An approach to scalable security monitoring

Christof Brandauer and Peter Dorfinger
Salzburg Research Forschungsgesellschaft
Salzburg, Austria
Email: firstname.lastname@salzburgresearch.at

Pedro Yuri Arbs Paiva
Instituto Tecnológico de Aeronáutica
São José dos Campos/SP, Brasil
Email: paiva@ita.br

Stefano Salsano
University of Rome Tor Vergata / CNIT
Rome, Italy
Email: stefano.salsano@uniroma2.it

***Abstract*—A long time ago Industrial Control Systems were protected from attacks due to the use of proprietary technologies and physical isolation. This situation has changed dramatically and the systems are nowadays often prone to severe attacks executed from remote locations. In many cases, intrusions remain undetected for a long time and this allows the adversary to meticulously prepare an attack and maximize its destructiveness. The ability to detect an attack in its early stages thus has a high potential to significantly reduce its impact. To this end, the SCISSOR project proposes a holistic, multi-layered, security monitoring and mitigation framework spanning the physical- and cyber domain. The comprehensiveness of the approach demands for scalability measures built-in by design. In this paper we present how scalability is addressed by an architecture that enforces geographically decentralized data reduction approaches that can be dynamically adjusted to the currently perceived context. A specific focus is put on a robust and resilient solution to orchestrate dynamic configuration updates in the periphery of the monitoring and data collection subsystem. Experimental results obtained from a prototype implementation are in favor of the feasibility of the approach.**

***Index Terms*—Cyber-physical systems, Industrial control systems, SCADA systems, Computer security, Cyber defense.**

I. Introduction

In the early days of industrial control systems (ICS), security in the meaning of defense against cyber-attacks and threats was not regarded as a main issue. The systems were protected by physical access restrictions and they were isolated from public communication networks. They employed proprietary technologies and afforded inherent security by obscurity of its niche applications.

In the last decade, however, many boundary conditions have changed dramatically. On one side, for obvious cost and market reasons, industrial control processes and SCADA systems have made a significant move towards the progressive adoption of widespread low-cost equipment and they rely on the same standard Internet protocols (e.g. TCP/IP, HTTP, etc.) and ICT solutions used in ordinary networks and systems. As such, they are already extensively scrutinized and challenged by attackers, who may bring into the ICS networks the same vulnerabilities and attacks extensively exploited in the Internet at large. Many of these attacks are documented well and require very little technical skills to be executed.

Today, while most ICS deployments still lack any cryptographic protection in the internal network, it is no longer true that ICSs are always disconnected from the Internet [1], [2]. According to [3], the number of vulnerability disclosures increased enormously, from 19 in 2010 to 189 in 2015. According to [4], many of these vulnerabilities remain unpatched for longer than one month in SCADA software. The infection/intrusion of a system is often undetected for a long time. As an example, the analysis of the attack on the Ukrainian Power Grid infrastructure [5] reasons that the adversary appears to have gained access to the system more than 6 months before the attack. Being undetected throughout the whole period allowed the attacker to meticulously prepare the attack by performing network reconnaissance, harvesting credentials, escalating privileges etc.

Obviously, the best way to protect critical infrastructures would consist in redesigning ICSs from scratch and by leveraging secure operating system technologies. While this solution is in principle appealing, it would not be viable in the short/medium term, as it would require the redesign of all the software for industrial systems. The EU H2020 project "SCISSOR" thus primarily targets a complementary approach that is focused on the *detection and mitigation* of ongoing attacks. It can be integrated into existing systems, affords incremental deployment, and holds the promise to concretely impact in a much shorter time frame.

Specifically, the SCISSOR framework consists in the design of a holistic, multi-layered, security monitoring and mitigation framework, spanning all the dimensions present in a critical infrastructure deployment: the physical environment, the network traffic, the hardware and software system components, the people accessing the infrastructure, and, of course, the industrial process itself. Due to this comprehensive monitoring approach and to the fact that ICSs can reach an overwhelming size (e.g. nation-wide critical infrastructures comprising millions of nodes), scalability may become a severe hurdle. Unlike conventional systems which limit themselves to collect monitoring data and deliver them are as they are to central analysis points, SCISSOR addresses scalability by design through an architecture that enforces geographically decentralized data pre-filtering and data reduction approaches. The filtering and data reduction can be dynamically adjusted to the currently perceived context. SCISSOR's *Control and Coordination Layer*, which is at the core of this approach, is the specific focus of this paper.

II. The SCISSOR Architecture

The SCISSOR architecture comprises 4 layers as shown in Fig.1. The *Monitoring Layer* hosts widely heterogeneous types

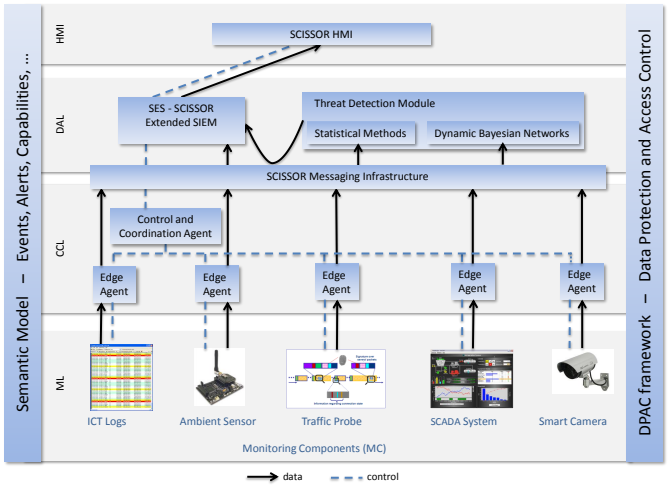

Fig. 1. SCISSOR security monitoring framework.

of remote data sources producing at least: i) environmental data gathered from cyber-physical ambient sensors; ii) alerts gathered from traffic analysis probes; iii) logs and integrity checks carried out on the critical infrastructure HW/SW components and subsystems, iv) events generated by surveillance cameras supporting object/pattern detection, and v) events natively generated by the SCADA control process. Additional data sources can easily be integrated incrementally as needed. In the following we refer to all entities on the Monitoring Layer (ML) as *Monitoring Components* (MC).

The *Control and Coordination Layer* (CCL) decouples this heterogeneous multi-source, multi-technology, and multi-purpose ML from the overlying *Decision and Analysis Layer* (DAL). It thereby abstracts the peculiarities of the various MCs, their native data formats, protocols, and configuration interfaces, and thus greatly simplifies the event correlation and triggering of dynamic reactions in the modules of the DAL. As described in detail in Section III, the CCL provides distributed data pre-processing capabilities that can be dynamically adjusted in an orchestrated fashion. The CCL emits the pre-processed data to the *SCISSOR Messaging Infrastructure* (SMI). The SMI provides a fault-tolerant, resilient, high-throughput and low-latency messaging infrastructure that scales very well with SCISSOR's cloud computing infrastructure. The messages are consumed by the DAL which comprises a conventional SIEM plus analysis modules that significantly extend its detection and correlation capabilities by the use of advanced probabilistic and statistical models. The *Human-Machine Interface* (HMI) is devised to present in real-time the system behavior to the human operators in a simple and usable manner.

III. Control and Coordination Layer

SCISSOR addresses the above mentioned scalability concerns by designing the CCL as a distributed, multi-stage, agent-based data collection and pre-processing subsystem that applies adaptable data filtering and reduction techniques in the periphery of the system. It is a fundamental feature that these

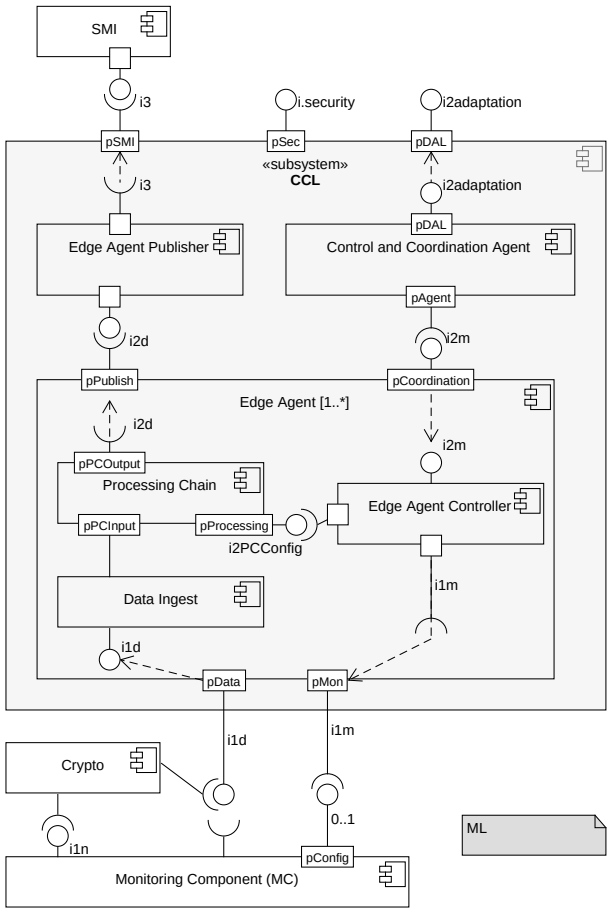

Fig. 2. Design of the CCL subsystem

processing steps are i) adaptive, and ii) programmable, using platform-independent programming interfaces. Adaptive since the level of data quality needed may dramatically vary in the course of an attack: if in normal conditions some specific data may be needed every several seconds or even minutes, when an attack is supposed to be in progress, the data resolution needed may drastically increase in order to track and recognize the threat, and control its evolution. Programmable, since the way in which data is filtered, again, may significantly differ depending on the contextual conditions: while a given type of data pre-processing is adequate in normal conditions, the emergence of alerts from alternative sensors (e.g. a physical intruder detected by a camera) may trigger the need to disable (or filter/aggregate differently) the data reduction from environmental sensors in that same geographic region.

A. CCL Design

The design of the CCL is formalized as a UML composite structure diagram which is shown in Fig. 2. *Edge Agents* play the major role in establishing the distributed data collection and pre-processing system. The CCL comprises many of them and they are located physically close to their respective MCs. An Edge Agent serves one or more MCs. It consumes the MC data and processes it before handing it over to the Edge Agent Publisher which sends the data to the SMI.

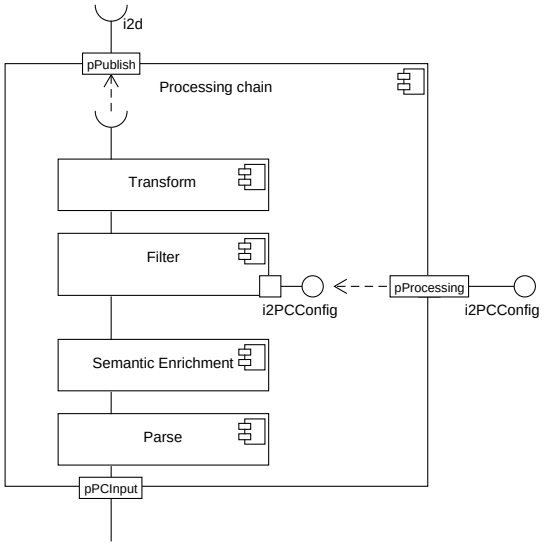

Fig. 3. Processing Chain component of the Edge Agent

An Edge Agent must be able to handle all sorts of common "output channels" or "transports" (e.g. stdout, file, TCP, UDP, syslog, etc.) and data formats. If a MC with a previously unknown data export interface has to be integrated, the integration effort must be achieved on the Edge Agent side. Therefore, the *Data Ingest* component of the Edge Agent is required to implement an extensible set of protocols that can be reused among the MCs. The data handover, which can occur in push or pull mode, respectively, is handled via the `i1d` interface. Data on this interface is encrypted according to the SCISSOR data protection mechanisms [6]. A MC may opt to implement the protection scheme (e.g. as an output plugin) and send data directly via the `i1d` interface. Alternatively, it outputs data via its native export interface (`i1n`) and a generic SCISSOR *Crypto* component protects the data transfer to the Ingest component.

The data processing of the Edge Agent is handled inside its *Processing Chain* component which is detailed below in Section III-B. The Processing Chain provides the configuration interface `i2PCConfig` which is accessed exclusively by the *Edge Agent Controller* component in order to dynamically reconfigure the data processing mechanisms of the Edge Agent. The Edge Agent Controller can also handle the reconfiguration of a MC in those cases where a MC provides the optional configuration interface `i1m`. Both types of reconfiguration are performed in collaboration with the *Control and Coordination Agent* which centrally coordinates and orchestrates dynamic adaptations as described in detail in Section III-C.

B. Data reduction

The Processing Chain component as shown in Fig. 3 represents a sequence of consecutive data processing steps. It is composed of functional blocks related to (in this order): parsing, semantic enrichment, filtering, and finally transformation to a SCISSOR internal unified format.

The *Parse* block parses incoming data with a regular expression based pattern that is tailored to the native data format of the MC. The *Semantic Enrichment* component enhances the data structure with e.g. location information, units, etc.

The Filter block of the Processing Chain is the core component related to the implementation of SCISSOR's distributed data reduction concept. It is used to filter out specific data and/or aggregate data by computing statistics over a given time interval and forwarding the statistics at the end of the interval.

The filter configuration is specified by an XML document that contains conditions and operations. The conditions define on which incoming data records an operation shall be performed. For the specification of a condition any field of the record can be used for numeric comparison and/or pattern matching. The conditions can be logically combined via AND and OR. When a record matches a condition it is subjected to a filter operation.

So far, a few simple operations have been implemented but with the given framework the set of operations can be extended easily. There is currently one operation that can compute simple statistics (like minimum, maximum, average) over a given period of time. During this period, the filter consumes incoming records, updates the statistic(s) and discards the record. At the end of the interval, one record containing the statistic(s) is emitted. It is, of course, properly annotated as a statistic computed over the given time interval so that an analysis module can distinguish it from a single measure. Another operation is available that forwards only every N-th record (and drops the other ones). When N is set to 0, all records matching the condition are dropped.

The Filter component can be dynamically reconfigured at runtime via the `i2PCConfig` interface (see Fig. 2). At any time, a new XML configuration can be provided via this interface. The Filter will pick it up immediately and it is guaranteed that no data is lost in the process of switching to the new configuration.

C. Orchestrated adaptation

As motivated above it is a central concept of SCISSOR that the data acquisition system can be adapted at runtime. This approach promises major benefits in terms of scalability compared to a conventional, mostly statically configured monitoring system while at the same time allowing capturing the current state of the system at a much higher resolution in alert situations. Dynamic reactions can e.g. be: activate/deactivate additional MCs; adjust the data filtering/reduction; capture data at a higher/lower temporal resolution etc.

An adaptation request can be triggered by a human operating the HMI or alternatively automatically by a module in the DAL. In any case an adaptation request is sent via the SMI to the `i2adaptation` interface which is served by the Control and Coordination Agent (CCA). Inside the CCL, 2 interfaces form the basis upon which dynamic adaptation actions can be built: the `i2m` interface as described above and the `i1m` interface which is an optional interface that can be provided

by a MC. Through this interface the MC configuration can be set (or modified). This configuration determines what metrics are measured (e.g. temperature) and how they are measured (e.g. sampling frequency). In some cases, the MC may provide extensive configuration options (e.g. the programmable network monitoring devices of SCISSOR [7]) while others are hardly configurable (if at all). A MC may also provide services beyond configuration, e.g. the rotation of a camera.

When the CCA receives an adaptation request it decomposes it and thereby creates a task for each Edge Agent that needs to be involved in the fulfillment of the request. The implementation of the request is then orchestrated by the CCA in collaboration with the Edge Agent Controller (EAC) (see Fig. 2). To this end, *Interaction Protocols* as standardized by the FIPA [8], provide valuable building blocks that address the problem of orchestrating distributed agents in order to achieve a common goal. As an example, the *Contract Net* protocol could by used by the CCA to orchestrate a task that requires the collaboration of at least a given minimum number of Edge Agents. In other cases where there is no such minimum restriction, a simpler Interaction Protocol requiring less message round-trips between the agents (e.g. Request-Response) can be used.

In the Contract Net Interaction Protocol the initiator sends a call for proposals to m agents who may accept or refuse it (or not answer at all within the given deadline). The initiator will then decide if it rejects or accepts the proposals and notifies the agents accordingly. In the later case, the proposing agents will carry out the task and inform the initiator about the result. Of course, carrying out the task may involve the use of additional interaction protocols.

IV. Prototype implementation

The prototype implementation of CCL Processing Chain is based on Apache Flume. This framework provides agents that comprise a *Source*, *Channel*, and *Sink* component. The Source is used to ingest data from some upstream entity (e.g. a log file or some other agent). The data can be processed as it moves from the Source via the Channel to the Sink which finally passes it on to a downstream entity (e.g. another agent, a database, or a messaging broker). Various concrete implementations of Source, Channel, and Sink are available. Powerful processing capabilities are available in a *Morphline*, which defines a chain of data transformation blocks called *Commands*. Our prototype implementation of the Processing Chain is based on existing (Parse, Transform) and newly developed (Semantic Enrichment, Filter) Morphline Commands.

As far as the orchestrated adaptation is concerned, the implementation of the first prototype was based on an agent framework which provided implementations of FIPA Interaction Protocols (see above). However, in the testing and evaluation phase of this prototype it turned out that the agent platform is not sufficiently robust under the occurrence of intermittent network outages that lead to temporary network partitions or disconnections of an agent from the platform. Especially the latter case led to unacceptable situations: if the temporary disconnection occurred at specific instants of time in the processing of an Interaction Protocol then the Interaction Protocol got stuck and the lock situation could not be resolved. Obviously, the security monitoring system must not be vulnerable to intermittent network problems that may be provoked by an attack.

In search for a more robust and resilient implementation in the presence of (intermittent) network problems, alternative means of communication have been investigated. In a distributed system, processes can communicate through message passing or using shared memory. Direct message exchange between processes, however, involves some issues that should be carefully thought out (e.g. the network protocols, handling of connections and disconnections, network partitions, etc.). The framework Apache Zookeeper [9] uses the (replicated) shared memory model with a hierarchical namespace and data nodes. Of course, processes still use messages to communicate with ZooKeeper, but the API handles the peculiarities of the communication and specifically the errors that may occur.

Zookeeper is a service for coordinating processes of distributed applications, called a *coordination kernel* [10], [11], [12]. It does not constitute a solution for specific problems, instead, it enables the implementation of distributed primitives, such as locks, leader election and group membership. ZooKeeper implements an API that manipulates simple wait-free data objects [13] organized hierarchically in a file system like data structure [11]. This structure is replicated within a cluster of servers - called *ensemble* - to achieve high availability and performance.

The data is replicated and each server has a copy of it. There is a leader that is elected by the servers participating in the quorum, i.e. the majority. The leader election and the agreement is performed by the ZooKeeper Atomic Broadcast protocol (ZAB) [14], [15]. Every operation that modifies data is forwarded to the leader which converts it into a transaction. When a client requests a write operation, for example, the leader proposes a transaction that may be accepted by the quorum. Once accepted, all the non-faulty servers update their own data and the client is notified. A client configuration contains the addresses of all servers of the ensemble and if a server becomes unavailable the client automatically connects to a different one.

In the second prototype, the dynamic adaptation service of the CCL was built on the Apache Zookeeper framework which is used to reliably, resiliently, and consistently exchange requests and notifications between the main actors of the coordination task: the DAL, CCA, and the various EACs. The DAL submits an adaptation request to the SMI and it is consequently received by the CCA. The CCA decomposes the request into tasks related to the involved EACs and engages in a Contract Net interaction protocol with theses EACs. The CCA evaluates the replies from the EACs and decides whether to progress in the implementation of the adaptation. When an EAC receives a task, it will evaluate the request and if possible implement it by updating the Filter configuration and/or invoking the management API of the MC.

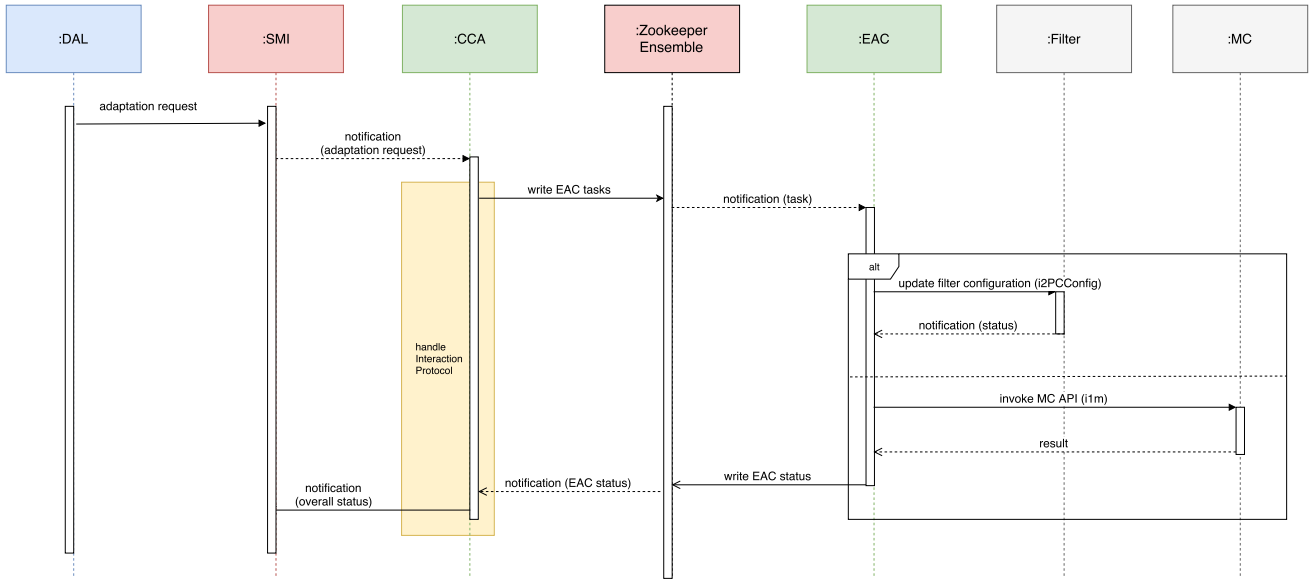

Fig. 4. Dynamic adaptation in the CCL

This interaction between the components is depicted in a simplified sequence diagram in Fig. 4. It is insofar simplified as the interaction between the CCL and the EACs doesn't show the internal message exchange of the Contract Net Interaction Protocol upon which it is built [8]. The notifications are realized by so called *Watches* in Zookeeper. A client can set a Watch to a specific data node and will subsequently be notified immediately when the data of the node is modified.

A. Experimental results

Executing and orchestrating the dynamic adaptation via Zookeeper affords several benefits in terms of e.g. helpful synchronization primitives, resilience, etc. A potential downside is the fact that communication between the involved agents is now taking place indirectly via the Zookeeper cluster. The cluster, which has an additional internal overhead related to leader election and the commit protocol, sends notifications to agents that set Watches for specific data nodes. Thereupon, an Edge Agent retrieves the new configuration from the cluster. Overall, the communication, while being more robust and resilient, must be expected to consume more time than a direct message exchange between agents. To evaluate whether the adaptation request can still be processed sufficiently fast, two experimental test were carried out in laboratory environment and the SCISSOR testbed, respectively. Both tests were made with a Zookeeper Cluster comprising 3 separate instances.

In the laboratory test, 2 additional machines with each one running an instance of an Edge Agent, are deployed. The 5 machines are all located in the same local network connected by a 1 Gbit/s Ethernet switch. At the beginning of the test, the CCA receives a configuration adaptation request and implements it by engaging with the 2 Edge Agents via Zookeeper as described in the previous section (see Fig. 4). The test covers 2000 requests (1 per second) in total. We measure the duration of an adaptation "transaction", i.e. between the reception of the request at the CCA and the instant when the CCA sends a notification to the SMI informing about the successful completion.

The laboratory test establishes a baseline against which the results of the real-world test are compared. In this second experiment, the Zookeeper cluster runs in a public cloud located in Geneva. One Edge Agent (serving a simulated SCADA system) is executed in a project internal testbed located in Paris; the second Edge Agent (serving environmental sensors, network probes and a smart camera) is running in an operational electrical cabin located on the island of Favignana in the west of Sicily. The connection from the cabin is established by a 4G mobile link.

The empirical cumulative distributions of the measured durations are shown in Fig. 5. The longest durations are measured for the Edge Agent in Favignana. This is an expected result as the path between the CCA (Geneva) and the Edge Agent (Favignana) exhibits the largest communication delays. With the given prototype implementation of a transaction, 3 communication round-trips between the CCA and the EAC are required. Some internal processing time is additionally needed inside the Edge Agent, where the EAC updates the configuration of the Filter component. Further analysis of the transaction durations confirmed that the 3 round-trips are the largest contributor to the overall delay. For the EAC located in Paris 90% of the transactions complete within 49 ms. This result matches the small communication delay between Geneva and Paris during the test.

Overall, 98% of the transactions complete within less than 500 ms. This is a satisfying result that fulfills SCISSOR's near real-time requirements, especially considering that the test environment spanned a noteworthy geographical area and included a mobile network. The highly-available Zookeeper infrastructure brings essential benefits in terms of resilience and robustness. The negligible added cost is 1 additional

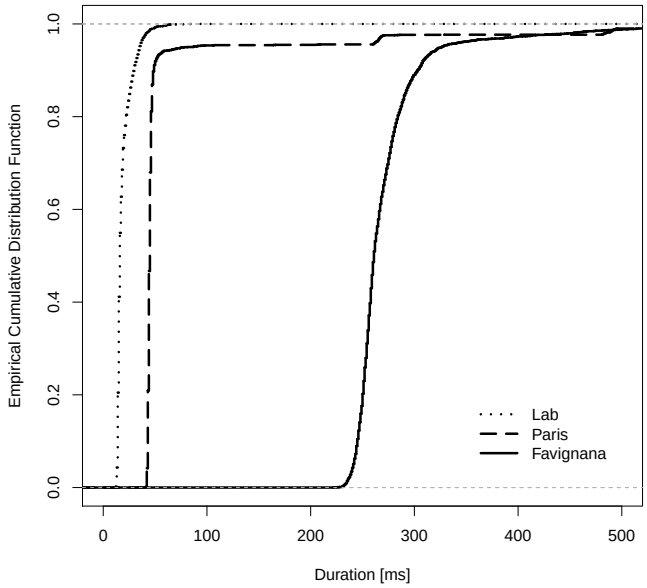

Fig. 5. Duration of the adaptation request transaction

communication round-trip time on top of the 2 round-trips that are needed at minimum for a Contract Net Interaction Protocol.

V. Conclusion

In today's Industrial Control Systems (ICS), many protocols still do not employ cryptographic protection mechanisms, attacks often remain undetected, and publicly know vulnerabilities remain unpatched for many months on average. The state of security in ICSs thus needs to be improved enormously but many measures bring improvements only in the medium/long term. In the short term, the ability to detect an attack in its early stages and apply appropriate countermeasures would be highly beneficial. The SCISSOR project tackles this topic by proposing a holistic monitoring and mitigation framework spanning all the dimensions of critical infrastructures. The comprehensiveness of the approach demands for scalability by design. The given paper focuses on the scalability aspect and presents how it is addressed by a distributed and hierarchical data aggregation system that is dynamically adjusted at runtime. Thereby, the distributed processing nodes and system components must be reconfigured in an orchestrated fashion. The proposed solution makes use of Interaction Protocols as developed by the software agent community and operates them over the highly reliable and resilient infrastructure for coordinating distributed processes as provided by Apache Zookeeper. The concept has been prototyped and tested in a lab and real-world environment. Experimental results show the effectiveness and resilience of the approach and confirm that the increased robustness can be achieved with only a small performance overhead. Based on these encouraging results more extensive tests will be carried out in the future. These tests will monitor a much larger infrastructure including a real-world Smart Grid deployment as well as a simulated SCADA system that can be attacked without restraints for testing purposes.

Acknowledgment

The SCISSOR project has received funding from EU's Horizon 2020 programme under grant agreement No 644425.

References

[1] M. J. Shodan, "Shodan," http://www.shodan.io, 2014, accessed: 2017-09-29.

[2] B. Radvanosky and J. Brodsky, "Project shine (SHodan INtelligence Extraction) findings report," https://de.slideshare.net/BobRadvanovsky/project-shine-findings-report-dated-1oct2014, 2014, accessed: 2017-09-29.

[3] O. Andreeva, S. Gordeychik, G. Gritsai, O. Kochetova, E. Potseluevskaya, S. I. Sidorov, and A. A. Timorin, "Industrial control systems vulnerabilities statistics," Kaspersky Lab, Report, 2016.

[4] Secunia, "Secunia vulnerability review," https://secuniaresearch.flexerasoftware.com/?action=fetch&filename=secunia_vulnerability_review_2015_pdf.pdf, 2015, accessed: 2017-09-29.

[5] E-ISAC Electricity Information Sharing and Analysis Center, "Analysis of the cyber attack on the ukrainian power grid," https://ics.sans.org/media/E-ISAC_SANS_Ukraine_DUC_5.pdf, 2016, accessed: 2017-09-29.

[6] M. Avril, L. Basta, L. Bouillet, A. Daif, G. Landais, and C. Tavernier, "Identity based cryptography for smart grid protection," in *Proceedings of 9th international conference on computer engineering and applications, Dubai, UAE, 22-24 Feb 2015*, 2015.

[7] P. L. Ventre, A. Caponi, G. Siracusano, D. Palmisano, S. Salsano, M. Bonola, and G. Bianchi, "D-streamon: From middlebox to distributed NFV framework for network monitoring," in *2017 IEEE International Symposium on Local and Metropolitan Area Networks, LANMAN 2017, Osaka, Japan, June 12-14*, 2017.

[8] M.-P. Huget, "The foundation for intelligent physical agents," http://www.fipa.org, 2017, accessed: 2017-09-29.

[9] The Apache Software Foundation, "Apache zookeeper," https://zookeeper.apache.org, 2017, accessed: 2017-09-29.

[10] F. Junqueira and B. Reed, *ZooKeeper: Distributed Process Coordination*, 1st ed. O'Reilly Media, Inc., 2013.

[11] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010.

[12] F. Junqueira. (2015, Aug.) Distributed consensus reloaded: Apache zookeeper and replication in apache kafka. Accessed: 24-05-2017. [Online]. Available: https://www.confluent.io/blog/distributed-consensus-reloaded-apache-zookeeper-and-replication-in-kafka/

[13] M. Herlihy, "Wait-free synchronization," *ACM Transactions on Programming Languages and Systems*, vol. 13, no. 1, Jan. 1991.

[14] B. Reed and F. P. Junqueira, "A Simple Totally Ordered Broadcast Protocol," in *Proceedings of the 2Nd Workshop on Large-Scale Distributed Systems and Middleware*, ser. LADIS '08. New York, NY, USA: ACM, 2008, pp. 2:1–2:6.

[15] F. P. Junqueira, B. C. Reed, and M. Serafini, "Zab: High-performance broadcast for primary-backup systems," in *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems&Networks*, ser. DSN '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 245–256.