Towards an Autonomic Bayesian Fault Diagnosis Service for SDN Environments based on a Big Data Infrastructure

Fernando Benayas, Álvaro Carrera and Carlos A. Iglesias
Intelligent Systems Group
Universidad Politécnica de Madrid,
Av. Complutense, 30, 28040, Madrid, Spain
f.benayas@upm.es, a.carrera@upm.es, carlosangel.iglesias@upm.es

***Abstract*—Software Defined Networks (SDN) are gaining momentum as a solution for current and future networking issues. Its programmability and centralised control enables a more dynamic management of the network. But this feature introduces the cost of a potential increase in failures, since every modification introduced on the control plane is a new possibility for failures to appear and cause a decrement of the quality for the offered service. Following a classical approach, this kind of problems could be solved increasing the number of high skilled human operators, which would dramatically increase network operation cost. Our approach is to apply Machine Learning and Data Analysis for monitoring and diagnosis SDN networks with the goal of automating these tasks. In this paper, we present an architecture for a self-diagnosis service which is deployed on top of a SDN management platform. In addition, a prototype of the proposed service with different diagnosis models for SDN networks has been developed. The evaluation shows encouraging results which will be explored in future works.**

***Index Terms*—SDN, Bayesian network, fault diagnosis, Machine Learning.**

I. Introduction

Computer networks are facing challenging times [1]. The increasing presence of Internet of Things (IoT) devices in everyday life and the expected growth in the number of mobile devices will be a considerable burden in current networks [2]. This is further aggravated by the adoption of 5G technologies and the development in video definition [3]. Moreover, progressive adoption of cloud services magnifies traffic overload. In fact, cloud traffic will reach 14.1 ZB in 2020, almost quadrupling in the 2015-2020 period [4]. In order to face these challenges, flexible networking policies are needed [5]. SDN provides a never seen before level of flexibility in computer networking [6]. This would enable fast re-routing depending on traffic demand and congestion. SDN decouples forwarding and control planes, centralises the latter and exposes it through an API. Therefore, a more responsive network management is possible by using software systems that interact with such APIs according to the data collected from the network and instructions or policies retrieved from managers. This is a vast improvement over legacy networking, where forwarding rules had to be manually configured in each network device by human operators.

As a consequence of these facts, networking industry is taking interest in SDN. According to a recent study [7], the adoption of SDN and Network Function Virtualization (NFV) is currently high up in the strategic technology agenda of major telcos and Multiple Service Operators (MSO). Furthermore, more than 80% Cloud Service Providers (CSP) network executives consider significant the impact of NFV and SDN to the operating model, and more than 50% of them are going to invest in these technologies for data center and mobile core [8]. Hence, the SDN and NFV market is expected to grow at a Compound Annual Growth Rate (CAGR) of 71.4% in the 2017-2022 period, reaching USD 54.41 billion of market value [9].

Notwithstanding the advantages mentioned before, SDN has some issues of resiliency. Frequent changes in networking rules made by software systems entails constant possibilities for faulty traffic policies to be introduced in the network. This could severely hinder the benefits of switching from legacy to SDN technology. Hence, a system that allows users auditing these faults is needed. This is the motivation behind the architecture proposed in this paper.

This paper is structured as follows. In Sect. II, we present related works in the field of failure management in SDN environments. Sect. III proposes an architecture for fault diagnosis based on Bayesian inference, which is applied in the prototype exposed in the case study in Sect. IV. Sect. V shows the results of evaluation of the diagnosis module. Finally, Sect. VI summarises some conclusions and explore possible paths for future work.

II. Related Work

Most of the research in fault diagnosis in SDN scenarios focuses on links/switches failover within the data plane, many aiming at agility improvement by decentralising tasks. For example, Kempf et al. [10] proposes that link failover is detected (and fixed) in switch level instead of the controller using Bidirectional Fowarding Detection (BFD) instead of Link Layer Discovery Protocol (LLDP) in order to improve restoration time. A similar approach is taken by Kim et al. [11], which also use switches for tasks that do not require

full knowledge of the network, switch and link failures are detected by LLDP, and repaired by calculating multiple paths with updated information. Cascone et al. [12] uses "heartbeat" packets sent by nodes to detect faulty routes: when a node does not receive them and packet rate drops below a threshold, a node can request heartbeats to detect if the link is down. Then, a "link down" message is sent along the primary path until a node can found a detour for that path.

Similarly, Capone et al. [13] proposes an extension to the OpenFlow protocol, called "OpenState". It introduces state machines into OpenFlow switches, triggering transitions by packet-level events. Therefore, when a node receives notification of a packet affected by a failure in some node or link, it activates a state transition that creates a pre-computed detour for that flow. This alternative path is calculated considering link capacities, and cycle and congestion avoidance.

In contrast, some effort has been put also in centralised management of link failures. Sharma et al. [14] propose that restoration and protection recovery methods for in-band networks are fully implemented at the controller, instead of delegating them to the switches, whose only involvement in the process is to detect failures through Loss of Signal (LOS) and BFD.

Another interesting approach to malfunctioning link detection is the one proposed by SDN-RADAR [15], which relies on agents, acting as clients that report bad performance on services consumed by them. When bad performance is reported, the system extracts and stores the path defined for the network flow related to each client and service, finding the link that appears in the most number of 'faulty' flows. Withal the original approach on failure detection using agents feedback, this work focuses only in malfunctioning links.

In the previous research works, failure was only considered in links or switches. Howbeit, this concept is severely limited, since failures could happen in any level of a SDN system: consequently, some research has been made in fault detection in SDN where failures at other levels are considered. Tang et al. [16] propose a system mapping by combining the network topology with the "Policy View" of each service. These views are reports consisting in a pair of logical end nodes, traffic pattern specifications for the service, and a list of required network functions for the provisioning of the service. This system mapping is known as "Implementation View". Using this view and a SDN reference model, a belief network is built for each service. Many belief networks are combined to infer the root cause and the fault location. Following the same global approach, Beheshti and Zhang [17] propose a system that studies all the possible locations of the controller and their related routing trees (Greedy Routing Tree, a modified shortest-path tree for more resiliency), choosing the location with the most resilient tree.

A different method is proposed by Chandrasekaran and Benson [18], considering SDN application failures instead of link/nodes ones. In order to achieve fault tolerance for SDN applications, a new isolating layer, called "AppVisor", is defined between applications and SDN controller. It separates the address space between both controller and applications by running them in different Java Virtual Machines (JVM), so software crashes are contained into their JVM, but keeps communication open between them. It also defines a module which enables network transforming transactions between application and controller to be atomic and invertible. Therefore, an application crash in the middle of a network transforming message does not affect the network itself. Lastly, it defines a module (the Crash-Pad module) that takes application snapshots before it process an event, so it reverts to the previous state in case of failure during the event processing. In order to avoid repeating the same failure, the snapshot is modified. This can be done by either ignoring the failure-inducing event or modifying it accordingly to pre-defined policies.

There is also other interesting works for our proposal in other fields related with different SDN aspects. For instance, one common approach consists in dealing with the issue of a centralised controller being a single point of failure, as presented by Botelho et al. [19]. They propose an architecture with a back-up controller and replicated data stores. A similar system is proposed by Li et al. [20]. In this paper, many controllers are defined for a single network, but in this case the network is also protected against "hacked". Katta et al. [21] consider a situation where a controller fails during handling of a switch state. This is because the entire event-processing cycle is managed as a transaction, instead of just keeping a consistent state between the controllers. Nevertheless, all of them are passive systems. In contrast, an active approach to the single point of failure issue has been proposed by Bouacida et al. [22]. In this case, different classification algorithms are able to predict if work loads at the controller (as the ones caused by the simultaneous introduction of multiple new flows) are short or long-term loads, which is quite useful since the latter is capable of bringing down the controller and shutting down the network if none actions are taken.

Finally, it is interesting the approach proposed by Sánchez et al. [23], where a self-healing architecture for SDN is presented, including monitoring of data, control and service levels. In this architecture, self-modelling techniques are applied to dynamically build diagnosis models synthesised as Bayesian networks. These models are based on topology information requested from the controller. Then, the diagnosis result is sent to the recovery block, which chooses the appropriate strategies to fix the failure diagnosed. In order to complement this approach, our proposal focuses more on the behaviour of the network than in the topology itself, as described below.

As we can see, most of the related work focuses on managing link failures, or failures that completely disable SDN elements (either links, nodes, SDN controllers, or applications). We would like to take a different approach, extending failure diagnosis to faulty traffic configurations (which could hamper the provision of specific services within the network). We also intend to address the design of a self-healing service.

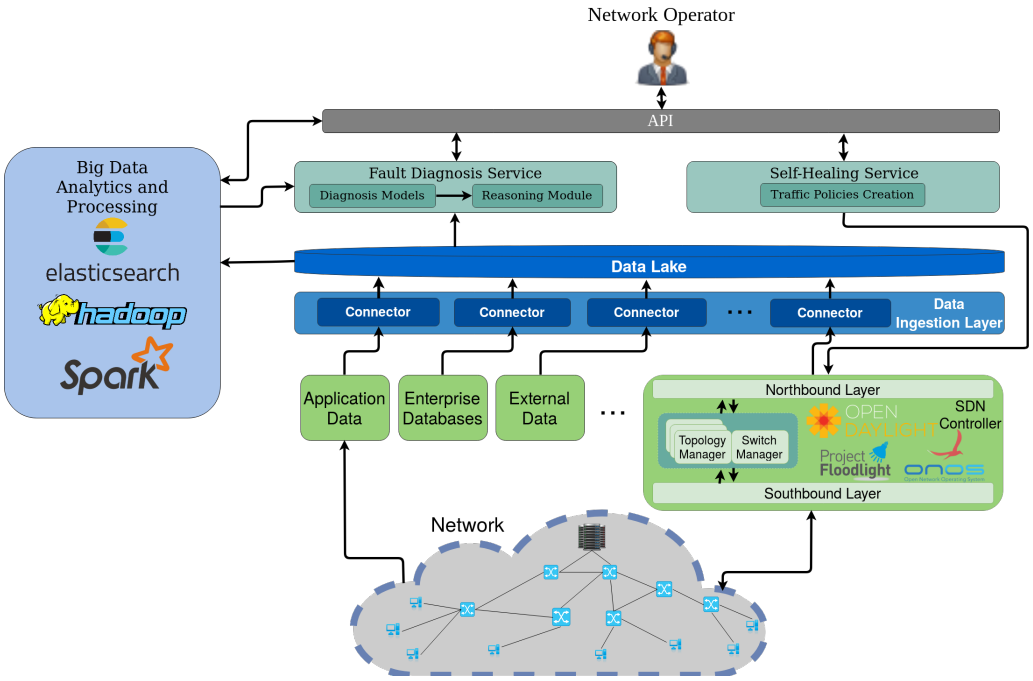

Figure 1: Overview of the proposed Fault Diagnosis Architecture.

III. Fault Diagnosis Architecture

This section presents a Fault Diagnosis Architecture based on Bayesian Reasoning for SDN Environments. Fig. 1 shows an overview of the architecture which is based on a Data Lake Architecture which ingests data from different sources, such as enterprise databases, application monitoring systems, or SDN controllers which are used to monitor the network behaviour. Data collected from these sources is ingested to the Data Lake and analysed applying Big Data techniques to provide refined and useful data and models to the Fault Diagnosis Service, which will start a complete diagnosis process when any symptom is detected analysing data in the Data Lake. Models required to reasoning about fault root causes are generated using Machine Learning algorithms inside the Big Data Analysis and Processing Module. Once those models are available, they are used to infer a possible cause of fault in the network. This result can be used by Self-Healing Service, which implements network policies in order to fix the failure or mitigate its effect, or by a human operator. The following subsections detail the main modules of the proposed architecture.

A. Data Ingestion Layer and Data Lake

The aim of this layer is to implement connectors to ingest data from all sources to the Data Lake, which is a large storage repository that holds a vast amount of raw data in its native format until it is needed. Every data source must use its own connector to ingest data in streaming or in batches. If the ingestion is performed through a real time streaming, each data item is imported as generated by the source. But, if it is performed in batches, data items are imported in bundles at periodic time intervals.

In our case, one of the most important data sources is the SDN Controller. It performs monitoring tasks over the network and provided detailed information about its status. We propose collecting the data via requests to the SDN controller, using its northbound API. Here there are several possible information sources. One of them is requesting the data directly to the nodes. In this case, we would have to design a module that implements the OpenFlow protocol, in order to communicate with network nodes. The other one is to collect some metrics using probes directly in the physical network. This could be used to measure congested or abnormally empty links, or resource's consumption at nodes.

B. Big Data Analytics and Processing

Independently of the data source, collected data must be stored and analysed. Big Data software platforms, such as ElasticSearch, Spark or Hadoop [24], must be used to index and classify high volume of collected data in order to simplify further processing. This would also facilitate conversion to multiple formats, which would enhance collaboration with external data units and diagnosis modules, and improving general flexibility.

As mentioned previously, network traffic and status data must be processed to be ready to be used in the reasoning module. Moreover, raw data contains much non-relevant information for reasoning process. That information could be not only useless, but also pernicious, since it could lead to wrong reasoning processes with a lot of noise and false statistical patterns. Also, some variables may have missing values due

to network failures altering the ability to monitor them. To avoid such cases, historical data could be used as knowledge base.

Summarising, we propose the following features for this module. Based on data collected from the SDN environment, we process them and select specific variables that can suggest a type of failures. Some of them will need further processing, such as time series analysis, ontology-based reasoning or other variables can be directly discretised, assigning a class depending on the range that contains the value. Finally, processed data is converted to the adequate format to generate diagnosis models or to infer the most probable cause of fault using those models.

C. Fault Diagnosis Service

The reasoning process for hypothesis discrimination required to that essential phase of any diagnosis process is carried out in this module inferring with data provided by the processing module presented in Section III-B. Multiple reasoning techniques can be used for this task. For instance, rule-based reasoning provides a straightforward method to express domain knowledge. That knowledge can be represented as rules expressing cause-effect relationships. However, it is incapable of dealing with situations uncovered by those rules. Other alternative is the application of probabilistic reasoning techniques. Specifically, causal models based on Bayesian reasoning are interesting for our uncertain and complex environment. They make a heuristic model that relates symptoms and cause of fault, obtaining the probability of a failure based on those observed variables.

Thus, we apply Bayesian networks as *causal models* for fault root cause inference models. This reasoning process starts with information retrieved from the network and properly processed in the Reasoning Module, as explained in Section III-B. Then, this information is sent as a set of evidences to the Reasoning Module. In the Reasoning Module, a Bayesian network model will infer a hypothesis of the most probable status of the network. This status will be predicted according to a Conditional Probability Table (CPT), where the conditional relations between the values of the set of evidences and the status of the network are stored. This diagnosis is offered through an API to human operators and other modules.

D. Self-Healing Service

After diagnosing a failure in the network, some actions must be taken to either fix or bypass the failure. Therefore, network policies are modified by this module. Information about failure, guidelines provided by network managers and further information about network are taken into account to decide any policy's change. Then, these policies are sent to the controller for their application. We take advantage of the ability of SDN controllers to present a centralised API where we can inject new traffic policies. In a legacy network, we would have to send policies to each network element separately, thus creating scalability issues. Some reasoning is also needed in this module, but requirements for this reasoning process are quite different to the ones required by the hypothesis discrimination process performed in the *Fault Diagnosis Service*. Thus, here is not required to use probabilistic reasoning techniques and other approaches can be considered. But this module could be not completely autonomous since some decisions can be delegated to human operators. For example, network policy templates could be provided by network managers.

The current version of the proposed architecture does not focus on this module as exposed in the case study in Section IV. Thus, further work must be done to define appropriate reasoning techniques for decision making about network policies updates.

IV. Case Study

The proposed architecture has been implemented as a prototype based on a simulated network environment. As mentioned above, we have focused on covering data ingestion and processing and generating diagnosis results, leaving self-healing and decision making about reconfiguration for future work.

The following subsections detail the experimentation testbed as follows. Sect. IV-A presents the network simulation module to generate realistic topologies and traffic. Sect. IV-B details the selected SDN controller used in the case study. Then, Sect. IV-C specifies the data selected to be processed in the *Big Data Analytics and Processing* module. Finally, Sect. IV-D explains how the *Reasoning Module* of the Fault Diagnosis Service manages the Bayesian networks used to reason about the most probable cause of fault.

A. Network Simulation

Our experiments will be carried out with synthetic data obtained from a network simulation environment. For this purpose, we have selected the network virtualization tool Mininet [25] which allows users to simulate an entire SDN network in a single computer. This tool is appropriate for SDN scenarios, since OpenvSwitch nodes and external controllers are supported. An OpenvSwitch switch is a virtual switch that implements the OpenFlow protocol among others. It also provides an API, through which a plethora of features can be configured. We will use this API to define a scale-free network composed by multiple subnetworks. These scale-free networks have several benefits, such as high error tolerance and very small diameter [26]. Then, we will connect that network to an external SDN controller.

A scale-free subnetwork will represent the role of the core network of the simulated Internet Service Provider (ISP), as exemplified in Figure 2. This subnetwork will have a random, but configurable, size (i.e. number of nodes, connections and datacenters). Each datacenter will be represented by three hosts. Every of those host provides a streaming service using Real Time Session Protocol (RTSP) over Real Time Protocol (RTP). Furthermore, background traffic such as chat, web or Peer to Peer (P2P) will be emulated using Socat tool. Then, a configurable number of sub networks will be joined to that "core" network. They represent ISP's "last mile" networks.

The number of nodes and hosts (representing end users) is also configurable. These hosts will communicate among themselves and will establish connections with streaming servers housed in datacenters.

To generate faulty data, some failures will be purposely injected and fixed in the network periodically. Those faults will be executed periodically either through the SDN controller or directly at network level through Mininet API. For simulation purposes, the occurrence of these failures is also configurable. These changes will be also stored in a network log to enable machine learning with collected data and then be able to generate causal models as explained in Section IV-D.

B. SDN Controller

Regarding the SDN controller, we have used Opendaylight [27], supported by companies such as Cisco, Ericsson, Huawei or ZTE. Opendaylight is designed around a Model-Driven Service Abstraction Layer (MD-SAL) that describes network devices and applications as objects. These models are defined using YANG modelling language. Since a generalised description of a device or application is given, the specific implementation does not need to be known. Interactions with these objects (or models) are processed within the MD-SAL. Opendaylight communicates with the network through "southbound" interfaces anchored to the MD-SAL. These interfaces implement the protocol used for communication with devices in the network.

Inside the controller platform, multiple service functions are defined.

We use some of them to oversee the network status. Specifically, *Topology Manager*, *Switch Manager* and *Host Tracker* functions will allow us to monitor the network and collect data. Access to the data granted by these functions is provided by the Opendaylight controller through its "northbound" Representational State Transfer (REST) API. This API allows not only obtaining data, but also make changes in the network to update configuration and policies if required.

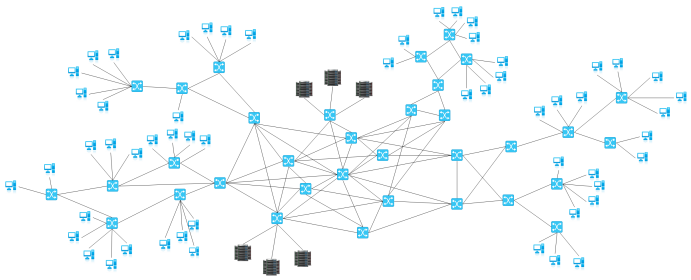

Figure 2: Example of the random network generated for simulation purposes.

C. Data Ingestion from SDN Controller

The main component of this module is the indexing engine provided by ElasticSearch which is used to store data from both network simulator and controller, as explained below. At simulation beginning, two essential data requests about topology and nodes status reports are sent to the controller. These requests are made periodically until simulation finishes. The network log is also monitored, storing every modification made in the network, both generated or fixed errors. This information is used to train the causal models used in the *Fault Diagnosis Service*. This monitoring is implemented using Filebeat [28]. Filebeat monitors the log file, and sends a "beat" (i.e. a message) to ElasticSearch whenever a new message is written into it.

Then, data processing is implemented getting time-ordered reports from Elasticsearch for specific fields that have been selected according to their capability to show changes provoked by failures introduced in the network. Some fields valueless for the diagnosis are ignored. For this case study, the output variables of this module are the followings for every switch: presence of flows, dropping rules in LLDP flows, presence of any kind of timeout, recent changing of in-ports, out-ports and priorities in a given time window, and the presence of hosts in the network. After data processing, these variables are sent to the *Reasoning Module* to start the hypothesis discrimination process for every switch.

D. Fault Diagnosis Service

The goal of the reasoning module of the Fault Diagnosis Service is to infer the correct output (failure in the network) given a set of inputs (network status datasets). For this case study, we have focused on the development of a switch diagnosis model. In further developments, this model can be combined with other causal models for specific services, platforms or network topologies.

We have applied supervised learning algorithms to generate different models and evaluate them, as shown in Section V. This consists in providing training datasets with status-fault pairs. These labelled data have been generated by the *Big Data Analytics and Processing Module* which receives information from both network controller and network simulator. Then, training is performed in order to obtain a Bayesian network model that relates the status of the network with a specific kind of fault.

The Bayesian model is described using a Directed Acyclic Graph (DAG), which is a graph composed by nodes and edges. Each node is a variable considered relevant for the decision process. Each edge connects two nodes, is directed from one to another, and represents dependency of the destination node on the origin node. In a DAG, a cycle of dependency is not allowed, since a node can not depend on itself.

Our model can diagnose a generic switch. Therefore, when applying it to a network, we will be able to diagnose the failure, but we need know what node is the source of those status variables. This can be done by instantiating different Bayesian inference processes for each node in the network, and combining the outputs of each one.

The learning and reasoning library used in this prototype was GeNIe [29], which is a modelling and learning software for Bayesian networks and influence diagrams. Particularly,

we have applied the *BayesianSearch* supervised learning algorithm for network learning on a training dataset obtained from different simulations. This algorithm will attempt to search the full space of graphs for the best graph. The suitability of each graph is measured using the accuracy of the model as a scoring function. This accuracy is measured using a 10-fold cross-validation technique. It consists in partitioning the dataset in 10 fragments, using nine fragments for training of the model, and one fragment for testing. This process is repeated 10 times, each time using a different fragment as 'test' fragment.

V. Evaluation

In this section, we evaluate the quality of two different models for switch faults. These faults generated in the simulated network are the followings:

- **S0** - No faults - OK status.
- **S1** - Shutting down a node.
- **S2** - Disconnecting a datacenter server from the network.
- **S3** - Modifying the out-port rules in a node.
- **S4** - Modifying the in-port rules in a node.
- **S5** - Adding idle-timeouts in a node.
- **S6** - Adding hard-timeouts in a node.
- **S7** - Changing flow priorities in a node.
- **S8** - Forcing a node to drop LLDP packets.
- **S9** - Modifying both out-port and in-port rules in a node.

We feed the following information to the models: the presence of flows (represented by *existence_of_flows*), changes in the number of hosts in the network (*modified_hosts*), changes in the output and in ports in any flow (*changed_output* and *changed_inport*), the detection of rules involving dropping packages with a 35020 Ethernet code (*not_dropping_lldp*), changes in any timeout (*modified_timeout*), changes in the priority order of the flows (*changed_priority*) and any change in any flow (*changed_flow*), as shown in Figs. 3 and 4.

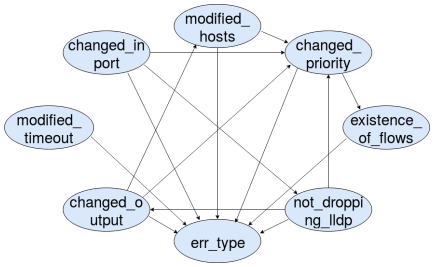

Figure 3: DAG for Switch Diagnosis - Model 1.

Two models have been evaluated in this experiment. Model 1 makes use of all variables obtained in the *Big Data Analytics and Processing Module*, as shown in Fig. 3. Since this model could have a low generalisation capability, we have explored the impact of reducing the number of collected and processed variables. Thus, we have defined Model 2 shown in Fig. 4. In this model, instead of including the details of flow changes, binary variables are defined to state if a flow has been modified.

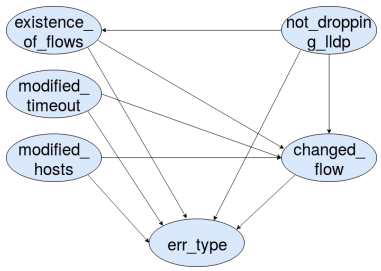

Figure 4: DAG for Switch Diagnosis - Model 2.

These models have been validated using a balanced dataset with 184 fault diagnosis cases from simulated networks. The results obtained from both models can be seen in Table I and Table II. These tables show different metrics for both causal models. The overall metrics of both models can bee seen in Table III.

Fault Type	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
F1-Score	0.69	0.75	1.00	0.96	0.95	1.00	0.95	0.94	0.93	0.87
Recall	0.59	1.00	1.00	1.00	1.00	1.00	0.90	1.00	1.00	0.87
Precision	0.83	0.6	1.00	0.92	0.90	1.00	1.00	0.89	0.88	0.87
Accuracy	0.90	0.97	1.00	0.99	0.98	1.00	0.99	0.99	0.98	0.98

Table I: Metrics for Model 1.

Fault Type	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
F1-Score	0.77	0.89	1.00	0.31	0.00	0.97	0.90	0.00	0.86	0.00
Recall	0.69	0.89	1.00	1.00	0.00	0.94	0.83	0.00	1.00	0.00
Precision	0.87	0.89	1.00	0.18	0.00	1.00	1.00	0.00	0.76	0.00
Accuracy	0.92	0.99	1.00	0.72	0.86	0.99	0.98	0.96	0.95	0.92

Table II: Metrics for Model 2.

Model	M1	M2
F1-Score	0.904	0.570
Recall	0.936	0.635
Precision	0.889	0.573
Accuracy	0.978	0.929

Table III: Comparison of both models.

As shown in Table III, Model 1 significantly outstrips Model 2. Taking a closer look into Table I and Table II, Model 1 obtains perfect scores in the diagnosis of S2 and S5 statuses. However, Model 2 obtains better results in all four metrics for the S0 status diagnosis. Particularly, Model 2 shows a better recall in S0 fault diagnosis than Model 1, due to more S0 statuses being diagnosed as such. Model 2 is also more effective for S1 faults. In S1 diagnosis in Model 1, a recall of 1.00 combined with a precision of 0.6 is a symptom of some false positive predictions. However, in the rest of the cases, Model 1 shows similar or better results than Model 2.

Model 2 shows similar satisfactory results in the diagnosis of most status. In fact, as stated before, it shows better results for S0 and S1 faults. However, it struggles in the diagnosis of S3, S4, S7 and S9 statuses. This shows that, as expected, there is a compromise between generalisation capability and effectiveness. A more-specific model, as Model 1, shows better results than Model 2.

VI. Conclusions and Future Work

New technologies such as SDN provide a number of benefits in the virtualisation and management of network services. Nevertheless, it is needed to research on the application of techniques for enabling its autonomic management. For this purpose, Big Data technologies provide the foundation for collecting and processing huge amounts of raw data from the telecommunication network.

In this article we have proposed a Big Data based architecture that takes advantage of Big Data technologies and explores the use of Big Data analytics based on Bayesian networks. Main effort has been dedicated to the creation of the testing environment, since there are not yet available benchmarks for diagnosis purposes.

We have implemented a prototype which is presented in the case study section. Finally, two diagnosis models have been generated following different generalisation criteria. Their evaluation showed the more specific model has better accuracy while non-variable reduction is performed in the data processing phase.

Therefore, the proposed architecture shows potential for failure diagnosis in SDN. However, this work is still in progress. Next steps in our research line include combining diagnosis models to cope complex cases, not only switch faults. This can be done by further processing more data sources, such as final user applications, including probes in the network and/or servers or implementing testing agents which could execute specific tests when symptoms or anomalies are detected in the network. Moreover, the *Self-Healing Service* is another important aspect that will be addressed as future work.

Acknowledgments

This research has been funded by Spanish Ministry of Industry, Energy and Tourism under the R&D project BayesianSDN (TSI-100102-2016-12), the Spanish Ministry of Economy through the R&D project SEMOLA (TEC2015-68284-R) the Autonomous Region of Madrid through programme MOSI-AGIL-CM (grant P2013/ICE-3019, co-funded by EU Structural Funds FSE and FEDER). We also acknowledge the use of an academic license of the GeNIe bayesian modeler.

References

[1] S. Alexander, "When networks hit the wall," *NetworkWorld*, Sept. 2017.

[2] Cisco VNI, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021," *Cisco White Papers*, 2016.

[3] Cisco, "The zettabyte era: Trends and analysis," *Cisco White Papers*, 2016.

[4] Cisco, "Cisco Global Cloud Index: Forecast and Methodology, 2015–2020," *Cisco White Papers*, 2015.

[5] A. Vidal, "Flexible networking hot trends at SIGCOMM 2016," *Ericsson Research Blog*, 2016.

[6] S. Baines, "Enterprises want SDN to build flexible networks," tech. rep., Orange, June 2015.

[7] F. Groene, C. Isaac, H. Nalinakshan, and J. Tagliaferro, "The software-defined carrier: How extending network virtualisation architecture into IT BSS/OSS architectures opens up transformational opportunities for telecom and cable operators," *Communications Review*, Dec. 2016.

[8] A. C. D. C. S. 2015, "Network transformation survey 2015, final results," tech. rep., Accenture, 2015.

[9] Markets and Markets, "Software-Defined Networking and Network Function Virtualization Market by Component (Solution (Software (Controller, and Application Software), Physical Appliances), and Service), End-User, and Region - Global forecast to 2022," 2017.

[10] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takács, and P. Sköldström, "Scalable fault management for openflow," in *Communications (ICC), 2012 IEEE international conference on*, pp. 6606–6610, IEEE, 2012.

[11] H. Kim, M. Schlansker, J. R. Santos, J. Tourrilhes, Y. Turner, and N. Feamster, "Coronet: Fault tolerance for software defined networks," in *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pp. 1–2, IEEE, 2012.

[12] C. Cascone, L. Pollini, D. Sanvito, A. Capone, and B. Sanso, "Spider: Fault resilient sdn pipeline with recovery delay guarantees," in *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*, pp. 296–302, IEEE, 2016.

[13] A. Capone, C. Cascone, A. Q. Nguyen, and B. Sanso, "Detour planning for fast and reliable failure recovery in sdn with openstate," in *Design of Reliable Communication Networks (DRCN), 2015 11th International Conference on the*, pp. 25–32, IEEE, 2015.

[14] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Fast failure recovery for in-band openflow networks," in *Design of reliable communication networks (drcn), 2013 9th international conference on the*, pp. 52–59, IEEE, 2013.

[15] G. Gheorghe, T. Avanesov, M.-R. Palattella, T. Engel, and C. Popoviciu, "Sdn-radar: Network troubleshooting combining user experience and sdn capabilities," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pp. 1–5, IEEE, 2015.

[16] Y. Tang, G. Cheng, Z. Xu, F. Chen, K. Elmansor, and Y. Wu, "Automatic belief network modeling via policy inference for sdn fault localization," *Journal of Internet Services and Applications*, vol. 7, no. 1, p. 1, 2016.

[17] N. Beheshti and Y. Zhang, "Fast failover for control traffic in software-defined networks," in *Global Communications Conference (GLOBECOM), 2012 IEEE*, pp. 2665–2670, IEEE, 2012.

[18] B. Chandrasekaran and T. Benson, "Tolerating sdn application failures with legosdn," in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, p. 22, ACM, 2014.

[19] F. Botelho, A. Bessani, F. M. Ramos, and P. Ferreira, "On the design of practical fault-tolerant sdn controllers," in *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, pp. 73–78, IEEE, 2014.

[20] H. Li, P. Li, S. Guo, and A. Nayak, "Byzantine-resilient secure software-defined networks with multiple controllers in cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 4, pp. 436–447, 2014.

[21] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: Controller fault-tolerance in software-defined networking," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, p. 4, ACM, 2015.

[22] N. Bouacida, A. Alghadhban, S. Alalmaei, H. Mohammed, and B. Shihada, "Failure mitigation in software defined networking employing load type prediction," in *Communications (ICC), 2017 IEEE International Conference on*, pp. 1–7, IEEE, 2017.

[23] J. M. Sánchez, I. G. B. Yahia, and N. Crespi, "Thesard: On the road to resilience in software-defined networking through self-diagnosis," in *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*, pp. 351–352, IEEE, 2016.

[24] D. Singh and C. K. Reddy, "A survey on platforms for big data analytics," *Journal of Big Data*, vol. 2, no. 1, p. 8, 2015.

[25] K. Kaur, J. Singh, and N. S. Ghumman, "Mininet as software defined networking testing platform," in *International Conference on Communication, Computing & Systems (ICCCS)*, pp. 139–42, 2014.

[26] L. Gyarmati and T. A. Trinh, "Scafida: A scale-free network inspired data center architecture," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 5, pp. 4–12, 2010.

[27] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pp. 1–6, IEEE, 2014.

[28] V. Sharma, *Beginning Elastic Stack*. Springer, 2016.

[29] BayesFusion, "Genie modeler." Available at https://www.bayesfusion.com/, 2017.