Received January 7, 2020, accepted February 11, 2020, date of publication February 27, 2020, date of current version March 9, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2976613

An Advance Reservation System to Improve Network Resource Utilization in Software-Defined Networks

ALAITZ MENDIOLA[1], JASONE ASTORGA[2], EDUARDO JACOB[2], (Senior Member, IEEE), AND JUANJO UNZILLA[2]
[1]Keynetic Technologies, Faculty of Engineering, University of the Basque Country UPV/EHU, 48013 Bilbao, Spain
[2]Department of Communications Engineering, Faculty of Engineering, University of the Basque Country UPV/EHU, 48013 Bilbao, Spain

Corresponding author: Jasone Astorga (jasone.astorga@ehu.eus)

This work was supported in part by the Spanish Ministry of Economy, Industry, and Competitiveness through the State Secretariat for Research, Development, and Innovation through the Adaptive Management of 5G Services to Support Critical Events in Cities (5G-City) under Project TEC2016-76795-C6-5-R and through the Department of Economic Development and Infrastructure of the Basque Government under the Strategic Research Projects Cognoms4.0 and Ciberprest.

ABSTRACT Research and education networks (RENs) worldwide, attracted by the benefits of software-defined networking (SDN) for services involving traffic engineering (TE) such as bandwidth on demand (BoD), have started to include SDN in their network evolution plans. BoD allows users to request end-to-end connectivity services of known duration with a guaranteed bandwidth. Considering such a scenario, this paper presents a set of algorithms and two novel data structures for provisioning of the BoD service in SDN-based RENs. First, network snapshots are used to represent a period of time during which resource availability remains constant. Second, the network snapshot tree (NSTree) allows for different network snapshots to be arranged hierarchically using a time-based node aggregation policy. Additionally, each network snapshot stores multiple alternative paths for each source-destination pair. This innovative approach allows for the network resource utilization and number of service requests being accepted in the network to be improved compared to the most common approach of a single shortest path being considered for each destination. As a result, our approach allows for network resource utilization to be improved by approximately 30%. Furthermore, the two-phase procedure used to pre-compute alternative paths and perform admission control makes it possible to achieve response times on the order of milliseconds, enabling the real-time provision of the service.

INDEX TERMS Advance reservations, bandwidth on demand, software-defined networking, traffic engineering.

I. INTRODUCTION

The appearance of software-defined networking (SDN) has enabled the development of next-generation networks. In a nutshell, the SDN approach entails a separation of forwarding and control planes so that control of the network is performed by means of a logically centralized control plane that resides in an external element and programs the forwarding plane using open interfaces [1]. With this new paradigm in mind, network operators worldwide have started to evolve their network architectures [2] to reduce their capital and operating expenditures and more efficiently facilitate the introduction of new services.

Among the benefits introduced by SDN, its capability to control the network while taking into account the network's overall state is of interest to network operators. This capability of SDN can revolutionize traffic engineering (TE) [3], one of the most challenging topics in communication networks. First, the logically centralized control plane enables the utilization of novel TE strategies. Second, being able to forward packets with fine granularity facilitates the utilization of alternative load balancing approaches [4].

Moreover, research and education networks (RENs) offering high-availability and high-quality services to the research

The associate editor coordinating the review of this manuscript and approving it for publication was Zehua Guo.

and academic community have started to include SDN in their network evolution plans, again due to the attraction of the impact of SDN on TE. In RENs, TE plays a key role in scenarios where the nature of the data being transmitted requires fast failure recovery capabilities and an efficient utilization of network resources to cope with the growing demand.

One of the services typically provided by RENs to the research community is bandwidth on demand (BoD), which allows for end-to-end (E2E) connectivity services to be established during a specific period of time with a guaranteed bandwidth. For instance, ESNet [5], a high-speed computer network serving the United States Department of Energy, provides BoD through the On-Demand Secure Circuits and Advance Reservation System (OSCARS) [6]. Similarly, GÉANT [7], a pan-European REN, offers BoD to its users through the AutoBAHN [8] provisioning tool. Provisioning of the BoD service requires the utilization of the advance reservation mechanisms [9] typically used in grid computing and optical networks [10]. Nonetheless, the emergence of SDN makes it possible to re-examine how TE should be supported and how the resource and timing constraints that facilitate advance reservations should be handled.

Considering such a scenario, this paper presents a set of algorithms and data structures that support advance reservations to provide an SDN-based BoD service in the REN environment. The proposed algorithms and data structures have been specifically designed to leverage the logically centralized control plane and high programmability available in SDN. The solution is based on the generation of *network snapshots*, a data structure that represents a period of time during which resource consumption of a given set of concurrent service reservations remains stable. Such network snapshots are arranged using a hierarchical data structure called a network snapshot tree (NSTree). An NSTree makes it possible to represent longer time intervals during which a subset of common resources is available as a result of aggregating multiple network snapshots into auxiliary nodes, taking into account time constraints.

The algorithms and the data structures defined in this paper (network snapshots and the NSTree) are used for the optimal advance reservation of bandwidth in SDN networks, but other parameters of QoS (delay, jitter, loss, etc.) could be easily accommodated. The NSTree structure is, by definition, hierarchical and dynamic, so it could be adapted for use in other scenarios in which not only bandwidth reservation but also other QoS parameters, or a combination of several quantities, are needed. This feature would allow for the prioritization of the selected QoS parameter by using weighting coefficients.

Taking into account that the joint solution of the combined routing and resource allocation problem is NP-hard, some authors have examined the use of function splits for achieving QoS in SDN networks [11]. The pursued goal is to avoid or at least reduce the computational overhead of path computation (routing) and resource allocation. To reduce the computational effort of such algorithms, our approach relies on a path pre-computation phase for selecting a subset of all possible paths to connect each source-destination pair. The second phase, in charge of actual bandwidth allocation, is the on-demand phase. The proposed data structures (network snapshots and the NSTree) could also be useful in the wider context of function split frameworks for QoS in SDN as a mechanism for representing the dynamic resource allocation status on the network.

Therefore, the main contribution and novelty of this paper entail a formal definition and assessment of these algorithms and data structures. To prove the suitability of the solution for the REN environment, the data structures and algorithms presented in this paper have been included in DynPaC, a framework for TE in software-defined networks that has already been used in experimental deployments [12]. It is worth pointing out that although the proposed algorithms and data structures are currently integrated in DynPaC, they are general enough to be extrapolated to other frameworks. Additionally, the performance of the solution has been evaluated using the topology of the ESNet network. The experiments show that the proposed solution satisfies the requirements of RENs and that the data structures and algorithms used to handle advance reservations are able to accept or reject a service reservation request within a time interval on the order of milliseconds.

The remainder of this paper is structured as follows. Section II presents the related studies, in which existing data structures used to handle advance reservations are analysed. Section III summarizes the architecture of the DynPaC framework. Section IV introduces the reservation model supported by the framework, and Section V describes the path computation algorithms used. Section VI details the proposed data structures, and Section VII presents the admission control and update procedures. Section VIII demonstrates the suitability of the solution by means of a performance evaluation, and Section IX concludes the paper.

II. RELATED STUDIES

The main contribution of this paper is the presentation and evaluation of a set of algorithms and data structures specifically tailored for efficient provisioning of the BoD service and to support advance reservations in SDN. General unicast QoS routing mechanisms have been extensively studied in the past, and several papers have been published more recently, re-examining these mechanisms in the light of the SDN paradigm [13]. One of the most analysed topics is resource reservation mechanisms; it is the focus of our proposal. As a consequence, this section reviews a comprehensive list of advance reservation approaches and the impact of the network resource being managed, the timing constraints being considered and the data structures being used in such approaches. In addition, it also presents various techniques designed to increase the number of accepted services.

A. IMPACT OF THE MANAGED NETWORK RESOURCE

In an analysis of advance reservations, the resource being managed plays a key role. The first solutions dealing with advance reservations considered link-based resource management [14], given that the bandwidth of a link is typically the resource being shared by the reservations. However, keeping track of the bandwidth utilization profile of each link in the network requires the utilization of multiple instances of the data structure [15]. This imposes a penalty in terms of memory usage and the running time of the admission control process. The reason for this is that to compute the path that satisfies the reservation constraints, the admission control process must be repeated k times, where k is the number of links in the network.

Although this approach is required by distributed architectures, centralized architectures can benefit from a more efficient approach to advance reservations. For instance, Andreica [16] propose a solution that can be used on a per-path basis, whereas Balman *et al.* introduce a graph-based resource management approach [17]. However, Andreica's per-path proposal is less efficient than per-link approaches. Additionally, the Balman's proposal relies on linked lists that as described later in this section, are not the most suitable data structures to be used for advance reservations.

In the case of SDN networks, apart from bandwidth, a scarce resource that must be efficiently used is the space in the flow table. In a related study, Guo *et al.* propose a real-time routing solution for OpenFlow-based SDN networks called STAR [18]. In STAR, the cost of each path is computed as a combination of the delay introduced by the flow table's utilization level and the actual length of the path. The final objective of the proposed solution is to use the scarce flow table resources of the OpenFlow switches in an efficient and balanced manner. The authors demonstrate that as a result, the overall network performance is improved, compared to classic routing approaches, in terms of the controller's workload, end-to-end packet delay and server throughput. The proposed architecture is based on path pre-computation, following an approach similar to that used in the DynPaC framework, as detailed in Section III-A. However, the STAR approach does not support advance reservations and therefore lacks a scheduling mechanism and the corresponding data structures and algorithms.

B. TIMING CONSTRAINTS

A very important issue to consider in advance reservation systems is how the timing constraints are handled, as they directly affect the efficiency of the solution. Supporting advance reservations requires taking into account the evolution of resource consumption over time. To this end, there are two different approaches. The first utilizes fixed time slots [19] that divide the time spectra equally, and the second uses dynamic time slots of variable length [20].

If fixed time slots are used, then their granularity plays a key role in the efficiency of the solution. If time slots are large, then the number of time slots to be analysed will be lower. Thus, memory will be saved and response time will be reduced; however, as a result, the reservations may last more than the requested time [15]. In addition, the size of fixed time slots can result in unnecessary resource and time fragmentation and impact the optimality of the solution. For all of these reasons, the current trend is to rely on dynamic time slots, even though their management is more complex.

In fact, if dynamic time slots are used, then the reservations can be tailored to the specific time intervals requested, and memory usage will be reduced. However, the data structures that need to be used are more complex, and their utilization may affect the running time of algorithms. In [21], Zuo *et al.* present a bandwidth reservation service as an enabler of the transfer of large amounts of data with guaranteed performance. More specifically, the researchers propose two optimization algorithms: one that minimizes cost and another that minimizes the data transfer time. The proposed system model is based on the concepts of *time step* and *time window*. A time step is defined as a time lapse in which the available bandwidth does not change for any of network links. A time window, in turn, consists of a single time step or multiple consecutive time steps, and the available bandwidth of each link during a time window corresponds to the smallest bandwidth available among all the time steps contained in that time window.

In [22], Barshan *et al.* present a bandwidth reservation model specifically tailored to the characteristics of media production networks. The proposed solution is based on time slots of variable size, and a timeslot is a time interval in which reservations remain invariant. The authors define the concept of a *scenario* as a set of video transfers that are dependent on each other. The proposed advance bandwidth reservation algorithm sorts all scenarios according to the earliest average start time of all of the scenario's requests. Then, for each scenario, requests are prioritized according to, first, the estimated hard deadline and, next, the traffic volume. For the bandwidth reservation, two algorithms are used for comparison: one is based on fixed-size timeslots, and another is based on variable-size timeslots. The evaluation's results show that the solution based on variable-size timeslots executes faster and allows for slightly increasing the request admittance ratio.

Also using dynamic time slots, Wang *et al.* [23] investigate bandwidth reservation requests that imply the maximum available bandwidth within a fixed-length floating time window and with a deadline constraint. This scenario represents the case of very large amounts of data that must be transferred in a timely manner. The authors formulate a periodic bandwidth request problem and attempt to maximize the number of successfully accepted requests over an already provisioned path. Each request is defined by a floating window size, an earliest possible start time and a deadline. Since the objective is to maximize the number of accepted requests, the proposed algorithm prioritizes requests that consume less resources.

Following a different approach, in [24], Chung *et al.* propose an architecture to allow for advance reservations in multi-domain environments based on per-domain reservation agents and a centralized orchestrator. The proposed system allows for exploiting different reservation options available in different domains. As a result, it is possible to benefit from the flexibility provided by novel advance reservation schemes while maintaining compatibility with more rigid legacy systems. In this regard, the proposed multi-domain orchestrator complements intra-domain reservation systems, such as the ones analysed previously.

C. IMPACT OF DATA STRUCTURES

A major aspect to take into account when dealing with advance reservations is the data structure that is used to keep track of resources' consumption or availability. If arrays [25] or linked lists with fixed time slots are used, then the running times of algorithms are always linearly dependent on m, the number of time slots considered in the book-ahead interval. In addition, time intervals where no reservations at all exist need to be handled as well, increasing memory usage. Nonetheless, although the utilization of dynamic time slots may solve the aforementioned problems, it results in more complex data structures. Schneider *et al.* report this issue in [26] and propose using a list of free blocks to support dynamic time slots. Using hierarchical data structures presents some benefits. First, allowing for the analysis of wider time slots leads to the running time of the algorithms being reduced to a logarithmic function [14], [27], [28]. Second, hierarchical data structures are able to support non-consecutive time slots, therefore reducing the need to keep track of resources during the entire book-ahead interval. The latter results in an additional benefit: there is no need for a book-ahead interval [27].

In summary, there are currently no solutions based on hierarchical structures such that resource management is performed on a per-graph basis. Nevertheless, SDN relies on a logically centralized control plane that facilitates an automatic abstraction of the network graph. Therefore, it represents the perfect environment for rethinking how advance reservations are handled and for leveraging the lessons learned throughout the preceding decades about the impact of data structures on such systems.

D. TECHNIQUES USED TO INCREASE THE NUMBER OF ACCEPTED SERVICES

Among the techniques proposed to increase the number of accepted services, establishing multi-path traffic flows over SDN-based networks is a common trend. However, such operations result in increased signalling overhead between the SDN controller and the networking devices and a high workload of the SDN controller related to admission control tasks. This affects the scalability of the solutions. To cope with these issues, researchers in [29] leverage path pre-computation as an approach to achieving scalability in their SDN-based routing and resource management system. Therefore, instead of computing paths for each flow request, paths are pre-determined offline and fixed for certain time periods (e.g., 15 minutes), which results in the controller not having to perform time-consuming tasks. The obtained results show that the admission control time is barely affected by the traffic load, while the proposed multi-path approach allows for increasing the number of accepted services.

A related approach is that of Hou *et al.* [30], where the authors propose a multi-path solution to improve the throughput and robustness of large data transfers. For each transfer, the authors select two node-disjoint paths and consider two alternatives for the selection of these paths: (1) fixed paths with bandwidth fixed during the entire duration of the data transfer and (2) fixed paths with variable bandwidth availability during the duration of the data transfer. In both cases, the optimization objective is to minimize the data transfer's completion time. Therefore, in the case of a variable bandwidth, the data transfer should always start immediately. However, in the case of a fixed bandwidth, it might be convenient to delay the start of the data transfer to be able to reserve a larger bandwidth for the transfer. A simulation-based performance evaluation shows that the factors that have the greatest impact on the transfer's end time are the data size and the bandwidth, while the effect of network size is not as noticeable. The proposed algorithms outperform commonly used greedy algorithms and attain near-optimal performance.

Similarly, in [31], Chung *et al.* propose a multi-domain advance reservation system that takes advantage of services provided by software-defined exchanges (SDXs), which allows different domains to share computing, storage and networking resources. Due to this information and the development of a negotiation protocol, it is possible to perform multi-path splitting of bandwidth requests. Accordingly, the proposed system allows for the success rate of multi-domain advance reservation requests to be increased from approximately 50% to almost 99% using four paths.

A different alternative approach to improving the service acceptance ratio is to provide requesting users with some degree of flexibility. In [32], Zuo *et al.* provide the user with two alternative reservation options if the network does not have sufficient available resources during the time interval specified by the user. The two alternatives are the closest time intervals before and after the time interval specified by the user, in which there are sufficient available resources for accepting the request. In a similar study [33], the authors deal with the problem of multiple bandwidth reservation requests received by the network at the same time and propose two alternatives: to maximize the number of accepted requests and to maximize the overall amount of transferred data. Neither of the proposed solutions considers traffic engineering mechanisms to reorder already established services and thus make room for new requests.

In [34], Barshan *et al.* add reliability guarantees to their previous approach. To this end, apart from the primary data transmission path, a totally disjoint backup path is also computed for each reservation request. To mitigate the bandwidth waste effect resulting from the reservation of backup paths when there is no link failure, the authors propose a dynamic adaptation algorithm. This algorithm operates in two phases, executed at the end of every time slot. The first phase—a periodic update—consists of monitoring the current actual network status. Then, the second phase—periodic adaptation—analyses and reschedules data transfers during the next time slot to improve efficiency by using idle bandwidth capacity. Additionally, whenever a failure is detected, the algorithm is automatically invoked to adapt the network transfers to the current state. The results obtained through a simulation show that the proposed algorithm allows for increased performance of the advance bandwidth reservation algorithm. Additionally, the execution times of the algorithm allow it to react timely to sudden network changes.

Similarly, in [35], Zuo investigates bandwidth preemption strategies in bandwidth reservation systems, considering two alternatives when a bandwidth reservation request with a higher priority arrives: to minimize first the number of preempted bandwidth reservations or the total amount of preempted bandwidth. Using simulations, the author shows that the proposed algorithms outperform the basic greedy algorithm. In a related paper [36], the authors aim at maximizing the number of accepted bandwidth reservation requests. To this end, they propose dividing the requested bandwidth among different paths when it is impossible to provide the requested bandwidth by using a single path. For the multi-path solution, they select link-disjoint paths and propose two optimization algorithms: to minimize the average earliest completion time and to minimize the average duration of data transfers.

III. ARCHITECTURE OF THE DYNPAC FRAMEWORK

Since the proposed data structures and algorithms have been integrated in the DynPaC framework, this section briefly reviews its architecture. As previously mentioned, DynPaC is a generic framework aiming to facilitate the adoption of TE strategies in software-defined networks; its architecture is depicted in Figure 1. It consists of 6 well-defined modules and a REST API that facilitates its integration with external elements, such as traffic analysers, NFV orchestrators or multi-domain agents [12]. The following subsections describe the modules of the DynPaC architecture.

A. PATH COMPUTATION ELEMENT (PCE)

PCE is a dedicated element in charge of path computation that enables an easy introduction of different routing algorithms such as Dijkstra's or Ford-Fulkerson algorithms. This element makes it possible to utilize novel and customized path computation algorithms with various objective functions. For example, power consumption can be minimized if the framework is used for green computing, or the QoE can be maximized if the framework is used to provide video on-demand services.

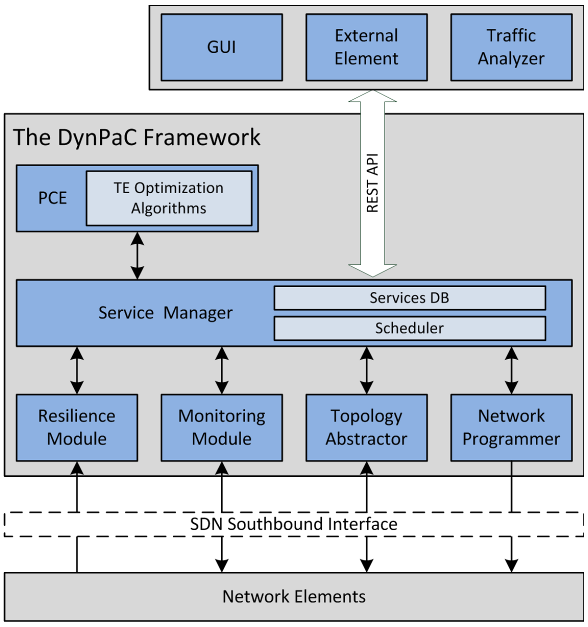


FIGURE 1. Architecture of the DynPaC framework for advance path computation in SDN.

B. TOPOLOGY ABSTRACTOR

PCE requires a model of the network to perform path computation. Once the topology has been discovered (in SDN, this is usually achieved by means of the LLDP and BDDP protocols), the optimal way of representing it may vary depending on network type, the use case and especially the routing algorithm being used. In such a scenario, the main objective of the topology abstractor is to construct the correct network model that will facilitate path computation.

C. RESILIENCE MODULE

The DynPaC framework includes a resilience module that aims to minimize the service disruption and packet loss in the event of network failure. This module monitors the state of the network and informs the remaining modules when it receives an event notifying it about a node or a link failure. Thus, a failure can be mitigated by installing alternative routes in the network to continue providing the affected connectivity services.

D. MONITORING MODULE

A monitoring module keeps track of resources consumed by each connectivity service provisioned in the network. It allows for verifying that the SLAs to which the user subscribed are being fulfilled. This module does not impose any particular mechanism for obtaining the QoS metrics, which could be done by means of the southbound protocol [37] or by retrieving the data from a NetFlow or sFlow collector. Upon a violation of a given SLA or detection of an anomaly on the network (i.e., congestion), the monitoring module informs the remaining modules so that the appropriate mitigation procedure is executed.

E. NETWORK PROGRAMMER

The network programmer module is in charge of programming the network elements involved in the provisioning of a given service. In summary, it translates the information contained in the form of a path, i.e., a sequence of network elements, into the appropriate messages to program the network devices. For instance, in the case of OpenFlow devices, this implies the generation of the necessary *flow modification* messages to install or remove the flow entries at the OpenFlow switches.

F. SERVICE MANAGER

The *Service Manager* is the core module of the DynPaC framework. All modules of the framework and the REST API communicate with the Service Manager that orchestrates the interaction of various modules to provide the requested connectivity service. This module also handles the lifecycle of services and keeps track of the state of services (if they are reserved, installed in the network or have expired). In addition, since the framework has been designed to support advance reservations, the Service Manager keeps track of resource consumption over time. Therefore, it acts as a *stateful PCE* described in RFC 4655 [38]. This feature allows for the path computation to be performed using information not only on topology but also on the resources consumed by already accepted connectivity services.

IV. RESERVATION MODEL

This section describes the reservation model used in the DynPaC framework and the timing conditions that need to be fulfilled for acceptance of new service reservation requests into the system.

A. DEFINITION

In the DynPaC framework, a new service reservation request between two endpoints for a certain amount of time is defined as follows:

Definition 1: Every new service reservation request is identified by the source and destination endpoints (node and port), the start time at which the service needs to be provisioned and the end time at which the service needs to be terminated. In addition, a service reservation request is also characterized by the peak bandwidth that will be available through the E2E circuit and the desired failure recovery policy, which can be either path protection or path restoration. It is, therefore, a "specified time specified duration" reservation [39].

B. TIMING CONSTRAINTS

Each new service reservation request requires the execution of two different phases: *check* and *update*, as depicted in Figure 2. First, it is necessary to check whether there are enough resources (i.e., bandwidth, VLANs, etc.) in the network to satisfy the service request. This phase takes the amount of time denoted by t_{check} and results in the new service request being accepted or rejected.

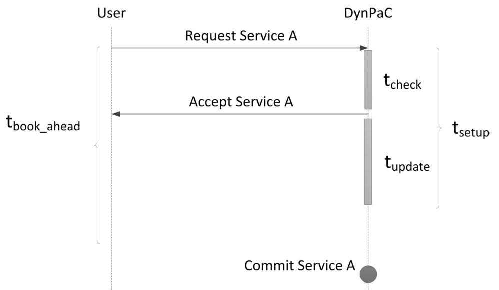


FIGURE 2. Service request being accepted by the DynPaC framework.

Second, the optimal path is selected among all available paths satisfying the connectivity requirements, and the necessary data structures are updated. This takes the amount of time denoted by t_{update}. The second phase is only executed for accepted services. Variable p_A denotes the probability of a service being accepted by DynPaC, also known as the *service acceptance ratio (SAR)*. SAR is one of parameters most commonly used to evaluate advance reservation systems. It measures the number of service reservation requests that are accepted by the system relative to the total number of service reservations requested from the system, as described in equation 1:

$$SAR = \sum service_{accepted} \; / \; \sum service_{requested} \tag{1}$$

The time required to execute both check and update phases is denoted by t_{setup} and is the minimum time that must elapse between two consecutive service requests to avoid inconsistencies during snapshot updates. In fact, service requests are always handled in the order of arrival, and an amount of time of at least t_{setup} must pass after a service request has been received and before the next one can be processed. In this case, it is possible to guarantee that the system is stable and that the data structures represent the network state correctly when the second request is processed.

Furthermore, t_{setup} also delimits the time at which the requested service can be committed to the network, which is commonly referred to as the *book-ahead interval* (t_{book_ahead}) [25]. Therefore, two conditions must be met for a service request to be accepted in the system: (1) at least t_{setup} has passed since the last service request, and (2) no t_{book_ahead} time exists between the time at which the request was received and the start time of the service. The relationships among t_{check}, t_{update}, t_{setup} and t_{book_ahead} are defined by equations 2 and 3.

$$t_{setup} = t_{check} + t_{update} \tag{2}$$

$$t_{setup} \leq t_{book\_ahead} \tag{3}$$

Since t_{setup} for a given service depends on whether the service has been accepted, we define the average t_{setup} as follows:

$$t_{\langle setup \rangle} = t_{check} + p_A * t_{update} \tag{4}$$

V. PATH COMPUTATION ALGORITHMS

As previously mentioned, having an advance reservation solution that leverages the centralized nature of the SDN control plane allows for per-network resource management, as opposed to the classic per-link approach. This characteristic of the advance reservation solution presented in this paper makes it possible to utilize path pre-computation strategies. In a nutshell, the approach followed for the DynPaC application for BoD consists of pre-computation of a set of possible paths for all pairs of source and destination nodes. Then, once a particular service reservation is requested, the possible paths are checked against the available resources until a valid path has been found.

It is worth noting that prior to path pre-computation, it is necessary to retrieve and abstract the network topology. In this regard, once the topology has been retrieved, an undirected graph is formed. This approach allows for a simplification of path computation since usually in BoD, the service requests are considered symmetric and bidirectional. In other words, for a given service, the same bandwidth is reserved for both directions of communication.

This section describes both algorithms: the first is used in the *pre-computation phase*, and the second is used in the *on-demand phase*.

A. PRE-COMPUTATION PHASE

The first phase of the computation of the optimal path for a given service reservation request consists of the computation of a subset of all possible paths that will be used to establish the connecting circuits between particular source and destination nodes. The most basic approach would consist of a computation of all possible paths between all possible combinations of source and destination nodes. However, this approach is infeasible since obtaining all possible paths between two nodes is a well-known NP-hard problem [40].

To be computationally feasible, the path pre-computation algorithm needs to compute only a subset of possible paths in the network. Although this approach makes using path pre-computation feasible, it also presents several drawbacks that need to be taken into account. First, by pruning the list of possible paths, the DynPaC algorithm can reject a service reservation request even in the case of available resources. Therefore, a request can result in a *false negative*, a typical problem when using heuristics. Second, the path pre-computation phase will depend on the use case and the topology of the network. The better the selection of possible paths is, the lower the number of false negatives. In the particular case of BoD, the path pre-computation algorithm has been customized, taking into account two requirements particular to the use case.

First, services are provided E2E between two endpoints, where an endpoint is represented either by a combination of a node and a port or by a combination of a node, a port and a VLAN tag. In terms of path computation, what this requirement represents is that only the possible paths between

Algorithm 1 Path Pre-Computation

```
1:  function PathPreComputation(topology)   ▷ Computes a set of possible paths between pairs of source and destination nodes
2:      nodes ← filterEgressNodes(topology)
3:      for all pair of nodes do
4:          paths ← kShortestPaths(srcNode, dstNode)
5:          for i ← 1, max_cost do
6:              pathLists[i] ← paths.getPathsOfLength(i)
7:          end for
8:          pathMap[srcNode, dstNode] ← pathLists
9:      end for
10: end function
```

Algorithm 2 On-Demand Path Computation

```
1:  function OndemandPathComputation(srcNode, dstNode)   ▷ Provides an ordered list of possible paths between the given source and destination nodes
2:      pairOfNodes ← [srcNode, dstNode]
3:      listPaths ← pathMap(pairOfNodes)
4:      pathsList ← []
5:      for i ← 1, max_cost do
6:          paths ← randomizePathList(listPaths.get(i))
7:          pathsList.append(paths)
8:      end for
9:      return pathsList
10: end function
```

pairs of ingress or egress nodes must be computed. It is unnecessary to go through every combination of nodes since only those facing users or external administrative domains can be selected in service reservation requests. Second, it is desirable to consume the least possible amount of network resources to provide a connectivity service. Therefore, we are interested in the paths with the lowest cost. Hence, to pre-compute a set of alternative paths between the source and destination nodes, the K-shortest path algorithm [41] is used.

As described in algorithm 1, for each pair of ingress and egress nodes the K-shortest paths are computed. Next, these K paths are stored in a map called *pathsLists* that groups the paths of the same length. Finally, all pre-computed paths are stored in a map called *pathMap*, where the source and destination nodes are used as a key to retrieve the path lists.

B. ON-DEMAND PHASE

The second phase of path computation consists of a randomization of lists of possible paths of the same length provided by PCE for a given pair of source and destination nodes. As previously mentioned, PCE in the pre-computation phase generates a map in which it stores a list of possible paths for each combination of ingress and egress nodes. This list is arranged from the smallest to the largest number of hops.

When a new service reservation is made, the service manager requests the list of possible paths to PCE by simply specifying the source and destination nodes. Next, PCE checks pathMap and returns the list of possible paths for that pair of nodes. It is worth noting that this approach results in PCE providing identical path lists for services originating and terminating at the same nodes. As a consequence, there is an increasing probability of congesting some network resources, while others remain underutilized. Algorithms of this kind are usually referred to as *greedy algorithms* since they always try to provide the best path for every request, which in turn results in the congestion of those paths.

To solve the *greedy algorithm* problem, the algorithm used in the on-demand phase randomizes the list of possible paths. As a result, requests with the same source and destination nodes are assigned different lists of possible paths. Note that the set of possible paths is the same, and what varies is the order in which the paths are listed. A randomization is applied to the set of paths of the same length to prioritize the shortest paths at all times. This is precisely the reason pathMap arranges the list of paths for each source and destination pair using blocks of paths of the same length. The procedure used to obtain a randomized list of possible paths is described in algorithm 2.

As previously mentioned, the architecture proposed in this paper includes a resilience module. To support link failures, and when the user requests protection, the on-demand path computation phase generates a pair of disjoint paths. This approach ensures that in the case of a link failure, the backup path will not be affected. Although this approach guarantees a restoration of service in the case of a single link failure, it requires an execution of the path pre-computation phase to accurately reflect the new state of the network in subsequent service reservation requests.

VI. DATA STRUCTURES FOR ADVANCE RESERVATIONS IN SDN

This section presents the data structures that make it possible to support advance reservations for provisioning of the BoD service. First, the *network snapshot* data structure is used to represent the network state in terms of resource availability at a specific period of time. Second, the *NSTree* data structure provides the means to arrange multiple network snapshots hierarchically to ease the admission control and update processes.

A. NETWORK SNAPSHOT

One of the main contributions of the proposed solution is the utilization of *network snapshots*, defined as follows:

Definition 2: A network snapshot represents the state of the network during a specific period of time in terms of network resource availability for a set of concurrent services that have been assigned an optimal path. It is identified by a unique identifier and its start and end times. A network snapshot stores the following information:

- *Identifier (ID):* A unique identifier.
- *Start Time (sT):* The time at which the network snapshot starts.
- *End Time (eT):* The time at which the network snapshot ends.
- *Concurrent Services:* Connectivity services to be provided simultaneously during the specific period of time that the network snapshot represents.
- *Availability Vector:* A vector that summarizes resource availability in terms of available bandwidth per network edge and VLAN tag availability.
- *Optimal Paths:* A map that stores the optimal path for each service in a given network snapshot.

In other words, leveraging the centralized nature of SDN, a network snapshot represents the available resources in the entire network and provides information about concurrent services during a specific period of time. For instance, Figure 3 depicts the evolution of network resource availability when two services are requested. In this case, three different network snapshots are generated: the first when only *service 1* of 20 Mbps exists, the second when *service 1* and *service 2*, of 5 Mbps, coexist and the third when only *service 2* exists. Network resource availability is represented by means of a vector, where each index represents bandwidth availability for a given link. The maximum available bandwidth of all links is 100 Mbps.

B. NSTREE

The need to keep track of all bookable resources over time is a characteristic of advance reservation systems. As a consequence, one of the main problems of systems of this kind is that checking for available resources needs to be done for each time interval affected by the new service request and, in this case, for each network snapshot. For such a scenario, we present the NSTree, a custom hierarchical data structure designed to facilitate the admission control of new service reservation requests and the corresponding update of the data structure when a request is accepted into the network.

The NSTree arranges network snapshots in a way that reduces the number of network snapshots being analysed in the check phase to minimize the response time to the user. Additionally, a minimal penalty in terms of memory consumption is achieved due to the utilization of auxiliary nodes. The structure of the NSTree is depicted in Figure 4.

1) NODE TYPES

Three different types of nodes are used to construct the NSTree, namely, *root*, *network snapshot (NS)* and *common ancestor (CA)* nodes. Each node in the NSTree keeps track of the information specified in the network snapshot's definition. In addition, each node keeps pointers to the next node in chronological order, to the parent node, and to the child nodes. These pointers facilitate arranging various nodes and make traversal of the NSTree feasible.

a: ROOT

The root node is an auxiliary node that represents the root of the NSTree. By definition, only a single node of this type may

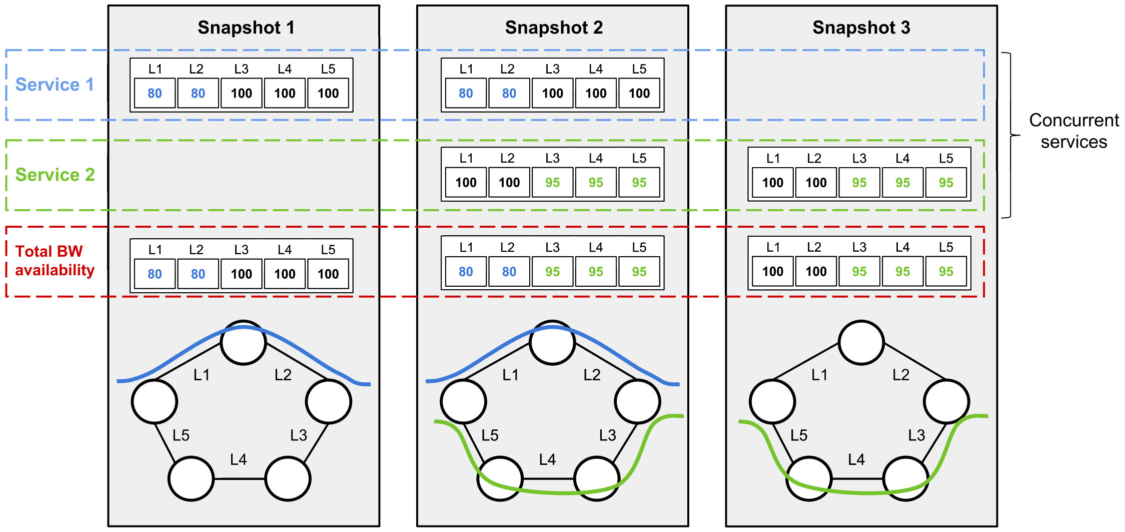


FIGURE 3. Network resource consumption's evolution represented by network snapshots. The maximum bandwidth available in each link is 100 Mbps.

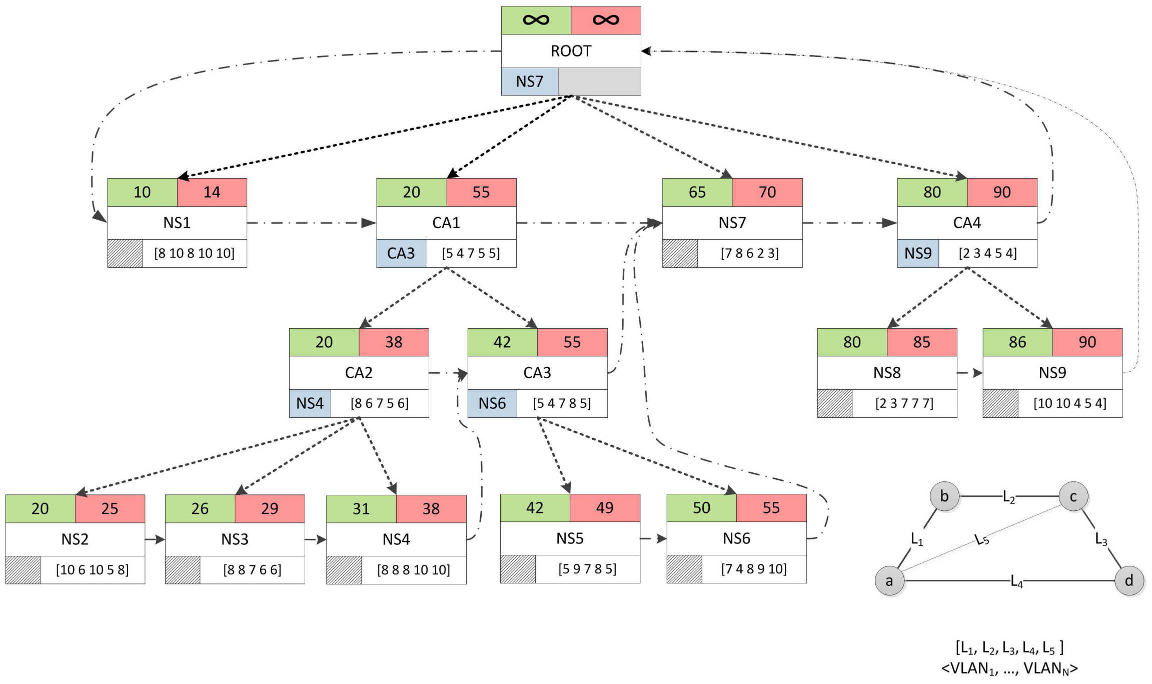


FIGURE 4. Representation of the NSTree data structure used to keep track of resource availability over time in the network depicted in the lower right corner. For each node, the green box represents the start time, and the red box signifies the end time. At the bottom of each node, a vector represents resource availability for each link in the network.

exist; it does not have a parent node. To ease computation, the start and end times of the root are equal to infinity (which in practice takes the highest value of the data type used to represent time). This allows for the root node to be treated as a regular node in the NSTree and facilitates the insertion of the first and last nodes. In contrast to NSs and CAs, it does not keep track of any resource availability, concurrent services or optimal paths.

b: NETWORK SNAPSHOT

A network snapshot node is a node in the NSTree that represents a network snapshot. Resource availability is represented by means of an *availability vector*, where each index represents a network link. Each position of the availability vector provides information about the amount of available bandwidth and the set of available VLAN tags in a specific link. It is worth noting that an NS represents resource availability for a given set of coexisting services, where each service is assigned to an optimal path. Each NS points to the next node in the chronological order, which can be either another NS or a CA. In addition, it also points to its parent node, which in this case can be the root node or a CA but never an NS node. A particular feature of NS nodes is that they do not have any children since they are always leaf nodes in the NSTree.

c: COMMON ANCESTOR

A common ancestor node is an auxiliary node that aggregates a set of consecutive child nodes (both NS and CA nodes). Each CA contains a list of child nodes arranged in the chronological order and keeps track of available resources by using an availability vector built using the minimum values of each index in the availability vectors of all child nodes it aggregates. It represents a longer time interval where a set of common resources are guaranteed to be available. It is identified by a start time equal to the start time of the first child and an end time equal to the end time of the last child. The next node is equal to the next node of its last child in the chronological order. It is worth noting that a CA node does not keep a list of optimal paths since the same service can be provided through different paths in NS nodes aggregated under the same CA.

2) NODE AGGREGATION POLICY

Using CA nodes imposes a penalty in terms of memory usage, and they are only useful when they allow for a reduction in the number of NSs being analysed when a new service reservation is requested. Hence, the number of CA nodes created in the NSTree needs to be kept to a minimum to improve t_{setup}. The criteria used to aggregate multiple NSTree nodes into a CA depend on the nature of the connectivity service being provided and the use case. As such, this section presents the node aggregation policy used in the advanced reservation system presented in this manuscript to provide BoD in the REN environment.

In advance reservation systems, particularly those used in the REN environment, services are usually requested for a long period of time. As a result, it is highly probable for service reservations separated by less than the average duration of service requests to be simultaneously overlapped by new requests. Hence, the choice has been made to use a time-based aggregation policy. Two consecutive nodes—either two NSs, two CAs or an NS and a CA—are paired together into the same CA as defined below:

Definition 3: Two consecutive nodes can be aggregated into a CA if and only if there is less than δ of separation between the end time of the former and the start time of the latter.

Therefore, δ defines the minimum amount of time that needs to exist between two consecutive nodes that are not to be aggregated into a common CA. With such a strategy, every time a new node is created in the NSTree, it is necessary to check whether a rearrangement of the NSTree is necessary. In other words, it is necessary to check if new CAs have to be created or if existing ones have to be updated with new child nodes. The rearrangement of the NSTree is described in the next section.

VII. ADMISSION CONTROL AND THE STATE UPDATE PROCEDURE

Every time a new service reservation is requested, the system must first check in every affected network snapshot whether there are sufficient resources available to meet demand. This phase, called the *check phase*, implements the actual admission control procedure. If the check phase returns a positive response, then the NSTree must be updated accordingly to reflect the new conditions of network usage over time. This second phase is known as the *update phase*.

As previously mentioned, the NSTree has been designed to make both the admission control and update procedures for advance reservations in software-defined networks more efficient even for large network topologies. For this reason, this section describes in detail the algorithms used in such procedures.

A. PHASE I: CHECK

The approach followed to determine whether there are sufficient resources consists of traversing the NSTree chronologically. Every time a node overlaps in time with the service reservation request, it is necessary to check whether the node has enough resources to accept the request during the time interval it represents. This is achieved by comparing the availability vector of the node with the vector describing the resource consumption of the service using a specific path. Since PCE provides a set of possible paths for every service request, ordered according to the number of hops, the algorithm checks all possible paths until a suitable one is found. Once a suitable path has been found, the time interval being analysed is subtracted from the original service request. This procedure is repeated until there is no time pending to be analysed.

The algorithm ends with a negative result when an NS does not have enough resources to provide the service. Otherwise, the algorithm presumes that service demand can be satisfied. The output of the algorithm will be null if the service is rejected. Conversely, it will be the predecessor node, i.e., the last node unaffected by the new service reservation request, if the service is accepted.

To keep track of the time interval pending to be analysed, we use the remaining service time (RST) variable. RST is a duple that specifies the start time (sT) and the end time (eT) of the time interval pending to be analysed. RST is initialized with the start and end times of the requested service, and sT is updated every time it has been checked that an NS or a CA have sufficient resources. The details of the check phase are shown in Algorithm 3.

It is worth mentioning that prior to the execution of the algorithm, a procedure is run to discard the service reservation requests that do not satisfy the timing constraints presented in Section IV or exceed the resources handled by

Algorithm 3 Check resource algorithm

```
function Check($service$, $NSTree$)
    $RST \leftarrow (service.sT, service.eT)$
    $aNode, pNode \leftarrow root$
    $nodes \leftarrow [\,]$
    $predecessorFound \leftarrow false$
    $children \leftarrow aNode.getChildren()$
    if $!children.isEmpty()$ then
        $nodes.push(children)$
    end if
    while $!RST.isEmpty()$ && $!nodes.isEmpty()$ do
        if $!predecessorFound$ then
            $pNode \leftarrow aNode$
        end if
        for all $aNode \leftarrow nodes$ do
            if $RST.sT < aNode.sT$ then
                if $RST.eT < aNode.eT$ then
                    $nodes.remove(aNode)$
                end if
                update RST
                $predecessorFound \leftarrow true$
            else
                if aNode has resources then
                    update RST
                    $nodes.remove(aNode)$
                    $predecessorFound \leftarrow true$
                else
                    if aNode is CA then
                        $children \leftarrow aNode.getChildren()$
                        $nodes.push(children)$
                    else
                        return null
                    end if
                end if
            end if
        end for
    end while
    return $pNode$
end function
```

the advance reservation system (i.e., VLAN tags outside of a specific range or bandwidth reservation requests that exceed the maximum bandwidth of the network).

B. PHASE II: UPDATE

During this phase, the NSTree is updated to reflect new resource availability, which results in new NS nodes being added or updated, and the generation of CA nodes for their aggregation. This characteristic of the NSTree necessitates the evaluation of consecutive nodes to determine if they have to be aggregated into a new CA or an existing one. The algorithm (described in Algorithm 4) uses the RST variable initialized with the service start and end times to detect the termination of this phase.

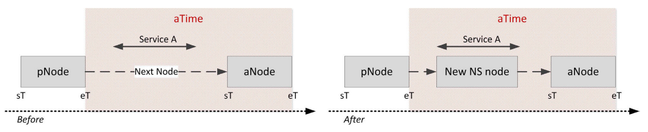


FIGURE 5. Node generation process, case 0: no overlapping.

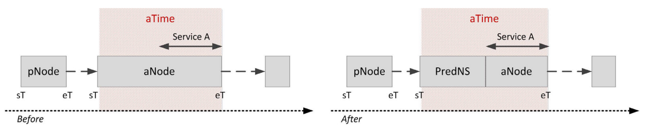


FIGURE 6. Node generation process, case 1: back overlapping.

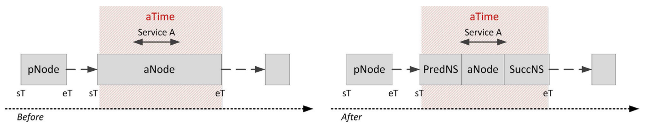


FIGURE 7. Node generation process, case 2: partial overlapping.

Every time a new service reservation is requested, as it happens during the check phase, it is compared chronologically to all nodes that it overlaps in time. In every iteration of the process, a subset of time for which the service reservation has been requested is processed, which is referred to as analysed time (aTime). The aTime interval covers the period of time between the end time of the pNode (the node analysed in the previous step) and the end time of the aNode (the node being analysed in the current step). Once the analysis of a given aTime has finished, that period of time is subtracted from RST to continue analysing the next node in the chronological order. It is worth noting that in every iteration, the analysis encompasses two well-defined phases. First, the lapse of time between the end time of the pNode and the start time of the aNode is considered since it requires the generation of a new NS node. Second, the lapse of time corresponding to the duration of the aNode itself is analysed. The latter requires updating the aNode to update resource availability with resources consumed by the new service reservation, and the generation of additional NSs and CAs depending on how the service overlaps the aNode.

This approach allows for a reduction in node generation or update to five different cases that differ from each other depending on how the service reservation overlaps aTime. The five cases considered in this solution are **Case 0** No overlapping (see Figure 5), **Case 1** Back overlapping (see Figure 6), **Case 2** Partial overlapping (see Figure 7), **Case 3** Front overlapping (see Figure 8) and **Case 4** Full overlapping (see Figure 9). The case of post-overlapping[1] is not considered since it can be regarded as a no-overlapping case over the next node being analysed. This is achieved by considering the root as the pNode of the first node in the chronological order and the last aNode.

[1]Post-overlapping will correspond to a scenario in which the service reservation ends after the aNode.

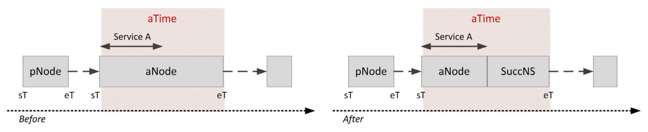


FIGURE 8. Node generation process, case 3: front overlapping.

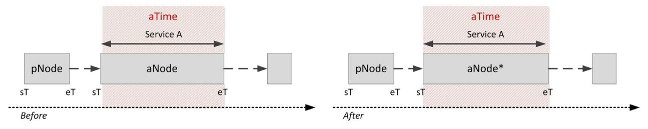


FIGURE 9. Node generation process, case 4: full overlapping.

The following list summarizes the specific procedures depending on the overlapping case.

- **Case 0** requires the creation of a new NS node to reflect resource availability after service acceptance since a gap between the pNode and the aNode represents a period of time in which no services are reserved. During the analysis of aTime, it is possible to have a service reservation request with end time after the start time of the aNode. If this happens, the period of time related to the no-overlapping case is subtracted from RST to continue the analysis since Case 3 or Case 4 can occur.
- **Case 1:** First, a new NS is created equal to the aNode, but with an end time equal to the start time of the service request minus one. This node is named PredNS and represents the part of the original aNode in which the service did not overlap. Second, the aNode is shortened, with a new start time equal to the start time of the service request, and resource availability is updated.
- **Case 2:** First, PredNS is created as a copy of the aNode but with the end time equal to the service start time minus one. Second, the aNode's start and end times are updated to match those of the service, and resource availability is updated. Third, SuccNS is created also as a copy of the aNode prior to the insertion of the new service, with a start time equal to the service end time.
- **Case 3:** First, the original aNode is updated with an end time equal to that of the service reservation request and the new resource availability. Second, the SuccNS node is created, which maintains the same resource availability as that of the aNode before the addition of the new service and with a start time equal to the end time of the requested service.
- **Case 4:** Resource availability of the aNode is updated.

As previously mentioned, the generation of new NSs requires a rearrangement of the tree. As such, in Cases 1-3, the aNode is modified, and additional NSs are created. Since the created NSs are adjacent in time to the aNode, the generation of a CA is required. For example, Figure 10 depicts for Case 2 how a CA would be created from the original aNode descendant of the root to aggregate a copy of the aNode ($A*$), PredNS ($A-$) and SuccNS ($A+$). Note that in *Case 1* or *Case 3*, SuccNS or PredNS, respectively, will be missing.

Algorithm 4 NSTree update for an accepted service

```
 1: function NSTreeUpdateAccept($service, NSTree, pNode$)
 2:     $RST \leftarrow (service.sT, service.eT)$
 3:     $aNode \leftarrow pNode.getNext$
 4:     $solvedNodes = [\ ]$
 5:     while $RST$ is not empty do
 6:         if aNode is solved then
 7:             update RST
 8:             $pNode \leftarrow aNode$
 9:             $aNode \leftarrow aNode.nextNode()$
10:         else
11:             if Case 0 then
12:                 create newNS
13:                 add service to newNS
14:                 rearrangeTree()
15:             else
16:                 if Case 1 then
17:                     create PredNS from aNode
18:                 end if
19:                 if Case 2 then
20:                     create PredNS from aNode
21:                     create SuccNS from aNode
22:                 end if
23:                 if Case 3 then
24:                     create SuccNS from aNode
25:                 end if
26:                 update times from aNode
27:                 rearrangeTree()
28:                 add service to aNode    ▷ Includes Case 4
29:                 $pNode \leftarrow aNode$
30:                 $aNode \leftarrow aNode.nextNode$
31:             end if
32:             update RST
33:         end if
34:     end while
35: end function
```

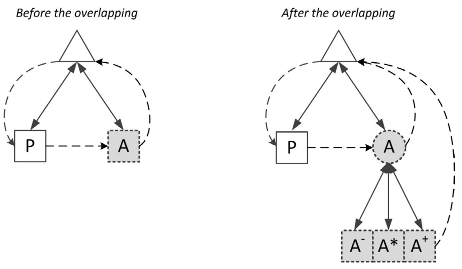


FIGURE 10. Creation of a CA from an NS with a root parent.

Similarly, Figure 11[2] depicts the case of the aNode already being the child of a CA. In this case, the rearrangement is simpler since only PredNS and SuccNS need to be added as children of the same CA. The process of performing such a rearrangement is described in Algorithm 5.

[2]For simplicity, the rest of child nodes of the CA have been omitted from the figure.

Algorithm 5 Creation of a new CA

```
1:  function CreateCA($PredNS$, $node$, $SuccNS$)
2:      $parent \leftarrow node.parent$
3:      if $parent$ is not $CA$ then
4:          $parent \leftarrow newCA$
5:          $node.parent \leftarrow parent$
6:          $newCA.addChildren(node)$
7:      end if
8:      $PredNS.parent \leftarrow parent$
9:      $SuccNS.parent \leftarrow parent$
10:     $parent.addChildren(PredNS)$
11:     $parent.addChildren(SuccNS)$
12:     return $parent$
13: end function
```

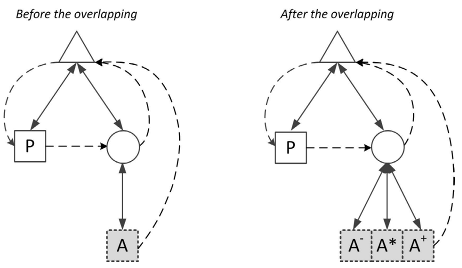


FIGURE 11. Creation of a CA from an NS with a CA parent.

In addition to the immediate creation of CAs as a result of Cases 1-3, it is also necessary to check whether additional rearrangements are necessary, considering the separation between the pNode and the aNode. An aggregation is performed if there is at most δ between the end time of the pNode and the start time of the aNode. The methods utilized for the rearrangement of the NSTree in this case are described in Algorithm 6 that distinguishes the following cases.

a: CASE A: CONSECUTIVE NETWORK SNAPSHOTS AGGREGATED INTO THE SAME COMMON ANCESTOR

When a new NS is created, if the time span between the nodes is less than δ, then it inherits the parent of its pNode. As such, it is possible to have two consecutive nodes separated by less than δ automatically aggregated into a common CA. In other words, in this case it is unnecessary to rearrange the NSTree since the nodes are already arranged accordingly.

b: CASE B: CONSECUTIVE NETWORK SNAPSHOTS BEING CHILDREN OF THE ROOT NODE

In this case, as depicted in Figure 12, the pNode labelled by P and the new node identified as A are both children of the *root* node. If spacing of less than δ exists between the two nodes, then it is necessary to create a new CA to aggregate them. The approach followed in this solution is to upgrade the pNode into a CA, which allows for the retention of pointers that guarantee a chronological ordering. Then, a copy of the pNode identified as $P*$ in the figure is aggregated as the first child of the CA, while the new node is aggregated as the second child.

Algorithm 6 NSTree rearrangement method

```
1:  function RearrangeTree($pNode$, $aNode$)
2:      if $pNode.eT + \delta > aNode.sT$ then
3:          if $Case\ B$ then
4:              $pNodeBis \leftarrow copy(pNode)$
5:              $pNode \leftarrow upgrade2CA(pNode)$
6:              $pNode.addChildren(pNodeBis, aNode)$
7:          end if
8:          if $Case\ C$ then
9:              $ca \leftarrow pNode.getParent()$
10:             $ca.addChildren(aNode)$
11:         end if
12:         if $Case\ D$ then
13:             $pNodeBis \leftarrow copy(pNode)$
14:             $pNode \leftarrow upgrade2CA(pNode)$
15:             $ca \leftarrow aNode.getParent()$
16:             $pNode.addChildren(pNodeBis, ca.getChildren()$
17:         end if
18:         if $Case\ E$ then
19:             $ca1 \leftarrow pNode.getParent()$
20:             $ca2 \leftarrow aNode.getParent()$
21:             $ca1bis \leftarrow copy(ca1)$
22:             $ca \leftarrow upgrade2CA(ca1)$
23:             $ca.addChildren(ca1bis, ca2)$
24:         end if
25:     end if
26: end function
```

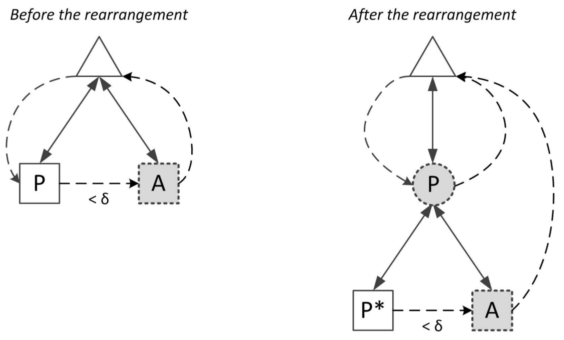


FIGURE 12. Rearrangement, Case B: Two consecutive NS children of the root node.

c: CASE C: PREDECESSOR NETWORK SNAPSHOT AGGREGATED INTO A COMMON ANCESTOR

The case where the pNode is aggregated into a CA, but the new node is a child of the *root* node is illustrated in Figure 13. If there is less than δ of separation between the pNode denoted by P and the new node marked by A, then it is necessary to convert the new node into a child node of the CA. Once this has been done, the CA node is updated with the resources consumed by the new node, and the pointer to the next node of the CA is retrieved from the new node.

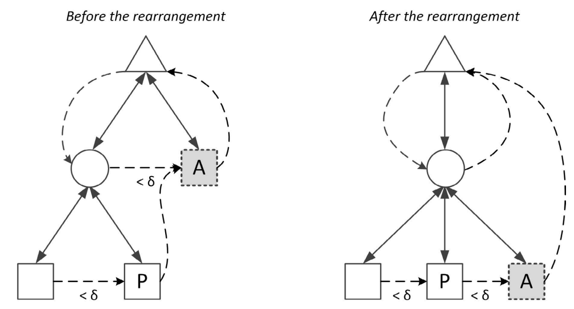


FIGURE 13. Rearrangement, Case C: Predecessor network snapshot aggregated into a common ancestor.

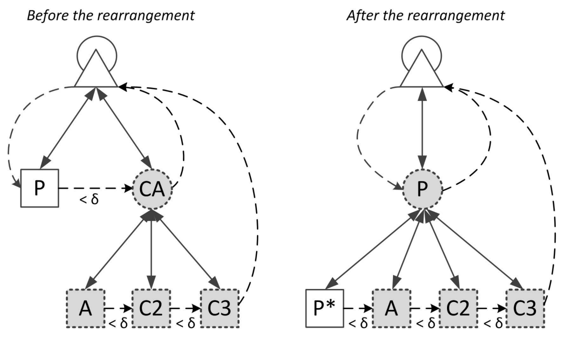


FIGURE 14. Rearrangement, Case D: New node aggregated into a common ancestor.

d: CASE D: NEW NODE AGGREGATED INTO A COMMON ANCESTOR

In this case, the pNode identified as P in Figure 14 is a child of the *root* node, while the new node denoted by A is aggregated into a CA. To ensure a chronological ordering, it is necessary to upgrade the pNode to a CA. Then, a copy of the pNode labelled as $P*$ is added as the first child of the newly created CA. Additionally, the new node and the rest of the children of the initial CA are transferred to the new one. Once all children have been aggregated into the new CA, the initial CA is removed from the NSTree.

e: CASE E: CONSECUTIVE NODES AGGREGATED INTO DIFFERENT COMMON ANCESTORS

Finally, as depicted in Figure 15, when the pNode identified as P is aggregated into $CA1$, but the new node identified as A is aggregated into a different CA denoted by $CA2$, it is necessary to create a higher-level CA to aggregate both $CA1$ and $CA2$, with the result labelled by $CA3$ in the figure. This situation can occur if Case 0 (no overlapping) is followed by Case 1 (back), Case 2 (partial) or Case 3 (front) overlapping. In this case, the new NS node generated in Case 0 can be aggregated in $CA1$, and the aNode has resulted in the generation of $CA2$ as a consequence of the aNode overlap. Although the simplest approach would be to transfer the child nodes of $CA2$ to $CA1$, the algorithm checks whether the numbers of child nodes in both $CA1$ and $CA2$ are equal or, on the contrary, are unbalanced. In the latter, it is preferable to keep the

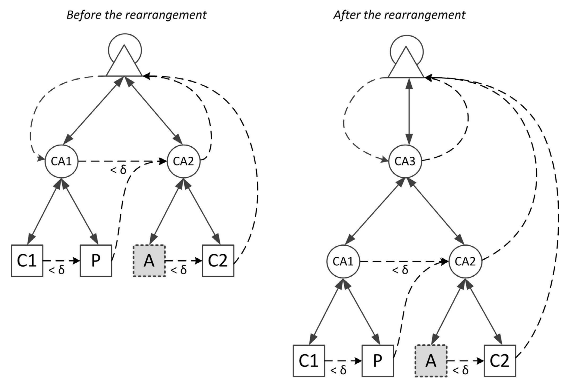


FIGURE 15. Rearrangement, Case E: Consecutive nodes aggregated into different common ancestors.

pNode aggregated in $CA1$ and the aNode aggregated in $CA2$. Nonetheless, since less than δ of separation exists between the pNode and the aNode, it is guaranteed that there is less than δ between $CA1$ and $CA2$, and therefore, it is possible to aggregate these two CA nodes into the higher-level $CA3$.

Once the new nodes have been created and the rearrangement of the NSTree has been performed, it is necessary to update the affected nodes with the selection of the optimal path and the addition of the new service. This means that as in the previous cases, not only NSs but also CAs of which NSs are descendants need to be updated, as described in Algorithm 7.

To ensure that services are not being switched constantly from one path to another when the network snapshot changes, the algorithm uses two strategies. First, it always tries to assign the most stable path to a given service request. This is achieved by using the aggregate utilization information held at the CAs for as long as possible. Therefore, it is necessary to go from the NS being updated to the highest-level CA and add the service using the CA availability vector. If there is a common solution, then all children nodes affected by the service request are marked as solved after the solution for the service has been propagated downwards.

As previously mentioned, after adding the service to all possible nodes, it is necessary to update the availability vector of the CAs affected by the update. To facilitate this task, when a possible path is found for the highest-level CA, that CA and the other CAs previously identified in *Step 1* of algorithm 7 are added to a queue. That queue is later used to update, in the order from the lowest- to highest-level CA, the aggregated availability vectors. As in the case of the *check* phase, the algorithm is executed for each node until RST becomes empty.

VIII. PERFORMANCE EVALUATION

The objective of this section is to confirm the suitability of algorithms and the data structure proposed for efficient provisioning of the BoD service with advance reservations. To this end, it is shown how the proposed solution improves the utilization of network resources, while at the same time,

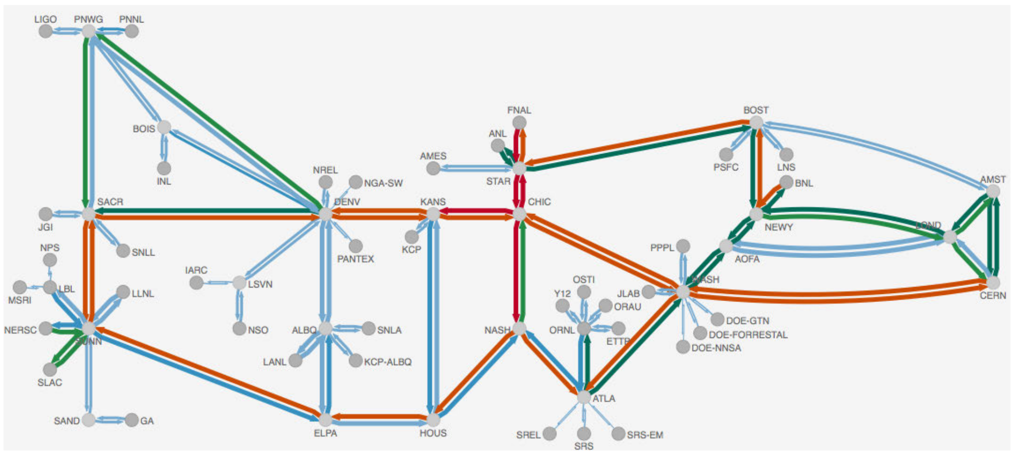


FIGURE 16. Topology of the ESNet network.

Algorithm 7 Adding a Service to an NS

```
1:  function AddService(RST, aNode, pNode)
2:      CAstack.add(aNode)
3:      parent ← aNode.getParent()
4:      while parent.overlaps(RST) do                ▷ Step 1
5:          CAstack.add(parent)
6:          parent ← parent.getParent()
7:      end while
8:      while node ← CAstack.peek() do
9:          path ← node.getOptimalPath()
10:         if path exists then
11:             propagate solution to descending NSs
12:             update CAs
13:         else
14:             CAstack.pop()
15:         end if
16:     end while
17:     return list of solved nodes
18: end function
```

the time and computational complexity of the proposed algorithms as well as the storage impact of the proposed data structures are kept low.

A. EXPERIMENTS

To validate the suitability of the proposed system, two experiments have been performed. The first analyses the impact of the number of alternative paths on network resources' utilization, computational complexity and storage requirements, and the second evaluates the impact of node aggregation policy on the same performance indicators.

An Ubuntu 14.04 virtual machine with 6 GB of RAM and an Intel CORE i7 vPro CPU has been used to perform the experiments. The DynPaC framework has been implemented as an application running in ONOS 1.7. The ESNet network comprising 22 nodes and 66 links (see Figure 16) has been emulated using mininet, where the switches are OVSs with OpenFlow 1.3 support. Each test has been repeated 30 times to gather sufficient data for applying statistics for a normal distribution.

1) EVALUATION OF THE IMPACT OF THE NUMBER OF ALTERNATIVE PATHS

As described in Section V, path computation is handled by means of two different algorithms: (1) pre-computation of K-shortest paths and (2) randomization of the alternative paths. The first experiment aims to assess the impact of the number of alternative paths generated during the path pre-computation phase on the resulting network resource utilization and on the duration of the admission control and state update processes.

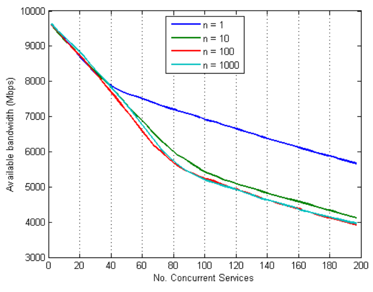
The tests have been performed for various values of K, i.e., the number of alternative paths ranging from 1 to 1000. For each test, 200 services have been requested with a peak bandwidth ranging from 300 Mbps to 1 Gbps in steps of 50 Mbps. To simplify the analysis, all concurrent services have been requested for the same period of time to have a single network snapshot.

2) EVALUATION OF THE IMPACT OF THE AGGREGATION POLICY

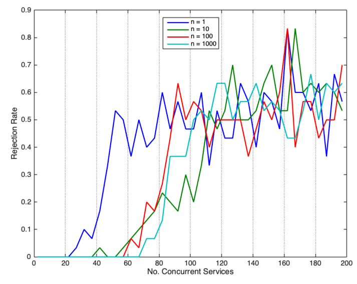
This experiment evaluates the impact of the selected δ on the performance of the solution, where δ refers to the maximum time that may separate two nodes in the NSTree for them to be aggregated into a CA. More specifically, four different values of δ have been used: none,[3] one day, one week and one month. In this case, 500 services have been requested with a random bandwidth between 100 Mbps and 2 Gbps, random start and end times ensuring a duration between 1 day and 4 weeks, and random source and destination nodes to ensure a uniform distribution of service requests.

It is worth noting that even if the value of δ is modified, the number of network snapshots that are generated remains

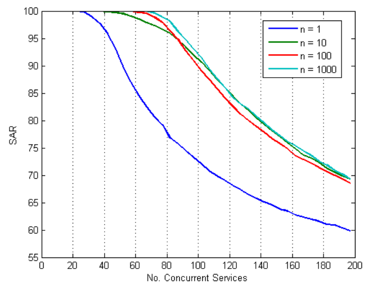
[3]Note that for chronologically adjacent nodes, CAs are still created.



(a) Overall available bandwidth



(b) Service rejection rate



(c) Service acceptance ratio (SAR)

FIGURE 17. Impact of the number of alternative paths on network resources' utilization and SAR.

constant (see Figure 22a afterwards). This is the expected behaviour since the variation of δ does not affect how network snapshots are generated, which only depends on the service start and end times and their relative overlapping.

B. DISCUSSION OF NETWORK RESOURCES' UTILIZATION AND SAR

To assess the improvement of network resource utilization, the following parameters are measured: the service acceptance ratio (SAR), the overall bandwidth available in the network and the probability of a service request being rejected due to a lack of resources. Each of these parameters is evaluated as a function of the number of concurrent services in the network and for various values of the number of alternative paths and parameter δ associated with the aggregation policy

Figure 17a shows the evolution of the overall available network bandwidth with the number of requested services. As can be anticipated, taking into account that in this test all services are requested for the same period of time, as the number of requested services grows, the network is increasingly more congested. Additionally, increasing the number of alternative paths allows for a higher utilization of network resources to be achieved since the number of services that can be accepted in the network increases. As a result, for the same number of requested services the available bandwidth decreases with the number of alternative paths.

As to the service rejection rate, it increases with the number of requested services, as depicted in Figure 17b. The reason is that as previously mentioned, for a higher number of requested services, there are fewer available resources, and therefore, services will be rejected more frequently. Nevertheless, this figure shows a less stable tendency than that in the previous one. The reason is the random selection of source and destination endpoints and bandwidth capacity for each service request.

Considering SAR, Figure 17c shows how SAR decreases with the number of service requests since the network is increasingly more congested. It is worth noting that the availability of more alternatives paths reduces the probability of a new service being rejected and the overall number of rejected services. Therefore, SAR improves, and more network capacity is used to route requested services. In other words, the use of network bandwidth is optimized to improve SAR and minimize rejection of new service requests. It is worth mentioning that as also summarized in Table 1, the difference between utilizing one alternative path or ten paths results in an improvement of SAR by more than 50%. Nonetheless, although using more than ten alternative paths continues to improve SAR, the effect is not as prominent.

This is a consequence of the topology being used, where only the points that connect ESNet with other RENs have been included as source and destination endpoints. Given the low level of path diversity that exists in these nodes, the links rapidly become oversubscribed, while the inner links of the topology remain underutilized. This is also the reason for selecting 40% resource utilization as the optimization objective for results presented in Table 1 since in RENs such as GÉANT, the inter-REN traffic is usually close to that percentage.

Therefore, it can be concluded that using more alternative paths makes it possible to achieve optimal network resource utilization earlier and with a lower penalty in terms of the number of rejected services.

The enhancement achieved in terms of network resource utilization due to pre-computation of multiple alternative paths is also depicted in Figure 18. This figure represents the overall network resource utilization achieved for a given service rejection rate depending on the number of alternative paths. As shown in this figure, for a given rejection rate the overall utilization of network resources increases with the number of alternative paths being used. This demonstrates the benefit of pre-computing multiple path alternatives. This figure also shows that the network utilization level and the rejection rate are positively correlated. In other words, the remaining available bandwidth corresponding to a higher network utilization is lower, and therefore, it is more difficult to accommodate new service requests in the network.

As to the impact of the aggregation policy on network resources' utilization, Figure 19 depicts the number of rejected services and the overall bandwidth availability in the network as functions of the number of concurrent services

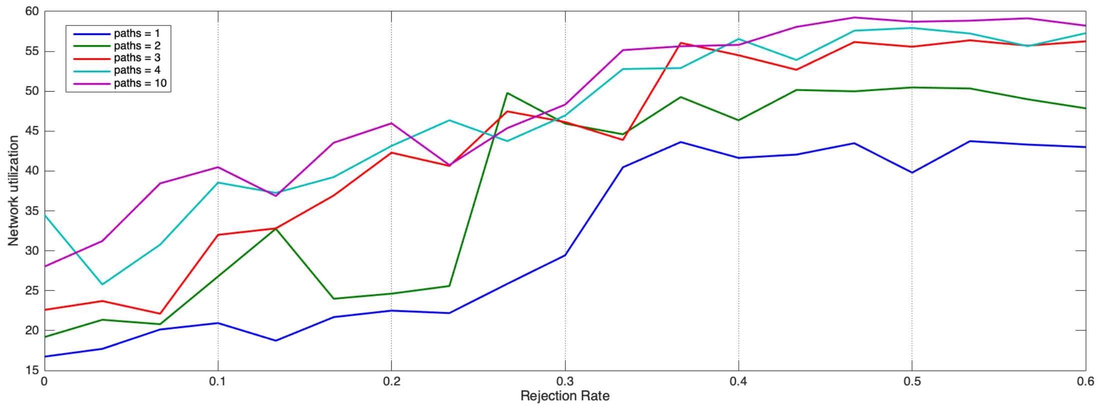
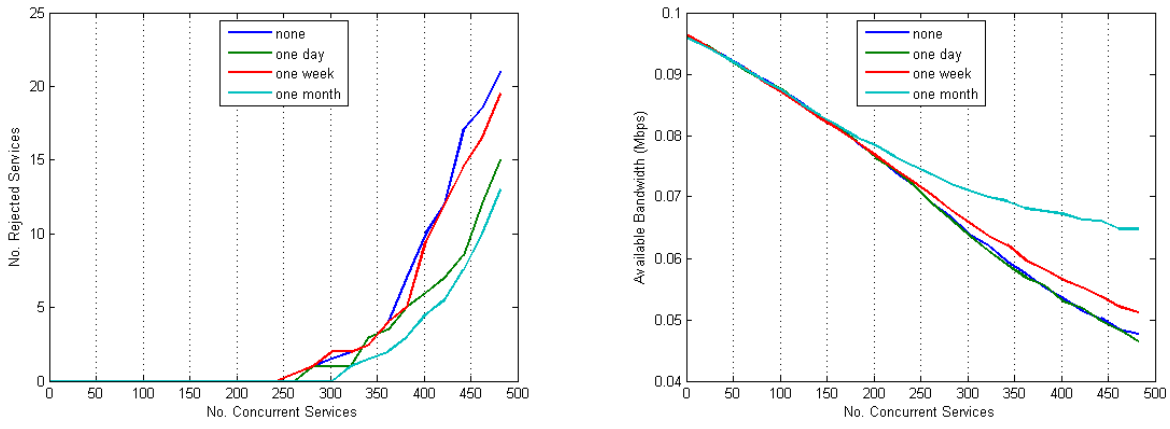


FIGURE 18. Overall network resource utilization depending on service rejection rate.



(a) Number of rejected services

(b) Overall available bandwidth

FIGURE 19. Impact of parameter δ associated with the aggregation policy on network resources' utilization.

and presents the results for various values of parameter δ associated with the aggregation policy.

As Figure 19a shows, the average number of rejected services increases with the number of service requests since the overall network utilization is higher. An important outcome of the results depicted in this figure is that there is no difference in the number of rejected services for different values of δ. This is the expected behaviour since the rejection of a service ultimately depends on the resources available in each network snapshot, the source and destination endpoints of the requested service and the requested bandwidth. Since these parameters behave randomly in the tests, the obtained results also exhibit a degree of randomness. For example, for 500 concurrent services the number of rejected services varies between 12 and 20, which represents a difference of 0,016%. This value is not statistically significant and is due to random requests of resources. In summary, the generation of CAs reduces the time needed to determine whether a service can be accepted but does not impact the fact that a service reservation will only be accepted if there are sufficient resources in the network.

The same is observed for the overall network utilization presented in Figure 19b. The generation of CAs has the outcome of easing the admission control process. However, the network ultimately has a set of specific resources, and the use of the same service reservation pattern results in the same overall network utilization.

C. DISCUSSION OF COMPUTATIONAL COMPLEXITY

An assessment of computational complexity of the algorithms involved in the proposed solution is performed by measuring the time needed to execute each algorithm. It is evaluated as a function of the number of concurrent services installed in the network and for various numbers of alternative paths and values of parameter δ associated with the aggregation policy. More specifically, the time needed to execute the on-demand path computation algorithm as well as the time required by the check and update phases and the entire setup time are evaluated.

Considering the setup time (t_{setup}), it is worth mentioning that it is clearly affected by the number of alternative paths being computed. This is a consequence of all measured times

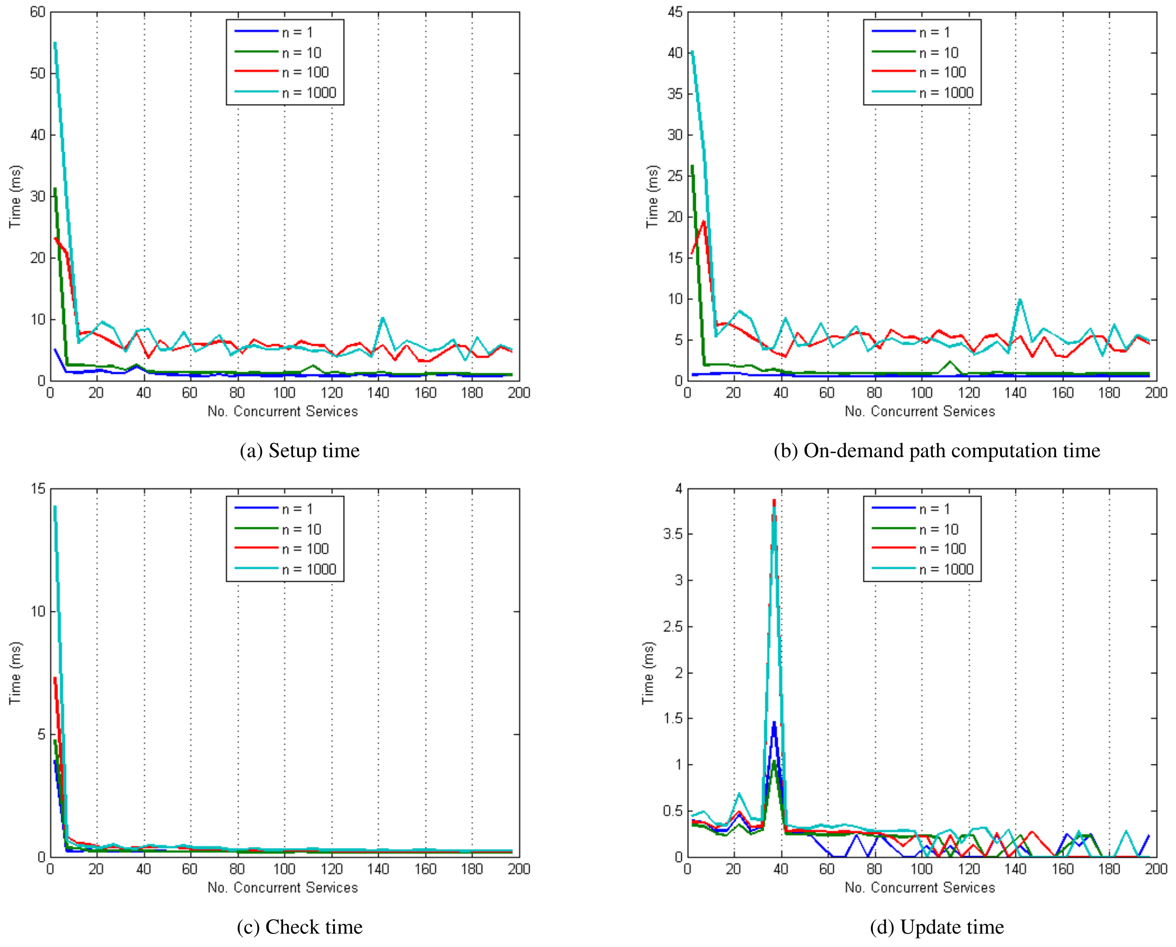


FIGURE 20. **Impact of the number of alternative paths on the computational complexity of the solution.**

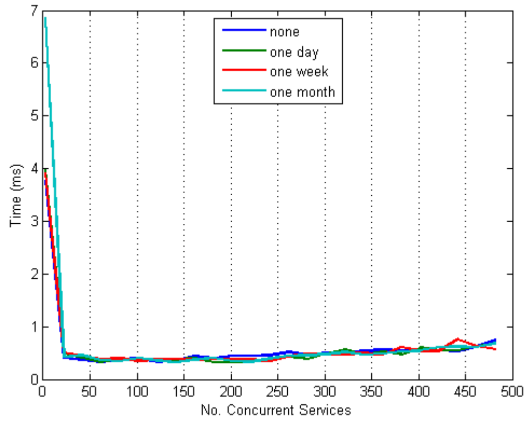
($t_{ondemand}$, t_{check} and t_{update}) being affected by the number of alternative paths. Nonetheless, as shown specifically by Figures 20c and 20d, the impacts on t_{check} and t_{update} are minimal, compared to the impact on $t_{ondemand}$, and within a fraction of a millisecond. The reason is the type of operation performed in each phase. On the one hand, both t_{check} and t_{update} require comparison operations on the array used to represent the consumption vector, which are not computationally intensive. On the other hand, the on-demand path computation time is strongly affected by the number of alternative paths because the randomization process needs to be performed over the entire set of alternative paths. Therefore, a larger number of alternative paths leads to a longer $t_{ondemand}$.

It is also worth mentioning that Figures 20d exhibits a peak at 40 concurrent services. The reason is that up to around this point, the creation of CAs increases very rapidly, as will be explained later in Section VIII-D. As a result, a rearrangement of the NSTree structure must be performed at this point. This operation implies a greater consumption of resources and, as a consequence, an increase in the update time.

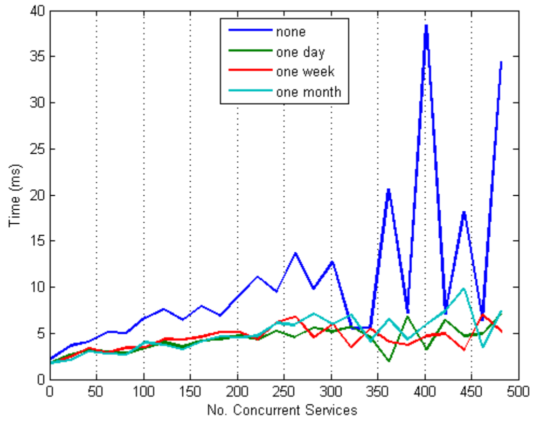
Based on the analysis of charts shown in Figure 21, it can be concluded that on the one hand, the aggregation policy does not have a significant impact on the check time (see Figure 21a). Even if the number of generated CAs varies (see Figure 22b below), the only operation performed in each of the nodes is a comparison of availability vectors since there is no need to create or update the nodes in the NSTree.

On the other hand, the effect on the update phase can be seen in Figure 21b. It must be taken into account that in the update phase, a higher number of CAs leads to a lower number of calls to the algorithm 7 since the selected path can be propagated downwards to the aggregated NSs. As such, in the case where no δ is specified, and therefore, a lower number of CAs are generated, the time required to update the NSTree is longer. The reason for this is the necessity to execute the path selection algorithm more frequently.

It is also worth mentioning that both Figures 20 and 21 show several peaks in the initial parts of the charts that represent the check and on-demand computation times. These out-of-range values are a product of the specific implementation of the algorithm and are caused by the setup of the initial data structures when the tree is first created. Although visually



(a) Check time



(b) Setup time

FIGURE 21. Impact of parameter δ associated with the aggregation policy on the computational complexity of the solution.

these points seem to be high, in fact, they represent numerically very low delays. Additionally, the interesting analysis corresponds to a greater number of requested services. In this case, all necessary data structures have already been created, and this effect does not occur.

It can be concluded that using a hierarchical structure, where the CAs aggregate multiple NSTree nodes and advertise a set of common resources available across a wider time span, results in a lower t_{update}. However, the effect of aggregation is not very prominent in the case of t_{check} due to the simple operations performed in that phase.

Table 1 provides a summary of the results shown so far pertaining to network resources' utilization and computational complexity of the involved algorithms. In this table, the average median t_{setup}, expressed in *ms*, is presented for various values of K, along with the 95% confidence intervals' corresponding minimum and maximum values. Additionally, the number of concurrent services at which 40% network resource utilization is achieved is indicated by SAR, expressed as a percentage. The network resource utilization target has been set to 40%, as this is the usual value used by RENs.

Overall, the obtained results are very promising, taking into account that the average median setup time remains on the order of milliseconds in the case of an application to a real-world REN topology. These results validate the suitability of the advance reservation mechanism for use in RENs.

D. DISCUSSION OF THE STORAGE IMPACT OF THE USED DATA STRUCTURES

The aim of this section is to analyse the impact of the proposed hierarchical data structures in the memory of the server running the proposed system. To this end, we first study the numbers of CA and NS nodes generated as a function of the number of concurrent services in the network and the used aggregation policy. In other words, we explore the numbers of CAs and NSs created for different values of parameter δ used

TABLE 1. Setup time and SAR depending on K at 40% network utilization.

Paths (k)	Mean t_s (ms)	t_s 95% Conf (ms)	SAR (%)
1	1,05	0,91 - 1,19	62,19
2	1,13	0,96 - 1,29	70,46
3	1,24	1,05 - 1,43	86,84
4	1,34	1,08 - 1,61	92,23
10	1,72	1,35 - 2,09	96,25
100	6,28	5,44 - 7,11	98,84
1000	8,29	6,24 - 10,34	99,34

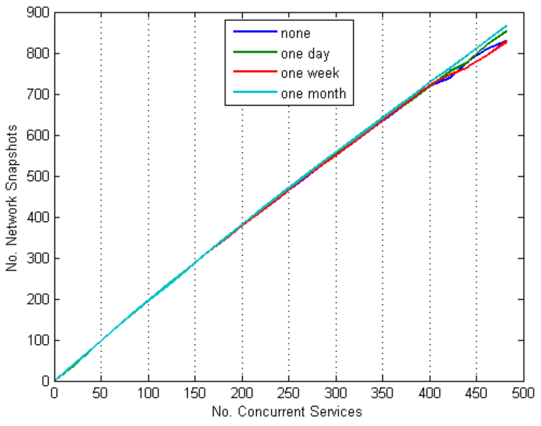
TABLE 2. Relationships among node types, stored fields and sizes.

Field	Object type	Node type		
		Root	CA	NS
ID	int	✓	✓	✓
Start time	long	✓	✓	✓
End time	long	✓	✓	✓
Concurrent services	[int](::max(n))			✓
Availability vector	[int + [short](::max(4096))](::max(m))		✓	✓
Optimal Paths	{service_id(int):int}(::max(n))			✓
Parent	int	✓	✓	
NextNS	int	✓	✓	✓
Children	[int](::max(p))	✓	✓	

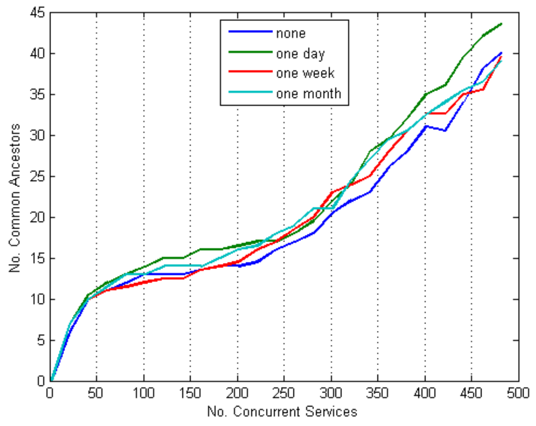
to customize the aggregation policy. As can be concluded from the data represented in Figure 22, the introduction of a hierarchical data structure and the proposed aggregation strategy do not have an impact on the number of the generated network snapshots and have little impact on the number of generated CAs.

Then, we approximate the memory cost of each type of node (root, NS and CA) according to the information contained by such nodes. Table 2 details the relationship among the fields stored in each type of node and their sizes.

The values gathered in Table 2 correspond to a worst-case scenario with respect to memory usage. In such a case, m represents the number of links in the network, n is the number of services, and p is the number of NSs, where $p = 2n - 1$



(a) Number of NS nodes



(b) Number of CA nodes

FIGURE 22. Impact of parameter δ associated with the aggregation policy on storage space required by the used data structures.

is the maximum number of NSs created when n services are enabled in the network. Additionally, *[](::x)* denotes a list of length x, and *(::x)* represents a dictionary of length x. Taking into account that *short* type values use 2 bytes, *int* values use 4 bytes, and *long* values use 8 bytes, we can approximate the maximum size of each node type as shown in equations 5, 6 and 7. Note that the maximum number of VLAN tags available for each link is 4096 and that the overhead introduced by lists and dictionaries is not considered in these values because they are machine- and language-dependent.

$$\begin{aligned} Size\_root &= 4+8+8+4+4p = 24+4(2n-1) \\ &= 20+8n \simeq 8n \; bytes \end{aligned} \tag{5}$$

$$\begin{aligned} Size\_CA &= 4+8+8+m(4+4096*2)+4+4p \\ &= 24+8196m+4p = 20+8196m+8n \\ &\simeq 8196m+8n \; bytes \end{aligned} \tag{6}$$

$$\begin{aligned} Size\_NS &= 4+8+8+4n+m(4+4096*2)+8n \\ &\quad +4+4 = 24+4(2n-1) = 28+12n \\ &\quad +8196m \simeq 8196m+12n \; bytes \end{aligned} \tag{7}$$

Once the maximum sizes of the involved data structures have been computed, we approximate the number of CA and NS nodes by linear regressions based on the results depicted in Figure 22. The amount of memory needed by various data structures, determined using all of these values, is shown in Figure 23 for the case of the ESNET topology and as a function of the number of concurrent services in the network.

The results presented in Figure 23 show that memory usage is clearly dominated by the storage impact of NS nodes. The storage impact of CA nodes included to improve the efficiency of the admission control procedure is close to being negligible. As to the storage impact of NS nodes, the main contributing factor is the storage needed by the availability vector.

Nevertheless, the obtained results show that the memory needs of the proposed solution are acceptable for any current commodity server. Additionally, it must be taken into account that the presented values are maximum values.

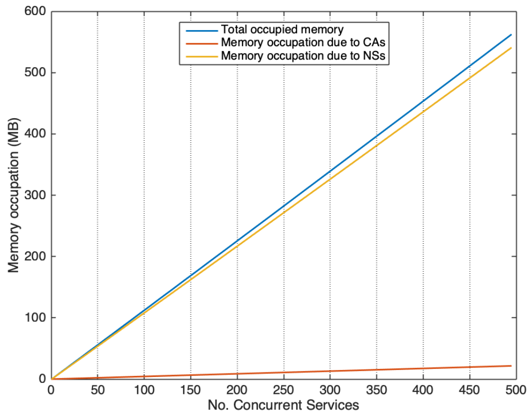


FIGURE 23. Maximum amount of memory needed by the proposed data structures.

E. SUMMARY OF LITERATURE REVIEW AND DISCUSSION

To allow for an easier comparison of all studies analysed in the ‘‘Related Studies’’ section and the proposal presented in this paper, Table 3 provides a structured summary of their main characteristics.

The analysed studies are focused on optimizing data transfers through communication networks, and therefore, the network resource being managed in most of them is bandwidth. An exception is the study of by Guo *et al.* [18] that aims at optimizing a cost parameter computed as a combination of flow table utilization and path length. To achieve their objectives, most studies follow the link-based resource management approach, although some of them [16], [18], [23], [29] propose path-based approaches. Additionally, two studies by Chung *et al.* [24], [31] aim at developing a multi-domain solution relying on the information provided by software-defined exchanges where scheduling and actual bandwidth reservation are performed internally by each domain controller.

TABLE 3. Summary of literature review.

Authors	Main objective	Managed network resource	Approach used to manage network resources	Advance reservations supported	Time intervals
Andreica [16]	Optimize scheduling of file transfers over multiple disjoint paths	bandwidth	Per-path	No	n/a
Balman *et al.* [17]	Optimize data transfer for the earliest completion and the shortest duration	bandwidth	per-link	Yes	fixed
Guo *et al.* [18]	Improve use of an Openflow-aware flow table	flow table utilization, path length	per-path	No	n/a
Zuo *et al.* [21]	Optimize large amounts of data transfers for the shortest path and the earliest completion	bandwidth	per-link	Yes	dynamic
Barshan *et al.* [22]	Improve admittance ratio of video transfers in media production networks by means of an advance reservation system based on flexible timeslots	bandwidth	per-link	Yes	fixed & dynamic
Wang *et al.* [23]	Maximize acceptance ratio for multiple concurrent bandwidth reservation requests	bandwidth	per-path	No	n/a
Chung *et al.* [24]	Improve provisioning time and success rate in advance reservations in multi-domain environments	bandwidth / data size	per-domain	Yes	handled by each domain
Celenlioglu *et al.* [29]	Improve scalability and use of resources in SDN networks	bandwidth	per-path	No	n/a
Hou *et al.* [30]	Speed up completion of data transfers by means of multiple node-disjoint paths	bandwidth	per-link	No	n/a
Chung *et al.* [31]	Improve the success rate of multi-domain advance reservation requests by means of multipath bandwidth splitting	bandwidth	per-domain	Yes	handled by each domain
Zuo *et al.* [32]	Improve the acceptance ratio of bandwidth reservation requests by providing the user with alternative reservation intervals close to the requested one if there are no available resources	bandwidth	per-link	Yes	dynamic
Zuo *et al.* [33]	Scheduling of concurrent bandwidth reservation requests to maximize the amount of transferred data or the number of accepted requests	bandwidth	per-link	Yes	dynamic
Barshan *et al.* [34]	Increase the success rate in failure-prone media production networks by means of totally disjoint backup paths	bandwidth	per-link	Yes	fixed
Zuo [35]	Improve preemption performance when a higher-bandwidth reservation request arrives to minimize the number of preempted requests or the total preempted bandwidth	bandwidth	per-link	Yes	dynamic
Zhang *et al.* [36]	Maximize the acceptance ratio of bandwidth reservation requests over link-disjoint multiple paths to achieve the earliest completion or the shortest duration	bandwidth	per-link	Yes	dynamic
Mendiola *et al.*	Improve the acceptance ratio and the response time of a BoD service with advance reservations	bandwidth	per-link	Yes	dynamic

As to scheduling, among the studies that allow for advance reservations, only a few are based on fixed time slots [17], [34]. Most of them opt for the flexibility provided by dynamic time slots even if managing such structures is more complex. In [22], Barshan *et al.* propose a mechanism for improving the admittance ratio of video transfers in media production networks, which are a specific use case characterized by an interdependence among different video transfers.

Zuo *et al.* have published several papers [21], [32], [33], [35] dealing with the bandwidth reservation service as an enabler of large amounts of data transfers with a guaranteed performance. In their successive studies, the authors explore various optimization options with respect to the scheduling of data transfers: earliest completion vs. shortest duration, maximization of the overall amount of transferred data vs. maximization of the number of accepted requests, etc. In another study [36], researchers also consider the possibility of splitting traffic requests among multiple paths to maximize the number of accepted requests.

The data structure used by these authors to model their bandwidth reservation system exhibits some similarities with the NSTree data structure. In fact, Zuo *et al.* introduce the concepts of a *time step* and a *time window*. A time step is defined as a time lapse in which bandwidth availability remains constant for all network links and therefore is similar to the network snapshot concept. A time window is a concept used to aggregate one or multiple consecutive time steps where the bandwidth availability of each link in the time window is the smallest among all the grouped time steps. Therefore, the time window concept is similar to the CA concept. However, in the cited studies, the authors propose neither an aggregation policy nor a mechanism to keep the tree balanced. Additionally, they do not analyse the impact of the aggregation policy on the execution time of the bandwidth reservation algorithm.

In fact, execution time is not considered in most of the papers as a parameter in the evaluation of performance of the solution and is measured only in [33]. The obtained results show that the average time needed to process a bandwidth reservation request increases exponentially with network size. However, in our proposal, the network size only affects the pre-computation phase, which is performed only once at the network initialization stage, and not the on-demand phase. Therefore, in our approach, bandwidth reservation requests can be processed with a linear rise in execution time as network size increases.

In fact, for each service reservation request, the on-demand path randomization and the setup are performed. As explained above, the setup time considers both the check time and the update time for the accepted services. These two times are dependent on the number of links in the network since to check or update the values of nodes in the NSTree, it is necessary to operate over the consumption vector. Note that the latter represents the aggregate available bandwidth for each unidirectional link in the network graph. Since this vector is implemented as a one-dimensional array, the check or update operations are dependent on the size of this array. Nonetheless, using a consumption vector that summarizes the availability of network resources allows these operations to be performed in O(k), where k is the number of unidirectional links in the network graph. On the one hand, the checking phase only requires a comparison between the consumption vector computed for the alternative path being tested and the consumption vector of the given node in the NSTree. On the other hand, the update phase only requires a subtraction of the consumption vector computed for the alternative path being tested and the consumption vector of the given node in the NSTree.

As to resource utilization and the number of accepted services, it must be taken into account that the acceptance of a bandwidth reservation request ultimately depends on the available resources in the network for the request's duration. This depends heavily on the distribution of service requests with respect to endpoints and bandwidth as well as on the mesh level of the network. A popular mechanism for improving the service acceptance ratio and consequently resource utilization is splitting bandwidth reservations into multiple smaller sub-flows that can be routed through different paths. However, such approaches are mostly limited to the literature, and practical implementations are scarce due to the additional complexity and buffering capacity necessary to store and reorder packets received out of order.

Our approach successfully improves the service acceptance ratio and resource utilization by maintaining multiple alternative paths for each possible source-destination pair in the network. Accordingly, if it is impossible to route a service request through the first alternative path due to bandwidth unavailability, other possible alternative paths will be considered. The obtained results show that there is a significant improvement in the number of accepted services for up to 3 alternative paths. For higher path counts, the relative increase is not as prominent. The reason is that the path randomization algorithm executed in the on-demand phase results in balanced network usage, and therefore, if a service request cannot be routed through the preferred path because the latter does not have available bandwidth, the remaining alternative paths are usually similarly in use.

IX. CONCLUSION

In this paper, an approach to supporting advance reservations that leverages the SDN paradigm has been proposed. This approach uses two custom data structures, namely, network snapshots and the NSTree, and its aim is to evolve BoD service provisioning in the REN environment.

The novelty of this solution relies on per-network resource management designed to handle the timing constraints of the advance reservations support. Network snapshots allow for representing a constant resource availability during a fixed period of time. Arranging network snapshots in the NSTree hierarchically allows for the running time of the admission control procedure to be kept on the order of milliseconds. In summary, the utilization of a hierarchical structure, where the CAs aggregate multiple NSTree nodes and advertise a set of common resources available during a longer time span, results in shorter time needed to perform the admission control procedure. Incidentally, the introduction of this strategy does not have an impact on the number of generated network snapshots, the number of rejected services or the overall network resource utilization since these parameters are independent of the number of CAs.

The proposed solution has been validated using the ESNet topology and the DynPaC framework running as an ONOS application where the data structures and algorithms designed to support the advance reservations have been included. The average median setup time, i.e., the time needed to accept or reject a service, remains on the order of milliseconds in the case of an application to a real-world REN's topology. The path computation strategy followed in this approach relies on pre-computation and posterior randomization of a set of alternative paths. The experiments show that it is unnecessary to pre-compute a very large number of paths to optimize network resource utilization. Moreover, in the case of the ESNet topology, pre-computation of four paths results in network resource utilization of 96,25%, a figure that is 33% higher than that in the case of using a single path, and the service acceptance ratio is 89 %. As a consequence, these results validate the suitability of the advance reservation mechanism for use in the REN environment.

Although the solution presented in this paper has been demonstrated to be a step forward in the support for advance reservations in software-defined networks, the authors of this paper are interested in continuing to improve it. Among the topics to be studied further, it is worth mentioning the introduction of new path computation algorithms that will take into account the actual network congestion or additional parameters such as latency. Moreover, there are plans to extend the data structures and algorithms to take into account not only bandwidth availability but also VLAN availability.

REFERENCES

[1] *Software-Defined Networking: The new norm for networks*, Open Netw. Found., Menlo Park, CA, USA, 2012.

[2] AT&T. (Jan. 2015). *AT&T Domain 2.0 Vision White Paper*. Accessed: Jan. 5, 2020. [Online]. Available: http://www.att.com/

[3] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus, *Requirements for Traffic Engineering Over MPLS*, document RFC 2702, Internet Requests for Comments, Sep. 1999.

[4] A. Mendiola, J. Astorga, E. Jacob, and M. Higuero, "A survey on the contributions of software-defined networking to traffic engineering," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 2, pp. 918–953, 2nd Quart., 2017.

[5] *ESnet*. Accessed: May 14, 2017. [Online]. Available: http://www.es.net/

[6] *ESnet's OSCARS with FloodLight*. Accessed: Jan. 5, 2020. [Online]. Available: https://github.com/hsr/oscars-gui

[7] *Géant*. Accessed: Jan. 5, 2020. [Online]. Available: https://www.geant.org/

[8] *AutoBAHN*. Accessed: Jan. 5, 2020. [Online]. Available: http://geant3.archive.geant.net/service/autobahn/pages/home.aspx

[9] M. Degermark, T. Köhler, S. Pink, and O. Schelen, "Advance reservations for predictive service," in *Network and Operating Systems Support for Digital Audio and Video*. Berlin, Germany: Springer, 1995, pp. 1–15.

[10] N. Charbonneau and V. M. Vokkarane, "A survey of advance reservation routing and wavelength assignment in wavelength-routed WDM networks," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 4, pp. 1037–1064, 4th Quart., 2012.

[11] J. W. Guck, M. Reisslein, and W. Kellerer, "Function split between delay-constrained routing and resource allocation for centrally managed QoS in industrial networks," *IEEE Trans Ind. Informat.*, vol. 12, no. 6, pp. 2050–2061, Dec. 2016.

[12] A. Mendiola, J. Astorga, E. Jacob, K. Stamos, A. Juszczyk, K. Dombek, J. Vuleta-Radoicic, and J. Ortiz, "Multi-domain bandwidth on demand service provisioning using SDN," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, Jun. 2016, pp. 353–354.

[13] M. Karakus and A. Durresi, "Quality of service (QoS) in software defined networking (SDN): A survey," *J. Netw. Comput. Appl.*, vol. 80, pp. 200–218, Feb. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804516303186

[14] T. Wang and J. Chen, "Bandwidth tree–a data structure for routing in networks with advanced reservations," in *Proc. Conf. Proc. IEEE Int. Perform., Comput., Commun. Conf.*, Apr. 2002, pp. 37–44.

[15] R. A. Guerin and A. Orda, "Networks with advance reservations: The routing perspective," in *Proc. IEEE INFOCOM . Conf. Comput. Commun.*, vol. 1, Mar. 2000, pp. 118–127.

[16] M. I. Andreica and N. Tapus, "High multiplicity scheduling of file transfers with divisible sizes on multiple classes of paths," in *Proc. IEEE Int. Symp. Consum. Electron.*, Apr. 2008, pp. 155–158.

[17] M. Balman, E. Chaniotakisy, A. Shoshani, and A. Sim, "A flexible reservation algorithm for advance network provisioning," in *Proc. ACM/IEEE Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2010, pp. 1–11.

[18] Z. Guo, R. Liu, Y. Xu, A. Gushchin, A. Walid, and H. J. Chao, "STAR: Preventing flow-table overflow in software-defined networks," *Comput. Netw.*, vol. 125, pp. 15–25, Oct. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128617301779

[19] M. Barshan, H. Moens, and B. Volckaert, "Dynamic adaptive advance bandwidth reservation in media production networks," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, Jun. 2016, pp. 58–62.

[20] L. C. Wolf and R. Steinmetz, "Concepts for resource reservation in advance," *Multimedia Tools Appl.*, vol. 4, no. 3, pp. 255–278, 1997.

[21] L. Zuo, M. M. Zhu, and C.-H. Chang, "Optimizing trade-off between cost and performance of data transfers using bandwidth reservation in dedicated networks," *J. Netw. Syst. Manage.*, vol. 27, no. 1, pp. 166–187, Jul. 2018, doi: 10.1007/s10922-018-9463-2.

[22] M. Barshan, H. Moens, B. Volckaert, and F. De Turck, "Design and evaluation of a flexible advance bandwidth reservation algorithm for media production networks," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2017, pp. 142–150.

[23] Y. Wang, C. Q. Wu, and A. Hou, "Periodic scheduling of deadline-constrained variable slot-bandwidth reservations for scientific collaboration," in *Proc. 26th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2017, pp. 1–9.

[24] J. Chung, R. Kettimuthu, N. S. V. Rao, and I. Foster, "Democratizing network reservations through application-aware orchestration," in *Proc. 27th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2018, pp. 1–9.

[25] L.-O. Burchard, "Analysis of data structures for admission control of advance reservation requests," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 3, pp. 413–424, Mar. 2005.

[26] J. Schneider and B. Linnert, "Efficiently managing advance reservations using lists of free blocks," in *Proc. 23rd Int. Symp. Comput. Archit. High Perform. Comput.*, Oct. 2011, pp. 183–190.

[27] O. Schelen, A. Nilsson, J. Norrgard, and S. Pink, "Performance of QoS agents for provisioning network resources," in *Proc. 7th Int. Workshop Qual. Service (IWQoS)*, Jun. 1999, pp. 17–26.

[28] M. I. Andreica and N. Tapus, "Efficient data structures for online QoS-constrained data transfer scheduling," in *Proc. Int. Symp. Parallel Distrib. Comput.*, 2008, pp. 285–292.

[29] M. R. Celenlioglu, M. F. Tuysuz, and H. A. Mantar, "An SDN-based scalable routing and resource management model for service provider networks," *Int. J. Commun. Syst.*, vol. 31, no. 8, p. e3530, Feb. 2018.

[30] A. Hou, C. Q. Wu, D. Fang, Y. Wang, and M. Wang, "Bandwidth scheduling for big data transfer using multiple fixed node-disjoint paths," *J. Netw. Comput. Appl.*, vol. 85, pp. 47–55, May 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804516303022

[31] J. Chung, R. Kettimuthu, N. Pho, R. Clark, and H. Owen, "Orchestrating intercontinental advance reservations with software-defined exchanges," *Future Gener. Comput. Syst.*, vol. 95, pp. 534–547, Jun. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18301687

[32] L. Zuo, M. M. Zhu, C. Q. Wu, and A. Hou, "Intelligent bandwidth reservation for big data transfer in high-performance networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[33] L. Zuo, M. M. Zhu, and C. Q. Wu, "Bandwidth reservation strategies for scheduling maximization in dedicated networks," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 2, pp. 544–554, Jun. 2018, doi: 10.1109/TNSM.2018.2794300.

[34] M. Barshan, H. Moens, B. Volckaert, and F. D. Turck, "Design and evaluation of a dual dynamic adaptive reservation approach in media production networks," *J. Netw. Comput. Appl.*, vol. 80, pp. 109–122, Feb. 2017.[Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804516303083

[35] L. Zuo, "Bandwidth preemption for data transfer request with higher priority," in *Proc. IEEE 36th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Dec. 2017, pp. 1–2.

[36] X. Zhang, C. Q. Wu, L. Zuo, A. Hou, and Y. Wang, "Bandwidth scheduling with flexible multi-paths in high-performance networks," in *Proc. 18th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2018, pp. 11–20.

[37] C. Argyropoulos, D. Kalogeras, G. Androulidakis, and V. Maglaris, "PaFloMon - a slice aware passive flow monitoring framework for OpenFlow enabled experimental facilities," in *Proc. Eur. Workshop Softw. Defined Netw.*, Oct. 2012, pp. 97–102.

[38] A. Farrel, J.-P. Vasseur, and J. Ash, *A Path Computation Element (PCE)-Based Architecture*, document RFC 4655, RFC Editor, Internet Requests for Comments, Aug. 2006. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4655.txt

[39] J. Zheng and H. T. Mouftah, "Routing and wavelength assignment for advance reservation in wavelength-routed WDM optical networks," in *Proc. IEEE Int. Conf. Commun. Conf. (ICC)*, vol. 5, May 2002, pp. 2722–2726.

[40] F. Keshavarz-Kohjerdi, A. Bagheri, and A. Asgharian-Sardroud, "A linear-time algorithm for the longest path problem in rectangular grid graphs," *Discrete Appl. Math.*, vol. 160, no. 3, pp. 210–217, Feb. 2012.[Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X11003088

[41] D. Eppstein, "Finding the K shortest paths," *SIAM J. Comput.*, vol. 28, no. 2, pp. 652–673, 1999.

ALAITZ MENDIOLA received the B.Sc. and M.Sc. degrees in telecommunication engineering and the Ph.D. degree from the University of the Basque Country (UPV/EHU), in 2011 and 2017, respectively. She worked as a Researcher with the I2T Research Group, from 2012 to 2017, participating in the EU-funded projects GN3plus, DynPaC, and GN4. She joined GEANT in 2018 as an SDN Developer. She is currently an Research and Development Manager with the Keynetic Technologies, University of the Basque Country UPV/EHU focused on digital identity and SDN-based access control. Her research interests include traffic engineering, software-defined networking, and network function virtualization (NFV).

JASONE ASTORGA received the B.Sc. and M.Sc. degrees in telecommunication engineering and the Ph.D. degree from the University of the Basque Country (UPV/EHU), in 2004 and 2013, respectively. From 2004 to 2007, she worked at Nextel S.A., a telecommunication company. In 2007, she joined UPV/EHU as a Lecturer and as a Researcher in the I2T Research Lab. She is currently an Assistant Professor with the Communications Engineering Department. She has participated in multiple research projects at the local, national, and European levels, and has supervised two Ph.D. theses. Her research interests include software-defined networking, network function virtualization, and IP-enabled wireless sensor networks and cybersecurity.

EDUARDO JACOB (Senior Member, IEEE) received the B.Sc. degree in industrial engineering and the M.Sc. degree in industrial communications and electronics from the University of the Basque Country (UPV/EHU), in 1991, and the Ph.D. degree from ICT, in 2001. He worked for two years in a public Research and Development telecommunication enterprise (currently Tecnalia). Later, he served for several years as an IT Director in the private sector before returning to the Faculty of Engineering in Bilbao (UPV/EHU), where he was the Head of the Communications Engineering Department, from 2012 to 2016. He is currently an Assistant Professor with the Faculty of Engineering in Bilbao. He also leads the I2T (Engineering and Research on Telematics) Research Laboratory. He has directed several Ph.D. theses and managed several research projects at the local, national, and European levels. His research interests include applying software-defined networks to industrial communications, cybersecurity in distributed systems, and IP-enabled wireless sensor networks.

JUANJO UNZILLA received the B.S. and M.S. degrees in electrical engineering and the Ph.D. degree in communications engineering from the UPV/EHU, in 1990 and 1999, respectively. He was the Head of the Electronic and Telecommunications Department, from 2001 to 2004, and the Vice-Chancellor of UPV/EHU, from 2004 to 2013. He is currently a Professor with the Communications Engineering Department, UPV/EHU, where he teaches subjects related to telecommunication networks and services. He is a member of the I2T Research Lab, where he participates in several regional, national, and European Research and Development projects. His research interests include SDN and NFV in 5G networks and its applications to industrial communications and cybersecurity in distributed systems. He is one of the co-founders of the spin-off Keynetic in 2017 focused on cybersecurity services to SMEs.