Catalog-driven services in a 5G SDN/NFV self-managed environment

Henriques, Nuno; Sargento, Susana; Neves, Pedro; Gil Perez, Manuel; Martinez Perez, Gregorio; Bernini, Giacomo; Wang, Qi; Alcaraz Calero, Jose Maria; Koutsopoulos, Konstantinos

Published in:
2018 IEEE Symposium on Computers and Communications (ISCC)

DOI:
10.1109/ISCC.2018.8538659

Published: 25/06/2018

Henriques, N., Sargento, S., Neves, P., Gil Perez, M., Martinez Perez, G., Bernini, G., Wang, Q., Alcaraz Calero, J. M., & Koutsopoulos, K. (2018). Catalog-driven services in a 5G SDN/NFV self-managed environment. In *2018 IEEE Symposium on Computers and Communications (ISCC)* [8538659] (IEEE Conference Proceedings). IEEE. https://doi.org/10.1109/ISCC.2018.8538659

Catalog-Driven Services in a 5G SDN/NFV Self-Managed Environment

Nuno Henriques[1, 2, 3,*], Susana Sargento[1, 2,*], Pedro Neves[2, 3], Manuel Gil Pérez[4], Gregorio Martínez Pérez[4], Giacomo Bernini[5], Qi Wang[6], Jose M. Alcaraz-Calero[6] and Konstantinos Koutsopoulos[7]

[1]Department of Electronics, Telecommunications and Informatics, University of Aveiro - Aveiro, Portugal
[2]Instituto de Telecomunicações - Aveiro, Portugal
[3]Altice Labs - Aveiro, Portugal
[4]Departamento de Ingeniería de la Información y las Comunicaciones, University of Murcia - Murcia, Spain
[5]Nextworks - Pisa, Italy
[6]School of Engineering & Computing, University of the West of Scotland - Paisley, United Kingdom
[7]Creative Systems Engineering - Athens, Greece

***Abstract*—With the Fifth-Generation (5G) mobile networks set to arrive within the next years, this new generation will transform the industry with a profound impact on its customers as well as on the existing technologies and network architectures. Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) will play key roles for the network operators as they prepare the migration to 5G, allowing them to quickly scale their networks. This paper presents a research work undertaken on this new paradigm of virtualized and programmable networks, aiming to address Self-Organizing Networks (SON) scenarios in a NFV/SDN context, focusing on detection and prediction of potential network and service anomalies. Towards this end, the performance management system performs aggregation, correlation and analysis of data gathered from the virtualized and programmable network elements. In particular, customized catalog-driven tools are developed, and the results show that they are able to successfully address these requirements. Current performance management platforms in production are designed for non-virtualized (non-NFV) and non-programmable (non-SDN) networks, and the knowledge gathered from this research brings some new understanding on how management platforms must evolve in order to be prepared for the upcoming next-generation mobile networks.**

***Keywords*—5G, SON, SDN, NFV, Virtualization, Monitoring, Aggregation, Catalog-Driven**

I. Introduction

The Fifth-Generation (5G) mobile networks will bring a new era in which connectivity will become significantly enhanced and increasingly flexible and network performance will be tailored to the needs of the user. The standardization of 5G is still underway, although as we move closer to 2020, the vision of 5G is becoming clearer. 5G is expected to feature exceptionally fast speeds and extremely low latency, and allow various use cases such as demanding mission-critical applications and massive device-to-device communications for the Internet-of-Things (IoT). The performance management platforms in production usually support a large set of tools to conduct performance measurements on the network and services, and process and analyze information collected from non-virtualized (non-NFV) and non-programmable (non-SDN) resources. The static nature of these non-virtualized and non-programmable resources allows the configuration processes of the information models to be enforced manually. However, in the 5G NFV/SDN context, the agility required for the on-boarding and deployment of new functions increases considerably, with the support of automatic configuration processes for the information models of the network functions and virtualized services becoming necessary. The EU 5G-PPP project SELFNET targets to achieve automatic and autonomous network management for 5G networks, and a catalog-driven architecture is proposed for network monitoring and analytics purposes. In catalog-driven architectures, the information models of the virtualized network functions are typically stored on a central catalog of the network operator. After on-boarded on the catalog, publish/subscribe mechanisms are used to propagate the information models to the components of the architecture that requires them. The SELFNET project provides the necessary research platform and environment to develop a proof-of-concept solution to meet these requirements, allowing operators to understand how their performance management platforms must evolve in order to be prepared for the upcoming 5G mobile networks. Moreover, these system advancements in a 5G scenario also requires aggregation, correlation and analysis of data gathered from these virtualized and programmable network elements. The data themselves can be processed either in batch (non-real-time processing) or streaming (real-time processing) to create indicators that will be used to generate Health-of-Network (HoN) symptoms to be processed by Machine Learning (ML) algorithms, endowing the network with the necessary intelligence to automatically react upon imminent threats and failures. In this wider context, the scope of this paper is focused on two specific components or services of the SELFNET 5G network management architecture that has been developed from scratch and integrated within its logical structure throughout the research work: 1) **Monitoring Catalog**: a catalog devised to comprise the information on the sensors available for deployment on the platform (its integration with the orchestration layer is out of the scope of this work though), as well as the batch/streaming aggregation and thresholding rules that need to be enforced on the aggregation layer (within the scope of this work); 2) **Aggregation Configuration Manager**: the component responsible for the

*Corresponding authors, Email: {nuno.henriques, susana}@av.it.pt

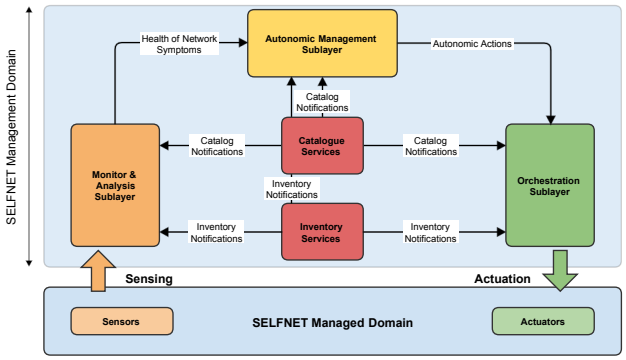

Fig. 1. SELFNET's Architecture (high-level)

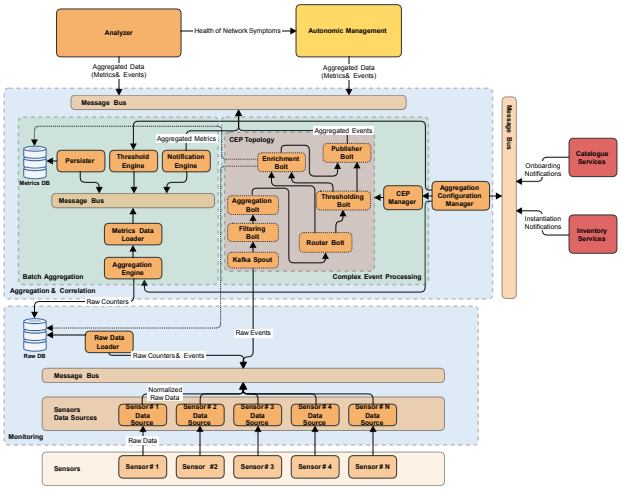

Fig. 2. SELFNET's Aggregation Framework (low-level view)

enforcement of the aggregation and thresholding rules on the aggregation layer.

II. Architecture Overview

SELFNET's high-level architecture is represented in Fig. 1, showing the Monitor & Analyzer, Autonomic and Orchestration sub-layers, as well as the transversal Catalog and Inventory services. The main objective of the Monitor & Analyzer sub-layer is to collect, aggregate and analyze information provided by sensors in a heterogeneous network environment composed by NFV, SDN and legacy network functions. Prediction algorithms are involved to analyze and identify Health of Network (HoN) symptoms to proactively indicate potential problems, and the Autonomic sub-layer, which is the core of the SELFNET Autonomous management logic, uses these HoN symptoms to diagnose the most probable cause(s) and decides on the most appropriate tactic(s) and action(s) to proactively prevent the end-to-end services from being affected. The Orchestration sub-layer is responsible to enforce all the actions defined by the Autonomic sub-layer, by orchestrating all the available heterogeneous network functions (e.g. SDN-Apps, VNF, PNFs) implementing sensing and actuation logics, supporting both deployment and (re)configuration. Finally, the Catalog and Inventory services are responsible for maintaining and expose all the information related to the on-boarding and instantiation of resources that are common to the sub-layers (e.g.: rules of aggregation of metrics, deployed sensors, NFVs and so on). The Monitor & Analyzer sub-layer is further divided in three frameworks. The Monitoring Framework, responsible for collecting, persisting and forwarding the data gathered by the sensors, the Aggregation Framework which aggregates and/or correlates data and provides the results to the last remaining framework, the Analyzer, which is responsible for providing HoN symptoms. Nevertheless, the Aggregation Framework may also, in some cases, forward aggregated data directly to the Autonomic Framework, so the latter can perform its autonomic functionalities based either on the HoN symptoms and/or the aggregated data.

A. Aggregation Framework

The Aggregation Framework is in turn sub-divided in two sub-frameworks, as shown in Fig. 2. The Batch Aggregation Framework (BAF) which is responsible for the non-realtime data processing and the Complex Event Processing Framework (CEPF) which addresses the realtime processing. Both these sub-frameworks rely on the data provided by the Monitoring Framework, which is responsible for collecting raw data from heterogeneous sensors and normalizing the collected data to a common and uniform monitoring information model. This allows the data gathered by the multiple and heterogeneous sensors to be handled seamlessly. The collected and normalized data are persisted on the Raw DB to be processed at a later stage by the BAF and forwarded also directly to the Aggregation Framework for real-time processing by the CEPF. Collected and normalized data can be grouped into two main categories: raw counters, which are produced periodically by the sensors (e.g. total packet count, communication frequency) and raw events which are produced only when a specific occurrence takes place (e.g. virtual machine instantiated, hardware failure). The BAF mainly aggregates, compresses and persists data retrieved from the Raw DB, for trend and prediction analysis by external components (the Analyzer framework or the Autonomic sub-layer), but it also analyzes the produced aggregated metrics to determine if any of defined thresholds is crossed, also providing alarm notifications to the external components. The CEPF processes raw data (counters and/or events) and it is producing aggregation events similar to the aggregated metrics provided by the BAF but in real-time. However, in the CEPF case, the aggregation events might be triggered by a threshold crossed by a metric calculated in real-time or by a threshold crossed by alarms and/or events correlated in real-time; they can also be aggregation events reporting metrics being calculated in real-time. Although the nature of the CEP flow is for real-time data processing, the output is always persisted on the Metrics DB to enable further analysis from autonomic components leveraging on machine learning techniques and algorithms. In order to instruct both the BAF and the CEPF on which aggregated metrics/events and thresholds should be set, the Aggregation Configuration Manager (ACM), which belongs to the Aggregation Framework, and the Monitoring Catalog (MC), which resides within the Catalog Services, as seen in Fig. 2, were designed and developed. The Monitoring Catalog is responsible for the on-boarding of aggregation and threshold rules that will be enforced on the BAF and CEPF components, as well as for

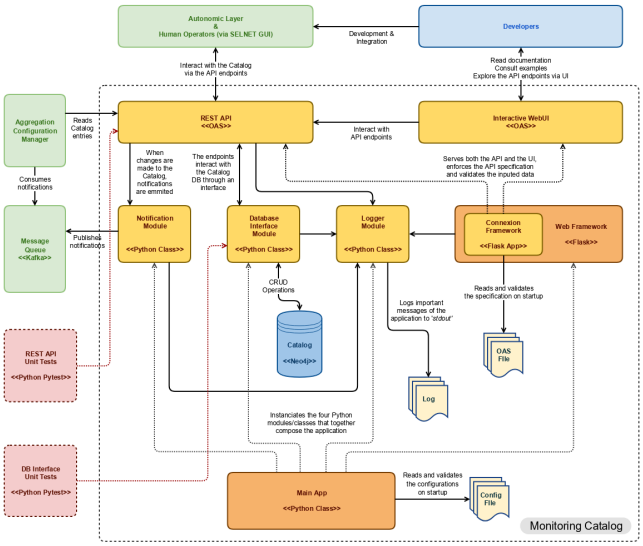

Fig. 3. SELFNET's Monitoring Catalog Architecture

the information on all types of sensors that are available: what kind of sensing data they provide and how the rules can make use of them to produce aggregated metrics (real-time or not). The enforcement of the rules on-boarded on the Monitoring Catalog is done by the Aggregation Configuration Manager. Whenever a change occurs on the catalog, the Aggregation Configuration Manager is notified about that through a message bus. In Fig. 2 it is possible to see that the ACM reaches out three specific components of the Aggregation framework: the Aggregation Engine (AE), the Threshold Engine (TE) and the CEP Manager. All the real-time aggregation rules will be enforced on the CEP Manager, the non real-time ones on the Aggregation Engine and, of course, the threshold rules will be enforced on the Threshold Engine.

III. Implementation

This section discusses how the Aggregation Configuration Manager and the Monitoring Catalog were implemented as well as their respective architectures.

A. Monitoring Catalog

The Monitoring Catalog (MC) is the unique entry point within the SELFNET architecture allowing external components (e.g. Autonomic sub-layer) to create, read, update and delete aggregation and threshold rules that are enforced on the Aggregation Framework. Its internal architecture is represented in Fig. 3, but in its most basic sense, it is comprised by a database where the catalog itself is stored, a RESTful API to interact with the service and a notification module to notify external services about the modifications or updates of the catalog.

The Monitoring Catalog API is designed following the OpenAPI Specification (OAS) [1], formerly Swagger, which is a specification for machine-readable interface files for describing, producing, consuming, and visualizing RESTful Web services created by a consortium of industry experts with an

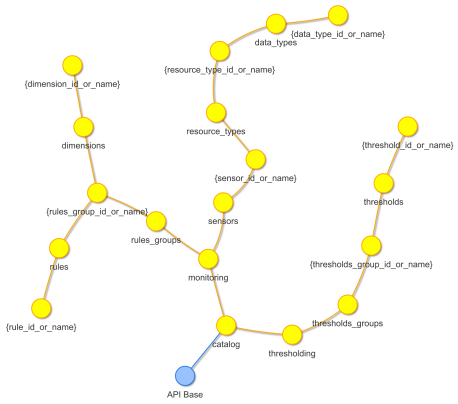

Fig. 4. Graph of the Monitoring Catalog API's Endpoints

open governance structure under the Linux Foundation, the OpenAPI Initiative (OAI) [2]. Flask [3] is used as the Web Framework, as well the Connexion Framework [4], which works on top of Flask. The latter handles the HTTP requests based on the OAS, allowing the API specification to be written first (API first approach). Later on, the framework will handle the mappings of the endpoints to the routing functions, guaranteeing that it will work as specified, even before any code is written. The Web Framework serves both the RESTful API of the MC as well an interactive Web UI that contains rich documentation of the API endpoints, providing the developers the complete information they need to develop and integrate their services. The complete set of endpoints supported by the API is represented by the graph shown in Fig. 4, and all the data submitted to and retrieved from the API is in JSON format. The catalog has two major areas: *monitoring* and *thresholding*. The *thresholding* section is organized by thresholding groups which may contain one or more threshold rules. Likewise, the *monitoring* section has aggregation rules groups which may contain one or more aggregation rules. Furthermore it also contains a sub-section that holds the information about the types of sensors that exist on the network as well the resource and data types they support, which is extremely important for the statistical formulas and filters that are used both on the aggregation and thresholding rules.

Neo4j [5] is used as the database to store the Monitoring Catalog. It is a Graph Database that is able to represent the catalog. In fact, the catalog can be considered the database itself, as the catalog is stored in the database without versioning, i.e. there is only a single version of the catalog present in the database. The graph database models the data in the form of a graph, as it is evident, and it compares to the traditional relational databases as represented in TABLE I. With that in mind, it will be easier to understand the catalog

TABLE I
RDBM vs Graph DB

RBDMS	Graph Database
Tables	Graphs
Rows	Nodes
Columns and data	Properties and their values
Constraints	Relationships
Joins	Traversal

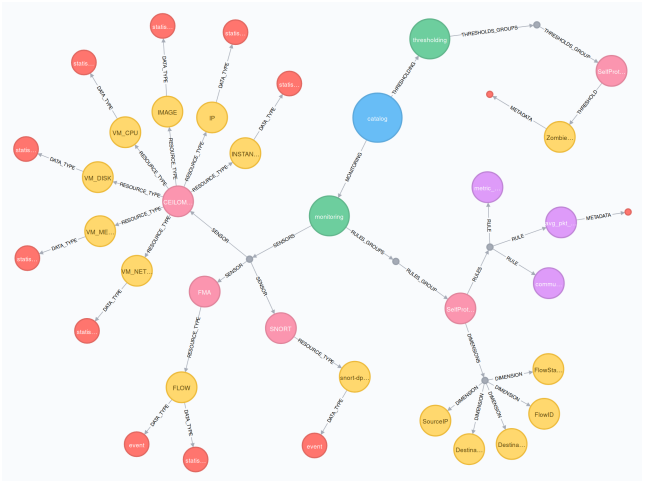

Fig. 5. SELFNET's Monitoring Catalog Graph DB

example represented in Fig. 5 as stored in the database. The blue node represents the root of the catalog, and the green ones the *monitoring* and the *thresholding* areas mentioned before. Attached to the *thresholding* node it can be seen a single thresholds group also containing a single threshold rule. The *monitoring* node contains a rules group node with three aggregation rules (purple nodes) and five aggregated metrics dimensions (yellow nodes). It also contains three sensor node with their respective resource and data type nodes (yellow and red nodes respectively). As stated in TABLE I, all the properties and their respective values are associated with the nodes, e.g. the name of a sensor or the period of an aggregation rule.

The Monitoring Catalog also contains a notification module that creates an abstraction layer to handle the notifications that need to be published on the message bus whenever changes are made to the catalog. This mechanism is necessary to make the Aggregation Configuration Manager aware of these changes, so it can enforce them on the Aggregation Framework.

B. Aggregation Configuration Manager

The Aggregation Configuration Manager is the component responsible for enforcing the on-boarded rules on the Monitoring Catalog throughout the Aggregation framework. Specifically, the batch aggregation rules are enforced on the Aggregation Engine, the stream (realtime) aggregation rules are enforced on the CEP Engine, and the threshold rules are enforced on the Threshold Engine.

The ACM internal architecture is represented in figure Fig. 6, and it follows a master/slave or master/worker model where Apache Zookeeper [6] is used to coordinate the services of the Aggregation Configuration Manager. The tasks are assigned by the Master Service to the Client Services and the tasks in this context are directly associated to the operations that were made on the Monitoring Catalog (*CREATE*, *UPDATE* or *DELETE*).

MongoDB [7] was used as a database, with three distinct collections of documents, one for each client service, i.e one collection for the Aggregation Engine client, another collection

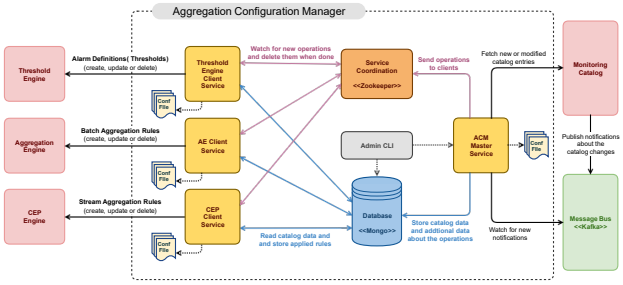

Fig. 6. SELFNET's Aggregation Configuration Manager Architecture

for the CEP Engine client and another one for the Threshold Engine client.

The Master Service is responsible for watching any incoming notifications sent by the Monitoring Catalog, and instructing the Client Services to apply the changes made on the catalog to their respective end clients or services. During its start up, the Master Service reads from its configuration file all the information required for it to interact with the four services it communicates with directly: Zookeeper, MongoDB, Kafka and of course the Monitoring Catalog.

Although there are three distinct types of Client Services, they all follow the same steps to handle their tasks. Their difference lies in the way they transpose the rules from the Monitoring Catalog to their respective end client components.

The Threshold Engine Client Service translates the threshold rules from the catalog to a structure that can be interpreted by the Threshold Engine's API, and likewise the AE Client Service and the CEP Client Service will translate the aggregation rules from the catalog to structures that can be interpreted by the Aggregation Engine's API and the CEP Engine's API respectively.

Just like it happens in the Master Service case, during their start up the Client Services read from their configuration files all the information required for them to interact with the three services they communicate with directly, Zookeeper and the MongoDB which are common to all three of them, and the Threshold Engine, Aggregation Engine and CEP Engine depending on the Client Service.

IV. Case Study

The Catalog-Driven solution presented in the previous sections has been experimentally validated in a lab environment. This experimental validation aimed at deploying the SELFNET's Self-Protection Use Case (SP-UC), and assess the solution presented in this paper against a scenario with real mobile network traffic generated by mobile users. The main goal of the SP-UC is to detect bots shaping a botnet (also known as zombies) that successfully managed to infiltrate the network by infecting some hosts, and take action to sever their communications to the botnet's Command & Control (C&C) in order to neutralize them. The overall process is divided into three loops with several steps (Fig. 7 depicts the first two).

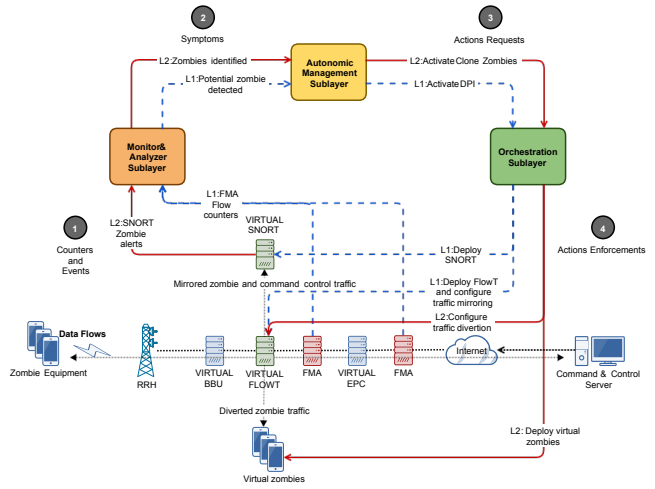

Fig. 7. First (blue dashed) and second (red straight) loops of the Self-Protection use case

A. SP-UC Loop 1

Preliminary identification of suspect communication patterns that may indicate that a host or group of hosts may be infected with malicious software of a botnet (Fig. 8), becoming zombies. To achieve this, the following steps are taken: 1) two batch aggregation rules are (manually) on-boarded on the MC which will be enforced in the AE by the ACM; 2) a threshold rule, which will be applied over these two batch aggregation rules, must be also (manually) on-boarded on the MC to be enforced on TE, which when crossed, will produce alarms and consequent notifications for the Autonomic Framework; 3) the Autonomic Framework produces a symptom based on the alarm notification and aggregated metrics, and instruct the Orchestration Framework to deploy a VNF with Deep Packet Inspection (DPI) functions and mirror the suspicious traffic to it; 4) in parallel with the previous step, the Orchestration Framework is also instructed to deploy a DPI VNF to identify possible zombies of the botnet.

B. SP-UC Loop 2

Confirmation (or dismissal) of the suspect communication patterns and diversion of the botnet's traffic. To achieve this, the following steps are taken: 1) at this phase of the process the DPI VNF is already collecting information and generating alerts, if any, that need to be aggregated in realtime and, for this purpose, a stream aggregation rule is (dynamically) on-boarded on the MC that will be enforced on the CEPF by the

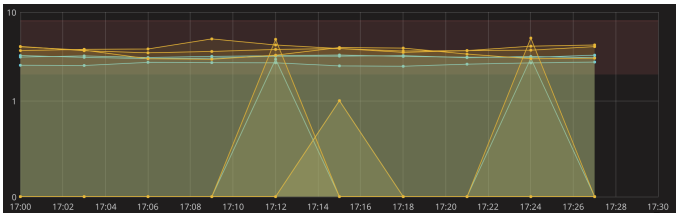

Fig. 8. SP-UC Loop 1 - Suspicious Attack Symptoms: resulting from the manually on-boarded batch aggregation and threshold rules, avg. pkt. count (blue), communication freq. (yellow), threshold (red stripe)

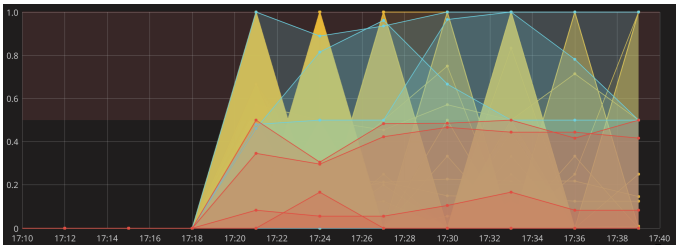

Fig. 9. SP-UC Loop 3 - Botnet Attack Symptoms: dynamically on-boarded batch aggregation and threshold rules (red/blue - botnet communications, yellow - false positives)

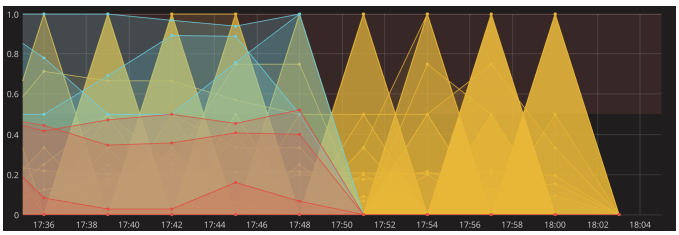

Fig. 10. SP-UC Loop 3 - Botnet Attack Symptoms: botnet communications (red/blue) dropped, false positives (yellow) not affected

ACM; 2) the stream aggregated metrics produced by the CEPF will confirm (or dismiss) if the communications indeed fall into the botnet pattern and will start generating alerts for the Autonomic Framework; 3) the Autonomic Framework produces a new symptom and instructs the Orchestration Framework to deploy a HoneyNet VNF to isolate the communications from the zombies to the botnet C&C, by diverting the traffic to the HoneyNet; 4) at the same time the Autonomic Framework initiates a learning phase using a Machine Learning (ML) process with neural networks, with the goal of filtering out false positives and acting more quickly upon new zombies that may appear afterwards, which when over will (dynamically) on-board a new batch aggregation rule and a new threshold rule on the MC, starting then the 3rd loop.

C. SP-UC Loop 3

Severing the botnet's communications to neutralize it (Fig. 9 and Fig. 10). To achieve this, the following steps are taken: 1) once the new batch aggregation rule and the new threshold rule are in place, the following communications from new zombies will generate new alarms for the Autonomic framework (different from the previous alarms of Loop 1); 2) upon receiving these new alarms, the Autonomic framework will generate a new symptom and instruct the Orchestration framework to simply drop the related communications, severing or disrupting any communication attempts from the botnet's C&C to its zombies, i.e. the botnet is from this moment onwards effectively neutralized.

V. Load Tests

In order to determine the performance of the developed services (the MC and ACM), a series of load tests were devised and performed, either by using freely available tools or by developing custom ones. The Monitoring Catalog has a RESTful API and therefore the load tests were done by checking and stressing its web services using simple client

TABLE II
MONITORING CATALOG - LOAD TEST RESULTS

Service	Avg. Time / Operation	Elapsed Time
Overall (avg. of CRUD operations)	14.00ms	1m 08s

Number of Operations : 5000, Avg. Operations / sec: 71

TABLE III
ACM MASTER SERVICE - LOAD TEST RESULTS

Service	Avg. Time / Operation	Elapsed Time
Monitoring Catalog	4.91ms	0m 36s
MongoDB	0.44ms	0m 04s
Zookeeper	3.86ms	0m 22s
Overall	9.86ms	1m 10s

Number of Operations : 5000, Avg. Operations / sec: 75

TABLE IV
ACM THRESHOLD ENGINE CLIENT - LOAD TEST RESULTS

Service	Avg. Time / Operation	Elapsed Time
Thresh. Engine	86.49ms	3m 08s
MongoDB	1.87ms	0m 18s
Zookeeper	2.79ms	0m 13s
Overall	93.91ms	3m 39s

Number of Operations : 2500, Avg. Operations / sec: 11

TABLE V
ACM AGGREGATION ENGINE CLIENT - LOAD TEST RESULTS

Service	Avg. Time / Operation	Elapsed Time
Agg. Engine	77.58ms	2m 57s
MongoDB	2.52ms	0m 16s
Zookeeper	4.95ms	0m 14s
Overall	91.57ms	3m 27s

Number of Operations : 2500, Avg. Operations / sec: 12

invocations on HTTP verbs (GET, POST, PUT, DELETE) and the overall results of the tests are presented in TABLE II. As to the Aggregation Configuration Manager, a series of timestamp variables were carefully placed within its source code to collect information about the times spent in specific operations, providing statistics when activated and through load tests stimulating this component by "hammering" the service with notifications through the message bus to simulate the operations done in the Monitoring Catalog. The results obtained for the ACM's Master Service and the ACM's Threshold Engine and Aggregation Engine client services are presented in TABLE III, TABLE IV and TABLE V (no results are shown for the ACM's CEP client service as the integration is currently underway).

VI. CONCLUSIONS

This paper presents a catalog-driven network management system for 5G networks. Starting with the Monitoring Catalog, its main goal was to address the catalog-driven needs of the 5G network management architecture defined in the SELFNET project. It effectively addressed the information content and structure that needed to be enforced on the Aggregation Framework. It employed both the batch/streaming aggregation rules to be enforced on the AE/CEP and the thresholding rules to be enforced on the TE. As to performance, the MC proved to have very good results, above the expectations and requirements of the project, as it was not foreseeable to have a performance demand close to the obtained average time of the operations as well as the number of requests per second for this particular service. The ACM provided a bridging service between the MC and the Aggregation Framework, effectively "translating" the rules on-boarded on the former and enforcing them on the latter. The results of the load tests performed on the ACM's Master Service show that it is able to cope with the maximum rate imposed by the Monitoring Catalog (75 op/sec against 71 op/sec). However, when it comes to the ACM's Client Services, the operations on the coordination service and the database have little impact on the overall performance of these components. It is clearly shown that the operations performed on their end-services, the TE and the AE, contributed to most of the processing time, which shows that the ACM's Client Services are the components of the ACM that can cripple the overall performance of this service. The reason behind this is the fact that both the TE and AE are currently operating 24/7 on the testbed under heavy load, which affects the results of the load tests. Even so, it is expected that these components to be under heavy load on a regular basis and the truth is that the ACM's performance may be compromised or limited by the very own performance of the end services it is connected to. Nevertheless, the average number of operations per second obtained for the ACM's Client Services is enough to cope with the project requirements. In summary, the two components that were the focus of this paper generally fulfilled their purposes and the requirements of the SELFNET project, and they can cope with the demanded throughput, despite their limitations. There is room for improvement, although they are prototypes to assert the validation of the proposed and designed architecture of the project, and hence they are not required to have stellar performances.

ACKNOWLEDGEMENTS

This work was funded by the European Commission Horizon 2020 5G-PPP Programme under grant agreement number H2020-ICT-2014-2/671672, SELFNET (Self-Organized Network Management in Virtualized and Software Defined Networks) [8] and by FCT/MEC through national funds and when applicable co-funded by FEDER PT2020 partnership agreement under the projects UID/EEA/50008/2013. The authors wish to thank all the SELFNET partners for their support in this work.

REFERENCES

[1] OpenAPI Initiative. OpenAPI Specification Website. [Online]. Available: https://github.com/OAI/OpenAPI-Specification

[2] ——. OpenAPI Initiative Website. [Online]. Available: https://www.openapis.org/

[3] A. Ronacher. Flask Website. [Online]. Available: http://flask.pocoo.org/

[4] Zalando SE. Connexion Website. [Online]. Available: https://jobs.zalando.com/tech/blog/meet-connexion-our-rest-framework-for-python/?gh_src=4n3gxh1

[5] Neo4j, Inc. Neo4j Website. [Online]. Available: https://neo4j.com/

[6] Apache Foundation. Apache Zookeeper Website. [Online]. Available: https://zookeeper.apache.org/

[7] MongoDB, Inc. MongoDB Website. [Online]. Available: https://www.mongodb.com/

[8] 5G-PPP. SELFNET - Framework for Self-Organized Network Management in Virtualized and Software Defined Networks. [Online]. Available: https://selfnet-5g.eu