A Northbound Interface for Software-based Networks

Daphne Tuncer, Marinos Charalambides, Gioacchino Tangari, George Pavlou
Department of Electronic and Electrical Engineering, University College London, UK

***Abstract*—The current shift from traditional network architectures to software-based solutions is offering new opportunities to allow network functionality to be managed in a flexible way. Substantial efforts have been invested in the recent years in the development of new network management approaches taking advantage of emerging paradigms such as software-defined networking and network function virtualization. Until now however there has not been much progress in the development of a northbound interface (NBI) linking high-level requirements (HLRs) capturing business objectives to management operations. This is a crucial functionality to facilitate faster service deployment and realization of business objectives. In this paper we extend the efforts towards the development of a NBI and propose a novel approach for the automatic decomposition of HLRs to network management operations. We demonstrate its functionality based on representative use cases and evaluate its feasibility through prototype implementation. The results obtained show that our solution can translate new technical requirements to network configurations in the order of a few seconds, as such enabling management of network functionality and services in short timescales.**

I. Introduction

The configuration of network devices is a result of high-level requirements (HLRs) that capture business objectives. These are usually communicated through human channels and find their way to network administrators who are responsible for configuring the network in a way that, according to their knowledge, best meets the objectives. This process is time-consuming but also error-prone, given the large set of configuration parameters and the wide range of services supported by the network. Automating the decomposition of HLRs to configuration settings is a key feature that network and service providers alike would desire to be supported by their systems.

Recent trends are showing a shift from the current rigid network architecture and largely manual configuration processes towards software-based solutions. This is evidenced by the large body of work on virtualized network functions and their orchestration [1][2], as well as the standardization of the southbound interface for software-defined infrastructures [3]. Jointly, these technologies can allow the development of platforms through which network functionality can be flexibly managed and programmed. Despite these efforts, there has not been much progress in the development of a northbound interface (NBI), which links the HLRs to the management and control functions/algorithms that compute operational parameters. Such an interface would not only facilitate faster service deployment / realization of business objectives, and reduced OPEX (due to less human involvement), it would also allow more frequent changes in the HLRs ensuring the fulfillment of service level agreements.

The functionality offered by the NBI interface has been recently dubbed intent-based networking (IBN), in which an intent (of declarative nature) is translated to a form that lower-level entities, such as SDN controllers, can understand. Preliminary work by the Open Networking Foundation (ONF) [4] described the basic properties and structure of the NBI including an engine that is responsible for the translation. The latter is based on mapping tables that can potentially take various sources of information into account. Although useful, this work can only be considered as a set of guidelines for realizing the interface. A more specific solution for the translation of intents has been proposed in [9], but this is limited to the selection and chaining of virtualized network functions. Looking back in the research timeline, IBN has its roots in policy-based management systems and more specifically policy refinement. Several mechanisms have been developed in the past, *e.g.,* [10] and [11], which can be useful in guiding the design of the NBI. Their main limitations however, include the dependence on differentiated services configuration parameters, the tight relation to specific QoS architectures, and the need for domain-specific libraries.

In this paper we extend the efforts towards the development of a NBI and propose an approach that facilitates the automatic decomposition of HLRs to network management operations. The core part of the approach is based on mapping functionality that associates HLRs to services offered by the network operator and functions that manage the network resources. This is realized through matching procedures with the support of operator-defined descriptors that encode distinct features and uniquely identify services and functions. A key attribute of the approach is the use of different abstraction levels for representing information with clear separation of concerns. This allows (i) a global view of the infrastructure for decision making by the functions of a network management system, (ii) simple mapping procedures by the NBI that are generic to multiple functions and services, and (iii) compact and human-friendly representation of HLRs. We demonstrate the functionality of our approach based on representative use cases and evaluate its feasibility through prototype implementation.

The rest of the paper is structured as follows. Sect. II provides an overview of the problem addressed together with representative examples. Sect. III presents the proposed ap-

proach, while Sect. IV focuses on the mapping functionality of the interface. Sect. V describes the use cases considered and Sect. VI presents the evaluation settings and results. Sect. VII discusses related work and Sect. VIII concludes the paper and provides future directions.

II. Decomposing High-Level Requirements

In this section, we present an overview of the problem of decomposing high-level technical requirements into management operations and introduce the main challenges in developing the functionality of a NBI.

A. Problem Overview

Network operators have two main goals when managing their infrastructure. On one hand they need to ensure that they meet their agreements with respect to the services offered to their customers over their infrastructure. These services are diverse and range from basic operational services, such as connectivity between end points, to over-the top services with rich functionality, such as video on demand. On the other hand they strive to make the most out of their infrastructure in order to enhance their offer while accommodating ever growing needs in terms of demand and service quality.

From an operational perspective, these goals translate to technical requirements that can be classified as either *client-facing* or *operator-facing*. The former are concerned with performance guarantees with respect to the offered services (*e.g.,* delay guarantees when delivering traffic), whereas the latter relate to internal operational requirements (*e.g.,* in terms of energy-savings). To meet these requirements, appropriate operational parameters that define how resources in the infrastructure are configured and commodities are regulated have to be computed. More specifically, each requirement can be associated with a specific operational procedure that dictates what is needed from a network perspective to satisfy the requirement. This includes the type of functionality required to provide the service (*e.g.,* forwarding, storage), the type of resources (*e.g.,* switch, link, cache), but also the type of commodities (*e.g.,* traffic, content).

Decisions on the value of the operational parameters are the responsibility of management functions. These are software programs implemented and executed in a network management system. They realize the necessary logic for deciding on resource configurations using real-time information from the underlying network infrastructure. Examples of such functions include adaptive traffic load-balancing, server selection, and virtual machine placement. To drive management decisions in a way such that configurations satisfy high-level technical requirements, the management functions need to be made aware of these requirements. Given that management functions operate at a lower level of abstraction, *e.g.,* representation of resources and actions, a mechanisms needs to be in place to decompose/translate the high-level requirements.

B. Illustrative Examples

We illustrate the requirements decomposition problem through the following two examples.

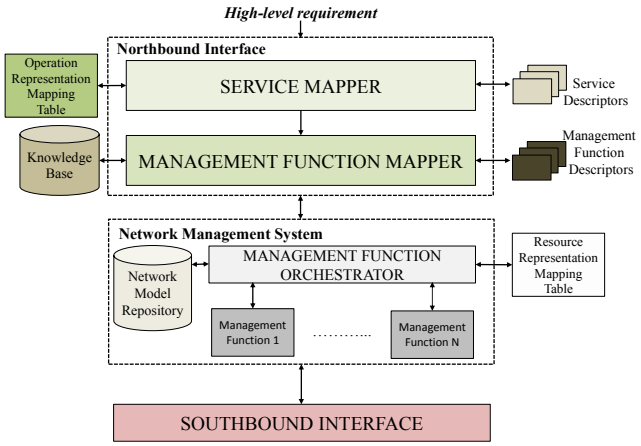


Fig. 1. Architecture.

Example 1 In this example, we consider the case of a new customer request for a connectivity service between two sites (*e.g.,* between two of the customer premises) with delay guarantees. From an operational point of view, this constitutes a new client-facing technical requirement. To satisfy the requirement, a procedure needs to be triggered that first identifies the necessary functionality (forwarding in this case) and subsequently configures the associated resources (*i.e.,* switches) in the infrastructure. From a network management perspective, it necessitates the execution of two functions: one that determines available paths (*i.e., path computation*) and one that decides on the path(s) to use for routing the traffic (*i.e., path selection*). Path computation is based on the relevant end points that map to the two customer sites, whereas path selection uses the delay constraint extracted from the high-level requirement, as well as the type of customer traffic.

Example 2 In this second example, we take the case of a network operator that decides to apply additional processing to the traffic flowing over its infrastructure and, as such, requires traffic between any pair of end points to be redirected to a server that hosts a virtual network function (VNF) realizing the appropriate traffic processing, *e.g.,* for intrusion detection. From an operational perspective, this can be expressed as an operator-facing requirement that will result in first selecting forwarding as the functionality between the relevant network entities but also processing for the corresponding VNF, and then configuring the associated resources (*i.e.,* switches and server). In this case, three types of decisions from three different management functions need to be computed: i) the server to redirect the traffic to (*server selection*), ii) available paths between the relevant end points (*path computation*), and iii) path(s) to use for routing the traffic (*path selection*). Constraints on the type of traffic as well as the end points involved are extracted from the high-level requirement (in the example, *all* traffic and *any* pair of end points) and used by the management functions for computing the new configurations.

III. Proposed Approach

A key challenge is to enable the decomposition of high-level technical requirements in an automated fashion without

TABLE I
HIGH-LEVEL REQUIREMENT EXAMPLE

Forward traffic from client1 between A and B with delay lower than 20 ms

Predicate	Forward
Commodity	Traffic
Target	Resource A → Resource B
Constraint	Delay lower than 20ms
Condition	Traffic from client1

the oprator's involvement. From a system perspective, such a process needs to implement translation functionality linking HLRs to management functions through a NBI. In this section we present a novel approach to realize such functionality.

A. System Architecture

The process of decomposing HLRs to management decisions involves two main operations. First, it needs to identify the best set of management functions to invoke, among available options, that satisfy the requirements. It then needs to instruct and execute the selected functions with the appropriate inputs derived from the requirements.

An overview of the proposed approach is depicted in the architecture of Fig. 1, which relies on two main components. The northbound interface implements the functionality to map the high-level technical requirements to network management functions. In particular, this interface provides an intermediate common abstraction based on which it is possible to, on one hand, parse and express technical requirements and, on the other, expose properties and attributes of management functions so that the mapping from one to the other can be easily realized. It encompasses two main functions that are responsible for (i) determining the association of a HLR to the relevant service offered by the provider and (ii) selecting the set of management functions to execute so as to satisfy the technical requirements of that service.

The second component is the network management system, which hosts management functions realizing the management functionality. These are usually implemented in a modular fashion [12] along with an orchestration entity that supervises their operation. To decide on the configurations (*i.e.,* management decisions), each function takes into account values of long-lived parameters, such as static infrastructure attributes (*e.g.,* link/storage/processing capacity *etc.*), as well as short-lived parameters, representing the context dynamics (*e.g.,* resource utilization, demand *etc.*). These are defined based on an abstracted view of the network infrastructure (including topology and paths) that is maintained in a network model repository. Compared to the NBI, management functions operate at a lower level of abstraction.

Based on this information, each function analyzes the context in which it is operating, decides on the adjustments to perform and computes the new configurations that will need to be enforced on the underlying infrastructure, *e.g.,* new traffic splitting ratios. The resulting configuration decisions are translated into sets of commands and are communicated to the network equipment through a southbound interface (*e.g.,* OpenFlow for forwarding devices), which defines the sequence of actions to be enforced for updating the network parameters.

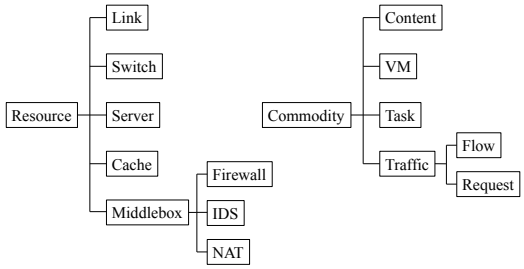


Fig. 2. Snapshot of the NBI network environment model.

B. High-Level Requirements

High-level requirements represent directives specified by the network operator regarding what must be achieved in the infrastructure so as to satisfy business objectives. As explained in Section II-A, these can be either client-facing or operator-facing. In our system, high-level requirements are expressed in a structured way using a general format inspired by the high-level intent examples discussed in [4] and [5]. The proposed format is based on the following five attributes:

1) *Predicate*: represents the type of actions to be executed. Examples include *forward*, *store*, *process*, *etc.*
2) *Commodity*: defines the type of commodity regulated by the actions, for instance *traffic*, *video content*, *etc.*
3) *Target*: identities the network resources involved in the execution of the relevant actions.
4) *Constraint*: stipulates constraints on the actions to execute and by extension on the configurations to enforce.
5) *Condition*: defines the circumstances under which the predicate applies. For instance, this field is used to specify the value of the associated commodities.

An example of a high-level requirement and its representation by the five attributes is shown in Table I.

C. Northbound Interface

The NBI acts as the interface between the high-level technical requirements and the network management system. It provides a common abstraction model of the network environment based on which the attributes of services and management functions can be expressed. In our system, the network environment is represented through a taxonomy of generic types of resources and commodities available in the infrastructure, *e.g.,* switch, server, cache, traffic, request, which is defined by the operator. A snapshot of the NBI network environment model used in this paper is shown in Fig. 2.

From a functional point of view, the NBI has two components: the service mapper and the management function mapper. Their objective is to associate the received HLR to the relevant service and management functions using mapping procedures. These work by comparing available services

TABLE II
SERVICE DESCRIPTOR EXAMPLE

Service Name		Connectivity
OP		Forward
IR	**CO**	Traffic
	RS	Switch;Link
TS	**CO**	All traffic
	RS	Resource A $\rightarrow$ Resource B
	CST	Delay lower than 20 ms
PAF		Dominated by delay_connectivity

and management functions against the attributes of high-level requirements, based on small set of generic descriptive features that are encoded in service and management function descriptors. The descriptors are defined by the operator for each service and management function based on the NBI abstraction model. Details on their structure are provided in Sections IV-A1 and IV-A2.

The service mapper is responsible for associating a high-level requirement with a specific service and subsequently updating the technical specifications of that service with the received input. The objective of the management function mapper is to determine, among the set of functions implemented by the network management system, the ones whose configuration decisions contribute in meeting the technical requirements of the identified service. To aid the selection, the management function mapper makes use of prior knowledge derived from the operator expertise and past experience with respect to management operation best practices. This is represented in the form of rules encoded in a knowledge base.

D. Interface to Network Management System

Information about the selected management functions is passed to the network management system where it is processed by the management function orchestrator. This component is responsible for supervising the execution of the relevant functions. More specifically, it performs two main tasks. It first instructs the relevant management functions about the new execution parameters derived from the HLRs (*e.g.,* constraint values, pair of end points). Given that technical specifications are expressed using a different abstraction model than that used by management functions, a preliminary step involves resolving the correspondence between the two different views. This is done through a resource representation mapping table (Fig. 1) that associates each network resource and commodity to their counterpart in the network model repository. The second task of the orchestrator is to supervise the execution of each function, which, depending on the implementation, can be sequential or parallel.

The orchestrator is also responsible for collecting feedback about the execution outcome of each function, *i.e.,* whether feasible configurations could be computed given the current HLR. Such information is essential for updating the knowledge base, but also for informing the operator in case a HLR cannot be satisfied given current operating conditions.

TABLE III
MANAGEMENT FUNCTION DESCRIPTOR EXAMPLE

Function Name	Path Selection
CO	Traffic
RS	Switch;Link
FCT	Allocate(end_points,delay_constraint)
OPE	delay_connectivity

IV. SERVICE AND MANAGEMENT FUNCTION MAPPERS

In this section, we elaborate on the functionality of the northbound interface providing details on the service mapper and management function mapper components.

A. Descriptors

As explained in Section III-C, HLRs are associated to services and management functions using mapping procedures that compare their representative attributes based on the common abstraction model offered by the NBI. These attributes are captured in templates, called descriptors, which are defined for each service and management function. From a content point of view, descriptors are rich enough to enable each service and management function to be uniquely identified and sufficiently concise to allow for a simple matching procedure.

1) Service Descriptor: It contains four main fields.

Operation (OP): generic description of the basic functionality required to provide the service. It is expressed in the form of one, or a combination of active verbs. Examples include *Forward* for a basic connectivity service, *Store* for a storage service, and *Process+Forward* for a connectivity service involving traffic processing (*e.g.,* through a middlebox).

Infrastructure Requirements (IRs): definition of what is needed in the infrastructure for the service to be operated. This is decomposed into the type of commodities regulated by the service (IR.CO), for instance traffic, and the type of resources involved (IR.RS), for instance a switch. Both commodity and resource types are defined based on the abstraction model of the NBI. IRs are derived at the design phase of a service and, as such, can be represented as static parameters.

Technical Specifications (TSs): specifications based on which the service must be operated. In contrast to IRs, TSs are defined at the deployment phase of a service and periodically updated as new HLRs are specified. They include specifications of the actual commodities regulated by the service (TS.CO), for instance traffic from client1, the resources on which the service is deployed (TS.RS), for instance Resource A $\rightarrow$ Resource B, and constraints specific to the service performance (TS.CST), for instance end-to-end delay.

Performance Affecting Factors (PAF): specification of dominating factors affecting the performance of the service. These are derived by the operator based on past experience; for instance the delay experienced by a service is mostly imputable to transportation delay. In case no predominant factors can be identified, the field is left empty. Information on PAF is particularly useful for identifying the set of management functions that have the potential to make the best configuration

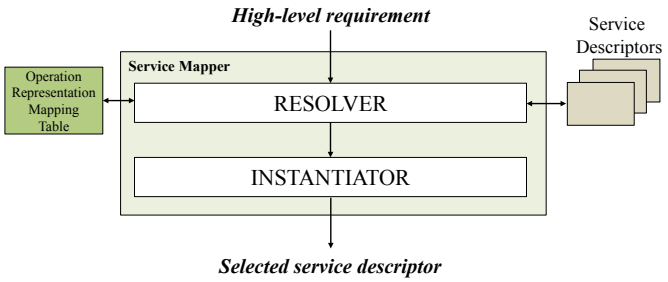


Fig. 3. Service mapper functions.

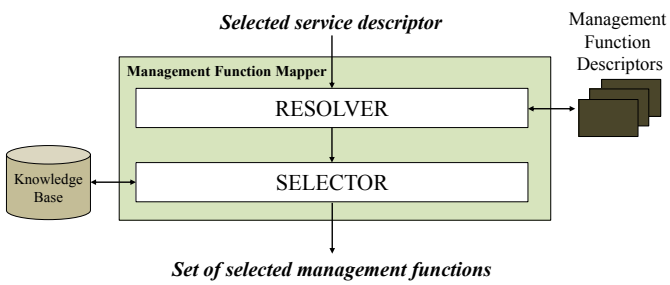


Fig. 4. Management function mapper functions.

decisions with respect to satisfying the service requirement, *e.g.,* a function that has a direct positive impact on delay.

An example descriptor for a basic connectivity service is shown in Table II.

2) Management Function Descriptor: In a similar fashion to the offered services, each management function comes with an abstract representation of its characteristics based on a restricted number of generic features. This representation does not expose any details pertaining to the logic of the function (algorithm, decision variables, *etc.*). Management function descriptors capture the following information.

Commodities (CO): definition of the type of commodities regulated through the actions derived from the decisions of the management function.

Resources (RS): definition of the type of resources whose configurations are computed by the management function.

Function (FCT): generic description of the operations performed by the management function on the associated commodities and resources. It is expressed in the form $function\_name(arguments)$. While the $function\_name$ signifies the actual operations (*e.g., allocate*, *order*, *fetch*, *etc.*), the $arguments$ are the parameters used by the management function to compute resource configurations. These are quantified according to the high-level technical requirements (*e.g.,* delay constraint value, pair of end points, *etc.*).

Operation Effects (OPE): specification of the performance factors affected by the decisions of the management function. This can include different performance factors such as connectivity delay or compute delay.

An example of descriptor for a path selection management function is shown in Table III.

B. Service Mapper

Given that HLRs specify guidelines on how services should be operated and perform, the first step in the decomposition process is to identify the associated service so that its technical specification can be updated. This is realized by the service mapper, which, as shown in Fig. 3, has two components: the resolver and the instantiator.

The objective of the resolver is to extract from the library of service descriptors the template of the service matching a HLR. This is achieved by resolving the correspondence between the *predicate* attribute of a HLR and the *operation* field the service descriptors using the operation representation mapping table. More specifically, each entry in the table is defined as a <key,value> pair where keys correspond to possible predicates and values to service operations. The table is initialized by the operator based on existing services deployed in its infrastructure. The selected template is the one with a matching *operation* field.

Once selected, a service template is passed to the instantiator, which is responsible for updating the technical specifications of the service. This involves amending the TS sub-fields with the value of the commodities, targets and constraints retrieved from the HLR. The instantiated template is returned as output and passed to the management function mapper.

C. Management Function Mapper

Having selected a service along with the associated technical specifications, the second step of the decomposition process is to determine the best set of functions to execute so as to satisfy the given requirements. This is realized by the management function mapper, which, as shown in Fig. 4, has two components: the resolver and the selector.

In a similar fashion to the resolver of the service mapper, the resolver of the management function mapper executes a resolution function. The objective of the resolver is to extract from the library of management function descriptors the template of functions matching the technical requirements of the service selected by the service mapper. This is achived using a matching procedure that compares attributes of the selected service against those of available management functions. More specifically, a management function is said to be relevant to the satisfaction of the service technical requirement if the following three conditions are met:

- function CO field matches service IR.CO field
- function RS field matches service IR.RS field
- function OPE field matches service PAF field

The templates of the matching functions are passed to the selector whose objective is to revise the initial choice of functions using additional knowledge extracted from the knowledge base. For instance, the selector can discard a selected function if previous experience has shown that it only has a marginal effect in meeting the requirements. It can also associate the functions with different priority levels, which can be further used to guide their execution (*e.g.,* according to a specific order). The set of the selected functions along with

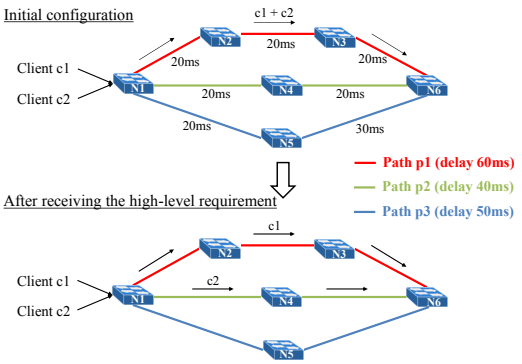


Fig. 5. Use case 1.

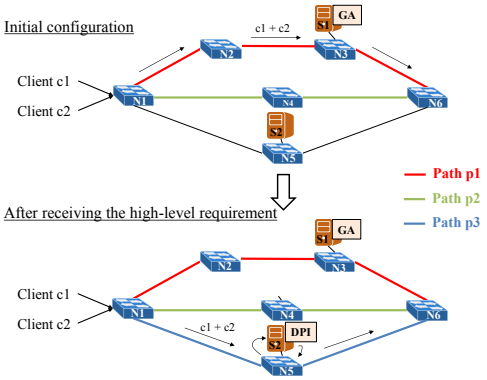


Fig. 6. Use case 2.

their descriptors are sent to the network management system for their execution, as described in Section III-D.

V. Use Cases

In this section, we demonstrate the functionality of the proposed approach based on three realistic use cases. The first two use cases are based on the examples presented in Section II-B. The third use case concerns an on-demand gaming service. In all cases, we assume a SDN-based environment. Initial network configurations need to be updated to reflect new operator- or client-facing high-level technical requirements. For the sake of demonstration, all scenarios are based on a toy network topology.

A. Use case 1: connectivity service with delay constraints

The scenario is depicted in Fig. 5 where traffic from two clients c1 and c2 is forwarded from nodes N1 to N6. Three paths p1, p2 and p3 with different delay can be used to forward the traffic. Initially both c1 and c2 request a basic connectivity service with no delay requirements. Operational parameters are thus set so that all traffic between N1 and N6 is routed on p1 with 60 ms delay. At time t, c2 decides to subscribe to a better connectivity service with delay guarantees of less than 40 ms. To comply with the client's request, new operational parameters need to be computed so as to direct c2 traffic to a path satisfying the delay constraints.

To enforce new network configurations, a high-level requirement is in the form *forward traffic from client c2 between N1 and N6 with less than 40ms delay* is passed to the network management system through the NBI. Given that all paths between N1 and N6 are already instantiated, this results in the selection of only one management function: path selection. Its objective is to select for each client the path(s) on which traffic between the pair of end points is forwarded. While the assignment of c1 to p1 is unchanged, c2 is moved to p2 that satisfies the 40ms delay requirement.

The decisions are enforced in the infrastructure through the southbound interface.

B. Use case 2: traffic redirection through a virtual network function

The scenario is depicted in Fig. 6. Two servers S1 and S2, connected to nodes N4 and N5, respectively, are deployed in the network to host virtual network functions (VNFs). Initially only one VNF is instantiated in the network on S1 that runs an online-gaming application, and two paths, p1 and p2, from nodes N1 to N6 are installed, with all traffic between the two nodes forwarded on p1.

At time t, the operator decides that additional processing should be performed on the traffic routed over its infrastructure and, as such, requests all traffic entering its network to be redirected to a VNF performing deep packet inspection (DPI) before reaching its destination. New operational parameters are therefore needed so that an instance of the appropriate VNF is instantiated on at least one of the servers and all traffic from N1 to N6 is redirected to the relevant server(s) for processing.

To enforce the new network configurations, a high-level requirement in the form *apply DPI to traffic from N1 to N6* is passed to the network management system through the NBI. This results in the selection of four management functions: VNF placement, server selection, path computation and path selection. VNF placement determines on which server(s) to instantiate the DPI VNF. Two servers can be chosen and the function selects the one with the lowest CPU utilization, *i.e.,* S2 since no VNF instances are currently running on that server. Server selection selects the server(s) hosting the relevant VNF to use for processing the redirected traffic. There is only one option in this scenario, *i.e.,* S2. Path computation determines the set of paths between, on one hand, N1 and the server(s) hosting one instance of the DPI VFN, and on the other, this server(s) and N6 (*i.e.,* path p3 in the figure). Finally path selection selects the path(s) to use for redirecting the traffic to the selected server, *i.e.,* path p3 here.

An aspect of the proposed approach that comes into play in this scenario is the mode of execution of the four management functions. We use here a simple rule by which functions are executed in a sequential manner as follows: VNF placement, server selection, path computation and path selection[1]. The rule is retrieved from the knowledge base by the management function mapper that computes priorities for the selected functions. These are passed to the orchestrator component in

[1]This order is chosen for the sole purpose of demonstration. In practice, more complex methods involving the parallel execution of the functions could be used, for instance to achieve some performance trade-off between the decisions of the different functions.

the network management system that supervises the execution of the four functions.

C. Use case 3: traffic redirection through a virtual network function

In this use case, we consider a scenario where an on-demand gaming (cloud gaming) service is offered by the network provider/operator, in a similar fashion to well-known platforms such as Gaikai [6]. To offer such a service, specialized hardware resources, such as GPUs and fast memory, are deployed in the operator's infrastructure, so as to support the computation required for the user game experience, which is offloaded from the end-user devices. The use case scenario is depicted in Fig. 7.

We assume that two different service classes are available: a basic class and a premium class providing personalized user gaming experience. Requests of clients subscribing to the basic class are served from a server supporting standard gaming experience, *i.e.,* server S1 attached to node N3, while requests for the premium service are directed to a server with higher processing capabilities supporting enhanced gaming experience, *i.e.,* server S2 attached to node N6. Initially, two clients c1 and c2 subscribe to the basic service class and their requests are directed to S1 on path p1. At time t, c2 decides to upgrade his subscription to the premium service. Operational parameters are updated so that c2 can be served from S2.

A new high-level requirement in the form *serve requests from client c2 from the server supporting enhanced gaming experience* is formulated, which results in the selection of three management functions: server selection, path computation and path selection. Server selection selects the appropriate server for each client, *i.e.,* S1 for c1 and S2 for c2. Path computation determines the set of paths between the clients and the servers. In this case, only the paths between c2 and S2 need to be computed, *i.e.,* paths p2 and p3. Finally path selection selects the path(s) to use for routing client requests to the relevant servers. In case multiple paths are available, all requests are initially routed on the same path until a utilization threshold is reached, in which case a new path is used. This results in the selection of p1 for c1 and p2 for c2. In a similar fashion to use case 2, management functions are executed in a sequential manner with server selection taking precedence over path computation, followed by path selection.

VI. Proof-of-Concept

We evaluate the feasibility of our approach for the automatic decomposition of HLRs to management decisions based on the three use case scenarios and network topology described in Sect. V. Our objective is to show that the proposed approach is applicable to different types of services and technical requirements, as well as management functions.

A. Implementation

We developed a prototype implementation of the different components of the NBI and network management system, including each management function module, in Java. To

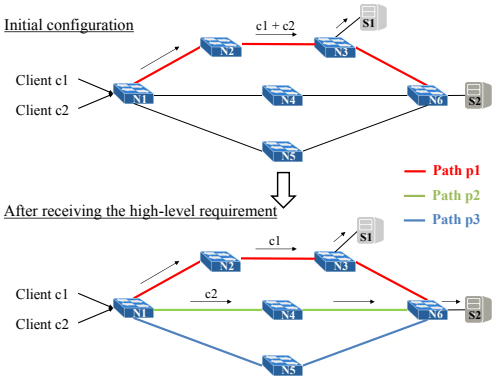


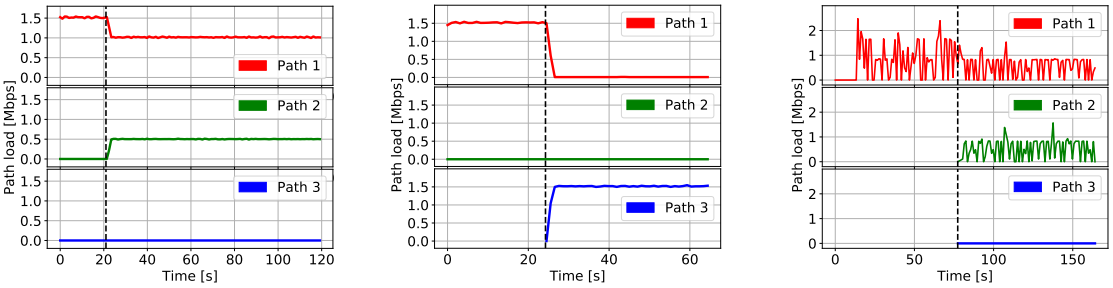
Fig. 7. Use case 3.

emulate the network topology, including hosts (*i.e.,* clients and servers), as well as OpenFlow switches, we used the Mininet environment. The network management system is interfaced to Mininet via a REST API. In addition, we reused a small set of APIs from the SDN controller POX [7] to implement the southbound interface functionality. High-level requirements, service descriptors and management function descriptors are encoded in the JSON file format. All links in the topology have 10Mbps bandwidth and latency between 20 ms and 30 ms as shown in Fig. 5. In all use cases, c1 and c2 are attached to N1. In use case 2, servers S1 and S2 are attached to nodes N4 and N5, respectively, and to nodes N3 and N6, in use case 3. In use cases 1 and 2, we use fixed-rate UDP flows generated with *Iperf* to emulate clients traffic. In the case of cloud gaming (use case 3) we emulate the corresponding network traffic by generating separate client and server-originated packet streams, with characteristics based on the results reported in [8].

B. Evaluation

In each experiment, the JSON file of the corresponding HLR is given as input to the NBI where the relevant operations are performed to select the management functions to invoke. These are instructed with the new input parameters and executed through the management system. Decisions are passed to the controller where they are translated into instruction commands to be enforced in the switches (*i.e.,* new flow rules).

The result of this process is shown in Fig. 8 for the three use cases, where the traffic load over time on the different paths is reported. In all cases, the time at which the requirement is passed to the NBI is marked by a dotted vertical line. As observed in Fig. 8a for use case 1, the receipt of the requirement at time t=21s results in traffic from c2 on p1 to be shifted to p2. The load on p3 remains 0 as no traffic is redirected to that path. In the case of use case 2 (Fig. 8b), the received requirement at time t=24s leads to the instantiation of a new path (p3) and the redirection of all traffic from p1 to that path. Path p2 is not used and its load is 0 throughout the experiment. Finally Fig. 8c depicts the result of the receipt of the requirement in use case 3. Two new paths, p2 and p3, are instantiated after time t=75s with p2 only receiving c2 traffic originally assigned to p1.



(a) Use Case 1. (b) Use Case 2. (c) Use Case 3.

Fig. 8. Result on network configurations of new high-level requirements.

In all cases, it should be noted that the time between receiving the high-level requirement and enforcing the new network configurations is less than few seconds. This time is composed of three main factors: the time to execute mapping procedures in the NBI, the time for management functions to compute new configurations and the time to enforce the new configurations in the network devices, with the latter being the dominating factor in our evaluation (*i.e.,* time for the SDN controller to update flow rules in the switches). In the proposed approach, the time-complexity of the mapping procedures performed by the NBI is constant with the number of services and management functions. As such, it can be inferred that in practice the NBI time will always be dominated by the two other factors.

VII. Related Work

The problem of decomposing high-level goals to network configuration parameters was initially studied in the context of policy-based management. This is known as policy refinement and a number of solutions have been proposed in the literature. The authors of [13] and [10] base their approach on goal elaboration and formal reasoning techniques, where so called operational policies are derived through strategies that can achieve a given high-level goal. Another approach, which is described in [15][14] and [11], is based on the same goal elaboration method, but uses model checking instead, to verify that low-level policies fulfil the objectives. Despite their novelty at the time, both approaches are based on application-specific policy refinement patterns, which limit their applicability. Not only did they target policies specific to the QoS domain, they also relied of a specific QoS provisioning architecture and associated network technologies. The authors of [16] proposed an approach that is not bound to a specific application domain. However, part of their solution is based on predefined static transformation rules for each of the high-level goals, which cannot flexibly accommodate new administrative goals. Also, the case-based reasoning part of the solution, where table look-ups are used for decomposing the goals, is limited to cases for which it is feasible to build a database of the requirements and configuration parameters.

In the context of software-based networking, a number of initiatives have focused on the issue of consistent policy composition [17][18][19], where the objective is to combine multiple independent and potentially conflicting policies into a unique global policy satisfying both individual requirements and infrastructure invariant. These initiatives are complementary to the problem addressed in this paper. In particular, the proposed approaches could be used upstream of the NBI for checking the coherence between multiple high-level technical requirements and harmonize them in case of incompatibility.

Finally related to the work presented in this paper are the research efforts reported in [20][21][22][23] that focused on the development of common abstraction models to enable the smooth integration and co-execution of independently developed SDN control/application modules (*i.e.,*, for instance implemented in different languages). These approaches are particularly relevant to support the modular implementation of management functions in the network management system.

VIII. Conclusions

In this paper, we propose a novel approach for the decomposition of high-level technical requirements to management operations. A key feature of the proposed approach resides in the functionality of a NBI that automatically associates HLRs to network management functions. We developed a prototype implementation of the proposed approach and demonstrated its functionality based on three use cases. We showed that with our solution, new technical requirements can be translated to new network configurations in the order of few seconds, as such enabling management of network functionality and services in short timescales. Our approach is simple - all components of the architecture were implemented with less than $\sim$1000 lines of code. In addition, it is lightweight. It relies on simple mapping procedures with the time-complexity being constant with the number of services and management functions. Finally, it uses simple forms to represent technical requirements so that the communication overhead towards both the NBI and the network management system is small, and as such, does not consume substantial amount of resources. In the future, we plan to extend this work by taking into account other use case scenarios, in particular security management and network monitoring, and apply the solution to realistic network topologies.

REFERENCES

[1] ETSI GS NFV-MAN 001,"Network Functions Virtualisation (NFV); Management and Orchestration," http://www.etsi.org/, December 2014, ETSI Industry Specification Group (ISG).

[2] R. Mijumbi, et. al, "Orchestration Challenges in Network Function Virtualization", *IEEE Communications, Topics in Network and Service Management*, Vol. 54, No. 1, pp. 98-105, January 2016.

[3] ONF TS-025,"OpenFlow Specifications v.1.5.1", http://www.opennetworking.org/images//openflow-switch-v1.5.1.pdf [Online; accessed 22-Jun 2018].

[4] ONF TR-523,"Intent NBI Definition and Principles", https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-523_Intent_Definition_Principles.pdf [Online; accessed 22-Jun 2018].

[5] Huawei Developer, "Intent NBI for Software Defined Networking", http://developer.huawei.com/ict/en/site-sdn/article/08 [Online; accessed 22-Jun 2018].

[6] Gaikai video game streaming platform, https://www.playstation.com/en-us/explore/playstation-now/ [Online; accessed 22-Jun 2018].

[7] POX OpenFlow controller, http://www.noxrepo.org [Online; accessed 22-Jun 2018].

[8] M. Manzano, J.A. Hernandez, M. Uruena, E. Calle, "An empirical study of Cloud Gaming," *in Proc. Netgames*, Venice, Italy, Nov. 2012.

[9] E.J. Scheid, et. al, "INSpIRE: Integrated NFV-baSed Intent Refinement Environment," *in Proc. IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Lisbon, Portugal, May 2017.

[10] A.K Bandara, et al.,"Policy Refinement for IP Differentiated Services Quality of Service Management," *IEEE Transactions on Network and Service Management (TNSM)*, Vol. 2, No. 2, 2006.

[11] J. Rubio-Loyola, et al.,"A Methodological Approach toward the Refinement Problem in Policy-Based Management Systems," *IEEE Communications, Topics in Network and Service Management*, Vol. 44, No. 10, October 2006.

[12] D. Tuncer, M. Charalambides, S. Clayman, G. Pavlou, "Adaptive Resource Management and Control in Software Defined Networks," *IEEE Transactions on Network and Service Management (TNSM)*, Vol. 12, No. 1, pp. 18-33, March 2015.

[13] A.K Bandara, et al., "Policy refinement for DiffServ quality of service management," *Proceedings of 9th IEEE/IFIP Int. Symposium on Integrated Network Management (IM 2005)*, Nice, France, May 2005.

[14] J. Rubio-Loyola, et al.,"Using linear temporal model checking for goal-oriented policy refinement frameworks," *Proc. of 6th IEEE Workshop on Policies for Distributed Systems and Networks (Policy 2005)*, Stockholm, Sweden, June 2005.

[15] J. Rubio-Loyola, et al.,"A Functional Solution for Goal-oriented Policy Refinement," *Proceedings of 7th IEEE Workshop on Policies for Distributed Systems and Networks (Policy 2006)*, Ontario, Canada, June 2006.

[16] J. Rubio-Loyola, et al.,"Policy Transformation Techniques in Policy-based Systems Management," *Proceedings of 5th IEEE Workshop on Policies for Distributed Systems and Networks (Policy 2004)*, New York, USA, June 2004.

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

[18] C. Prakash, et al., "PGA: Using graphs to express and automatically reconcile network policies," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 29-42, 2015.

[19] A. Abhashkumar, et al., "Supporting Diverse Dynamic Intent-based Policies using Janus," *in Proc.of the ACM 13th International Conference on emerging Networking EXperiments and Technologies*, Nov. 2017, pp. 296-309.

[20] R. Soulé, S. Basu, R. Kleinberg, E.G. Sirer, and N. Foster, "Managing the network with Merlin," *in Proc. of the 12th ACM Workshop on Hot Topics in Networks*, Nov. 2013.

[21] M. Aouadj, E. Lavinal, T. Desprats, and M. Sibilla, "Towards a modular and flexible SDN control language," *in Proc. Global Information Infrastructure and Networking Symposium-GIIS*, Sept. 2014.

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

[23] V. Heorhiadi, M.K. Reiter, and V. Sekar, "Simplifying Software-Defined Network Optimization Using SOL," *in Proc. NSDI*, Mar 2016, pp. 223-237.