to examine how to detect - among all threats and hazards - unexpected detrimental emerging phenomena.

3.2 Anomaly Detection

As mentioned above, anomaly detectors gained popularity especially when detection mechanisms such as fingerprint-based, event logs, heartbeats are not effective [38] e.g., when the complexity of the system is too high. Antiviruses and intrusion detectors can detect hazards when they identify a behavior that is compliant with a known fingerprint of an attacker or a malware, but they need also rules to detect zero-day attacks or attacks from unknown adversaries [9]. Moreover, unexpected or previously unknown system failures can be predicted observing specific indicators to characterize if the runtime system behavior is compliant with generic performance expectations [2], [3].

Despite the topic of bringing anomaly detection into SoSs is still not adequately explored, it is possible to find frameworks where anomaly detection is applied in complex systems e.g., Service Oriented Architectures or Cloud environments. In most of these studies, authors challenged the complexity of their system designing strategies that can be used as basis for a discussion that specifically tackles SoSs. In Table 1 we reported a set of frameworks that performs anomaly detection in complex systems. Some of them deal with dynamicity and evolution properties of complex systems [4], [6], [7], while others tackle systems composed of OTS components [2], [3]. Moreover, a subset of them [5], [10] is addressing emergent behaviors as side topic. All the listed frameworks are realized either for dependability [2], [3], [4], [5], [6], [7], [8] or security [9], [10] purposes.

4. DESIGNING A MONITORING AND ANOMALY DETECTION FRAMEWORK

Here we explore the main design principles behind a monitoring framework for anomaly detection, highlighting: i) the purpose of the framework, ii) the monitoring approach, iii) the indicators to be monitored, and iv) the anomaly detection technique. In Table 1 we report several frameworks in which authors adopted different approaches to solve the design challenges discussed in this Section.

4.1 Purpose of the Framework

As discussed in Section 3, anomaly detection was proven effective to the purpose of security and dependability. Depending on the specific needs of the administrator or the owner of the system, a monitoring framework can be designed to improve security (i.e., intrusion detection) or dependability (i.e., error detection, failure prediction), identifying anomalous behaviors. This choice influences the whole planning of the framework, defining the threats we want to detect and affecting the choices of i) the monitoring and data analysis approach (see Section 4.2), ii) the monitored indicators (see Section 4.3), and iii) the performance targets to be achieved (Section 5).

Approaches in existing frameworks (Table 1). The frameworks in Table 1 use anomaly detection for different scopes. Frameworks for error detection [4], [5] investigate anomalies to interrupt the fault-error-failure chain. Failure predictors [2], [3], [6] assume that errors already manifested in the system, and try to avoid their escalation in failures or the propagation to unsafe states. In the security domain, we can classify i) intrusion detectors [9], [10], which represent a security layer preventing or blocking possible malicious attacks, and ii) malware detectors, which analyse the system to identify anomalous behaviours due to malicious modules that are already infecting the system.

4.2 Monitoring and Data Analysis Approaches

Several approaches [14], [15], [20] can be adopted depending on where we put the data analysis engine e.g., anomaly detector. Moreover, databases containing historical or generic support data that are used for analyses can be put on i) an external machine coordinating the detection activities or ii) distributed on the nodes of the complex system. This results in the following two monitoring and data analysis approaches.

Centralized: a coordinator manages the monitoring and the data analyses. The coordinator also keeps track of historical or support data to assist the data analyses. Monitored data is sent from the CSs to the coordinator, which analyses them and alerts the administrator if anomalies are detected.

Distributed: The coordinator only provides to CSs policies or rules for data analyses e.g., thresholds or parameters of the anomaly detector, allowing the autonomous CSs to share a common core of parameters for data analysis. With this approach, the coordinator is not a bottleneck; instead, each CS must allow running custom tasks that may drain system resources.

Approaches in existing frameworks (Table 1). Depending on the context, frameworks for anomaly detection can be designed to centralize or decentralize the heaviest computing operations. Distributing operations [7], [8] reduces the bottleneck around the coordinator, but requires well-developed distribution of loads and

tasks among the CSs. Nevertheless, anomaly detection frameworks [9], [10] targeting security do not consider a distributed data analysis approach. Instead, they prefer sending collected data to a central elaboration unit. This allows not sharing parameters of the anomaly detection strategy with all the CSs, blocking adversaries that want to intercept such communications in order to read, corrupt or modify such critical parameters.

4.3 Monitored Indicators

Nowadays software is becoming more complex and consequently a large number of performance indicators e.g., memory usage, cache hits, packets shared through the network, can be captured by specific probes at defined time instants. Observing indicators related to different layers of the system e.g., OS, network, can provide a more accurate view of the system. The observed data need to be transmitted and analyzed continuously, affecting the monitored system and potentially slowdowning its tasks. Thus, it becomes fundamental to select those indicators that are most useful to detect anomalies.

In fact, previous research shows that even in a complex system the set of relevant variables is typically quite small [11]. Moreover, depending on the specific analyses that will be conducted using the monitored data, indicators can be classified extracting a minimum set that allows reaching defined performance scores. For example, sets of indicators were extracted targeting failure prediction [12], anomaly detection through invariants [5] and errors due to software faults [13].

An important remark should be done to consider the requirement of having all CSs synchronized to a global time. Otherwise, it is not possible to build a reasonable global time base. This affects our ability of fusing information coming from different CSs ultimately providing polluted data to the data analysis modules. For example, consider the final report about a major power blackout occurred in parts of the US and Canada in 2003. Here the authors declare that i) the Task Force's investigators labored over thousands of items to determine the sequence of events, and that ii) the process would have been significantly faster and easier if there had been wider use of synchronized data recording devices [40].

Approaches in existing frameworks (Table 1). Most of anomaly detectors observe performance indicators targeting OS [4], [3], [5] and network [2], [3], [7], [9] layers. We explain this results as follows: i) these layers are always present in a complex system,

Table 1: Existing Monitoring and Anomaly Detection Frameworks for Complex Systems

Framework			Monitoring		Anomaly Detection			Performance	
Name	**Evaluation Environment**	**Purpose**	**Approach**	**Observed Elements**	**Targeted Anomalies**	**Strategy**	**Detection Efficiency**	**Performance**	**Overhead**
ALERT [7]	Cluster Environment	Anomaly Prediction	Distributed	On each host: IBM System S and PlanetLab	Processing Time and Throughtput anomalies	Decision Tree Classifier	TPR > 90%, FPR ~ 0%	Tens of seconds or several minutes lead time	Probes: 1% load Detector: 1-2 ms for training
CASPER [2]	Air Traffic Control	Failure Prediction	Centralized	Network	Resource (Memory, I/O) Stress	Hidden Markov Models (HMMs)	Precision: 88.5%, Recall: 75.8%, FPR: 11.2%	Stress Prediction: Memory [20.8, 27] s - I/O [19.2, 24.9] s	Probes: - Detector: -
[5]	Distributed Web Banking Application	Error Detection	Distributed	CPU use, memory use, in/out network packets	Mis/Reconfiguration, Denial of Service, Development faults	Invariants	F-Measure: 86%	Upper bound: 1 min. from fault activation	Probes: - Detector: -
SEAD [6]	Cloud Environment	Failure Detection	Centralized	Dom0 and Xen Hypervisor	Faults from CPU, Memory, Disk, and Network.	Support Vector Machines (SVM)	TPR: 92.1%, FPR: 83.8%	Not Provided	Probes: OTS Detector: -
TIRESIAS [3]	Distributed Environment	Failure Prediction	Distributed	CPU, Memory, Context Switch	Performance - Degrading Faults	Dispersion Frame Technique (DFT)	FPR: 2.5%	Look-Ahead times in different setups	Probes: no overhead Detector: -
[4]	Service Oriented Architectures	Error Detection	Centralized	OS, JVM and Network	Software Errors (Performance Degradation)	Statistical Predictor and Safety Margin (SPS)	Precision and Recall: Memory [33.5, 95.8]% - Network [50.0, 86.7]%	Evaluation of each observation: (32.10 ± 5.99) ms	Probes: 150MB memory, negligible CPU stress Detector: -
[8]	Hadoop, SILK	Workflow Error Detection	Distributed	Log Files	Low performance (i.e., limiting the bandwidth)	Finite State Automation (FSA)	Hadoop FPR: 88%, SILK FPR: 76%	Not Provided	Probes: - Detector: -
SSC [10]	Web Services	Intrusion Detection	Centralized	UNIX Proc and SysInfo, custom JMX, FS Monitor	DoS attacks (hPing tool)	Most Appropriate Collab. Comp. Selection (MACCS)	FPR: 0.11%, FNR: 0%	Average Processing Time:150 ms	Probes: - Detector: "minimal CPU and RAM"
McPAD [9]	Datasets	Intrusion Detection	Centralized	HTTP Traffic	Generic, Shell-Code and Polymorphical CLET attacks	SVM	Detection Rate: 95%	< 0.04 ms per Payload	Probes: Synthetic dataset Detector: -

and ii) enterprise monitoring tools [21], [22] offer probes to observe these two layers. Moreover, several indicators regarding the memory and cache management can be retrieved only at OS-level, because middleware e.g., *JVM*, application servers such as *Apache Tomcat*, act at an higher stack level.

4.4 Anomaly Detection Technique

As highlighted in [1], a key aspect of any anomaly detection technique is the nature of the input data. Each data instance might consist of only one attribute (univariate) or multiple attributes (multivariate). In the case of multivariate data instances, all attributes might be of same type or might be a mixture of different data types. The nature of attributes determines the applicability of anomaly detection techniques. For example, for statistical techniques [19] specific statistical models have to be used for continuous and discrete data. Similarly, for nearest-neighbour-based techniques [18], the nature of attributes would determine the distance measure to be used. Moreover, when aggregated measures instead of actual data are provided e.g., distance or similarity matrix, techniques that require original data instances such as classification-based techniques [17] are not applicable.

Most of the techniques mentioned above need training data to learn the characteristics of both normal and anomalous instances, becoming able to label the data that is monitored at runtime through the probes. Focusing on SoSs, we observe that these systems can be characterized by intrinsic dynamicity, often changing their behaviour and, consequently, the characteristics of both normal and anomalous behaviours. This calls for a new training phase, requiring i) to collect train data and ii) to train the parameters of the chosen techniques. When dynamicity is very high, this task can overcome the normal activity of the system, resulting in large periods of unavailability of the anomaly detector and slowdowning the usual tasks that run on the targeted CS. This means that anomaly detection techniques that do not need training data are more suitable because they do not require periods of unavailability for training [13], [34].

Approaches in existing frameworks (Table 1). Different studies use different data analysis approaches: as explored in [1], specific anomaly detection approaches call for a more suitable anomaly detection algorithm or technique. This results in a wide utilization of statistical (3 out of 10 in Table 1) and machine learning (5 out of 10) algorithms, while [5] and [10] respectively scores anomalies using invariants and component selection. As expanded in Section 6.4, despite statistical and machine learning worked very well in the studies reported in Table 1, from a SoS perspective the usage of these algorithms raises important concerns that cannot be ignored.

5. PERFORMANCE TARGETS

To guarantee the best support either for dependability or security purposes, anomaly detectors need to analyze monitored data and provide their results rapidly and with a low number of wrong interpretations. Consequently, the *notification time*, or rather the time between the observation of system data through probes and the evaluation of its anomaly degree, should be minimized. Moreover, an inaccurate evaluation can result in i) *false positives*, which can cause the execution of non-required reaction strategies by the administrator, or ii) missed detections (*false negatives*), with possible severe consequences.

Table 2: Detection Performance Measures

Measure	Formula
True Positives (TP)	# of correct anomaly detections
True Negatives (TN)	# of correct non-anomaly detections
False Positives (FP)	# of wrong anomaly detections
False Negatives (FN)	# of missed anomaly detections
Precision (P)	$Precision = \frac{TP}{TP + FP}$
Recall (R)	$Recall = \frac{TP}{TP + FN}$
F-Score	$FScore(\beta) = (1 + \beta^2)\frac{P \cdot R}{\beta^2 \cdot P + R}$

Taking into account the following performance targets is mandatory and it has to be part of the development phase of anomaly detection frameworks.

5.1 Detection Performance

The performance of the anomaly detection strategy is evaluated according to the main metrics [16] used in pattern recognition and information retrieval with binary classification. All of these measures are based on indexes representing the correct predictions - *true positives* (TP), *true negatives* (TN) – and the wrong ones, due to missed detections (*false negatives*, FN) or wrong anomaly recognitions (*false positives*, FP). More complex measures based on the abovementioned ones are *precision* (also called positive predictive value), the fraction of retrieved instances that are relevant, and *recall* (also known as sensitivity), the fraction of relevant instances that are retrieved (see Table 2).

Depending on the purposes of the targeted SoS, the reference metric may change: for example, in systems where false negatives (i.e., missed detection of an anomaly) can heavily damage the system, recall is more relevant than precision. Instead, when detection of anomalies (both TP and FP) calls for expensive reaction strategies, FP must be minimized, thus emphasizing precision more than recall.

5.2 Notification time

Another performance index that needs to be addressed is the notification time, that is the time between the observation of a snapshot and its evaluation. According to the block definition in Section 4.2 this quantity is the sum of (see Figure 1):

- *observation time (ot)*, the time slot spent from the probing system to get system data by the probes;
- *probe-monitor transmission time (pmtt)*, the time needed to transmit all the observed data to the monitor
- *data aggregation time (dat)*, the time used by the monitor to aggregate and parse the received data
- *storing time (st)*, time spent from the monitor to store the aggregated data in the chosen data container;
- *monitor-detector transmission time (mdtt)*, the time needed to transmit the data aggregated from the monitor to the anomaly detector tool;
- *detection time (dt)*, the time used from the anomaly detector to compute its calculations based also on previously collected historical data;

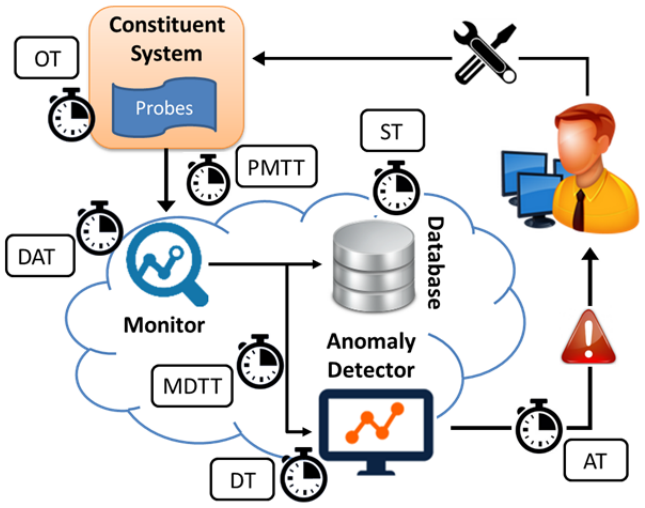

Figure 1: Time quantities through the workflow.

- *alert time (at)*, the time needed to deliver the anomaly alert to the system administrator.

Depending on the chosen monitoring approach, these quantities can be combined to obtain the *notification time* (*nt*) as follows.

Centralized. In this approach, the coordinator machine i) runs the monitor and the anomaly detector and ii) hosts the database in which historical and support data are stored. Considering the anomaly alert as a simple notification e.g., text message or email the quantities *mdtt* and *at* represent negligible instants of time. Assuming *nt* as the notification time, in such a context its value is expressed as the linear combination of the remaining time quantities:

$$nt = ot + pmtt + dat + st + dt$$

Distributed. Monitoring and data analysis logic are placed on the CSs, while the coordinator supports these activities providing parameters or rules e.g., set of indicators to monitor, rules for anomaly detection. Consequently, each CS runs dedicated modules that can interfere with the tasks that are usually executed on its CS resulting in a higher intrusiveness level that needs to be taken into account. Considering that i) data can be stored in the database simultaneously with the aggregations performed by the monitor, and ii) the possible alert need to be forwarded to the coordinator, the *nt* can be estimated as:

$$nt = ot + \max\{dat, st\} + dt + at$$

6. BRINGING ANOMALY DETECTION INTO SOS DESIGN

After describing the peculiarities of both anomaly detection frameworks and SoSs, in this section we list potential design approaches that can bring them together. Moreover, in Table 3, for each SoS viewpoint, we summarize the approaches for constructing an anomaly detection framework that can help adhering with the guidelines of a given viewpoint.

6.1 Purpose of the Framework

Building a framework that effectively uses anomaly detection for both *dependability and security* purposes can be a challenging goal. In fact, frameworks designed for intrusion detection are strongly dependent from the observation of network usage indicators. Further, malware oriented detection strategies should monitor OS attributes to understand if something is already damaging the system and maybe trying to steal or corrupt critical data from the hard drive. Regarding dependability monitoring, performance indicators observed in middleware e.g., thread number, cache usage and memory management, can reveal the manifestation of errors at application level that may escalate into failures in the near future. Regardless the chosen target, *governance* aspects play a decisive role in defining i) which CSs can be instrumented with probes, ii) the communication channels among them and iii) other general rules that could limit or support the effectiveness of the anomaly detection technique under consideration.

6.2 Monitoring and Data Analysis Approaches

Another key point is related to the *architecture* of the SoS, and mainly the characteristics, the roles and the ownerships of each CS and their interconnections. Monitored data must be labelled consistently in the whole SoS, since data acquisition through probes and monitors constitutes the basis for the anomaly detection process. This should include *handling time* issues that can lead to missed synchronizations or wrong timestamps assigned to each observation. As example, if the targeted SoS is

Table 3: Tackling Viewpoints Targeting Anomaly Detection in SoS

SoS Viewpoint	Description of the Technique	Frameworks Proposing or Implementing the Technique
Architecture	Consider Architectural Hybridization, i.e., link different CSs or blocks of CSs with a given level of safety that needs to be accomplished	CASPER [2] (Black Box)
Evolution and Dynamicity	Make Anomaly Detection able to tune its parameters when an evolution or a configuration change is detected. Algorithms and strategies for the detection of anomalies should work with poor knowledge of the history of the system e.g., online machine learning techniques, since this can change very often. Monitoring support needs to be adaptive as well.	[4], SEAD [6], SSC [10]
Emergence	Adopt models and libraries of anomalies targeting emerging behaviours, e.g., deadlock, livelock, unwanted synchronization	[5], ALERT [7], SSC [10]
Governance	Difficult to generalize. Communications must be fast enough to provide data observed by the probes to the monitor and to the anomaly detector, either if the approach is distributed or centralized.	-
Handling Time	Synchronize the clocks with an NTP server. The resulting clock precision is enough to label timestamps if real-time requirements are not intrinsic of the SoS.	CASPER [2] (Generic clock synchronization), [4] (NTP)
Dependability and Security	Build a Multi-Layer monitoring structure connected to adaptive Anomaly Detection modules	[4], SEAD [6], SSC [10]

under a (Distributed) Denial of Service attack, having an unsynchronized assignment of timestamps could lead to wrongly interpret anomalies in each threatened CS, without understanding the shared cause generating the anomalies.

More in general, CSs can perform tasks with heterogeneous levels of criticality. It follows that depending on the criticality of each CS the monitoring and data analysis approach must change, adopting an architectural hybridization [24] that allows checking more carefully the CSs that are responsible for the most critical tasks. In particular, we can envision an hybrid monitoring approach which i) runs a centralized coordinator that collects and analyzes data coming from critical CSs, and ii) provides a set of parameters or rules for the anomaly detection algorithms that will be executed directly in the CSs that do not execute critical tasks. This allows monitoring critical CSs without burdening the centralized coordinator, since it does not need to analyze data observed on less critical CSs. This choice also impacts notification time (see Section 5.2).

We remark that this hybridization might be tailored depending on the category of the SoS (see Section 3.1). In *directed* and *acknowledged* SoSs, it is easier to identify common thresholds or trends because the objective is mostly shared among CSs. Instead, when CSs act together (*collaborative* SoS) and have limited knowledge of the other components of the SoS (*virtual* SoS), identifying shared rules for anomaly detection becomes very hard. In this context, the monitoring strategy must be distributed and customized as much as possible to suit the characteristics of each CS.

6.3 Monitored Indicators

The adoption of a multi-layer monitoring approach [35] allows obtaining information about the state of the services (the macro-level from an SoS perspective) or the applications observing the underlying layers (SoS micro-level), without instrumenting the application or the service layer [4], [7]. The general idea is that when an application encounters a problem e.g., a crash in one of its functionalities, it generates an anomalous activity that can be observed looking at specific indicators of the underlying layers e.g., the number of active threads is abruptly decreasing. This solution is suitable even when services changes frequently. The result is a monitoring solution coping with *evolution end dynamicity* of the targeted SoS, giving a widespread and adaptive support to the modules responsible for the *dependability and security* assessment.

6.4 Anomaly Detection Technique

While a plethora of techniques for performing anomaly detection exist [1] in the literature, only a few of them can be considered suitable for anomaly detection in SoS. This is mainly due to i) *evolution and dynamicity* properties, which call for adaptive algorithms that can quickly reconfigure its parameters without needing of time-consuming testing phases, and ii) *emergence*, which can be unexpected, making techniques based on rules or on static pattern recognition less effective i.e., no rules or faulty patterns for unexpected phenomena are known. Consequently, the most suitable algorithms belong to the statistical and the online machine learning groups. In particular, statistical algorithms such as [35] work with a sliding window of past observations that are used to build a prediction. If the monitored value is not compliant with the predicted value, an anomaly is raised. Similarly, online machine learning techniques e.g., gradient-descend based [36], can build classifiers that change their behavior according to the evolution of the observed system, automatically tuning their main parameters. Emerging phenomena can be therefore detected because we assume that they cause the generation of values for specific parameters that are far from the nominal behavior.

7. CONCLUSIONS AND FUTURE WORKS

In this paper we discussed the main aspects and known issues behind the design of a monitoring and anomaly detection framework for systems of systems. Since this paradigm is arising and gaining a lot of interest in the recent years, we combined its main aspects and the characteristics of state-of-the-art monitoring and anomaly detection frameworks for complex systems. The result is a set of design guidelines that should be followed as “best practices” when designing such a framework for SoSs.

Future works will be directed to understand which anomalies are typically generated by emerging behaviors. In particular, we will revise the literature looking at the known emerging behaviors, conducting experimental campaign aiming at tracing the anomalies they generate. This will allow us to characterize these emerging behaviors in terms of their consequences on the trend of monitored indicators, ultimately improving our anomaly detection capabilities and, consequently, the connected dependability and security properties. In particular, existing works on emergence in complex systems [33] already list potentially detrimental behaviors that we would test with an experimental support.

Moreover, it will be important to investigate how the monitoring and anomaly detection system can adapt itself to work with newly added CSs. The need of global time synchronization among CSs will be further motivated also with an experimental support, showing how the notification time is affected by delays and misalignment regarding the clocks of CSs.

8. ACKNOWLEDGMENTS

This work has been partially supported by

removed row for double blind.

removed row for double blind.

removed row for double blind.

removed row for double blind.

9. REFERENCES

[1] Chandola Varun, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey." ACM computing surveys (CSUR) 41.3 (2009): 15.

[2] Baldoni, Roberto, Luca Montanari, and Marco Rizzuto. "On-line failure prediction in safety-critical systems." Future Generation Computer Systems 45 (2015): 123-132.

[3] Williams, Andrew W., Soila M. Pertet, and Priya Narasimhan. "Tiresias: Black-box failure prediction in distributed systems." Parallel and Distributed Processing Symposium, IEEE 2007 (pp. 1-8). IPDPS 2007.

[4] Zoppi, Tommaso, Andrea Ceccarelli, and Andrea Bondavalli. "Context-Awareness to Improve Anomaly Detection in Dynamic Service Oriented Architectures." International Conference on Computer Safety, Reliability, and Security (pp 145-158). Springer International Publishing, 2016.

[5] Aniello, Leonardo, et al. "Automatic Invariant Selection for Online Anomaly Detection." International Conference on Computer Safety, Reliability, and Security (pp 172-183). Springer International Publishing, 2016.

[6] Pannu, Husanbir S., Jianguo Liu, and Song Fu. "A self-evolving anomaly detection framework for developing highly dependable utility clouds." Global Communications Conference (GLOBECOM), 2012 IEEE. IEEE, 2012.

[7] Tan, Yongmin, Xiaohui Gu, and Haixun Wang. "Adaptive system anomaly prediction for large-scale hosting infrastructures." Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing (pp. 173-182). ACM, 2010.

[8] Fu, Qiang, et al. "Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis." ICDM. Vol. 9 pp. 149-158. 2009.

[9] Perdisci, Roberto, et al. "McPAD: A multiple classifier system for accurate payload-based anomaly detection." Computer Networks 53.6 (2009): 864-881.

[10] Shone, Nathan, et al. "Misbehaviour monitoring on system-of-systems components." 2013 International Conference on Risks and Security of Internet and Systems (CRiSIS) (pp. 1-6). IEEE, 2013.

[11] Hoffmann, G., Malek, M., "Call Availability Prediction in a Telecommunication System: A Data Driven Empirical Approach", 25th IEEE Symposium on Reliable Distributed Systems (SRDS 2006), Leeds, UK, October 2006.

[12] Irrera, Ivano, et al. "Towards identifying the best variables for failure prediction using injection of realistic software faults." Dependable Computing (PRDC), 2010 IEEE 16th Pacific Rim International Symposium on. IEEE, 2010.

[13] Bovenzi, Antonio, et al. "An OS-level Framework for Anomaly Detection in Complex Software Systems." IEEE Transactions on Dependable and Secure Computing 12.3 (2015): 366-372.

[14] Joyce, Jeffrey, et al. "Monitoring distributed systems." ACM Transactions on Computer Systems (TOCS) 5.2 (1987): 121-150.

[15] Massie, Matthew L., Brent N. Chun, and David E. Culler. "The ganglia distributed monitoring system: design, implementation, and experience." Parallel Computing 30.7 (2004): 817-840.

[16] Sokolova M., Japkowicz, Szpakowicz. "Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation." AI 2006: Springer Berlin Heidelberg, 2006. 1015-1021.

[17] Eskin, Eleazar, et al. "A geometric framework for unsupervised anomaly detection." Applications of data mining in computer security. Springer US, 2002. 77-101.

[18] Rajasegarar, Sutharshan, et al. "Distributed anomaly detection in wireless sensor networks." 2006 10th IEEE Singapore International Conference on Communication Systems (pp. 1-5). IEEE, 2006.

[19] Sotiris, Vasilis A., W. Tse Peter, and Michael G. Pecht. "Anomaly detection through a bayesian support vector machine." IEEE Transactions on Reliability 59.2 (2010): 277-286.

[20] Zanikolas, Serafeim, and Rizos Sakellariou. "A taxonomy of grid monitoring systems." Future Generation Computer Systems 21.1 (2005): 163-188.

[21] "Nagios Project", www.nagios.org [accessed on the 1st Sept. 2016]

[22] "Ganglia Monitoring", ganglia.sourceforge.net [accessed on the 1st Sept. 2016]

[23] "Zenoss | Own IT.", www.zenoss.com [accessed on the 1st Sept. 2016]

[24] P. Verissimo, "Travelling through wormholes: a new look at distributed systems models," SIGACT News 37, 1 (March 2006), pp. 66-81, 2006.

[25] Kopetz, H., Höftberger, O., Frömel, B., Brancati, F., & Bondavalli, A. (2015, May). Towards an understanding of emergence in systems-of-systems. In System of Systems Engineering Conference (SoSE), 2015 10th (pp. 214-219). IEEE.

[26] H. Kopetz, "Why a Global Time is Needed in a Dependable SoS?", Proc. of the Workshop on Engineering Dependable Systems-of-Systems, Univ. of Newcastle upon Tyne, 2014.

[27] E.Morris, P. Place, D. Smith, "System-of-Systems Governance: New Patterns of Thought", Technical Note CMU/SEI-2006-TN-036, Software Engineering Institute, Carnegie Mellon, October 2006.

[28] Bondavalli, Andrea, et al. "System-of-Systems to Support Mobile Safety Critical Applications: Open Challenges and Viable Solutions."

[29] M. W. Maier, "Architecting Principles for Systems-of-Systems," System Engineering, vol. 1, no. 4, pp. 267-284, 1998.

[30] D.A. DeLaurentis, "A taxonomy-based perspective for Systems-of-Systems design methods", IEEE International Conference on Systems, Man and Cybernetics, 2005.

[31] M. Jamshidi, Ed. (2009). System-of-Systems engineering - innovations for the 21st century, J. Wiley & Sons.

[32] Dahmann, J.S.; Baldwin, K.J., "Understanding the Current State of US Defense Systems of Systems and the Implications for Systems Engineering," in 2nd Annual IEEE Systems Conference pp.1-7, 2008.

[33] Mogul, Jeffrey C. "Emergent (mis) behavior vs. complex software systems." ACM SIGOPS Operating Systems Review. Vol. 40. No. 4. ACM, 2006.

[34] Su, Hui, and J. David Neelin. "Teleconnection mechanisms for tropical pacific descent anomalies during El Niño*." Journal of the atmospheric sciences 59.18 (2002): 2694-2712.

[35] Ceccarelli, Andrea, et al. "A multi-layer anomaly detector for dynamic service-based systems." International Conference on Computer Safety, Reliability, and Security (pp. 166-180). Springer International Publishing, 2015.

[36] Mukkamala, Srinivas, Guadalupe Janoski, and Andrew Sung. "Intrusion detection using neural networks and support vector machines." Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on. Vol. 2. IEEE, 2002.

[37] H. Kopetz, B. Fromel, O. Hoftberger. "Direct versus stigmergic information flow in systems-of-systems." System of Systems Engineering Conference (SoSE), 2015 10th (pp. 36-41). IEEE, 2015.

[38] Comar, Prakash Mandayam, et al. "Combining supervised and unsupervised learning for zero-day malware detection." INFOCOM, 2013 Proceedings IEEE (pp. 2022-2030). IEEE, 2013.

[39] US Government Accountability Office: "GPS Disruptions: Efforts to Assess Risk to Critical Infrastructure and Coordinate Agency Actions Should be Enhanced". Washington, GAO -14-15. 2013.

[40] Kopetz, Hermann. Real-time systems: design principles for distributed embedded applications. Springer Science & Business Media, 2011.

[41] AMADEOS Consortium, D2.3 – AMADEOS Conceptual Model Revised, 2016, http://amadeos-project.eu/documents/public-deliverables/ .