POLITECNICO DI TORINO
Repository ISTITUZIONALE

A twofold model for VNF embedding and time-sensitive network flow scheduling

Original
A twofold model for VNF embedding and time-sensitive network flow scheduling / Bringhenti, Daniele; Valenza, Fulvio. - In: IEEE ACCESS. - ISSN 2169-3536. - ELETTRONICO. - 10:(2022), pp. 44384-44399. [10.1109/ACCESS.2022.3169863]

Availability:
This version is available at: 11583/2962049 since: 2022-07-04T09:03:34Z

Publisher:
IEEE

Published
DOI:10.1109/ACCESS.2022.3169863

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

19 April 2024

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 15, 2022, accepted April 20, 2022, date of publication April 22, 2022, date of current version April 29, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3169863

A Twofold Model for VNF Embedding and Time-Sensitive Network Flow Scheduling

DANIELE BRINGHENTI, (Graduate Student Member, IEEE), AND FULVIO VALENZA, (Member, IEEE)
Dipartimento di Automatica e Informatica, Politecnico di Torino, 10129 Torino, Italy
Corresponding author: Daniele Bringhenti (daniele.bringhenti@polito.it)

ABSTRACT The revolution that Industrial Control Systems are undergoing is reshaping the traditional network management and it is introducing new challenges. After the advent of network virtualization, an enhanced level of automation has been required to cope with the safety-critical mission of industrial systems and strict requirements for end-to-end latency. In the literature, there have been attempts to automatically solve two strictly interconnected problems for the management of virtual industrial networks: the Virtual Network Function embedding and the time-sensitive flow scheduling problems. However, dealing with these problems separately can lead to unoptimized results, or in the worst case to situations where the latency requirements cannot be satisfied because of a poorly chosen function embedding. In light of these motivations, this paper proposes a formal approach to jointly solve the two problems through an Optimization Satisfiability Modulo Theories formulation. This choice also allows combining two vital features: a formal guarantee of the solution correctness and optimization targeting the minimization of the end-to-end delay for flow scheduling. The feasibility of the proposed approach has been validated by implementing a prototype framework and experimentally testing it on realistic use cases.

INDEX TERMS Network flow scheduling, time-sensitive SDN, VNF embedding.

I. INTRODUCTION

Nowadays, *Industrial Control Systems* (ICSs) are undergoing a deep transformation of their communication infrastructures towards increased connectivity of devices and extreme flexibility of industrial plants. Examples where this transformation is evident are within the Industry 4.0 and *Factory of the Future* (FoF) environments [1], [2]. Innovative industrial applications are also exploiting benefits from *Software-Defined Networking* (SDN) and *Network Functions Virtualization* (NFV) [3], [4], two virtualization approaches that determine a detachment from the traditional industrial network design and the power to administer the entire network environment from a centralized point. Specifically, there have been several attempts to introduce SDN support for the strict time requirements of *Time-Sensitive Networking* (TSN), resulting in the *Time-Sensitive Software-Defined Networking* (TSSDN) technology [5].

Two critical requirements emerge in a TSSDN environment: high security and low end-to-end latency. About the former requirement, the number of cyberattacks driven through the SDN-based industrial network infrastructure is increasing, and they can exploit inadequate segregations between different network environments [6]. Additionally, cyberattacks are increasing when the corporate network is not properly segregated from the industrial one, as well as when it exposes industrial systems that are potentially vulnerable. Maintaining industrial network systems safe is challenging also due to their safety-critical mission inside the manufacturing process, and it cannot be managed manually anymore. As the higher flexibility of TSSDN systems requires frequent reconfiguration, an enhanced level of automation in the management of cybersecurity has become necessary [7]. Regarding the latter requirement, TSSDN systems should regularly make decisions, report data to a centralized collector, and execute a remote command with a deterministic end-to-end delay. If the operation is not completed in a specific timeslot, the entire process may be invalidated [8]. The consequence could vary depending on the industrial environment type. In some environments, it can cause production line breaks and only human assistance can restore the proper system functioning. There are also safety-critical systems where issues

The associate editor coordinating the review of this manuscript and approving it for publication was Gerard P. Parr.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

in industrial systems communication could cause ecological disasters or incidents that result in loss of life. Bringing innovation inside these particular systems is very hard to achieve. It is imperative to meet several requirements before deploying solutions inside real ICSs.

Addressing these two requirements implies solving two different problems. On the one hand, it is necessary to establish the deployment scheme of the *Virtual Network Functions* (VNFs) composing the service, like firewalls or intrusion detection systems, in such a way that the scheme proficiently uses the resources of the physical network and ensures that the traffic can safely reach the destination. This problem is commonly known as the VNF embedding problem in the literature [9]. On the other hand, it is also essential to determine the optimal scheduling of time-sensitive flows, i.e., packets that cross a TSSDN-based network, so as to minimize their end-to-end delay. Making improvements in terms of security to the global network environment implies the introduction of middleboxes that can analyze traffic, keep track of it and possibly block flows considered malicious. The logic of the blocks can vary depending on the middleboxes referred to, but they all have in common the introduction of non-negligible latency for the time-sensitive flows.

Therefore, currently there is a high need to guarantee real-time achievement requirements for networked hosts that in a critical environment such as the industry remain of primary importance. Unfortunately, even though both the aforementioned problems (i.e., VNF embedding and scheduling of the time-sensitive flows) concur to this guarantee, they are commonly challenged separately in the literature. The results of applying two separate algorithms are thus sub-optimal, as the computed embedding scheme may not be the most suitable one to minimize the delay for time-sensitive flows. Additionally, most of the related work about VNF embedding overlooks common characteristics of TSSDN, so they cannot be applied at all to this peculiar environment.

In light of these considerations, this paper proposes a novel approach solving simultaneously and automatically the VNF embedding and time-sensitive flow scheduling problems. In doing so, it formalizes the problem with an *Optimization Satisfiability Modulo Theories* (OptSMT) formulation, with the aim of introducing two vital features for TSSDN: formal verification and optimization. The former can guarantee that the solution output by the proposed approach is effectively correct, and is devoid of all the manual errors that human users commonly produce. The latter is required for compliance with the strict time constraints characterizing communications among ICSs. This approach represents a step ahead in the literature related to TSSDN, as the following novelties enhance it:

- By jointly automating two critical processes, i.e., VNF embedding and flow scheduling, this approach improves their effective results, because the two operations work on the same network elements and strongly influence each other. This aspect has high importance for the management of TSSDN-based networks, because of the specific requirements in terms of efficiency and latency that have been previously illustrated.
- The OptSMT formulation avoids the application of an a-posteriori formal verification technique, differently from what commonly occurs in the literature, because it pursues a so-called ''correctness-by-construction'' approach where the guarantee of the solution correctness is automatically derived from the formal definition of network and flow elements on the one hand, from the problem constraints on the other one.
- The formal models not only formalize the interconnection among the functions composing the network topology, but also take into account their behavior (e.g., the filtering rules of firewalls). Therefore, the model complexity is higher than the models of networks simply composed of routers or switches. This feature is also crucial for establishing a correct and optimal mapping of VNFs onto the corresponding physical topology, as it helps in establishing paths that allow the time-sensitive flows crossing them, thus guaranteeing the effective reachability among hosts.

The remainder of this paper is structured as follows. Section II discusses related work, underlining the benefits provided by the proposed approach, which simultaneously solves the VNF embedding and time-sensitive flows scheduling problems. Section III provides a high-level description of the approach, aiming at explaining the main principles behind its behavior. Section IV delves into the formalization of the OptSMT problem that must be automatically solved. Section V describes a prototype framework developed for testing the formal approach, and it discusses its validation. Finally, Section VI briefly draws conclusions and discusses future work.

II. RELATED WORK

Two main literature areas pertain to the approach proposed in this paper: VNF embedding (Subsection II-A) and time-sensitive flow scheduling (Subsection II-B).

A. VNF EMBEDDING

The VNF embedding problem consists of mapping the virtual functions composing a network and security service to the candidate substrate resources, represented by general-purpose commodity servers. Only if an embedding plan where all virtual resources can be mapped is computed, the entire virtual network is then embedded and substrate resources are effectively employed. This problem has been extensively analyzed for NFV-based networks in the literature, as shown in a recent survey [9]. Most of the past work formulate the embedding problem using combinatorial approaches such as mathematical programming. In the most relevant papers proposing an exact formulation of the VNF embedding ([10]–[15]), integer programming or mixed integer linear programming are the most commonly used techniques to solve the problem, with the exception of [15] where an SMT formulation is used. Other papers ([16]–[21])

introduce heuristic algorithms to quicken the resolution of the VNF embedding problem, and provide higher compliance with the strict time requirements of virtual networks.

Nevertheless, all these works have limitations with respect to the formal approach presented in this paper. On the one hand, a formalization based on mathematical programming commonly entails the definition of problems with a bigger size than the OptSMT formulation, because it is limited to arithmetic expressions over integer, binary, or real variables [22]. Even though preliminary studies existed on transforming propositional calculus statements into mathematical programs ([23], [24]), which would avoid the manual formulation of those big problems, these studies acknowledged that the transformation is impractical in most cases and did not get significant traction in the literature. Besides, an OptSMT formulation is richer because it is sustained by theories such as bit-array and string ones, enabling higher fidelity for representing the network behavior and their incorporation into the problem constraints. On the other hand, even though heuristic algorithms may be faster than mathematical programming or OptSMT formulations, they cannot provide formal assurance for the correctness of the computed solution, and may output suboptimal VNF embeddings with consequent excessive resource consumption. Therefore, two main features characterizing our approach, i.e., formal verification and optimization, are commonly overlooked in those works.

Among the papers related to this literature area, [14], [18], [19], and [20] are the only ones dealing with the minimization of latency when computing the VNF embedding. However, not only they do not solve the time scheduling problem for time-sensitive flows, but they also have other shortcomings. [18] simply aims at minimizing the number of deployed network functions, but it fails in investigating resource-constrained scenarios of the embedding problem. [19] and [14] address resource-constrained cases, but they are limited to smart grids placement. [20] presents an optimization algorithm for the VNF embedding problem over a set of distributed substrate nodes taking into account the maximum allowed end-to-end delay, but it considers each network flow separately, without providing a full optimization at graph level.

B. TIME-SENSITIVE FLOW SCHEDULING

Scheduling network communications under strict real-time requirements is a problem that has been known in the literature for years. Few resource-constrained approaches were already proposed for the transmission scheduling at the hosts in multi-hop Ethernet-like networks [25], [26], and other ones tried to combine the computation of the scheduling for both transmissions and networking software tasks [27], [28]. However, all these initial approaches assume previous knowledge of the routes crossed by the communications, instead of using an appropriate algorithm to decide them in a way to optimize the flow scheduling (e.g., our approach computes the route with the minimum number of hops for each time-sensitive flow).

A larger batch of studies ([29], [30] [31], [32] [33], [34]) extended their scope to TSSDN-based environments. Their ideas aim at matching the propositions related to end-to-end latency requirements that were the focus for two large organizations in the world of networking: the IETF DetNets Working Group and the IEEE 802.1 Time-Sensitive Networking Task Group. The milestone in this literature area is represented by [29], the first work to exploit the logical centralization paradigm of SDN to compute a transmission schedule for time-sensitive flows on a global view. Multiple integer linear program formulations that solve the combined problem of routing and flow scheduling are also presented for the first time, and they were used as the foundation for future work. Variations of that initial study, having more features and higher efficiency, are [30], [32], and [34]. First, [30] adapts the job-shop scheduling problem [35] to a nowait packet scheduling problem for the offline calculation of schedules for periodic time-sensitive flows, aiming at minimizing the network delay and compacting the schedule length. They also side this formulation with a heuristic optimization algorithm based on the Tabu search meta-heuristic, thus allowing a more efficient schedule computation. Second, [32] presents an algorithm that reduces the time complexity in the schedule computation while maintaining the quality of the scheduling system. Third, [34] applies the Logic-Based Benders Decomposition on the time-sensitive flow scheduling problem, as according to their experimentation a model based on that logic exhibits better performance and significantly increases the schedulability. Finally, some studies attempt to solve corner cases of the scheduling problem. On the one hand, [31] proposes alternative integer programming formulations for managing the problem of incremental schedules, so as to approximate the optimal solutions of the corresponding static scheduling problem. On the other hand, [33] tries to ensure that no frames are lost during network update, and avoids the introduction of extra update overhead in the scheduling computation. In doing so, it proposes an online algorithm that consumes less time than a static one, even though it has slightly decreased schedulability.

Even though the studies belonging to this batch try to combine the routing and flow scheduling problems, they overlook the VNF embedding problem. Our approach is thus more feature-complete than the state of the art, both in terms of optimization and formal verification, because it can compute a solution that is globally optimal for both VNF embedding and flow scheduling, and due to the correctness-by-construction principle pursued through the OptSMT formulation.

III. APPROACH

The proposed approach aims at two simultaneous objectives:

- solving the VNF embedding problem, by placing the VNFs included in the topology of a virtual network onto the general-purpose servers composing the substrate physical infrastructure, so as to satisfy the resource consumption constraints;

- solving a critical problem for TSSDN networks, i.e., the flow scheduling, by establishing the optimal scheduling of time-sensitive network flows which minimizes the end-to-end delay without exceeding a threshold of maximum acceptable latency.

Multiple benefits derive from solving these two problems simultaneously. A first one consists in determining a VNF embedding scheme where also the time-sensitive flows are directly mapped, avoiding inconsistencies between the virtual paths and the physical paths that the flows effectively cross depending on their scheduling. A second one is that common errors due to the application of two separate algorithms for solving the two problems are avoided, because a solution is effectively reached only if it satisfies the requirements of both the problems. Then, a third benefit is that it is less problematic to introduce critical features for TSSDN environments such as formal verification and optimization. These characteristics are commonly counterbalanced by a higher time required for computing the problems to which they are associated. However, in this proposal, time is saved by combining two problems into a single one, so adding formal verification and optimization is feasible.

The inputs required by the approach, which the network operator can specify, consist in the description of the virtual and physical network topologies, and the declaration of the time-sensitive flows that should cross them (Subsection III-A). This information is employed to build an optimization problem, called OptSMT problem, aiming at reaching the two proposed objectives, i.e., minimization of resource consumption and end-to-end delay (Subsection III-B). At that point, an automated OptSMT solver outputs both the VNF mapping onto the physical infrastructure and the scheduling of the time-sensitive flows (Subsection III-C).

A. INPUTS OF THE APPROACH

The inputs required by the approach are: 1) the description of the topologies for the virtual and physical networks; 2) the specification of the time-sensitive flows.

1) VIRTUAL AND PHYSICAL TOPOLOGIES

The virtual network topology represents the logical interconnection of the VNFs, aiming at providing a complete end-to-end service. In the latest years, the idea of network devices seen as devices built to perform a specific function has been progressively abandoned. Thanks to the virtualization layer provided by the ETSI framework for the NFV environment, it is possible to define a series of functions that can then be deployed on one device rather than another in a completely transparent way. Various network reconfigurations, also depending on the general environment state, are managed by an orchestrator software. This element is also in charge of deploying the VNFs on a physical substrate that consists of general-purpose servers.

The entities involved in the virtual and physical topologies are different. In general, within the virtual graph VNFs act as middleboxes, depending on their configuration. Middleboxes are devices that perform an active function when forwarding packets within the network infrastructure. Examples of middlebox are firewalls, proxies, DPIs, NATs. The task of the network operator is to specify the configuration of these devices and then define the role they must play within the infrastructure in question. For example, an operator may want to define the firewall configuration, defining access lists that allow the transit of only one type of traffic. NAT configuration in the network peripheral areas may also be handled for publications and traffic management to the external network. An example of virtual network topology is depicted in FIGURE 1, where the network operator defines the role that the VNFs play. For example, ENDHOST A and ENDHOST B can be two client devices in an industrial network that need to exchange information with ENDHOST C and ENDHOST D, which instead act as servers. VNF1, VNF2, VNF4 and VNF5 can play the role of a firewall, filtering out not allowed traffic while VNF3 can be a DPI, analyzing the traffic to prevent possible attacks.

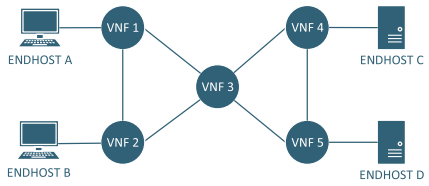

FIGURE 1. Virtual network example.

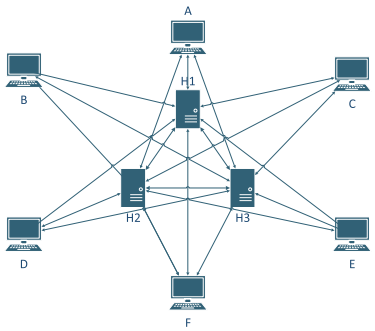

FIGURE 2. Physical network example.

Instead, the entities involved in the physical infrastructure are straightforward general-purpose servers. The network operator defines the characteristics in terms of available resources and therefore quantity of RAM, CPU power, storage and connections with other hosts. It should be noted that both the processing operations of the packet on each host and each physical connection among the various hosts influence the forwarding of the network packets, introducing a latency that is not always acceptable. In FIGURE 2 there is an example of a physical topology that can be exploited as a substrate for the virtual topology represented in FIGURE 1. When defining the physical topology, it is necessary to specify the physical resources available for each physical host.

These parameters are then taken into account during the embedding operations of the VNFs. Physical resources must not be saturated to ensure the correct behavior of the network and the correct management of all the flows.

2) TIME-SENSITIVE FLOWS

The network operator specifies the time-sensitive flows while using the virtual topology as a reference. For example, the network operator may want to put two machines in communication, but the flow must be processed through particular network functions. It could be required to keep track of the flow or verify that legal operations are performed during data exchange between the devices involved in that flow. In an environment like the industrial one, it is essential because of the criticality of the devices involved. Having a virtual abstraction of what happens inside the network can significantly help the network operator.

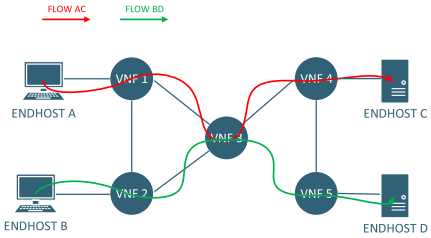

FIGURE 3. Virtual flows definition example.

Concerning the example illustrated in FIGURE 1, let us suppose to specify two time-sensitive flows, so with the need to be handled providing them guarantees about end-to-end latency communication. For the flow from ENDHOST A to ENDHOST C, packets must be processed by VNF1 and VNF4 before reaching their destination. Instead, for the flow from ENDHOST B to ENDHOST D, they must cross VNF2 and VNF5. These paths, graphically depicted in FIGURE 3, must be taken into account when mapping the virtual topology onto the physical infrastructure.

B. OPTSMT PROBLEM

The approach uses the inputs specified by the network operator to build an OptSMT (Optimization Satisfiability Modulo Theories) problem. This formulation is characterized by two types of clauses: hard constraints and optimization constraints. The first ones must always be satisfied in order to achieve a correct solution, and therefore they represent elements such as the connections among VNFs or physical hosts and the time-sensitive flow paths. The second ones are assignments that require minimizing an objective function, e.g., minimization of resource consumption or end-to-end delay. However, the bigger minimization is achieved compatibly with the satisfaction of the hard constraints. If an objective function could be further minimized by not satisfying a hard constraint, this solution would be forbidden and would not be produced as output.

Formulating the problem as OptSMT enables the introduction of formal verification and optimization. On the one hand, an OptSMT formulation lays its foundation on the correctness-by-construction principle. The produced solution is already correct without applying a-posteriori formal verification techniques because it is derived from formally defined hard constraints. On the other hand, the optimization constraints allow reaching the best solution among the ones that would only satisfy the hard constraints.

A detailed description of all the hard and optimization constraints composing the OptSMT problem will be presented in Section IV.

C. OUTPUTS OF THE APPROACH

An automated solver is fed with the formulated OptSMT problem and simultaneously produces two outputs: 1) the VNF mapping onto the topology of the substrate physical infrastructure; 2) the scheduling of the time-sensitive flows.

1) VNF MAPPING ONTO THE PHYSICAL NETWORK

For each pair of virtual and physical topologies, there are several possibilities for deploying VNFs on the physical substrate. All the various possibilities are taken into account by the solver, which at the end chooses the best VNF positioning that can ensure the correct management of time-sensitive flows declared by the user in the shortest possible time, and while satisfying the resource consumption constraints. FIGURE 4 shows a possible deployment example of VNFs declared in virtual topology of FIGURE 1 onto the substrate physical infrastructure of FIGURE 2, according to the requirements for the VNFs to properly operate and physical resources available on the substrate network.

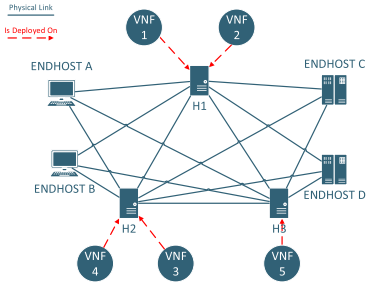

FIGURE 4. Deployment example.

Not only the VNFs, but also the time-sensitive flows that are declared in the virtual network topology must be mapped to the corresponding flows in physical topology. In fact, a goal of the proposed approach is to find an optimal positioning of the VNFs that can satisfy the user-defined requirements on the virtual topology, scheduling transmission over time. Resuming the deployment scheme example shown in FIGURE 4, the flows can be mapped as depicted in FIGURE 5. In this mapping, many overlappings between the flows must be managed to avoid the creation of queues in the network.

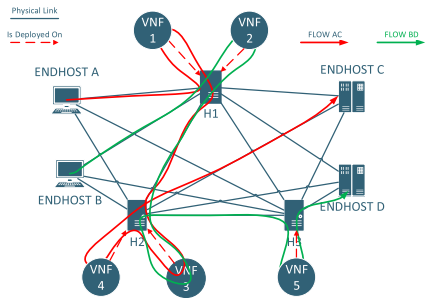

FIGURE 5. Physical flow mapping example.

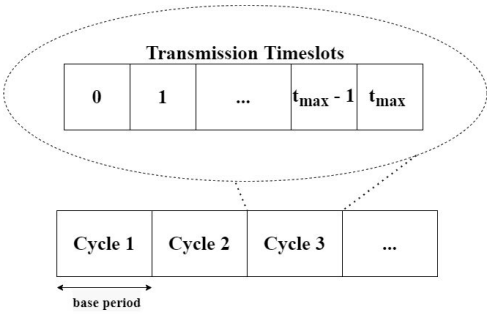

FIGURE 6. TSSDN timeslots partitioning.

2) SCHEDULING OF THE TIME-SENSITIVE FLOWS

The second component of the output is the scheduling of the time-sensitive flows that minimizes the end-to-end delay. Two factors must be taken into account for the delay estimation:

- **VNF execution**: the execution of a VNF on a given network flow takes some time. Assuming that queuing and overloads cannot occur since the objective is to schedule packet transmission to avoid these phenomena, it is possible to calculate the delay introduced by a VNF for a given flow in a deterministic manner. Instant by instant, all the resources are allocated to the VNF that must be executed. Using the DPDK technology, it is possible to bypass the kernel and work closely with the hardware [36]. Therefore, it is possible to perform tests that provide us in a deterministic way the time needed to perform the processing of an MTU-sized packet through the VNF, also considering the device's configuration. Some firewall implementations, for example, have a processing time that depends on the number of access lists declared in their configuration. It is necessary to have this information to calculate the introduced delay in a deterministic manner.
- **Physical Link Crossing**: crossing a physical link introduces a delay proportional to the characteristics of the medium crossed. It is advisable to use reliable transmission media in critical communications, possibly redundant to tolerate any faults.

On the basis of these considerations, it is possible to define some configuration parameters regarding the scheduling of transmissions in the network. In particular:

- **Single timeslot length**: this can be intended as the shortest time scheduling unit. It must be long enough to handle the flow element that takes the most time to run inside the network.
- **Base period length**: this is determined by the number of timeslots needed to manage the flows within the network.

As shown in FIGURE 6, transmission timeslots are allocated inside a base period that is repeated over time. Since these two parameters have been defined, it is possible to start estimating the maximum and average latency for each time-sensitive network flow verifying that all constraints defined by a network operator are satisfied. Concerning the example shown in FIGURE 5, supposing that the timeslots have been sized to host the VNF or physical link that takes the most time in the declared network environment, a possible scheduling that minimizes the number of used timeslots maximizing the number of time-sensitive flows handled could be the one shown in TABLE 1.

TABLE 1. Timeslot allocation example.

Timeslot	Flow AC	Flow BD
T1	AH1	BH1
T2	VNF1	-
T3	H1H2	VNF2
T4	VNF3	H1H2
T5	-	VNF3
T6	VNF4	H2H3
T7	H2C	VNF5
T8	-	H3D

The table shows how the AC flow starts to be managed at the T1 timeslot of each base period and ends at the T7 timeslot, while the BD flow management, even when it starts at the same time as the AC flow, ends at T8 timeslot. Note that if an AC flow packet is ready to be sent to the T2 timeslot, it will have to wait for the T1 timeslot of the next base period to be managed by the network, to be then delivered to the T7 timeslot. The same applies to the BD flow, but in return, there is the guarantee of having a deterministic network behavior.

IV. OPTSMT PROBLEM FORMULATION

The OptSMT problem is formulated by combining the following classes of clauses, built on the formal models defined for network topologies and time-sensitive flows (Subsection IV-A):

1) hard constraints expressing the resource consumption limitations which restrain the embedding of the VNFs onto the general-purpose servers of the substrate infrastructure (Subsection IV-B);
2) hard constraints determining the time scheduling of the flows, taking into account the crossed virtual paths, their mapping onto the physical ones, the possible flow ordering, their overlappings and the maximum acceptable latency (Subsection IV-C);

TABLE 2. Models for networks and time-sensitive flows.

Symbols	Notations
$V^s = (H^s, E^s), L^s$	Set of physical hosts/endpoints and connections
A^s_V, A^s_L	Set of physical hosts/endpoints and connections attributes
$G^s = (V^s, L^s, A^s_V, A^s_L)$	Substrate Network
$V^v = (N^v, E^v), L^v$	Set of virtual nodes/endpoints and connections
A^v_V	Set of virtual nodes/endpoints attributes
$G^v = (V^v, L^v, A^v_V)$	Service Graph
F^{ins}	Set of Time-Sensitive Flows
C^{ins}	Set of User-Defined Time-Sensitive Flow Constraints

3) optimization constraints for finding an optimal VNF embedding and flow scheduling solution (Subsection IV-D).

A. MODELS FOR NETWORKS AND TIME-SENSITIVE FLOWS

The formal models listed in TABLE 2 represent the computer networks and the time-sensitive flows for which the OptSMT problem is solved. The notation will be used for the definition of the hard and soft constraints.

According to this formalization, both the physical and the virtual environments are modeled as unidirectional graphs. The set of vertices in physical topology comprises hosts capable of hosting VNFs or fixed endpoints. In the virtual topology, the set of vertices is instead represented only by VNFs. Attribute sets contain additional information about the elements defined in the physical or virtual topology. For example, in the case of physical hosts, two valuable attributes for the VNF embedding are the storage and CPU power. Instead, concerning physical connections, an important parameter for the links is latency. The time-sensitive flows set is built on the network operator requests during real-time properties formulation. Each flow is associated with a constraint in terms of maximum acceptable latency for end-to-end flow management.

B. VNF EMBEDDING CONSTRAINTS

The function that maps a VNF to a host of the physical substrate is formalized as:

$$M_v(v^v) = v^s \tag{1}$$

The meaning of this function is that a vertex of the Service Graph should be mapped on a vertex of the substrate topology available.

Since there are two vertex types in the virtual topology, the M_v function has two declinations. The first one refers to endpoint mapping:

$$M_e(e^v) = e^s \tag{2}$$

A virtual endpoint has a reserved host in the substrate graph, so mapping function is very simple since it is fixed.

The second one refers to VNF mapping:

$$M_n(n^v) = h^s \tag{3}$$

This mapping function must satisfy a higher number of constraints. VNFs deployed on one single host, in fact, share the same physical resources (e.g., the same RAM capabilities). Since all physical resources are allocated to the management of one flow element in every instant, CPU consumption by other VNFs executed on the same physical host is not considered. However, it is important to guarantee that there is enough CPU power to meet single VNF requirements. Therefore, given the following elements:

- $|N^v|$, $|H^s|$ as N^v/H^s set cardinality;
- $memory(i)$, a function that returns the VNF memory for element i;
- $core(i)$, a function that returns the CPU core number for element i;
- $type(i)$, a function that returns the VNF type for element i;
- $supported\_types(i)$, a function that returns the set of supported VNF types by element i;
- $max\_vnf(i)$, a function that returns the maximum number of VNF that could be deployed on the i-th host;
- n_ih_j, a Boolean variable that is true when the i-th VNF is deployed on j-th host;
- h_i, a Boolean variable that is true when the i-th physical host is used;
- $int(x)$ (where x is a Boolean variable), a function that transforms true/false Boolean statement in 0/1 arithmetical number;

the following hard constraints, regarding physical host resources type and consumption, must be included in the formulation of the MaxSMT problem.

- **RAM usage**: each VNF requires a certain amount of available RAM to be executed properly, i.e., the required VNFs memory of nodes deployed on a physical host must be lower that RAM host capability.

$$\forall h_j \in H^s, \sum_{i=0}^{|N^v|} (memory(i) \times int(n_ih_j)) \leq (memory(j)) \times h_j \tag{4}$$

- **CPU power consumption**: each VNF requires a certain CPU power, measurable in core number, to be executed.

$$\forall n_i \in N^v, \forall h_j \in H^s, (core(i) \times int(n_ih_j)) \leq (core(j)) \times h_j \tag{5}$$

- **Maximum VNFs capability**: the number of VNFs deployed on a physical host must be lower than a specific threshold.

$$\forall n_i \in H^s, \sum_{j=0}^{|N^v|} int(n_jh_i) \leq max\_vnf(i) \times h_i \tag{6}$$

- **Supported VNF functional type**: a physical host can host only VNFs of the supported types.

$$\forall n_i \in N^v, \forall h_j \in H^s, n_ih_j \Rightarrow type(i) \in supported\_types(j) \tag{7}$$

Furthermore, two additional constraints must be accounted to perform a reliable embedding.

- **Single VNF deployment**: a VNF must be deployed on one single host exactly.

$$\forall n_i \in N^v, \sum_{j=0}^{|H^s|} n_i h_j = 1 \tag{8}$$

Thanks to this constraint, all VNFs are guaranteed to be deployed on available physical hosts at least once and, at the same time, multiple deployments of the same VNF are not considered.

- **VNF deployment implication on physical host state**: if a VNF is deployed on a physical host, this host must be put on active state.

$$\forall h_j \in H^s, h_j \Rightarrow \bigvee_{i=0}^{|N^v|} n_i h_j \tag{9}$$

C. FLOW SCHEDULING CONSTRAINTS

Flow scheduling constraints are responsible for the scheduling of time-sensitive network flows. The first operation to be managed is to find all the possible flows in the physical topology onto which the virtual flows that are declared by the user can be mapped. The network operator defines the flow by specifying the source, the destination, the reference service graph and a number of timeslots. This last specification represents the maximum latency that the network operator is willing to accept for the management of that flow. The algorithms and constraints related to this operation are explained in the following.

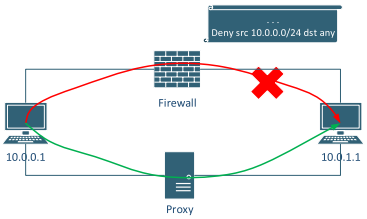

FIGURE 7. Virtual flow example.

1) VIRTUAL PATH COMPUTATION

Before the formulation of the flow scheduling constraints, a preliminary algorithm that is applied is the generation of the virtual paths. For each pair composed of a source node and a destination node, the shortest virtual path is identified by considering the number and configuration of the crossed VNFs. For instance, let us consider the example shown in FIGURE 7. A firewall rule blocks the flow that starts from host 10.0.0.1 and ends to host 10.0.1.1. Instead, host 10.0.1.1 is reachable through a proxy device. So the path between 10.0.0.1 and 10.0.1.1 is the best available one in terms of number of VNFs crossed. Besides, if another path between 10.0.0.1 and 10.0.1.1 were available but it included more VNFs in its path, the smallest one would still be the one that has been considered.

Algorithm 1 Virtual Flow Generation.

```
procedure VirtualFlowGeneration(source, destination)
    COMPUTE all possible flows between source and destination
    for flow in just computed flows do
        if reachability betweeen source and destination is guaranteed then
            STORE flow in the nominated flows
    SELECT the shortest flow between the nominated ones
return
```

Algorithm 1 formalizes how the best existing path between source and destination is computed. In order to obtain the best solution, all possible flows between source and destination should be discussed, if existing flows are more than one. In this way, overlapping flows can be avoided, as they should be otherwise managed carefully in the scheduling phase.

2) FLOW MAPPING CONSTRAINTS

Next, all possible mappings of a virtual flow onto the paths of the physical topology must be considered. For each possible mapping, a constraint is included in the OptSMT problem. This class of hard constraints is crucial for the problem formalization, because it represents the link between virtual and physical layers.

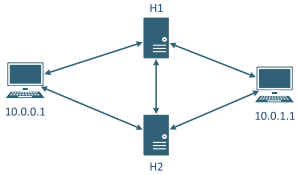

FIGURE 8. Physical topology example.

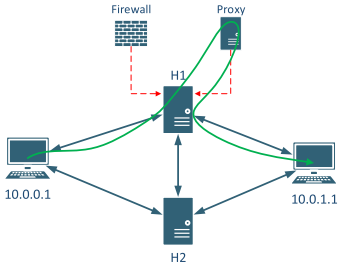

FIGURE 9. H1 deployment scenario.

For instance, taking as a reference the virtual topology depicted in FIGURE 7 and the physical topology depicted in FIGURE 8, there are four deployment possibilities: 1) both Firewall VNF and Proxy VNF are deployed on H1; 2) Firewall VNF is deployed in H1, Proxy VNF in H2; Proxy VNF is deployed in H1, Firewall VNF in H2; 4) both Firewall VNF and Proxy VNF are deployed on H2. For each deployment configuration and for each flow element, there must be a hard constraint composing the OptSMT problem.

For instance, considering scenario where both VNFs are deployed in H1 (FIGURE 9), naming Firewall VNF as n_1, Proxy VNF as n_2, flow from 10.0.0.1 to 10.0.1.1 as f_1, three constraints must be created, i.e., one for each node (f_{11}, f_{12}, f_{13}) of the path crossed by flow f_1:

$$n_1h_1 \wedge n_2h_1 \Rightarrow f_{11} \equiv 10.0.0.1 \rightarrow H1$$
$$n_1h_1 \wedge n_2h_1 \Rightarrow f_{12} \equiv Proxy\,Execution$$
$$n_1h_1 \wedge n_2h_1 \Rightarrow f_{13} \equiv H1 \rightarrow 10.0.1.1$$

All possible deployment scenarios must be contemplated for each flow. Taking this into account, Algorithm 2 concisely describes how the flow mapping constraints are generated for each flow.

Algorithm 2 Flow Mapping Constraints Generation Procedure.

```
1: procedure Constraints generation(...)
2:     for Flow declared as i do
3:         FIND all possible physical flow mapping
4:         for physical flow mapping as j do
5:             COMPUTE needed embedding condition as cond
6:             for Flow element as k do
7:                 GENERATE constraint cond ⇒ k ≡ j(n)
   ▷ n is an index
8:                 PUSH generated constraint
```

The constraints output by this algorithm can be represented with a formula. Given:

- a time-sensitive flow $f \in F^{ins}$, where notation f_i denotes the i-th time-sensitive flow element;
- a set P where each element $p \in P$ represents a possible physical flow mapping;
- a function $cond(x)$, where x is a physical flow corresponding to the time-sensitive one, returning the embedding condition that must be satisfied to have the x mapping;
- a function $length(x)$, where x is a flow, returning the flow length in terms of flow element number;

the hard constraints for flow mapping are represented as follows:

$$\forall p \in P, \forall i \geq 0 \wedge i < length(p), cond(p) \Rightarrow f_i \equiv p_i \quad (10)$$

3) FLOW ORDER CONSTRAINTS

To ensure that the flow scheduling operations do not alter the flow order previously calculated, adding another constraint regarding the beginning and the end of a single flow element is necessary. Three functions, operating on flow elements, are introduced to handle flow scheduling operations. Given a time-sensitive flow $f \in F^{ins}$ and the i-th time-sensitive flow element f_i:

- $start(f_i)$ returns the first timeslot from which the flow element f_i starts to be processed;
- $end(f_i)$ returns the last timeslot from which the flow element f_i ends to be processed;
- $duration(f_i)$ returns the number of timeslots required by the flow element f_i to be processed.

FIGURE 10. Time-sensitive flow elements.

For instance, for a time-sensitive flow as the one depicted in FIGURE 10, where each block represents a single flow element, the scheduling order must ensure the following conditions:

- F1 must be handled before F2, F3 and F4;
- F2 must be handled after F1, but before F3 and F4;
- F3 must be handled after F1 and F2, but before F4;
- F4 must be handled after F1, F2 and F3.

Algorithm 3 describes how the procedure for the generation of the flow order constraints is structured.

Algorithm 3 Order Constraints Generation Procedure.

```
1: procedure OrderConstraintsGeneration(flow)
2:     for i ← 0 to length(flow) do
3:         for j ← 0 to length(flow) do
4:             if f_i precedes f_j then
5:                 start(f_i) + duration(f_i) ≤ start(f_j)
```

The constraints output by this procedure are expressed as:

$$\forall f \in F^{ins}, \forall i \geq 0 \wedge i < length(f), \forall j \geq 0 \wedge j < length(f), \forall i < j, start(f_i) + duration(f_i) \leq start(f_j) \quad (11)$$

4) FLOW ELEMENT OVERLAPPING CONSTRAINTS

Another class of constraints to be considered concerns the management of a flow scheduling that requires the same physical resources to be processed. A primary objective is to avoid queues in network devices to provide guarantees regarding the maximum end-to-end delay between two or more hosts in the network. Therefore, the same physical resources cannot handle multiple flow elements simultaneously.

For instance, considering the case depicted in FIGURE 11, the two illustrated flows can be scheduled simultaneously since the physical resources involved, physical links and hosts on which the VNFs are deployed, are different. Therefore, it is possible to save considerably on the number of timeslots globally used for the management of declared flows. However, this is not always feasible. It depends on the source and destination host locations and the configuration of intermediate devices. In fact, considering instead the case depicted in FIGURE 12, VNF2 and VNF3 execution cannot be scheduled in the same timeslot since they require the same physical resources to be used. In this case, it must be guaranteed that:

$$start(VNF2) \geq end(VNF3) \vee end(VNF2) \leq start(VNF3)$$

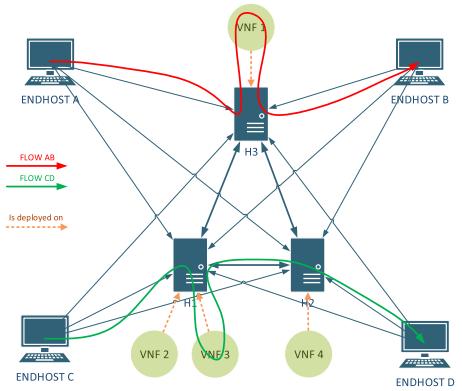

FIGURE 11. Non-overlapping flows example.

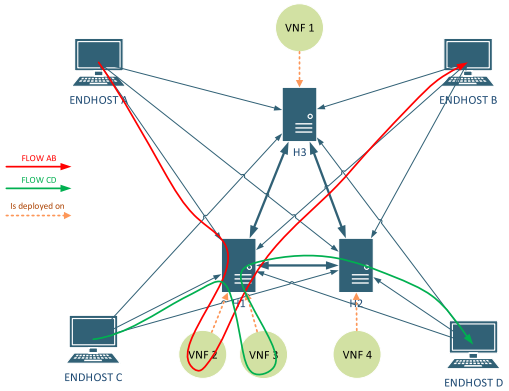

FIGURE 12. Overlapping flows example.

This statement must be verified for each flow pair that shares the same physical resources. The constraints for checking this statement are produced with the procedure described by Algorithm 4.

Algorithm 4 Overlapping Constraints Generation Procedure.

procedure OverlappingConstraintsGeneration(flows)
  for each flow f_i in flows data structure **do**
    for each flow f_j in flows data structure, different from f_i **do**
      for each f_{im} flow element **do**
        for each f_{jn} flow element **do**
          if f_{im} and f_{jn} are deployed on the same resource **then**
            GENERATE constraint
            PUSH constraint

The output constraints take into account also the relative timeslot allocation of different flow elements. For instance, supposing to have two different flow elements A and B belonging to two different flows deployed on the same physical resource, the only allowed relative allocation of these flow elements would be the one depicted in FIGURE 13.

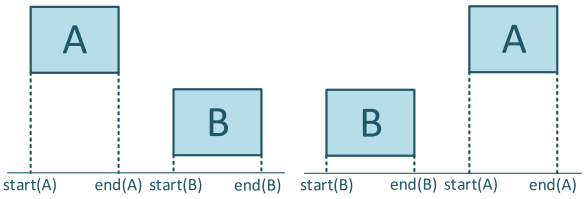

FIGURE 13. Flow element relationships example.

Namely, the only allowed relative allocation could be expressed with the following notation:

$$end(A) \leq start(B) \vee start(A) \geq end(B)$$

The formalization of the constraints output by Algorithm 4 requires the introduction of a new function. Given a flow element x, $deployedOn(x)$ returns the physical resource where the flow element x is mapped on. Thanks to this function, the flow element overlapping constraints are formalized as:

$$\begin{aligned} &\forall x \in F^{ins}, \forall y \in F^{ins}, x \neq y, \\ &\forall m \geq 0 \wedge m < length(x), \forall n \geq 0 \wedge n < length(y), \\ &deployedOn(x_m) == deployedOn(y_n) \Rightarrow \\ &start(x_m) \geq end(y_n) \vee end(x_m) \leq start(y_n) \end{aligned} \quad (12)$$

5) TOTAL TIMESLOT NUMBER VARIABLE CONSTRAINTS

A variable, named *makespan*, is introduced in the model to represent the total number of timeslots defined within the network environment for the management of all the time-sensitive flows. The usage of this variable has been made necessary because it is impossible to know in advance how many timeslots are necessary for the management of the flows. Some constraints must be defined over this variable.

First, the *makespan* variable must be limited between zero and the maximum number of timeslots needed to handle all time-sensitive flows declared in the network environment. In the worst case, this number can be computed as the algebraic sum of each single flow length. Two functions are required for the formulation of this constraint:

- $map(x)$, which returns the set of flows in the physical topology onto which the virtual flow x can be mapped;
- $max(x)$, which returns the longest path among the flows in set x.

By using these functions, the first constraint, which imposes a limit on the possible values for the *makespan* variable, is expressed as:

$$makespan \geq 0 \wedge makespan \leq \sum_{f \in F^{ins}} max(map(f)) \quad (13)$$

Then, other two constraints must be formulated, and for them the $last(x)$ function is required, so as to return the last flow element of flow x. These two constraints are expressed as:

$$\sum_{f \in F^{ins}} int(\bigvee (makespan, end(last(x)))) \equiv 1 \quad (14)$$

$$\forall f \in F^{ins}, makespan \geq end(last(f)) \quad (15)$$

Constraint (14) states that the *makespan* variable must be equal at least to the end of one flow in the physical topology, whereas constraint (15) states that the *makespan* variable must be greater or equal then the end timeslot of the last flow element for each flow.

Finally, constraints that limit the values returned by functions $start(x)$ and $end(x)$, where x is a flow element, are modeled as follows:

$$\begin{aligned} &\forall f \in F^{ins}, \forall i \geq 0 \wedge i < length(f), \\ &start(f_i) \geq 0 \wedge start(f_i) \leq makespan, \\ &end(f_i) \geq 0 \wedge end(f_i) \leq makespan, \\ &start(f_i) < end(f_i) \end{aligned} \quad (16)$$

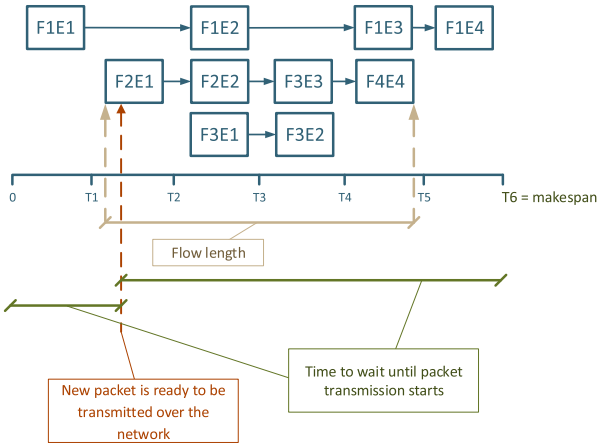

FIGURE 14. Max latency computation.

6) MAXIMUM ACCEPTABLE LATENCY CONSTRAINTS

A constraint must also be included in the OptSMT formulation regarding the maximum latency that the network operator is willing to have for the management of that specific time-sensitive flow. When declaring the time-sensitive flow, the operator can define this parameter, taking into account the duration of the timeslot that has been defined within that particular network. Alternatively, it can be computed with the following formula, where ts_{len} is the timeslot length defined by the network operator:

$$c_{ee_i} = ts_{len} \times (makespan + start(f_i) - endf_i) \quad (17)$$

The maximum end-to-end delay thus computed for the i-th flow is included in the C^{ins} set.

Computing the delay as shown in (17) is helpful to manage the worst cases. For instance, considering the case depicted in FIGURE 14, a new packet transmission request arrives a few moments after the beginning of the first flow element handling. So, the packet has to wait a number of timeslots equal to *makespan* until it starts to be handled. Then the packet will be delivered to the destination when flow handling finishes.

At this point, the maximum acceptable latency constraints can be computed. Given the set of time-sensitive flow F^{ins}, the set of constraints C^{ins} related to the maximum acceptable latency for the flows, a function $constraint(f)$ returning the constraint $c \in C^{ins}$ in terms of the maximum number of timeslots acceptable for flow $f \in F^{ins}$, the following constraint are included in the problem formulation:

$$\forall f \in F^{ins}, makespan + end(f) - start(f) \leq constraint(f) \quad (18)$$

D. OBJECTIVE FUNCTION

The declaration of the objective function represents the essential element for the formulation of the OptSMT problem. Specifically, the objective of the proposed approach is to find an optimal VNF embedding and flow scheduling solution able to minimize the end-to-end latency of the managed flows. For this reason, it was decided to include as an objective function the minimization of the end-to-end latency for each flow:

$$\forall f \in F^{ins}, min(makespan + end(f) - start(f)) \quad (19)$$

Besides the assurance that the flows will be managed in time according to the user-defined constraints, our approach attempts to minimize the end-to-end delay as much as possible, thanks to the definition of this objective function. If a solution is not found, that means it is impossible to find a VNF embedding and flow scheduling solution that addresses all constraints beforehand presented.

V. IMPLEMENTATION AND VALIDATION

This section describes how the proposed approach has been implemented with a prototype framework, and it discusses the validation.

A. IMPLEMENTATION

FIGURE 15 highlights the main elements composing the framework implementation, and it shows how it interacts with human users and other existing tools. The framework was developed in Java, and it requires a document written in XML (eXtensible Markup Language) format as input. The produced output provides information about the outcome of the work produced by the framework, i.e., it informs the user if a correct solution has been found for the VNF embedding and flow mapping problems. In the positive case, it provides the user with an indication of the optimal positioning of the VNFs on the available physical substrate and the optimal scheduling of the time-sensitive flows in the network. The goal is to find an optimal solution for VNF embedding on the physical substrate that uses the least possible number of timeslots to manage all the time-sensitive flows defined by the user.

The framework interacts with a series of already available tools:

1) *Microsoft Z3 Solver*, a theorem prover provided by Microsoft Research;

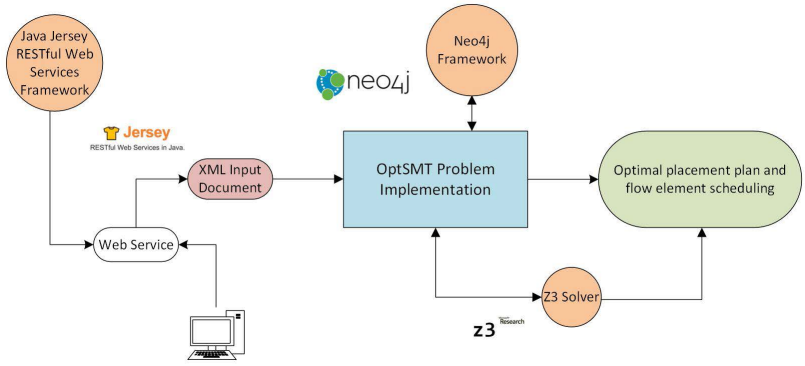

FIGURE 15. Framework design overview.

2) *Neo4J Graph Database*, a database to store graph information;
3) *Java Jersey RESTful Web Services Framework*, a framework that provides libraries to easily develop a Web Service.

Microsoft Z3 Solver is a tool that allows verifying the satisfiability of a set of logical formulas regarding a given formal theory. Z3 is a low-level tool. The optimization and search algorithms developed by Microsoft are accessible through a high-level interface, convenient to be used by a programmer. Different programming languages are supported by the tool. The most common and updated ones supported are C++, Python and Java. Classes that support a large variety of operations are displayed, depending on the needs of the programmer. Z3, therefore, allows verifying if a set of logical and mathematical conditions is respected.

Neo4J Graph Database allows easily schematizing the mapping between the physical substrate and the virtual service graphs. The interaction with Neo4j can occur in two ways:

- Cypher Query Language is a graph query language built to interact with the graph database efficiently. It is a SQL-inspired language that allows performing CRUD operations on graph databases. It is suited to handle nodes and relationships with straightforward instructions.
- Neo4j REST APIs make available a simple Web Service with which users can interact to operate with a graph database hosted in a machine. The framework has been developed to be compliant with these REST APIs. Many functions have been developed to interact with the database through REST calls, e.g., handling physical and virtual topology paths exploiting algorithms already implemented in the Neo4j framework, such as the minimum path research.

Java Jersey RESTful Web Services Framework is a REST framework, which provides a JAX-RS, Java API for RESTful Web Services implementation. This tool is used to implement a web service through which an interaction with the developed framework is feasible. The REST web service has been chosen because of its great flexibility and interoperability, as it allows resource organization. The resources can be managed through unique identifiers and many operations can be made available on each resource, since methods available inside HTTP protocol are used to distinguish operations. For instance, HTTP GET operation is often used to retrieve information on the resource requested, or HTTP POST and HTTP PUT operations are instead used to update a resource on the developed framework.

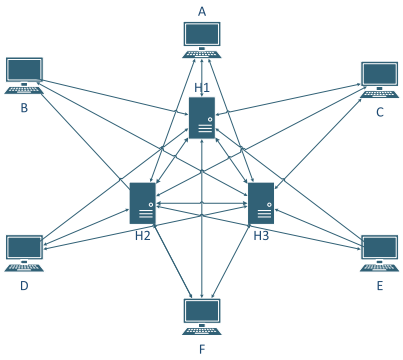

FIGURE 16. Physical topology test environment.

Interaction between the framework and these tools is handled through the main Java class that acts as a proxy, dispatching operations through other Java classes.

B. VALIDATION

The feasibility of the framework has been validated through some realistic use cases. The physical topology referred to during the test phase is presented in FIGURE 16, and the corresponding virtual topology is defined in FIGURE 17. The virtual topology is the one that the network operator wants to deploy on the given physical substrate.

Concerning the physical topology, it includes six endpoints that are used during end-to-end communications. Each endpoint appears both in the virtual and physical topology since its placement is fixed. Instead, for simplicity, only three physical middleboxes are defined. They can host VNFs defined in the virtual topology. The physical middleboxes

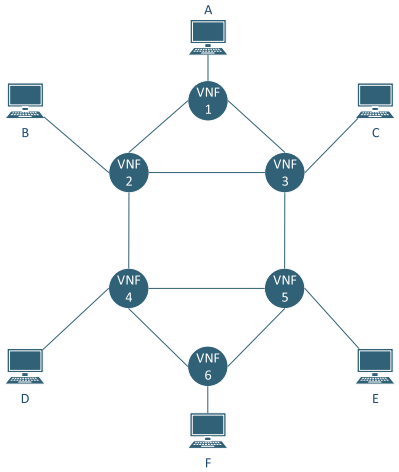

FIGURE 17. Virtual topology test environment.

resources (RAM capability, CPU power, maximum number of VNFs that the middlebox can support, list of supported VNF functional types) are taken into account during the embedding operation, by generating the proper constraints, as shown in Section IV. In addition, each endpoint is connected to each middlebox defined in the physical environment in order to guarantee high availability. If a failure occurs in one physical link that connects the endpoint to the middlebox, a network reconfiguration is triggered on the network controller. Reconfiguration obviously requires rerunning all the algorithms. So a new potential VNFs embedding scheme and flow scheduling result may be proposed. This operation takes time since all controlled hosts must be reconfigured to deploy the new configuration. The reconfiguration phase could be optimized by introducing a smart logic that considers failures and triggers a reconfiguration only when there is a real reachability problem between the declared flows.

Concerning the virtual topology, the VNFs involved in the network are represented more abstractly, showing relationships between endpoints and other VNFs in the studied environment. The presented flow scheduling results assume that the timeslots duration is equivalent to the time required for the longest flow element execution. In addition, we assume that all flow elements require exactly one timeslot to be completely executed.

On the basis of these two topologies, the behavior of the framework has been evaluated on five different scenarios, shown in FIGURE 18 differing for the number of declared time-sensitive flows and relative flows position in the virtual topology. The solutions for the VNFs embedding and flow scheduling problems will be presented in the following, with an explanation of how the framework can successfully compute them.

1) FIRST SCENARIO

In this first scenario, a single time-sensitive flow is considered. Each flow element is allocated to an available timeslot.

TABLE 3. First scenario placement results.

Physical Hosts	VNFs Deployed
H1	VNF1, VNF2, VNF3, VNF4, VNF5
H2	-
H3	VNF6

The total number of timeslots needed to manage this situation is equivalent to the number of flow elements of which the defined time-sensitive flow is composed. TABLE 3 explains the embedding scheme, whereas FIGURE 19 shows the flow scheduling.

2) SECOND SCENARIO

In the second scenario, two time-sensitive flows must be handled while guaranteeing a maximum end-to-end latency between hosts involved in communication. The two time-sensitive flows are not overlapping in their paths, as they have been scheduled to be handled in parallel. The framework computes the embedding scheme shown in TABLE 4 so that the VNFs involved in their paths are deployed on different hosts. In this way, there are no flow elements of the first time-sensitive flow that have intersections with some flow elements of second time-sensitive flow. As it can be seen from the flow scheduling representation in FIGURE 20, no more timeslots than the first scenario are needed to address this case.

TABLE 4. Second scenario placement results.

Physical Hosts	VNFs Deployed
H1	-
H2	VNF3, VNF5, VNF6
H3	VNF1, VNF2, VNF4

3) THIRD SCENARIO

In the third scenario, a flow that in the virtual topology has intersections with the two previously defined flows is introduced. The framework computes a different embedding solution, shown in TABLE 5. It allows the flows being handled in parallel, although their mapping onto the physical topology is not the optimal one. However, only one more timeslot than in the previous scenarios is needed to manage all the time-sensitive flows in this scenario. This result is illustrated in FIGURE 21.

TABLE 5. Third scenario placement results.

Physical Hosts	VNFs Deployed
H1	VNF2, VNF4
H2	VNF1
H3	VNF3, VNF5, VNF6

4) FOURTH SCENARIO

In the fourth scenario, the time-sensitive flow DE (i.e., the flow from D to E) is added to the already defined flows.

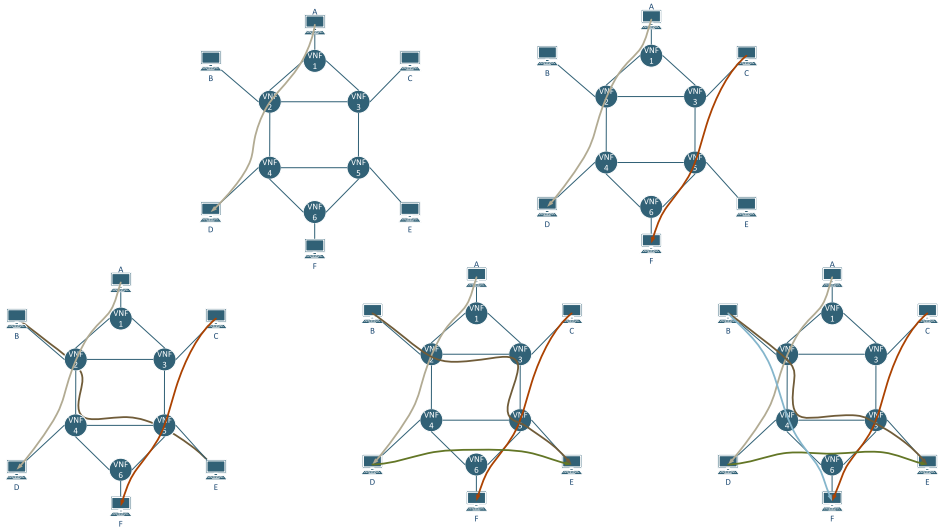

FIGURE 18. All test scenarios.

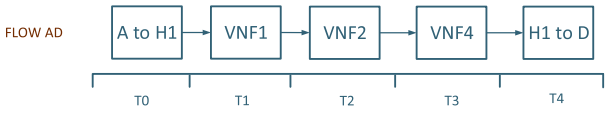

FIGURE 19. First scenario flow scheduling results.

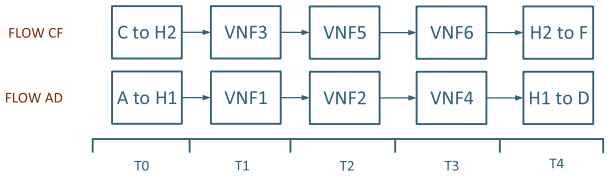

FIGURE 20. Second scenario flow scheduling results.

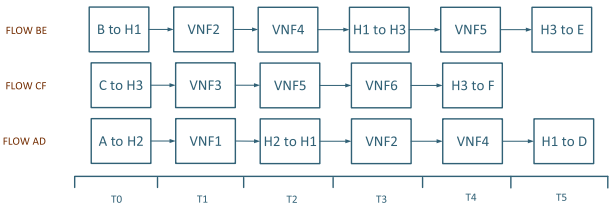

FIGURE 21. Third scenario flow scheduling results.

It must be noticed that, according to the VNFs configuration, two possible paths with the same cost are available for the previously defined flow BE:

- the path including VNF2, VNF4, and VNF5 avoids a possible overlapping with flow CF, since VNF3 is not crossed;
- the path including VNF3 generates an additional overlapping to be handled with flow CF.

The solutions for VNF embedding and time-sensitive flow scheduling may be different, according to the path that is selected. By running the framework, it selects the second path. Therefore, the VNFs are embedded as illustrated in TABLE 6, whereas seven timeslots are needed to handle all the time-sensitive flows as shown in FIGURE 22.

TABLE 6. Fourth scenario placement results.

Physical Hosts	VNFs Deployed
H1	VNF1
H2	VNF2, VNF3, VNF6
H3	VNF4, VNF5

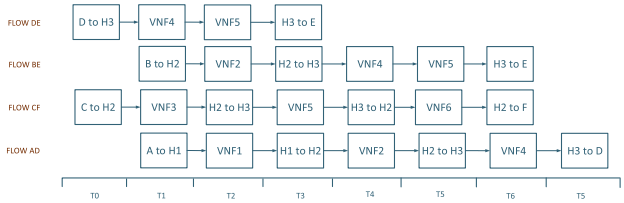

FIGURE 22. Fourth scenario flow scheduling results.

5) FIFTH SCENARIO

In the fifth scenario, one more flow is introduced. As it can be seen from the VNF embedding scheme shown in TABLE 7 and the timeslots allocation represented in FIGURE 23, only six timeslots have been used by the framework. This result could be misleading since, in the previous scenario with one less flow than the current scenario, seven timeslots were needed to handle all the time-sensitive flows. However, this is explained by the fact that no more constraints related to the virtual flow computation are included. In fact, for the flow definition phase, the network operator specifies the source and destination of each time-sensitive flow. The virtual path is computed automatically by the framework, in compliance with the VNFs configuration. At that point, since multiple

paths are available for flow BE, the algorithm chooses arbitrarily between the possible ones.

TABLE 7. Fifth scenario placement results.

Physical Hosts	VNFs Deployed
H1	VNF3, VNF5
H2	VNF1, VNF2
H3	VNF4, VNF6

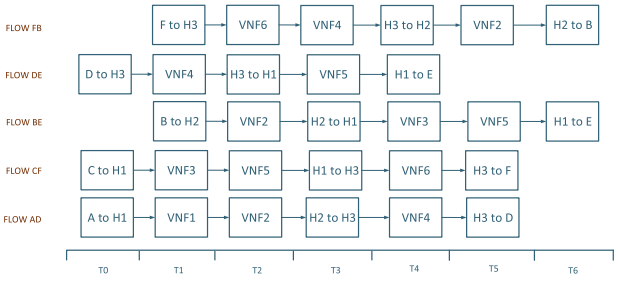

FIGURE 23. Fifth scenario flow scheduling results.

VI. CONCLUSION AND FUTURE WORK

This paper presented a novel approach for solving simultaneously the VNF allocation and the time-sensitive flow scheduling problems in TSSDN environments, where the presence of ICSs determines strict requirements in terms of security and maximum end-to-end delay for packet transmission. The design of the proposed approach lays its foundations on the formulation of an OptSMT problem, enabling formal correctness by construction and optimization for the automatically computed solution. The validation of the framework developed on the basis of this approach shows that it can be successfully employed for computing the time scheduling of the flows while correctly mapping the VNFs that process them onto the substrate infrastructure.

However, this preliminary study has room for improvement in future work. On the one hand, the OptSMT formulation is flexible enough to support the introduction of a larger number of optimization constraints. Therefore, an extensive evaluation of additional parameters whose optimization may benefit both VNF embedding and flow scheduling will be performed. We will also investigate methods to derive the problem constraints from real-time measures of VNF execution times directly in the kernel, in order to provide further optimization. On the other hand, we will try to integrate the resolution of a third problem, i.e., the automatic configuration of network and security functions, with the aim of enabling human users to specify which time-sensitive flows should be denied to reach their destination due to security reasons. With this integration, all-around management of time-sensitive networks may become feasible and match the high automation requirements characterizing industrial networks.

REFERENCES

[1] G. Aceto, V. Persico, and A. Pescape, "A survey on information and communication technologies for industry 4.0: State-of-the-art, taxonomies, perspectives, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3467–3501, Aug. 2019.

[2] C. Zunino, A. Valenzano, R. Obermaisser, and S. Petersen, "Factory communications at the dawn of the fourth industrial revolution," *Comput. Standards Interfaces*, vol. 71, Aug. 2020, Art. no. 103433.

[3] M. Cheminod, L. Durante, L. Seno, F. Valenza, A. Valenzano, and C. Zunino, "Leveraging SDN to improve security in industrial networks," in *Proc. IEEE 13th Int. Workshop Factory Commun. Syst. (WFCS)*, Trondheim, Norway, May 2017, pp. 1–7.

[4] S. Vitturi, C. Zunino, and T. Sauter, "Industrial communication systems and their future challenges: Next-generation Ethernet, IIoT, and 5G," *Proc. IEEE*, vol. 107, no. 6, pp. 944–961, Jun. 2019.

[5] N. K. Haur and T. S. Chin, "Challenges and future direction of time-sensitive software-defined networking (TSSDN) in automation industry," in *Security, Privacy, and Anonymity in Computation, Communication, and Storage* (Lecture Notes in Computer Science), vol. 11611. Atlanta, GA, USA: Springer, Jul. 2019, pp. 309–324.

[6] D. Bringhenti, G. Marchetto, R. Sisto, S. Spinoso, F. Valenza, and J. Yusupov, "Improving the formal verification of reachability policies in virtualized networks," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 713–728, Mar. 2021.

[7] J. C. C. Chica, J. C. Imbachi, and J. F. B. Vega, "Security in SDN: A comprehensive survey," *J. Netw. Comput. Appl.*, vol. 159, Jun. 2020, Art. no. 102595.

[8] B. Yan, Q. Liu, J. Shen, D. Liang, B. Zhao, and L. Ouyang, "A survey of low-latency transmission strategies in software defined networking," *Comput. Sci. Rev.*, vol. 40, May 2021, Art. no. 100386.

[9] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[10] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Proc. IEEE 4th Int. Conf. Cloud Netw. (CloudNet)*, Niagara Falls, ON, Canada, Oct. 2015, pp. 171–177.

[11] A. Gupta, M. F. Habib, P. Chowdhury, M. Tornatore, and B. Mukherjee, "On service chaining using virtual network functions in network-enabled cloud systems," in *Proc. IEEE Int. Conf. Adv. Netw. Telecommuncations Syst. (ANTS)*, Kolkata, India, Dec. 2015, pp. 1–3.

[12] J. Soares and S. Sargento, "Optimizing the embedding of virtualized cloud network infrastructures across multiple domains," in *Proc. IEEE Int. Conf. Commun. (ICC)*, London, U.K., Jun. 2015, pp. 442–447.

[13] I. Jang, S. Choo, M. Kim, S. Pack, and M.-K. Shin, "Optimal network resource utilization in service function chaining," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, Seoul, South Korea, Jun. 2016, pp. 11–14.

[14] M. M. Hasan and H. T. Mouftah, "Cloud-centric collaborative security service placement for advanced metering infrastructures," *IEEE Trans. Smart Grid*, vol. 10, no. 2, pp. 1339–1348, Mar. 2019.

[15] G. Marchetto, R. Sisto, F. Valenza, J. Yusupov, and A. Ksentini, "A formal approach to verify connectivity and optimize VNF placement in industrial networks," *IEEE Trans. Ind. Informat.*, vol. 17, no. 2, pp. 1515–1525, Feb. 2021.

[16] M. Bagaa, T. Taleb, and A. Ksentini, "Service-aware network function placement for efficient traffic handling in carrier cloud," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Istanbul, Turkey, Apr. 2014, pp. 2402–2407.

[17] R. Riggio, T. Rasheed, and R. Narayanan, "Virtual network functions orchestration in enterprise wlans," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, R. Badonnel, J. Xiao, S. Ata, F. D. Turck, V. Groza, and C. R. P. dos Santos, Eds. Ottawa, ON, Canada, May 2015, pp. 1220–1225.

[18] M. M. Hasan and H. T. Mouftah, "Latency-aware segmentation and trust system placement in smart grid SCADA networks," in *Proc. 21st IEEE Int. Workshop Comput. Aided Modelling Design Commun. Links Netw. (CAMAD)*, Toronto, ON, Canada, Oct. 2016, pp. 37–42.

[19] M. M. Hasan and H. T. Mouftah, "Optimal trust system placement in smart grid scada networks," *IEEE Access*, vol. 4, pp. 2907–2919, 2016.

[20] B. Martini, F. Paganelli, P. Cappanera, S. Turchi, and P. Castoldi, "Latency-aware composition of virtual functions in 5G," in *Proc. 1st IEEE Conf. Netw. Softwarization (NetSoft)*, London, U.K., Apr. 2015, pp. 1–6.

[21] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, R. Badonnel, J. Xiao, S. Ata, F. D. Turck, V. Groza, and C. R. P. dos Santos, Eds. Ottawa, ON, Canada, May 2015, pp. 98–106.

[22] E. Demirović, N. Musliu, and F. Winter, "Modeling and solving staff scheduling with partial weighted maxSAT," *Ann. Oper. Res.*, vol. 275, no. 1, pp. 79–99, Apr. 2019.

[23] E. Hadjiconstantinou and G. Mitra, "Transformation of propositional calculus statements into integer and mixed integer programs: An approach towards automatic reformulation," U.S. Armyï£¡s Eur. Res. Office, U.S Tech. Rep. TR/11/90, 1991.

[24] G. J. Gordon, S. A. Hong, and M. Dudík, "First-order mixed integer linear programming," 2012, *arXiv: 1205.2644*.

[25] Z. Hanzálek, P. Burget, and P. Sucha, "Profinet IO IRT message scheduling with temporal constraints," *IEEE Trans. Ind. Informat.*, vol. 6, no. 3, pp. 369–380, Aug. 2010.

[26] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks," in *Proc. 31st IEEE Real-Time Syst. Symp.*, San Diego, CA, USA, Nov. 2010, pp. 375–384.

[27] S. S. Craciunas and R. S. Oliver, "SMT-based task- and network-level static schedule generation for time-triggered networked systems," in *Proc. 22nd Int. Conf. Real-Time Netw. Syst. (RTNS)*, M. Jan, B. B. Hedia, J. Goossens, and C. Maiza, Eds. Versaille, France, Oct. 2014, p. 45.

[28] S. S. Craciunas and R. S. Oliver, "Combined task- and network-level scheduling for distributed time-triggered systems," *Real-Time Syst.*, vol. 52, no. 2, pp. 161–200, Mar. 2016.

[29] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive software-defined network (TSSDN) for real-time applications," in *Proc. 24th Int. Conf. Real-Time Netw. Syst. (RTNS)*, Brest, France, 2016, pp. 193–202.

[30] F. Dürr and N. G. Nayak, "No-wait packet scheduling for IEEE time-sensitive networks (TSN)," in *Proc. 24th Int. Conf. Real-Time Netw. Syst. (RTNS)*, Brest, France, 2016, pp. 203–212.

[31] N. G. Nayak, F. Dürr, and K. Rothermel, "Incremental flow scheduling and routing in time-sensitive software-defined networks," *IEEE Trans. Ind. Informat.*, vol. 14, no. 5, pp. 2066–2075, May 2017.

[32] N. K. Haur and T. S. Chin, "Time-sensitive-aware scheduling traffic (TSA-ST) algorithm in software-defined networking," in *Internet and Distributed Computing Systems* (Lecture Notes in Computer Science), vol. 11874. Naples, Italy: Springer, Oct. 2019, pp. 248–259.

[33] Z. Pang, X. Huang, Z. Li, S. Zhang, Y. Xu, H. Wan, and X. Zhao, "Flow scheduling for conflict-free network updates in time-sensitive software-defined networks," *IEEE Trans. Ind. Informat.*, vol. 17, no. 3, pp. 1668–1678, Mar. 2021.

[34] M. Vlk, Z. Hanzálek, and S. Tang, "Constraint programming approaches to joint routing and scheduling in time-sensitive networks," *Comput. Ind. Eng.*, vol. 157, Jul. 2021, Art. no. 107317.

[35] A. Mascis and D. Pacciarelli, "Job-shop scheduling with blocking and no-wait constraints," *Eur. J. Oper. Res.*, vol. 143, no. 3, pp. 498–517, 2002.

[36] R. Chen and G. Sun, "A survey of kernel-bypass techniques in network stack," in *Proc. 2nd Int. Conf. Comput. Sci. Artif. Intell. (CSAI)*, Shenzhen, China, 2018, pp. 474–477.

DANIELE BRINGHENTI (Graduate Student Member, IEEE) received the M.Sc. degree *(summa cum laude)* in computer engineering from the Politecnico di Torino, Italy, in 2019, where he is currently pursuing the Ph.D. degree in control and computer engineering. His research interests include novel networking technologies, automatic orchestration and configuration of security functions in virtualized networks, and formal verification of networks security policies.

FULVIO VALENZA (Member, IEEE) received the M.Sc. and Ph.D. degrees *(summa cum laude)* in computer engineering from the Politecnico di Torino, Torino, Italy, in 2013 and 2017, respectively. He is currently an Assistant Professor with the Politecnico di Torino. His research interests include networks security policies, orchestration and management of networks security functions in SDN/NFV-based networks, and threat modeling.