Extensible Signaling Framework for Decentralized Network Management Applications

Dario Valocchi*, Daphne Tuncer†, Marinos Charalambides†, Mauro Femminella*, Gianluca Reali*, George Pavlou†
†Department of Electronic and Electrical Engineering,University College London, UK
*Department of Engineering, University of Perugia, IT

***Abstract*—The management of network infrastructures has become increasingly complex over time, which is mainly attributed to the introduction of new functionality to support emerging services and applications. To address this important issue, research efforts in the last few years focused on developing Software-Defined Networking solutions. While initial work proposed centralized architectures, their scalability limitations have led researchers to investigate a distributed control plane, with controller placement algorithms and mechanisms for building a logically centralized network view, being examples of challenges addressed. A critical issue that has not been adequately addressed concerns the communication between distributed decision-making entities to ensure configuration consistency. To this end, this paper proposes a signaling framework that can allow the exchange of information in distributed management and control scenarios. The benefits of the proposed framework are illustrated through a realistic network resource management use case. Based on simulation, we demonstrate the flexibility and extensibility of our solution in meeting the requirements of distributed decision-making processes.**

I. Introduction

It is commonly admitted that network infrastructures have become over time very complex. To support an ever increasing range of applications and technologies, today's networks need to provide a heterogeneous set of functionalities, usually implemented using equipment from multiple vendors, which very often cannot easily inter-operate. As such, it has become essential to simplify the network management processes.

The Software-Defined Networking (SDN) paradigm has emerged as a promising solution to tame this complexity. The main principle behind SDN resides in the separation between the control and forwarding logic, which can be represented by a decoupling between the data plane and the control plane. In a SDN environment, the network infrastructure is controlled through a unified control plane independent of specific vendor equipment based on an abstract view of the resources. Flexibility, programmability, simplification of management tasks and application deployment are among the main advantages usually advocated for the development of SDN-based network solutions [1].

Early SDN proposals mainly focused on centralized control plane approaches to implement the control logic (*e.g.*, [2][3]). In addition, management applications are usually executed through a central management system which operates on a global view of the network resources (*e.g.*, [4]). Despite their advantages - relatively simple to implement and can result in optimized resource configuration - centralized solutions have limitations, especially in terms of scalability, as the size and dynamics of the network increase.

978-1-5090-0223-8/16/$31.00 © 2016 IEEE

To overcome these limitations, we developed in our previous work a novel SDN-based management and control framework for fixed backbone networks [5]. The proposed framework follows a layered architecture and relies on distributed planes to implement the management and control functionality. More specifically, management operations are executed through a set of distributed Local Managers (LMs), which interact in order to decide on the reconfigurations to apply for optimizing the utilization of the network resources. The resulting configuration is then communicated to a set of distributed Local Controllers (LCs), which determine the best sequence of actions to enforce for updating the network parameters.

In this paper, we go a step further in the implementation of the distributed management and control planes and develop a new signaling framework to enable the communication and interaction between the different entities in our framework. The development of a signaling mechanism to support interactions between distributed points is a key challenge in the design of any decentralized system. Such a mechanism should provide the means for multiple entities to communicate without incurring significant signaling overhead and complexity.

To satisfy these requirements, the proposed signaling framework relies on a modular structure that enables the decoupling between the transport logic, which is realized based on common transport protocols, and the management application logic, which exchanges application-level information. Such a modular structure offers several advantages in terms of flexibility and extensibility. By extracting the mechanisms common to all the applications (*e.g.*, transport, security *etc.*) into a commonly accessible layer, it enables application multiplexing: applications with different logic can implement their own specific requirements in terms of communication while relying on the common signaling layer to realize the exchange of information. In addition, the modularity allows the interoperability of components with different implementations (*e.g.*, LMs and LCs), which can make the framework also applicable to other types of environments, *e.g.*, distributed virtualized network functions.

While the management and control framework considered in [5] relies on several interfaces, in this paper we focus on

the interaction between the distributed management entities. In particular, we illustrate the role and benefits of the proposed signaling framework based on a representative resource management use case for content distribution in the context of an ISP-operated caching scenario [6]. The associated application has strong requirements in terms of the diversity and volume of information used by each management entity to take a decision and, as such, to be exchanged during a reconfiguration phase. The results of the evaluation, which are based on realistic settings, show that the signaling cost is directly influenced by the logic of the application. This demonstrates that the scalability of the approach is driven by the complexity of the communication scheme associated with the application rather than the internal mechanics of the proposed signaling framework.

The remainder of this paper is organized as follows. Section II provides background information on the network resource management framework and the caching application considered in this paper. Section III elaborates on the knowledge required by the different components of the management framework to interact with each other. Section IV describes the proposed signaling framework in detail. Evaluation results are presented in Section V. Section VI describes related work. Finally, conclusions are provided in Section VII.

II. Background

In this section, we provide background information about the SDN-based resource management and control framework considered in this paper, as well as the cache management application used to illustrate the functionality of the proposed signaling framework.

A. Management and Control Framework

In our previous work [5], we developed a novel SDN-based network resource management and control framework to support both static and dynamic resource management applications in fixed backbone infrastructures. In the proposed framework, the network infrastructure is managed and controlled by a set of software-based Local Managers (LMs) and Local Controllers (LCs), forming distributed management and control planes, respectively. The framework follows a three-layer architecture as depicted in Fig. 1. The bottom layer represents the underlying network infrastructure. The middle layer implements distributed management and control functionality. Finally, the top layer represents the central management system. As depicted in the figure, the interaction between the different components of the architecture is realized through a set of interfaces.

A key feature of this framework resides in its modular structure, which is represented by two levels of separation, *i.e.*, between the management and control functionalities, on one hand, and between centralized and distributed management operations, on the other. Short to medium term management operations are performed by the LMs, which implement the logic of management applications. These are responsible for computing the configuration of the set of network resources

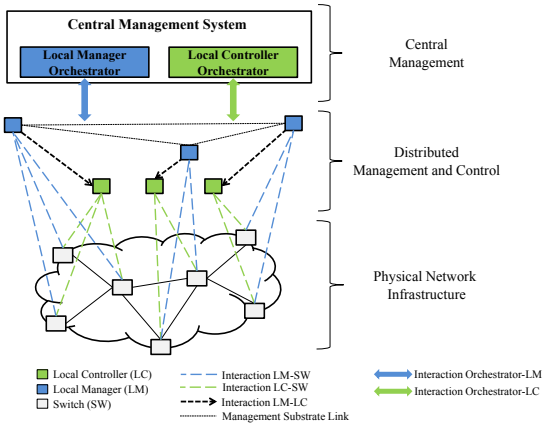

Fig. 1. Resource management and control framework proposed in [5].

under their supervision according to the objective of the applications which they implement. Configuration decisions taken by LMs are provided to the LCs, which define and plan the sequence of actions to be enforced for updating the network parameters. These actions are then translated to instructions sent to and executed by the relevant network devices. In contrast to the LMs, the centralized management system is responsible for longer term operations, for example those that pertain to the instantiation and life cycle of LMs and LCs.

The clear distinction between the management and control logic provides several deployment benefits. This not only allows the two concerns to evolve independently, offering increased design choices and flexibility for the system vendors, it also simplifies the integration of new network applications, while maintaining interoperability. It should be noted that this separation can be seen as a realization of the abstraction discussed in the network self-management literature, which proposed to partition the management process into separate functions represented by the Monitoring-Analyze-Plan-Execution ("MAPE") loop [7].

As depicted in Fig. 2, the logic of the management applications (MAs) is distributed across a set of LMs, which provide a common execution environment for different applications. Each MA is instantiated at the LM level as a module (*e.g.*, MA_1, MA_2), which maintains information tables and implements algorithms to decide on the configurations to apply. To support the decision-making process of a specific MA, the set of LMs involved in the execution of that application are organized into a management substrate [8][9]. The substrate is a logical structure used to facilitate the exchange of information between decision-making entities for coordination purposes. The number of substrates, as well as their structure, depend on the number and type of applications implemented in the distributed management plane (one substrate per application). It is worth noting that, although the same set of MAs is depicted on each LM in Fig. 2, this is not a requirement of the framework. In practice, each LM can instantiate a different number of applications.

We elaborate on the interaction between the different com-

TABLE I
MAIN ACRONYMS.

ASL	Application Signaling Layer
ASP	Application Signaling Protocol
LC	Local Controller
LM	Local Manager
LMO	Local Manager Orchestrator
MA	Management Application
MSL	Management Signaling Layer
MSP	Management Signaling Protocol

ponents of the management plane (*i.e.*, LMO, LMs and MAs) in Section III. For clarification purposes, the main acronyms used in this paper are summarized in Table I.

B. Cache Management Application

To illustrate the role and benefits of the proposed signaling framework, we investigate how this can be used to enable the content management application that we developed in [6] in the context of an ISP-operated caching scenario. In this scenario, each network node is associated with caching capability which is used to locally store a set of content items. The configuration of each cache is computed based on the logic of the content management application implemented by a set of LMs, which coordinate their decisions through the management substrate defined for that application. The objective is to determine which content items to cache, and where, based on content characteristics such as the popularity and origin (geographical) of requests.

More specifically, the decision-making process of this application relies on two phases. In the first phase, content placement decisions are computed concurrently by the LMs which coordinate between themselves to acquire global knowledge about the user demand distribution (*i.e.*, from which content popularity and geographical scope can be deduced), as well as cache configuration. In the second phase, decisions are taken independently by each LM with the objective of filling up any remaining local cache capacity.

III. KNOWLEDGE FOR DISTRIBUTED MANAGEMENT ARCHITECTURE

In this section we describe in detail the information maintained and shared by the management entities of the framework, in order to underline the main motivations behind our design choices.

A. Local Manager Orchestrator

As described in Section II, the LMO is the main centralized management entity. It is responsible for long-term configurations, such as deciding the number of LMs to deploy, their location in the network, and the subset of resources under their responsibility. The main LMO task is to compute the partitioning of the network resources into clusters and to identify the location of each LM inside each cluster. In [5], we proposed an algorithm to compute the clusters and the placement of the LMs. The output of this algorithm is used to deploy the distributed management infrastructure. The LMO communicates with the physical layer in order to instantiate the LMs in the selected locations (*i.e.*, bare metal servers, cloud managers, Network Function Virtualization platforms). Subsequently, the LMO communicates to each LM the list of the network resources under its responsibility and the list of the IP addresses of the other LMs in the network (LMO-LM interface). The second task performed by the LMO is to decide, based on the different management functions, the list of LMs where a specific MA must be installed. By computing this placement, the LMO also decides on the structure of the management substrate, defining the connectivity between the various modules of each MA. The list of MAs to be installed is then passed to each LM by the LMO, triggering the installation procedures. Finally, after the MA deployment, the LMO must communicate to each MA module the list of its neighbors in the relevant substrate (LMO-MA interface). To provide the services described above, the LMO must store the following information about the managed network:

- Network topology
- Network Resources (*i.e.*, routers, servers, switches, caches, *etc.*)

As an output of the placement procedures, the LMO will maintain two internal tables:

- <LM id, Server Id, <Rid>, <MA Id>>: a list where each entry represents a LM, the server hosting it, the list of the managed resources (*i.e.*, switches, routers, caches) under its responsibility, and the list of the hosted MAs.
- <MA Id, <LM Id>>: the list of LMs running the MA (*i.e.*, the MA substrate).

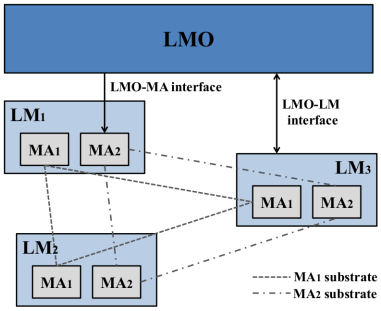

Fig. 2. Distributed management plane.

B. Local Manager

As explained in Section II-A, the LM represents a common execution environment for the MA modules, providing an abstraction of the managed resources to the hosted applications. As such, it must store the list of these resources, as well as the identifier (*e.g.*, address) of the LCs that will enforce its decisions on the them. In addition, it must also store the address of all the other LMs in the network, in order to enable the communication between the MAs hosted locally and the MAs hosted on remote LMs.

To provide the services described above, each LM maintains the following information:

- <R Id>: the list of the managed resources.
- <MA Id>: the list of the hosted MAs.
- <LM Id, IP address>: the translation table between LM identifier and IP address.
- <LC Id, IP address>: the translation table between LC identifier and IP address.

This information is retrieved via the LMO-LM interface during the deployment phase and updated, if needed, during a long term reconfiguration cycle.

C. Management Application

The MA implements the actual management and decision logic. As pointed out in Section II, MA modules run on the LMs deployed in the network and, depending on the application, they store different types of information according to the requirements of the specific management function, *e.g.*, link utilization, content demand, content location, network statistics, *etc.*

In order to make a decision, a MA module can act independently or share local information with a subset of its neighbors in the substrate, thus co-operating with other MA modules. The substrate could also be used to share additional information such as event triggers or to facilitate synchronization among the distributed entities during the decision-making process. To communicate through the substrate, MA modules must store the list of their neighbors, which is retrieved from the LMO once deployed. The information maintained by each MA module is the following:

- <LM Id>: the list of neighboring LMs in the management substrate.
- Application specific information.

IV. Signaling Framework

In order to enable communication between the entities of the proposed management and control framework, a suitable signaling protocol suite has to be designed. In this section we present our solution, describing the design principles, the signaling architecture and the functionality of the proposed components.

A. Design Principles

Different management applications can have different requirements in terms of how and when they need to share information. Due to this heterogeneity, a monolithic signaling protocol would be unsuitable to cope with the needs of different applications. Furthermore, the types of management applications that will be developed in the future, as well as their signaling requirements cannot be predicted. Another important aspect for the design of the signaling framework is the relationship between the management and control planes. Different types of applications would require to control different categories of network resources, which can be classified, for example, into SDN switches, routers, and network caches. Based on this classification, different controllers would be required to control and enforce decisions on different resource categories. As such, the signaling framework should be flexible and allow the MAs to interact and communicate with different types of LCs, through the abstraction provided by the LMs.

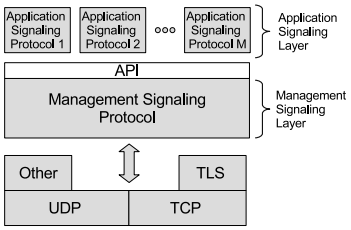

Fig. 3. Signaling architecture.

In addition, the signaling framework should also support the parallel evolution of the management and control planes. The definition of new features, or the introduction of new MAs, new categories of resources or new and more advanced LCs, should not require a re-design of the signaling architecture.

The proposed signaling framework is therefore designed to ensure both flexibility, in terms of enabling the communication between different types of applications and different types of controllers/resources, and extensibility to cope with the evolving nature of the management and control planes.

To achieve these objectives, our architecture decouples the infrastructure signaling, necessary to deploy and coordinate the system entities, from the management signaling required by the MAs to execute their decision-making process. This decoupling leads to a two-layer architecture as depicted in Fig. 3: the Management Signaling Layer (MSL) deals with the infrastructure signaling, whereas the Application Signaling Layer (ASL) implements the signaling logic needed by the MA modules.

The main functions of the MSL are the following:

1) Exchange packets used by the system to deploy modules over the network and update their internal status (*e.g.*, deploy LM, setup MAs, distribute substrate information, *etc.*).
2) Multiplex and demultiplex packets coming from and directed to the MAs.
3) Provide the communication interface between the MAs and the LCs.

The ASL has two main functions:

1) Exchange packets to share information needed by MA modules to make decisions.
2) Exchange packets to synchronize the different phases of the decision-making process of a given application.

We refer to the Management Signaling Protocol (MSP) as an implementation of the MSL, and to the Application Signaling Protocols (ASPs) as the implementations of different ASLs. The decoupling between the MSL and the ASL allows the MA developers to design their own information format and communication scheme, which relies on a common layer to provide message transport and addressing services. In addition, the MSL provides a common interface for the different MAs to push their decisions to the relevant type of LCs (*e.g.*, for SDN switches or caches) associated with the LM, regardless of the specific vendor implementations. The evolution of existing

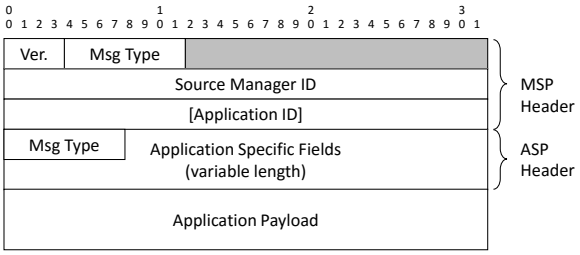

Fig. 4. MSP and ASP packet format.

LCs, or even the introduction of new LCs and/or resource categories, will not affect the signaling architecture, and will only require the definition of new identifiers for the different LCs, and the development of a relevant MA to interact with/manage them. Another advantage of this architecture is that, in case of LM relocation, the decoupling between the MSL and the ASL allows the LMO to inform the other LMs about the new address of the relocated LM. This alleviates the need to compute and distribute a new network-level substrate for all the MAs affected by the relocation, which provides gains in terms of computing time and network bandwidth.

B. Management Signaling Protocol Operations

Instead of designing a new IP based transport protocol, we designed the MSP to rely on existing transport and session protocols, which provide the communication primitives, such as datagram messaging (UDP), reliable sessions (TCP), or secure and reliable sessions (TLS over TCP).

To provide the address resolution service to the ASP, the MSP must maintain a table containing the list of the LMs deployed in the network with their relevant IP address. The ASP maintains information about the substrate of the relevant MA, represented by a list of LM identifiers. The interface between the ASP and the MSP allows the ASP to specify a list of LMs, which are the signaling destinations, its payload, and the signaling requirements (*i.e.*, reliability, security, *etc.*). The MSP adds its header to the payload, uses its internal table to resolve the IP addresses of the destinations and chooses, among the available transport services, the one matching the ASP requirements. It can then send the packet to the selected destinations. Fig. 4 shows the packet format of the proposed signaling framework. The header of the MSP is designed to be as general as possible in order to cope with the different requirements in terms of flexibility and extensibility, as described in Section IV-A. The Message Type field in the MSP header is used to identify packets carrying ASP payloads, and packets coming from or directed to LCs, or to the LMO. The MSP header also contains the Source Manager Id field that is fetched with the payload to the ASP when a message is received. As such, the ASP does not need to execute a reverse lookup on the MSP table in order to identify the source of the message.

In contrast, the ASP packet format strongly depends on the specific ASP implementation. A Message Type field should be available to allow the finite state machine of the ASP to identify the format of the carried MA payload. For example, the ASP header of a time-driven MA could provide synchronization fields, such as a configuration cycle identifier, an iteration identifier or a timestamp.

V. Performance evaluation

In this section we present the results of the experiments we performed in order to quantitatively assess the benefits of the proposed signaling framework. The performance of the communication scheme is evaluated both in terms of computational complexity and network cost based on a custom simulator. Although the signaling framework is designed to support various types of MAs, we designed and implemented a specific management application and its ASP. Section V-A describes this application and the associated ASP. Section V-B provides insights on the features of the simulator and on the evaluated performance metrics. Finally, the obtained results are presented and discussed in Section V-C.

A. ASP for Distributed Cache Management

An overview of the scenario used to illustrate the effectiveness of our solution is presented in Section II-B; a more elaborate description can be found in [6]. In this section, we focus on the interaction between the different MA modules and on the functionality of the associated ASP.

In the proposed scenario each MA module is associated with a fixed amount of caching space, used to locally store content items, and is in charge of selecting which content items to cache for the next configuration cycle. This decision is taken in a coordinated fashion among MA modules and is based on demand characteristics, such as content popularity. The cache reconfiguration algorithm is executed periodically by MA modules, once a quasi-synchronous timer expires. The decision process is composed of two phases. The first phase is iterative and involves the use of the ASP. At the start of this phase, MA modules exchange content popularity information, as perceived from their local view, through the ASP. The collected information is subsequently aggregated by each module separately to build a data structure representing the global view of content popularity. The next part of this phase involves an iterative process to decide on which content items to store at each of the available caching locations. At each iteration, every MA module selects a variable number of items to cache locally based on, (a) information extracted from the global popularity structure, and (b) the cache status of other modules in the substrate. Given that (b) needs to be updated at each iteration, MA modules use the ASP to share their cache status once a new placement has been computed. The first phase ends in an asynchronous fashion and a final cache status exchange is therefore required in order to synchronize the global content placement view between the different MA modules. The second phase of the configuration algorithm is carried out by each MA module independently and, as such, does not involve the ASP.

In the context of the scenario considered in this paper, it is evident that, along with the messages carrying application specific information, a synchronization mechanism is needed

in order to keep the MA modules in sync with the current step of the algorithm. We can divide each iteration of the first phase of the algorithm into two types of tasks: a compute task (*e.g.*, compute content placement) and a sharing task (*e.g.*, sharing the local cache status). As a result, the MA needs two synchronization mechanisms:

- A synchronization mechanism to notify the MA, during a sharing task, when the information exchange is finished and the information retrieved from the substrate can be processed.
- A synchronization mechanism to orchestrate the different MA modules in the substrate to ensure that they are all executing the same computing task of the algorithm.

The first synchronization mechanism is provided by the ASP, by exploiting the nature of the MA. At each sharing step, MA modules broadcast a set of messages knowing exactly the number of messages they expect to receive (*i.e.*, number of remote MA modules). A bitmap is maintained by the ASP for each type of signaling message, which has a number of bits equal to the number of remote MA modules in the substrate. All bits are initialized with a zero value at the beginning of a sharing task. A bit corresponding to a specific MA module is set to one if, during the sharing task, a message has been received from that module. The ASP can thus notify the MA that a particular sharing task has been completed when all the values maintained by the relevant bitmap are true.

In the case of the second synchronization mechanism, a set of synchronization messages is defined, so that upon completing a specific compute step of the algorithm, each module starts a new sharing task, exchanging synchronization messages relevant to that computing task. When the sharing task is completed all the modules can move to the next task of the iteration.

B. Evaluation Settings

Our simulator has been implemented in Java. It uses the Deltacom topology [10], composed by 92 nodes, for layer 3 routing. We assume that each link in the network has an available bandwidth of 10 Gbit/s, which is a typical capacity value in real networks (*e.g.*, [11]). The placement of LMs is provided as an input and is used to compute the relevant IP distances between pairs of managers. The number of requests per content, for each manager and for each reconfiguration cycle, is also provided as an input to the simulator. To determine the number of requests per content, we generate one hour of random requests based on a Poisson distribution with inter-arrival time $\lambda = 5000$ req/h (based on the hourly request pattern of the video on demand (VoD) trace presented in [12]) and a Zipf distribution with skew factor $\alpha = 1.2$ (based on the characteristics of the VoD dataset used in [13]). For the geographical distribution of interests (*i.e.*, number of distinct locations from which a content is requested), we use the model proposed in [6] with $\beta = 0.8$. In this case, the predominance of popular content items can be preserved without strongly discriminating less popular ones.

The performance has been evaluated both in terms of the time consumed by the MA during one reconfiguration cycle, and the traffic generated during the decision-making phase. The time consumed by the MA is divided into:

- The **CPU Time** used by each module of the MA to compute the required data structures and the output of the placement algorithm.
- The **Network Time** used by each module of the MA to send signaling messages to its peers in the substrate. This time is divided into two components; the transfer time and the network delay as follows:
  - The **Network Delay** is measured assuming that each IP hop in the topology incurs a 5 ms delay (based on the values reported in [14]).
  - The **Transfer Time** is computed by dividing the size of the packet with the link bandwidth.

As for the traffic generated, the **Network Traffic** is computed by summing up the size of each packet multiplied by the length of the path between its source and its destination, measured in terms of IP hops.

Each MA module is represented by an object that implements the content placement algorithm and the ASP described above. The simulation is time-driven and in real time. At the end of each reconfiguration cycle, the average value and the standard deviation for the network time and the CPU time are computed based on the values measured by each node, while the network traffic metric is obtained by summing up the contribution of each node. For each test we compute the average value along with the standard deviation for all the relevant metrics, collected for 10 reconfiguration cycles.

C. Simulation Results

We collected values for the described metrics for different configurations of three main parameters of the system: the content catalogue size, the number of managers (*i.e.*, caches), and the caching space available at each caching location. We consider a content catalogue with a number of items ranging from 10^4 to 10^5 based on the catalogue sizes reported in [15]. The number of managers, and their location in the topology, is computed using the placement algorithm proposed in [5], and it ranges from 4 to 82. As for the available caching space per location, we also used [15] and selected four values for the ratio between the total caching space in the network and the catalogue size (5%, 10%, 15% and 20%), but due to space limitations, we present results only with the two extreme cases (5% and 20%)[1]. For each ratio, we compute the available space per location by dividing the resulting total caching space by the number of managers.

Figure 5 shows the simulation results for the average CPU time, with a 95% confidence interval. The computing time needed to complete a reconfiguration cycle strongly depends on the size of the content catalogue, but as the size of the catalogue increases by a factor of 10, the average CPU time

[1]It is worth noting that similar results were obtained with the two other ratio values.

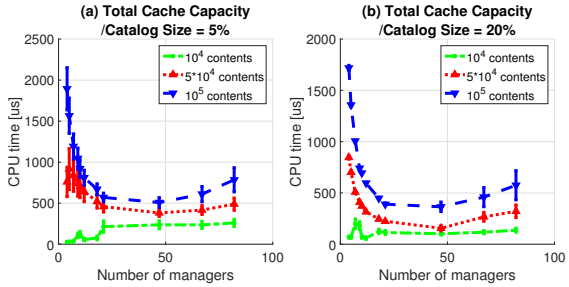

Fig. 5. Average CPU time with 95% confidence interval for a ratio between the total caching space and the catalogue size of (a) 5% and (b) 20%, for different values of the catalogue size, as a function of the number of managers.

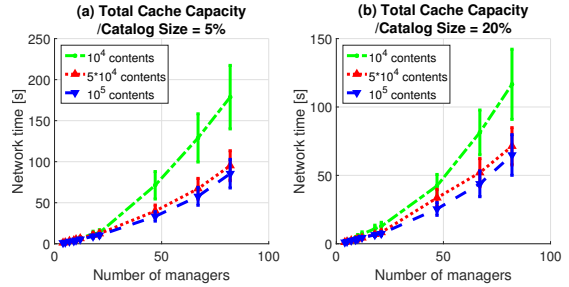

Fig. 6. Average Network time with 95% confidence interval for a ratio between the total caching space and the catalogue size of (a) 5% and (b) 20%, for different values of the catalogue size, as a function of the number of managers.

increases only by a factor of 2. Two interesting observations can be made from the results presented in Figure 5. Increasing the size of the total caching space (*i.e.*, increasing the size of each cache) leads to a lower computation time. This can be explained by the placement algorithm, which relies on a parameter p to control the number of content items to consider for caching at each iteration of the first phase. The value of p depends on the total available caching space in the network and an increase in its value means that more items can be cached at each iteration. This leads to a decrease in the number of iterations needed to complete the first phase. Given that the cost of each iteration is dominated by the contribution of sharing tasks, the effect of the value of p on this cost is negligible. As a result, a lower number of iterations leads to a lower CPU time.

The second observation is that an increase in the number of managers in the network leads to a decrease of the CPU time, but only up to a certain value, when the number of LMs account for around 50% of the network nodes. Increasing the number of managers above this value results to a decrease in the performance in terms of CPU time. This happens because, with a higher number of managers, the size of the data structures maintained by the MA modules grows. For example, the size of the global content demand structure grows linearly with the number of managers, since this is based on an aggregate of the local view of each manager. This is particularly evident when the catalogue size increases. In order to mitigate this effect, the scope of peering of each manager can be limited by partitioning the substrate into clusters of

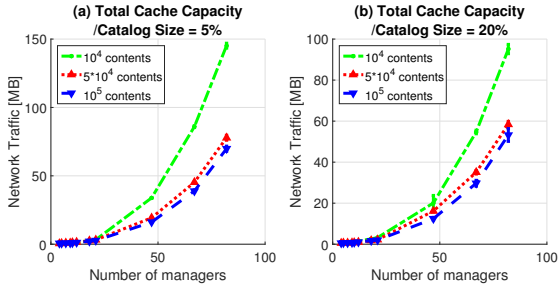

Fig. 7. Average Network Traffic with 95% confidence interval for a ratio between the total caching space and the catalogue size of (a) 5% and (b) 20%, for different values of the catalogue size, as a function of the number of managers.

managers. This will reduce the knowledge of each manager, decreasing the size of the structures maintained by the MA module, and thus improve the performance in terms of CPU time.

Figures 6 and 7 show the results for the network related metrics. These demonstrate how the signaling framework can support the complexity of the communication scheme of the considered MA. The MA relies on a N-to-N communication scheme, leading to a complexity in the order of $O(N^2)$, with N being the number of managers. The results show that the quadratic relationship between the number of managers and the network metrics is respected: no further overhead or complexity are incurred by the signaling system. In addition, an increase of the caching space at each manager leads to an increase in the performance in terms of network metrics. Increasing the caching space leads to a decrease in the number of iterations needed to complete a reconfiguration cycle (due to the effect of p), thus resulting to a decrease in the number of sharing tasks, and consequently to performance improvement.

An interesting result is the performance improvement in terms of network metrics when the number of contents in the catalogue increases. When a lower number of contents is considered, we observe that more signaling traffic is generated, which is due to the way in which local popularity is computed. As explained in Section V-B, we generate one hour of requests as a Poisson process. The total number of requests received during that period is then distributed between the contents in the catalogue according to the Zipf's Law, in order to compute the number of requests per content. The function proposed in [6] is used to compute the geographical distribution of the demand. The total number of requests per content is then divided by the number of geographical locations and assigned to a random set of managers, accordingly, if and only if the number of requests for that content at a specific location is greater than zero. Given that the total number of requests for each configuration cycle follows a constant stochastic model and the number of requests per content is driven by Zipf's Law, an increase in the total number of content items leads to a decrease in the total number of requests per content, resulting to more items having zero requests at certain geographical locations. This decreases the length of some of the messages exchanged in the first phase of the algorithm, thus decreasing

also the Network Time and the Network Traffic. In order to evaluate the effect of the function used to compute the number of requests per content, we also executed tests replacing the Zipf's law with a uniform distribution. This forces each content to be requested at least once and from at least one location. Due to space limitations, we do not show the results here, but it is worth mentioning that, in this case, the value of network related metrics increase with the size of the catalogue.

VI. Related Work

To overcome the limitations of early SDN solutions, which rely on a physically centralized control infrastructure, recent research efforts have focused on distributed control plane approaches (*e.g.*, [16][17]), which come with the set of traditional challenges associated with the implementation of any distributed system (*e.g.*, communication and interaction between distributed entities [18], synchronization issues [19] *etc.*).

To achieve the synchronization of network-wide view between distributed SDN controllers, Tootoonchian et al. developed in [16] a controller-to-controller communication mechanism based on a pub/sub system. A communication protocol to support the exchange of information between SDN controllers, referred to as SDNi, has also been discussed in [20], where the design of an interface mechanism for SDN-based domain controllers is proposed. While these approaches have focused on a horizontal communication design between the controllers, a different direction was followed by Yeganeh et al. in [17], in which communication between control entities is only permitted vertically between a root controller and a set of distributed local controllers. In [21], Levin et al. investigate the problem of interaction between distributed controllers from the perspective of the impact of distribution on the performance of management applications. In particular, the authors discuss two trade-offs which should be taken into account when designing applications for distributed control planes: between performance optimality and state distribution overhead, and, between complexity of the application logic and robustness to inconsistency. The primary objective of most of the communication mechanisms proposed in the SDN literature is to enable distributed controllers to build a global network view. While this is also one of the objectives of the signaling framework proposed in this paper, our solution provides additional functionality for the purpose of coordinating the decisions of distributed management entities.

In addition, while the approaches described above realize distributed control planes, they consider a centralized solution to implement network applications, which cannot achieve reactive and adaptive functionality. Distributed network management application solutions have received significant attention in the context of the research on autonomic networks. Previous efforts have focused on various issues associated with the communication and interaction between distributed management entities, such as the definition of the interaction type [7], the orchestration of distributed decisions [22], or the development of frameworks and protocols for the exchange of information between distributed decision points [23][8][24]. In this paper, we also design a communication protocol for distributed management entities. However, in contrast to previous work which mainly focused on specific use cases, we propose a general and extensible signaling framework that can be used to support any type of application.

The issue of a generic signaling framework, flexible enough to support virtually every kind of signaling application, has been also addressed by the IETF in [25], and further extended in [26]. However, in spite of its generality, the complexity of the framework prevented it from becoming widely used. A generic messaging system able to support peering messaging and to provide a messaging substrate service to overlaying applications has been proposed in [27]. This mainly focuses on the storage, duplication and retrieval of information and, although it is designed to be extensible, it is not suitable to provide a synchronization system to a set of management entities involved in distributed decision-making processes.

VII. Conclusions

This paper presents the design and performance evaluation of a novel signaling framework, which enables communication in a distributed management system. The proposed signaling framework provides both flexibility and extensibility, enabling different types of management applications to interact and control a heterogeneous set of network resources, and allowing the management and control planes to evolve without needing to modify the signaling stack. The results of the evaluation show that the evolution of the network related metrics with the number of managers follows the expected trend (*i.e.*, quadratic), which demonstrates that the signaling framework does not introduce scalability limitations or significant complexity. In fact, the cache management application used for evaluation purposes can be regarded as an extreme case, requiring each distributed application module to build a global view of the content demand and cache status. Other applications could take advantage of the decentralized management framework to limit the overhead, by relying on a gossip-based ASP to spread information in an epidemic fashion, tolerating partial knowledge. Furthermore, the scope of peering between managers can be limited by partitioning the substrate into clusters, or optimizing the structure of the substrate to cope with specific signaling mechanisms and requirements. Future work will investigate and evaluate the impact of such partitioning as well as other management applications with different communication requirements. Another aspect to be investigated concerns the control plane of the management framework, which will involve a complete definition of the interface between LCs and LMs, and a thorough study of the communication between the orchestrator and the distributed entities.

Acknowledgment

This research was funded by the EPSRC KCN project (EP/L026120/1) and by the Flamingo Network of Excellence project (318488) of the EU Seventh Framework Programme.

REFERENCES

[1] H. Kim and N. Feamster, "Improving network management with software defined networking," *Communications Magazine, IEEE*, vol. 51, no. 2, pp. 114–119, February 2013.

[2] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying NOX to the Datacenter," in *Proc. of HotNets-VIII*, 2009.

[3] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On Controller Performance in Software-defined Networks," in *Proc. of Hot-ICE'12*, 2012.

[4] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a Globally-deployed Software Defined Wan," in *Proc. of SIGCOMM'13*, 2013, pp. 3–14.

[5] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou, "Adaptive Resource Management and Control in Software Defined Networks," *Network and Service Management, IEEE Transactions on*, vol. 12, no. 1, pp. 18–33, March 2015.

[6] D. Tuncer, M. Charalambides, R. Landa, and G. Pavlou, "More Control Over Network Resources: An ISP Caching Perspective," in *Proc. of CNSM'13*, 2013.

[7] S. White, J. Hanson, I. Whalley, D. Chess, and J. Kephart, "An architectural approach to autonomic computing," in *Proc. of International Conference on Autonomic Computing*, May 2004, pp. 2–9.

[8] D. Tuncer, M. Charalambides, H. El-Ezhabi, and G. Pavlou, "A Hybrid Management Substrate Structure for Adaptive Network Resource Management," in *Proc. of ManFI'14*, Krakow, Poland, May 2014, pp. 1–7.

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

[10] "The Deltacom topology," 2010, http://www.topology-zoo.org/maps/Deltacom.jpg/.

[11] "The GEANT topology," 2012, http://www.geant.net/Resources/Media_Library/Documents/1084%20GEANT%20Topology%20MAR%2012.pdf.

[12] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng, "Understanding User Behavior in Large-scale Video-on-demand Systems," in *Proc. of EuroSys'06*, 2006, pp. 333–344.

[13] Y. Sun, S. K. Fayaz, Y. Guo, V. Sekar, Y. Jin, M. A. Kaafar, and S. Uhlig, "Trace-Driven Analysis of ICN Caching Algorithms on Video-on-Demand Workloads," in *Proc. of CoNEXT'14*, 2014, pp. 363–376.

[14] Y. Zhu, C. Dovrolis, and M. Ammar, "Combining Multihoming with Overlay Routing (or, How to Be a Better ISP without Owning a Network)," in *Proc. of INFOCOM'07*, may 2007, pp. 839 –847.

[15] G. R. D. Rossi, "Caching performance of content centric networks under multi-path routing (and more)," 2011, Telecom ParisTech, Paris, France.

[16] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proc. of INM/WREN'10*, 2010, pp. 3–3.

[17] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. of HotSDN'12*, 2012, pp. 19–24.

[18] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A Clean Slate 4D Approach to Network Control and Management," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, pp. 41–54, Oct. 2005.

[19] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.

[20] "SDNi: A Message Exchange Protocol for Software Defined Networks (SDNs) across Multiple Domains," Jun. 2012, https://tools.ietf.org/html/draft-yin-sdn-sdni-00. [Online; accessed 08-July-2015].

[21] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically Centralized?: State Distribution Trade-offs in Software Defined Networks," in *Proc. of HotSDN'12*, Helsinki, Finland, 2012, pp. 1–6.

[22] E. Lavinal, T. Desprats, and Y. Raynaud, "A generic multi-agent conceptual framework towards self-management," in *Proc. of NOMS'06*, April 2006, pp. 394–403.

[23] D. Tuncer, M. Charalambides, G. Pavlou, and N. Wang, "DACoRM: A coordinated, decentralized and adaptive network resource management scheme," in *Proc. of NOMS'12*, Maui, Hawaii, Apr. 2012, pp. 417–425.

[24] F. Wuhib, R. Stadler, and M. Spreitzer, "Gossip-based resource management for cloud environments," in *Proc. of CNSM'10*, Oct 2010, pp. 1–8.

[25] R. Hancock, G. Karagiannis, J. Loughney, and S. V. den Bosch, "Next Steps in Signaling (NSIS): Framework," IETF, RFC 4080, Jun. 2005.

[26] M. Femminella, R. Francescangeli, G. Reali, and H. Schulzrinne, "Gossip-based signaling dissemination extension for next steps in signaling," in *Proc. of NOMS'12*, April 2012, pp. 1022–1028.

[27] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol," RFC 6940 (Proposed Standard), Tech. Rep. 6940, Jan. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc6940.txt