LogiSig: Automatic Interaction Detection in Software-based Networks

Daphne Tuncer[*], Marinos Charalambides[†]

[*]Department of Computing, Imperial College London, UK
[†]Department of Electronic and Electrical Engineering, University College London, UK

***Abstract*—Emerging services with demand for high performance and resource availability call for network management solutions that enable distributed, flexible and adaptive implementation of their components. Software-based networks, characterized by modular structures, are key in supporting management functionality with the desired level of agility. An important challenge in realizing a modular management plane is the handling of interactions between independent decision-making applications operating in parallel, which can compromise service delivery. This paper presents LogiSig, a novel approach to automatically detect and classify interactions manifesting in the management plane of software-based networks. LogiSig builds upon a fine-grained taxonomy of possible interaction types and simple system-level operations to realize its functionality. Our evaluation results show that LogiSig can efficiently determine interactions between application pairs.**

I. Introduction

Recent years have witnessed the rapid emergence of new classes of services - from real-time big data analytics for Internet-of-Things infrastructures to ultra high-definition video streaming - characterized by stringent requirements in terms of resource availability (*e.g.,* high-bandwidth) and performance (*e.g.,* ultra-low latency, high-throughput), as well as strong quality of service guarantees [1]. Compared to dedicated middle-boxes, virtualized network functions can better cope with the requirements of such services due to their inherent flexibility in creating and modifying function instances on the fly in response to service demand. At the same time, the more dynamic nature of network functions introduces additional complexity with which current monolithic resource management solutions are not able to cope. The management plane should instead exhibit more agile properties, which can allow network functions to be added, removed, updated and deployed when and where needed.

A key design choice for meeting the desired level of flexibility is modularity, where individual resource management applications (*e.g.,* for server/path selection, energy saving, content placement, *etc.*) are implemented in a modular fashion. Such a fine grained management functionality is an important feature of software-defined networking (SDN), which builds on modular structures to implement the network architecture [2][3]. In particular, applications in the management plane are deployed as independent modules, each responsible for a specific resource configuration function. Despite having significant advantages (*i.e.,* extensibility, flexibility), these modular implementations raise questions regarding the interoperability between multiple processes operating in the same environment. Due to potential overlaps in the resources being configured, incompatible objectives or synchronization issues, interactions can occur between seemingly independent management applications (MAs). This can adversely affect the network operation as well as the delivery of services and several studies have acknowledged that devising mechanisms to handle these interactions is a challenging problem in SDN *e.g.,* [4][5].

Interactions between MAs can have different characteristics, especially with respect to how the applications and/or the environment are affected. As such, applying the same solution to mitigate all interaction types is likely to provide inefficient performance [6]. Detecting and classifying interactions is thus an important step that can subsequently guide the selection of the most effective mitigation strategy. In this paper, we propose LogiSig, an approach to automatically detect interactions manifesting in the management plane of software-based networks. The core of LogiSig is a taxonomy of possible interaction types, which is based on a model of the MAs attributes. In LogiSig, each interaction type is formally represented in terms of a logical signature that allows the detection to be systematically achieved by comparing attributes of MAs against all signatures. The proposed taxonomy provides a fine-grained classification of interactions and has been integrated in an implementation of the detection functionality, which can be used to efficiently and automatically determine interacting applications. Our solution can be easily realized as a component of any framework for the management of interactions in software-based networks.

The rest of the paper is organized as follows. Section II provides an overview of typical interaction management frameworks. Section III presents our model of management applications. Section IV describes the proposed taxonomy of interaction types and Section V presents the automatic interaction detection functionality along with evaluation results. Related literature is discussed in Section VI and conclusions are reported in Section VII.

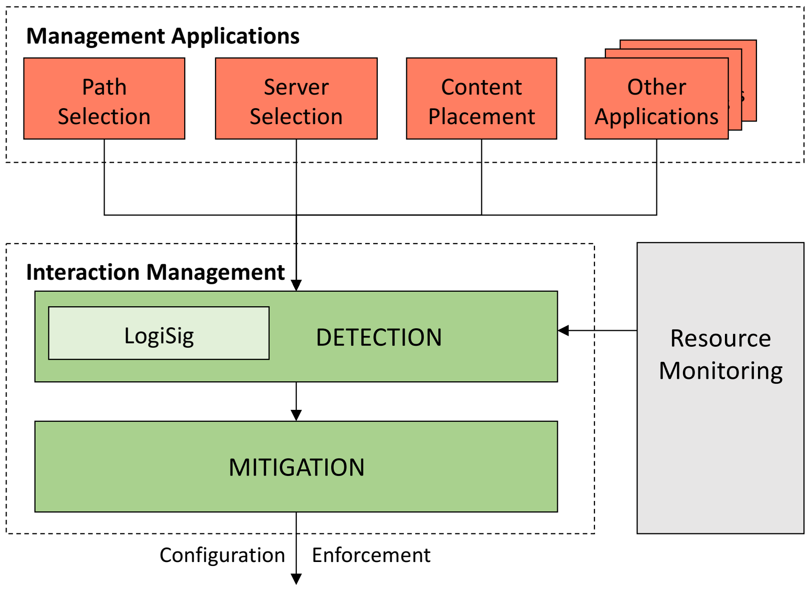


Fig. 1. Interaction management and LogiSig.

II. Interaction Management in a Nutshell

The management of interaction typically involves two main functions:

- Detection - to identify dependencies between individual decision-making processes and to check for potential inconsistencies and/or conflicts;
- Mitigation - to resolve identified inconsistencies using different strategies such as resource partitioning, priority enforcement or coordination.

A high-level functional view of a typical interaction management framework is presented in Fig. 1. To support each function, the framework is interfaced with a monitoring functionality that provides information about the environment and current state of the resources, as well as with a set of management applications. Fig. 1 shows how LogiSig, our automatic interaction detection and classification approach, is placed in such a framework. As can be seen, LogiSig sits within the detection component. It complements its tasks by determining the type of interaction that can occur between any pair of MAs, which is further used to guide the selection of the resolution strategy best suited to handle the identified interaction. Deriving taxonomies to guide selection processes is a common approach in system design. In [7], where emergent misbehaviors in software systems are discussed, Mogul argues that creating a taxonomy of these misbehaviors and their typical causes is key to the development of better tools and methods for their "anticipation, detection, diagnosis and management". In the subsequent sections, we describe the taxonomy used in LogiSig to classify interactions and explain in detail how this is integrated within its functionality.

III. Modeling Management Applications

Management applications are characterized by different aspects, for example in terms of their relation with the environment they operate in, their decision logic and their objectives. In this section we present a model to represent their functionality in a general and formal way. The proposed model forms the basis of LogiSig.

A. *Existing models and limitations*

Different modeling approaches have been proposed in the context of interaction management to represent MAs. Existing models can be classified in four categories as shown in Table I, each focusing on capturing different attributes of the functionality of an application, such as its objective [8][9], its logic [10], or its interaction with the environment [11]. We quantitatively evaluate the *goodness* of each model in enabling different types of interactions to be automatically identified based on the three following metrics:

- Expressiveness, *i.e.,* how rich the model semantic is to describe interactions;
- Complexity, *i.e.,* how complex it is to identify interactions;
- Scalability, *i.e.,* how costly it is to check for interactions.

A model qualifies as *good* if it provides high expressiveness, low complexity and high scalability. None of the existing approaches satisfies these requirements. While the approach in [8] provides a simple semantic to characterize collaborative behavior between applications, it cannot be used to identify the root cause of interactions (*e.g.,* overlapping resources being concurrently configured). In [10], the authors propose a path-based representation that is generic enough to express the logic of MAs involved in path or flow management, but it does not extend well to other application types (*e.g.,*

TABLE I
OVERVIEW OF EXISTING MANAGEMENT APPLICATION MODELS.

Model Type	**Captured Attributes**	**Expressiveness**	**Complexity**	**Scalability**
Utility Function [8]	Application Objective	Low	Low	High
Optimization Problem [10]	Application Logic	Low	High	Low
Deterministic Finite-State Transducer [11]	Application Environment	High	High	Low
Declarative Program [9]	Application Objective	High	High	Low

content/VM placement). As such, it does not offer the right level of expressiveness and complexity to identify interactions. While the model presented in [11] offers a high level of expressiveness, the richness of its semantic (each application's parameter and action is represented in the state machine) makes it far too complex as a general and practical solution for interaction detection. Finally, the solution proposed in [9] requires significant human involvement in the process of translating MA functionality in a declarative language, thus compromising the complexity and scalability of the approach.

We address these limitations by proposing an alternative model for MAs that can be used to automatically detect and classify interactions while meeting all three aforementioned requirements.

B. Proposed model

We model each MA as a function f, constituting an abstracted representation of the internal logic of the application, and a set of parameters $\mathcal{P}$, denoting the information required by the application to operate. The proposed model satisfies two important properties: *i)* it ensures that applications can be uniquely identified, and *ii)* it can be used to determine if and how two MAs interact by explicitly taking into account, via the set $\mathcal{P}$, the relationship between the MA and its environment. More specifically, by representing the decision process as a function, the impact of the MA decisions on the environment can be captured while hiding the actual algorithmic complexity.

The set $\mathcal{P}$ defined for each MA encompasses different types of parameters representing not only the information needed by the application to operate, but also the configuration decisions and the effects of these decisions on the environment. The set $\mathcal{P}$ can be further decomposed into two disjoint subsets: the set of constants $\mathcal{K}$ and the set of variables $\mathcal{V}$.

The set $\mathcal{K}$ contains all the parameters which are used as inputs by the application but are neither configured nor affected by the decisions of the application. In practice, these can be long-lived parameters describing the infrastructure and/or short lived parameters with dynamic attributes characterizing the context in which the application operates (*e.g.,* storage allocation, routing configuration *etc.*). In general, constant parameters define constraints on the decisions that can be taken by an application. In contrast, the set $\mathcal{V}$ contains all the parameters which are either configured by the application, *i.e.,* configuration parameters, or affected by the application decisions, *i.e.,* dynamic resource attributes. For instance, the configuration parameters of a network traffic load-balancing application (*e.g.,* [12][13]) are the traffic splitting ratios and the affected parameters are the links utilization. We denote as $\mathcal{C}$ the set of configuration parameters and as $\mathcal{A}$ the set of parameters affected by the decisions of a MA. By definition, $\mathcal{C}$ and $\mathcal{A}$ are disjoint. Based on the introduced notation, we have:

$$\mathcal{P} = \mathcal{K} \cup \mathcal{V} = \mathcal{K} \cup \mathcal{C} \cup \mathcal{A}$$

It is worth highlighting that the sets $\mathcal{K}$ and $\mathcal{V}$ associated with a MA are only defined from the perspective of that MA. A constant parameter for one application can well be a variable parameter for another. In addition, the proposed model implicitly assumes that the value of affected parameters is driven by the value of configuration parameters. This means that two applications controlling the same set of configuration parameters affect the same resources. The converse is however not true, *i.e.,* resources can be affected by a different set of configuration parameters.

C. Discussion

Although the model proposed in this paper cannot render any of the specifics of the decision logic of a MA, it allows the relationship between a MA and its environment to be described in a general and yet detailed enough way, which is essential to understand and detect interactions. In contrast to the approach in [11], it offers a good trade-off between expressiveness and complexity by focusing exclusively on the parameters, not on the actions. In addition, while the effects of MA decisions on the environment are directly driven by the MA's high-level objective(s), these are not taken into account *per se* in our model, as opposed to the solutions in [8] and [9]. They are, however, implicitly translated in the variables set, which, by definition, represents the effects of the MA decisions. In general, we believe that the objective of an application (how to configure resources) is not the main deciding factor when analyzing the interactions. As shown in [8], inconsistencies between decisions of uncoordinated MAs can arise even if their objectives are aligned. Finally, compared to [10], it can be used to represent any type of application implemented as part of the management plane.

IV. Interaction Signatures

In LogiSig, each type of interaction is represented by a logical formula derived from the properties of the parameter sets associated with each co-existing MA. The formulas are used to effectively detect and classify interactions in a formal and automatic way. In this section, we present a taxonomy of their possible types and associated signatures.

In all cases, the interaction between two applications is symmetric, *i.e.,* if MA_1 interacts with MA_2 according to type *t*, then MA_2 interacts with MA_1 according to *t*. Also, two MAs can be involved in only one type of interaction. Without loss of generality, we denote as MA_1 and MA_2 two applications represented by the functions f_1 and f_2 with parameter sets $(\mathcal{K}_1 \cup \mathcal{C}_1 \cup \mathcal{A}_1)$ and $(\mathcal{K}_2 \cup \mathcal{C}_2 \cup \mathcal{A}_2)$, respectively. The signatures associated with each type of interaction are presented in Table II.

A. Absence of interaction

Two MAs do not interact if they do not share any parameters. In some cases however, a relationship may exist between MAs due to parameter overlap, but without the two actually interacting, *i.e.,* not influencing each other. This happens when the only parameters in common are in the intersection of the constant set of each application (*i.e.,* $\mathcal{K}_1 \cap \mathcal{K}_2$).

B. Interaction by precedence with unaffected resources

Application MA_1 is said to interact with application MA_2 according to the *Interaction by Precedence with Unaffected Resources* type if (i) at least one of the variable parameters of MA_1 is used as a constant parameter by MA_2 (precedence), and (ii) none of the variable parameters of MA_1 is a variable parameter of MA_2 (unaffected resources). For completeness, this type also includes the supplementary case of strictly unidirectional precedence between MA_1 or MA_2, denoted as (2) in Table II.

This interaction type is representative of the case where the execution of one MA is conditional on the execution of the other and where the two MAs have different configuration parameters that do not affect the same resources. Multi-path computation and adaptive traffic load-balancing [13][12] are illustrative examples of this interaction type: the paths to the destination need to be computed before load-balancing can decide on the volume of traffic to send across each path.

C. Interaction by precedence with affected resources

Application MA_1 is said to interact with application MA_2 according to the *Interaction by Precedence with Affected Resources* type if (i) at least one of the variable parameters of MA_1 is used as a constant parameter by MA_2 (precedence), (ii) at least one of the affected parameters of MA_1 is an affected parameter of MA_2 (affected resources), and (iii) none of the configuration parameters of MA_1 is a configuration parameter of MA_2 (unchanged configurations).

This is representative of the case where the execution of one MA is conditional on the execution of the other and where the two MAs have different configuration parameters that however affect the same resources. Such applications are in general strongly coupled and very likely to be both implemented in the management system. A typical example concerns content placement and internal request redirection, with both aiming at reducing network bandwidth usage while controlling different configuration parameters [14]. This type can be generalized to the case where either MA_1 or MA_2 can have precedence over the other.

D. Interaction with affected resources

Application MA_1 is said to interact with application MA_2 according to the *Interaction with Affected Resources* type if (i) at least one of the affected parameters of MA_1 is an affected parameter of MA_2 (affected resources), (ii) none of the variable parameters of MA_1 is a constant parameter of MA_2 (no precedence effect), and (iii) none of the configuration parameters of MA_1 is a configuration parameter of MA_2 (unchanged configurations).

In contrast to the previous types, the MAs can be executed independently in this case, *i.e.,* MA_1 does not need any inputs from MA_2 to operate and vice versa. Interaction arises due to the fact that both applications, while controlling different configuration parameters, affect the same resources. A representative example is the case investigated in [6] which focuses on the interaction between a traffic engineering application and a multimedia content server selection application.

E. Interaction with conflicting decisions

This type of interaction covers two cases:

a) *By configuration parameters*: Application MA_1 is said to interact with application MA_2 if at least one of the configuration parameters of MA_1 is a configuration parameter of MA_2.

b) *By configuration and affected parameters*: Application MA_1 is said to interact with application MA_2 if (i) at least one of the affected parameters of MA_1 is a configuration parameter of MA_2, (ii) none of the affected parameters of MA_1 is an affected parameter of MA_2 and (iii) none of the configuration parameters of MA_1 is a configuration parameter of MA_2. For completeness, it also includes the supplementary sub-case (denoted as (2) in Table II) where at least one of the configuration parameter of MA_1 is an affected parameter of MA_2 but none of the affected parameter of MA_1 is a configuration parameter of MA_2.

This interaction type can be defined as a *conflict* since the decisions taken by one application are overwritten by the other. Adaptive traffic load-balancing [12] and energy saving [15] applications are illustrative examples of such an interaction. Another example concerns the antenna transmission power parameter configuration in the context of self-organizing networks, which can be configured by both mobility load-balancing and coverage optimization [16].

F. Taxonomy discussion

We show that the proposed taxonomy satisfies the following property:

TABLE II
INTERACTION TYPES AND SIGNATURES.

Interaction Type	**Signature**
Absence of interaction	$(\mathcal{V}_1 \cap \mathcal{K}_2 = \emptyset) \wedge (\mathcal{V}_2 \cap \mathcal{K}_1 = \emptyset) \wedge (\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset)$
Interaction by precedence with unaffected resources (1)	$(\mathcal{V}_1 \cap \mathcal{K}_2 \neq \emptyset) \wedge (\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset)$
Interaction by precedence with unaffected resources (2)	$(\mathcal{V}_2 \cap \mathcal{K}_1 \neq \emptyset) \wedge (\mathcal{V}_1 \cap \mathcal{K}_2 = \emptyset) \wedge (\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset)$
Interaction by precedence with affected resources	$(\mathcal{V}_1 \cap \mathcal{K}_2 \neq \emptyset) \wedge (\mathcal{A}_1 \cap \mathcal{A}_2 \neq \emptyset) \wedge (\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset)$
Interaction with affected resources	$(\mathcal{A}_1 \cap \mathcal{A}_2 \neq \emptyset) \wedge (\mathcal{V}_1 \cap \mathcal{K}_2 = \emptyset) \wedge (\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset)$
Interaction with conflicting decisions - By configuration parameters	$\mathcal{C}_1 \cap \mathcal{C}_2 \neq \emptyset$
Interaction with conflicting decisions - By configuration and affected parameters (1)	$(\mathcal{A}_1 \cap \mathcal{C}_2 \neq \emptyset) \wedge (\mathcal{A}_1 \cap \mathcal{A}_2 = \emptyset) \wedge (\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset)$
Interaction with conflicting decisions - By configuration and affected parameters (2)	$(\mathcal{C}_1 \cap \mathcal{A}_2 \neq \emptyset) \wedge (\mathcal{A}_1 \cap \mathcal{C}_2 = \emptyset) \wedge (\mathcal{A}_1 \cap \mathcal{A}_2 = \emptyset) \wedge (\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset)$

Property 1. *The proposed classification enables the detection of any possible set intersection under the model of management application presented in Section III-B.*

Proof. The proof of Property 1 can be decomposed in three main steps as follows.

1) To determine the dimension of the vector space $\mathcal{V}$ formed by all possible combinations of intersections between any of the parameter sets associated with each management application;
2) To create strictly disjoint sub-spaces of the vector space $\mathcal{V}$ and determine their dimension;
3) To compare the dimension of vector space $\mathcal{V}$ to the sum of the dimension of each of the created sub-spaces.

Step 1: Let $\mathcal{U}$ denote the set of dimension 9 constituted of all possible pair-wise intersections between the sets associated with each application, *i.e.,*

$$\begin{aligned}\mathcal{U} = \{&(\mathcal{K}_1 \cap \mathcal{K}_2), (\mathcal{K}_1 \cap \mathcal{A}_2), (\mathcal{K}_1 \cap \mathcal{C}_2), (\mathcal{A}_1 \cap \mathcal{K}_2), \\ &(\mathcal{A}_1 \cap \mathcal{A}_2), (\mathcal{A}_1 \cap \mathcal{C}_2), (\mathcal{C}_1 \cap \mathcal{K}_2), (\mathcal{C}_1 \cap \mathcal{A}_2), (\mathcal{C}_1 \cap \mathcal{C}_2)\}\end{aligned} \quad (1)$$

Let v denote the linear combination of all elements in $\mathcal{U}$ so that:

$$\begin{aligned} v = &v_1(\mathcal{K}_1 \cap \mathcal{K}_2) + v_2(\mathcal{K}_1 \cap \mathcal{A}_2) + v_3(\mathcal{K}_1 \cap \mathcal{C}_2) \\ &+ v_4(\mathcal{A}_1 \cap \mathcal{K}_2) + v_5(\mathcal{A}_1 \cap \mathcal{A}_2) + v_6(\mathcal{A}_1 \cap \mathcal{C}_2) \\ &+ v_7(\mathcal{C}_1 \cap \mathcal{K}_2) + v_8(\mathcal{C}_1 \cap \mathcal{A}_2) + v_9(\mathcal{C}_1 \cap \mathcal{C}_2) \end{aligned} \quad (2)$$

The coefficients v_i are binary values equal to 1 if the intersection is not null and 0 otherwise. The obtained vectors v are independent and represent all possible intersections between the sets of two applications. Let $\mathcal{V}$ be the vector space induced by the set of vectors v (*i.e.,* it forms a basis of $\mathcal{V}$). Given that there exist 2^9 possible vectors v, the dimension of $\mathcal{V}$ is equal to $2^9 = 512$.

Step 2: Let $\mathcal{P}_{i \leq 512}$ be a subset of vector space $\mathcal{V}$. In practice, each $\mathcal{P}_{i \leq 512}$ should be defined to represent a sufficiently generic type of interactions. We define here eight subsets $\mathcal{P}_{i \in [1;8]}$ corresponding to each interaction type reported in Table II and indexed according to the row ordering in the table. It can easily be proved *reductio ad absurdum* that the eight sub-spaces $\mathcal{P}_{i \in [1;8]}$ are strictly disjoint.

Step 3: The dimension of the $\mathcal{P}_{i \in [1;8]}$ can be easily determined by counting the number of elements in each subspace. More specifically, we have: $\dim(\mathcal{P}_1) = 2$; $\dim(\mathcal{P}_2) = (2^2 - 1) \cdot 2^3 = 24$; $\dim(\mathcal{P}_3) = (2^2 - 1) \cdot 2^1 = 6$; $\dim(\mathcal{P}_4) = (2^2 - 1) \cdot 2^5 = 96$; $\dim(\mathcal{P}_5) = 2^5 = 32$; $\dim(\mathcal{P}_6) = 2^8 = 256$; $\dim(\mathcal{P}_7) = 2^6 = 64$; $\dim(\mathcal{P}_8) = 2^5 = 32$.

It can be verified that the sum of the dimension of the sub-spaces is equal to $\sum_{i=1}^{8} \dim(\mathcal{P}_1) = 512$.

The sub-spaces $\mathcal{P}_{i \in [1;8]}$ are strictly disjoint and the sum of their dimension is equal to the dimension of the vector space $\mathcal{V}$. This proves that the proposed partitioning captures all possible types of set intersections, and hence Property 1. □

Property 1 ensures that the proposed taxonomy is exhaustive with respect to the model of management applications presented in Section III-B. As with any taxonomy, however, some decisions have to be made on the best grouping to apply. In particular, the classification presented here constitutes one among other possible groupings of set intersections. The grouping used to create the proposed taxonomy is the result of a qualitative analysis of various relevant interaction issues reported in the networking and network management literature. While the development of new management applications in the future might require a revision of the grouping presented in this paper, our model of MAs is generic and can easily be used to define the signature of possible new types of interactions[1].

G. Mitigation options

Each interaction type has its own characteristics with respect to how the applications and/or the environment are affected. These characteristics are essential to take into account when deciding how to mitigate an identified interaction given that different mitigating strategies come with different execution times and overheads. Techniques to mitigate interactions can be grouped in four categories [17]: *i)* avoidance, *i.e.,* to preclude the occurrence of interactions, *ii)* ordering, *i.e.,* to rank applications for their order of execution, *iii)* precedence, *i.e.,* to assign priorities to decide on the prevailing ones, and *iv)* harmonization, *i.e.,* to harmonize decisions in order to satisfy individual objectives. We discuss here the suitability of existing mitigation strategies to tackle the different types of interactions in the proposed taxonomy.

1) Interaction by Precedence with Unaffected Resources: This interaction type represents the case of two applications with disjoint sets of configuration and affected parameters, which need to be executed in a certain order. The resulting natural ordering between the two applications makes strategies based on ordering or precedence relevant and practical solutions to tackle this interaction. Determining the order/priorities according to which MAs should be executed is especially facilitated when the applications operate at different timescales or based on different operating conditions.

2) Interaction by Precedence with Affected Resources: A possible approach to tackle this interaction type is to avoid their occurence implementing a single global application with weighted objectives corresponding to each individual management application. This solution applies well when the applications are strongly coupled (as such likely to be developed at the same time) and neither operates at the same timescale nor triggered under the same operating conditions. In the other cases, it is preferable to resort using ordering or precedence since this can provide more flexibility in terms of implementation.

3) Interaction with Affected Resources: The most relevant approach to tackle this interaction type is to use a harmonization strategy. This ensures that a trade-off solution can be found, thus satisfying some level of performance for each application, while offering flexibility in the way each application is developed and implemented in the management system.

4) Interaction with Conflicting Decisions: As explained in Section IV-E, this interaction type qualifies as a *conflict*. A possible mitigating strategy is therefore to decouple the competing applications and replace them by a single MA. However, as already discussed, this solution has severe limitations in terms of flexibility and extensibility and we believe that this should only be used in cases where such interaction reflects a design issue rather than a resource contention problem. A more practical solution is to apply ordering or precedence and decide on the execution order/priorities. In case this cannot be easily determined (*e.g.,* applications operating at same timescales), an alternative approach is to convert the problem into an *Interaction with Affected Resources* type by logically partitioning the configuration parameters controlled by each application. The problem can then be resolved as described in Section IV-G3.

V. LogiSig Functionality

In LogiSig, each MA exposes its relation to the environment through a list of attributes. These are defined based on the abstraction used to represent network parameters in the management plane of the software-based infrastructure, *e.g.,* [4], and derived from the formal model of applications presented in Sect. III-B. The exposed attributes are used to check if interaction can arise between two applications and if so, to classify the interaction based on the signatures presented in Sect. IV.

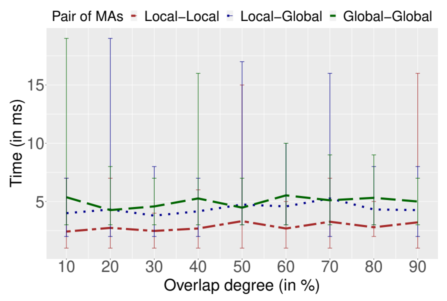
A. System-level realization

A key step in the implementation of the LogiSig functionality is to translate the parameter sets $\mathcal{K}$, $\mathcal{C}$ and $\mathcal{A}$ associated with each MA to their system-level realization. To perform the translation, we build on top of Statesman [18], the most prevalent solution proposed in the literature to-date for tackling interactions in the management plane of a network architecture based on SDN. Statesman is a state management service enabling management applications to operate independently through consistency enforcement between individual network views. In Statesman, the effect of an application's decisions on the network environment is represented in the form of *read* and *write* operations on the network state variables. We follow this rational for the implementation of LogiSig and define the attributes exposed by each MA as system-level operations on network parameters according to their management plane representation.

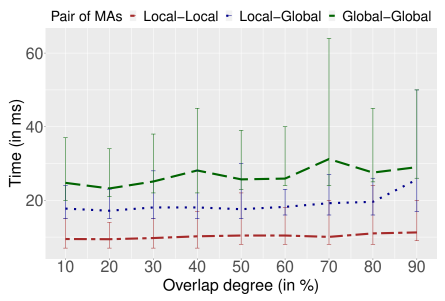
Attributes can be of three types:

- Read - the application performs a *read* operation on the value of the parameter, *i.e.,* the parameter is used as a constant in the application's logic (it belongs to set $\mathcal{K}$);
- Write - the application performs a *write* operation on the value of the parameter, *i.e.,* the parameter is modified by the application's logic (it belongs to set $\mathcal{C}$);
- Indirect Write - a *write* operation is indirectly performed on the value of the parameter, *i.e.,* the parameter is

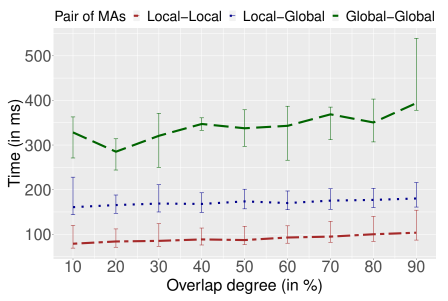
[1]The ability to correctly identify interactions depends on the correct definition of management applications based on our model. It is assumed that a mechanism is in place to check their correct specifications.



(a) 10,000 parameters.



(b) 100,000 parameters.



(c) 1,000,000 parameters

Fig. 2. Performance of LogiSig.

modified as a result of changes made by the application's logic to other parameters (it belongs to set $\mathcal{A}$).

As opposed to *read* and *write*, *indirect write* operations necessitate the availability of a mechanism to infer dependencies between parameters. In this work, we use the dependent variables model introduced in Statesman [18] to extract the *indirect write* attributes of all applications.

The direct mapping between our model of MAs (through sets $\mathcal{K}$, $\mathcal{C}$ and $\mathcal{A}$) and system-level operations makes it easy to integrate LogiSig as part of any interaction management framework. As opposed to the approach presented in [9] that requires each application to be translated in a declarative form, applications can be written in any language with LogiSig. The sole task of the operator is to map MA's attributes to network parameters, which can easily be achieved using a service like Statesman [18]. Based on this mapping, set intersections can be automatically computed and the type of interaction determined.

B. Performance evaluation

As explained in Sect. II, LogiSig operates upstream from the resolution of interactions. It is invoked when changes take place in the environment (*e.g.,* new resources added) or when MAs are added or modified. To effectively contribute to the interaction management task, LogiSig is expected to run fast with respect to the timescale at which the aforementioned changes occur. This is particularly important in the context of "softwarized" services deployed, for instance, over volatile infrastructures, where virtual resources are frequently added or removed at run-time [19], and flexible virtualized environments allowing the introduction of new or the modification of existing functionality on the fly, *e.g.,* [20]. In this section, we investigate the time taken by LogiSig to identify the type of interaction between any pair of MAs.

The time-complexity of LogiSig is mainly driven by the time to compute set intersections. More specifically, interactions are detected based on the pairwise analysis of MAs. Each pair of the *n* MAs deployed in the infrastructure is computed and the set intersection analysis of each pair is executed in parallel. The complexity is dominated by the slowest pair. The design of efficient methods for the set intersection problem has received a lot of attention in the literature, *e.g.,* [21]. A common approach in systems and networking is the use of Bloom Filters [22] (see [23] and [24] for an overview). While these data structures are particularly efficient in the case of very large dynamic sets, we believe that the size and relatively stable nature of the sets considered in our context do not make their use imperative[2]. In this paper, we implement a method to compute set intersections using hash table data structures and simple hash functions[3].

We focus on three main factors affecting the time complexity of LogiSig:

1) the scale of the infrastructure (denoted as S), *i.e.,* the total number of network parameters,
2) the scope of each application in the pair of MAs to examine (denoted as P), *i.e.,* the number of parameters on which each application operates;
3) the overlap in terms of parameters on which two applications operate (denoted as O), where the overlap degree between applications MA_1 and MA_2 is defined as the percentage of parameters used by MA_1 that are also used by MA_2.

We evaluate the performance for different combinations of S, P and O. More specifically, we set $S = 10^4$, 10^5 and 10^6. For P, we consider two types of applications – MA with a local scope operating on 20% of the parameters (Local) and MA with a global scope operating on 60% of the parameters (Global) – and three types of MA pair configurations: Local-Local (the two MAs have a local scope), Local-Global (one MA has a local scope and the other a global scope) and Global-Global (the two MAs have a global scope). Finally, we vary O in the interval $[10\%, 90\%]$ by increments of 10%.

The results are shown in Fig. 2 with the confidence intervals obtained for each setup (S,P,O) over 20 runs. Experiments were run on a laptop with 2.80 GHz Intel Core i7-2640M processor and 8 GB memory with 64-bit Windows 7 Operating System. As can be observed, the three factors S, P and O affect the complexity of LogiSig. As expected, the time increases as the scope of each application increases (there are more attributes to compare against each other), and the effect is

[2]We recognize that in cases not envisioned at the time of writing this paper, LogiSig might benefit from a Bloom Filters implementation.

[3]We use the Java Platform SE 8 HashMap object and the default String hashCode() method.

accentuated as S increases. While marginal with $S = 10^4$ and $S = 10^5$, the effect of the overlap degree between the two applications is more evident in the case where $S = 10^6$, *i.e.,* the time increases as the overlap degree increases. In terms of scalability, the total time increases exponentially as S increases by an order of magnitude. However, even in the case where $S = 10^6$, it takes less than 0.5 seconds for LogiSig to identify interactions, which makes the approach suitable even for scenarios where changes in the infrastructure/functionality are very frequent, *i.e.,* in the order of seconds, *e.g.,* [25].

VI. Related Work

Substantial efforts have been invested by the research community in understanding how interactions can arise between concurrent decision-making processes and in proposing strategies to handle uncoordinated decisions in a number of contexts, such as intelligent networks [29][30], policy-based management [27] and autonomic networks [26][35]. Kephart in his seminal work on autonomic computing [28] already highlighted the importance of coordination models to manage the coexistence of multiple management control-loops. This work paved the way for investigating conflicts in policy-based management, which can arise as a result of contradictory policy-driven operations simultaneously enforced on the managed system. Various conflict types have been reported in the literature, which have been broadly classified into domain-independent and domain-specific. The latter have been mainly studied in the areas of quality of service, security, and distributed systems management, and have been documented in [36].

Most policy conflict detection approaches are based on specifying the conditions under which an inconsistency would arise and the evaluation of those conditions during a detection process. Domain-independent conflicts can be detected by simple syntactic analysis, but more specialized inconsistencies require domain- and system-specific information. As reported in [36], the main detection methods are based on meta-policies, rule relations, applicability spaces, and information models. While each method has its merits, the representation of policies and the conflict conditions is of paramount importance. Logic-based approaches, e.g., [27], have demonstrated clear advantages because, in addition to detecting inconsistencies, they allow for advanced reasoning that can provide explanations for their occurrence. These are particularly important for selecting effective resolution strategies and appropriate configurations. For this reason we also adopt a logic-based approach, which in essence captures the conditions of interaction through signatures, but in a more generalized form that can apply to a wide range of network management functionality.

Software-defined Networking, as a network design based on layering and modular structure, has been reasserting the importance of managing interactions among parallel decision-making processes [4][31]. Research efforts in this domain have been mostly focusing on the resolution part of the problem, *i.e.,* on mechanisms to compute mitigating actions, *e.g.,* [5][32]. Deploying efficient methods to detect inconsistencies and provide explanations for their occurrence is however of paramount importance for deciding how to effectively mitigate them. LogiSig, by automatically determining the type of interaction between pairs of MAs in a time-efficient manner, contributes to the realization of such methods. In the recent years, Wang *et al.* proposed a novel framework to represent the network environment as a standard SQL database and to define management applications as SQL queries on database views [33][34]. By design, LogiSig is well suited to be integrated in such a framework and can complement the mechanism adopted in the work to coordinate applications.

VII. Conclusions

In this paper, we presented a novel approach for the detection and classification of interactions between applications in the management plane of software-based networks. We showed that it can be used as an automatic and computationally efficient method for determining how two applications interact. The approach overcomes important limitations of previous solutions and can be easily integrated with existing frameworks for the management of interactions.

In future work, we plan to compare the performance of different mitigation strategies and demonstrate how LogiSig can be used to support the automation of the selection of mitigating actions based on the current operating conditions and identified types of interactions.

Acknowledgment

The work of Daphne Tuncer is supported by the Imperial College Research Fellowship Scheme.

References

[1] M. Chiang, and W. Shi, "NSF workshop report on grand challenges in edge computing," Tech. Rep.. 2016.

[2] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou, "Adaptive resource management and control in software defined networks," *IEEE Transactions on Network and Service Management*, vol. 12(1), pp. 18-33, March 2015.

[3] E. Haleplidis, et al, "Software-defined networking (SDN): Layers and architecture terminology," no. RFC 7426, 2015.

[4] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing Software Defined Networks," in *Proc. of NSDI'13*, vol. 13, pp. 1-13, 2013.

[5] J. C. Mogul, et al., "Corybantic: towards the modular composition of SDN control programs," in *Proc. of the Twelfth ACM Workshop on Hot Topics in Networks (Hotnets'13)*, 2013.

[6] Wenjie Jiang, Rui Zhang-Shen, Jennifer Rexford, and Mung Chiang, "Cooperative content distribution and traffic engineering in an ISP network," in *Proc. of SIGMETRICS '09*, New York, NY, USA, pp. 239-250, 2009.

[7] J. C. Mogul, "Emergent (mis) behavior vs. complex software systems," in *ACM SIGOPS Operating Systems Review*, vol. 40, no. 4, pp. 293-304, April 2006.

[8] N. Samaan, "Achieving Self-management in a Distributed System of Autonomic BUT Social Entities," in *Proc. of the IEEE MACE*, vol. 5276 of Lecture Notes in Computer Science, pp. 90–101, 2008.

[9] W. Wang, W. He, and J. Su, "Redactor: Reconcile network control with declarative control programs In SDN," in *Proc. of IEEE 24th International Conference on Network Protocols (ICNP'16)*, pp. 1-10, November 2016.

[10] V. Heorhiadi, S. Chandrasekaran, M. K. Reiter, and V. Sekar, "Intent-driven composition of resource-management SDN applications," in *Proc. of the ACM 14th International Conference on emerging Networking EXperiments and Technologies (CoNEXT'18)*, pp. 86-97, December 2018.

[11] D. M. Volpano, X. Sun, and G. G. Xie, “Towards systematic detection and resolution of network control conflicts,” in *Proc. of the third workshop on Hot topics in software defined networking*, pp. 67-72, August 2014.

[12] D. Tuncer, M. Charalambides, G.Pavlou and N. Wang, “DACoRM: A coordinated, decentralized and adaptive network resource management scheme”, in *Proc. of IEEE/IFIP Network Operations and Management Symposium*, pp.417-425, April 2012.

[13] N.Wang, KH. Ho, and G. Pavlou, “Adaptive multi-topology IGP based traffic engineering with near-optimal network performance,” in *Proc. of IFIP-TC6 NETWORKING 2008*, pp. 654-666, 2008.

[14] M. Claeys, et al., “Hybrid Multi-tenant Cache Management for Virtualized ISP Networks,” in *Journal of Network and Computer Applications (JNCA)*, volume 68, pp. 28-41, June 2016.

[15] M. Charalambides, D. Tuncer, L. Mamatas and G. Pavlou, “Energy-Aware Adaptive Network Resource Management”, in *Proc. of the 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM’13)*, Ghent, Belgium, May 2013.

[16] H. Y. Lateef, A. Imran, M. A. Imran, L. Giupponi and M. Dohler, “LTE-advanced self-organizing network conflicts and coordination algorithms,” in *IEEE Wireless Communications*, vol. 22, no. 3, pp. 108-117, June 2015.

[17] D. Tuncer, M. Charalambides and G. Pavlou, “Management Application Interactions in Software-Based Networks,” in *IEEE Network*, vol. 33, no. 5, pp. 149-155, Sept.-Oct. 2019.

[18] P. Sun, R. Mahajan, J. Rexford, L. Yuan, M. Zhang, and A. Arefin, “A network-state management service,” in *Proc. of the 2014 ACM conference on SIGCOMM (SIGCOMM’14)*, pp. 563-574, 2014.

[19] E. Hernandez-Valencia, and B. Polonsky, “How will NFV/SDN transform service provider opex?,” *IEEE Network*, vol. 29(3), pp. 60–67, 2015.

[20] S. Clayman, et al., “The dynamic placement of virtual network functions,” in *Proc. of the IEEE Network Operations and Management Symposium (NOMS’14)*, pp. 1-9, 2014.

[21] H. Cohen, and E. Porat, “Fast set intersection and two-patterns matching,” *Theoretical Computer Science*, vol. 411, no 40-42, pp. 3795-3800, 2010.

[22] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” *Communications of the ACM*, vol. 13, no 7, pp. 422-426, 1970.

[23] A. Broder and M. Mitzenmacher, “Network applications of bloom filters: A survey,” *Internet mathematics*, vol. 1, no 4, pp. 485-509, 2004.

[24] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, “Theory and practice of bloom filters for distributed systems,” *IEEE Communications Surveys & Tutorials*, vol. 14, no 1, pp. 131-155, 2012.

[25] L. Chaufournier, et al., “Fast transparent virtual machine migration in distributed edge clouds,” in *proc. of the Second ACM/IEEE Symposium on Edge Computing (SEC’17)*, 2017.s

[26] M. Litoiu, M. Woodside, and T. Zheng, “Hierarchical model-based autonomic control of software systems,” in *Proc. of the 2005 Workshop on Design and Evolution of Autonomic application Software (DEAS’05)*, pp. 1-7, 2005.

[27] M. Charalambides et al., “Policy conflict analysis for diffserv quality of service management,” in *IEEE Transactions on Network and Service Management*, vol. 6, no. 1, pp.15-30, March 2009.

[28] J. O. Kephart, “Research challenges of autonomic computing,” in *Proc. of the 27th International Conference on Software Engineering (ICSE’05)*, pp. 15-22, 2005.

[29] J. Kamoun, and L. Logrippo, “Goal-Oriented Feature Interaction Detection in the Intelligent Network Model,” in *Feature Interactions in Telecommunications and Software Systems V, eds. K. Kimbler and LG Bouma*, pp. 172-186, 1998.

[30] S. Tsang and E. H. Magill, “Learning to detect and avoid run-time feature interactions in intelligent networks,” in *IEEE Transactions on Software Engineering*, vol. 24, no. 10, pp. 818-830, Oct. 1998.

[31] X. Jin, J. Gossels, J. Rexford, and D. Walker, “CoVisor: A Compositional Hypervisor for Software-Defined Networks,” in *Proc. of NSDI’15*, vol. 15, pp. 87-101, May 2015.

[32] A. AuYoung et al., “Democratic Resolution of Resource Conflicts Between SDN Control Programs,” in *Proc. of ACM CoNEXT*, pp. 391-402, 2014.

[33] A. Wang, X. Mei, J. Croft, M. Caesar, and B. Godfrey, “Ravel: A database-defined network,” in *Proc. of the ACM Symposium on SDN Research (SOSR’16)*, p. 5, March 2016.

[34] A. Wang and J. Croft, “Automating SDN Composition: A Database Perspective,” in *Proc. of the Symposium on SDN Research (SOSR’17)*, pp. 203-204, 2017.

[35] M. Charalambides, G. Pavlou, P. Flegkas, N. Wang, D. Tuncer, “Managing the future internet through intelligent in-network substrates,” *IEEE Network*, vol. 25, no. 6, pp. 34-40, 2011.

[36] M. Charalambides, “Policy Analysis for DiffServ Quality of Service Management,” *Ph.D. dissertation*, University of Surrey, 2009.