

Received January 28, 2020, accepted March 10, 2020, date of publication March 31, 2020, date of current version April 24, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2984500

Fast Deployment of Reliable Distributed Control Planes With Performance Guarantees

SHAOTENG LIU¹, REBECCA STEINERT¹, NATALIA VESSELINOVA¹, (Member, IEEE), AND DEJAN KOSTIĆ^{1,2}

¹Research Institutes of Sweden, RISE AB, 164 40 Kista, Sweden

²Royal Institute of Technology, KTH, 164 40 Kista, Sweden

Corresponding author: Rebecca Steinert (rebecca.steinert@ri.se)

This work was supported in part by the Swedish Foundation for Strategic Research (SSF) Time Critical Clouds under Grant RIT15-0075, in part by the Commission of the European Union in terms of the 5G-PPP COHERENT project under Grant 671639, and in part by the Celtic Plus 5G-PERFECTA (Vinnova) under Grant 2018-00735.

ABSTRACT Current trends strongly indicate a transition towards large-scale programmable networks with virtual network functions. In such a setting, deployment of distributed control planes will be vital for guaranteed service availability and performance. Moreover, deployment strategies need to be completed quickly in order to respond flexibly to varying network conditions. We propose an effective optimization approach that automatically decides on the needed number of controllers, their locations, control regions, and traffic routes into a plan which fulfills control flow reliability and routability requirements, including bandwidth and delay bounds. The approach is also fast: the algorithms for bandwidth and delay bounds can reduce the running time at the level of 50x and 500x, respectively, compared to state-of-the-art and direct solvers such as CPLEX. Altogether, our results indicate that computing a deployment plan adhering to predetermined performance requirements over network topologies of various sizes can be produced in seconds and minutes, rather than hours and days. Such fast allocation of resources that guarantees reliable connectivity and service quality is fundamental for elastic and efficient use of network resources.

INDEX TERMS Software-defined networking, distributed control plane, controller placement problem, latency, reliability, routability, optimization.


I. INTRODUCTION

The early definition of the control plane in a software-defined network (SDN) setting assumes that one controller handles flow requests over a set of associated switches. More recent solutions assume a distributed control plane, which consists of multiple physically distributed but logically centralized control instances. Deploying multiple control instances can help to decrease control latency, prevent a single controller from overloading, and tolerate controller failures.

Although distributing the control instances can enhance control plane scalability and reliability, this comes at a cost. Distributed control plane traffic in programmable SDNs encompasses controller-switch traffic and inter-controller traffic and is required to keep the shared network state and information consistent in the control plane [1]–[3]. As the number of deployed controller instances increases,

inter-controller traffic increases dramatically and creates a significant overhead [2], [4]–[6]. Regardless of the consistency level (strong vs. eventual), updating shared state at one of the C controllers intuitively requires a one-to-many style communication to update the $(C - 1)$ remaining instances.

Nonetheless, dealing with the traffic associated with a certain control plane definition is typically ignored. In addition to dealing with an increased control traffic volume, control plane traffic flows have to be forwarded timely and reliably through the network infrastructure with varying link capacities, availability, and other networking properties. Control traffic congestion, for example, is especially destructive since it may degrade control service performance, or worse, cause availability issues unacceptable in services critical to, e.g., human safety or tactile Internet. A highly available control plane is vital for the correct functioning of today's and future programmable networks. Hence, quick autonomous rescheduling and deployment of distributed control plane dynamically reallocating resources is fundamental to ensure

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

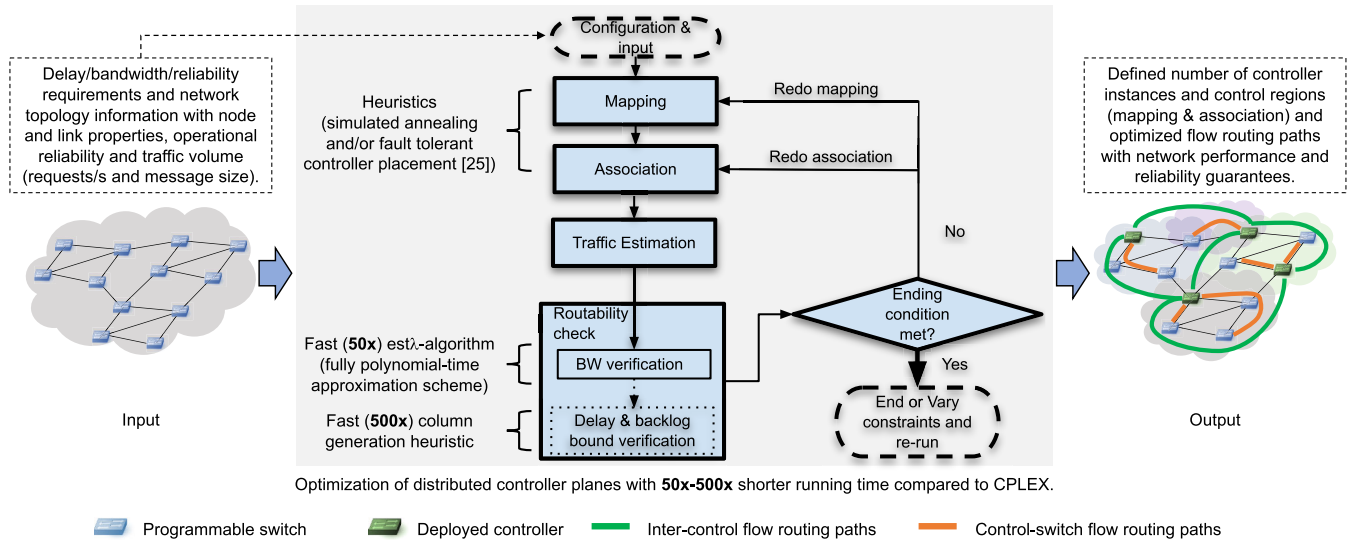


FIGURE 1. Conceptual overview of the approach and proposed methods for fast optimization.

the availability of critical control services under changing network conditions, such as, emergency flash-crowds and network failures.

In this work, we advance the state of the art of distributed control plane deployment by: 1) addressing control traffic routability with respect to required bandwidth allocations and control plane reliability; 2) addressing control traffic delay requirements; 3) outlining a generic black-box optimization process that outputs a distributed control plane deployment plan in line with bandwidth, reliability and delay constraints; and 4) introducing two fast algorithms for bandwidth verification as well as delay and backlog verification based on network calculus. The proposed optimization process facilitates flexible implementation and deployment of distributed control planes with bandwidth and reliability requirements, with or without transmission delay guarantees (Fig. 1).

In the case of routability verification considering only bandwidth and reliability requirements (excluding delay bounds), our *estλ* algorithm runs **50x** faster than the state-of-the-art. In scenarios including delay bounds too, our column generation heuristic (CGH) algorithm can reduce the running time at magnitudes of **500x** (or even more) while still offering near optimal routing solutions (Fig. 2). In practice, this translates to a running time reduction from days to seconds, thereby enabling elastic distributed control planes.

A. DISTRIBUTED CONTROL PLANE BACKGROUND

Fig. 3 illustrates two typical cases of the distributed control plane of a programmable network. An *aggregator* represents either an OpenFlow switch in an SDN or a radio access point in a software-defined radio access network (SoftRAN). In either case, the aggregator acts as a data forwarding device. A *controller* represents a distributed control instance, which is responsible for managing the associated aggregators and flows. In the out-of-band control setting (Fig. 3, left),

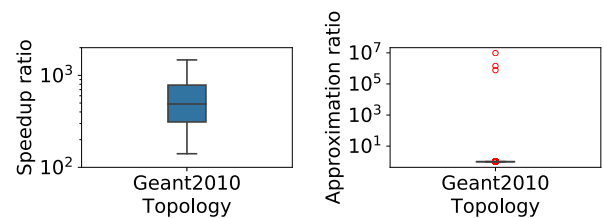


FIGURE 2. Performance of our column generation heuristic (CGH) algorithm used in a large network Geant2010: (left) speedup ratio of CGH over CPLEX, (right) performance approximation ratio of CGH relative to CPLEX. The CGH algorithm is substantially faster and in most cases produces equally optimal solutions as CPLEX.

all controllers are communicating via a dedicated control network. This is the case of running the controllers of an SDN on remote servers connected via dedicated communication links. In the in-band control setting (Fig. 3, right), inter-controller and aggregator-controller as well as data traffic share the same network. A control instance in this case can be co-located with an aggregator in one node.

Distributing the control plane can bring benefits related to both scalability and reliability. Scalability can be achieved by offloading across several control instances, where each instance exclusively controls a subset of aggregators [5] while propagating state changes related to these aggregators [4], [5] (Fig. 3). By placing a controller close to the associated aggregators, the control plane latency can be reduced. Further, using more controllers can also improve the reliability of each aggregator. As long as an aggregator can access at least one operational controller, the aggregator is said to be operational [7]. Deploying a distributed control plane can be demanding because of two grand challenges: 1) timely exchange of information for preserving a common and consistent network view, while 2) ensuring successful inter-controller messaging. The latter can be achieved by robust deployment and routing, which we call *routability*.

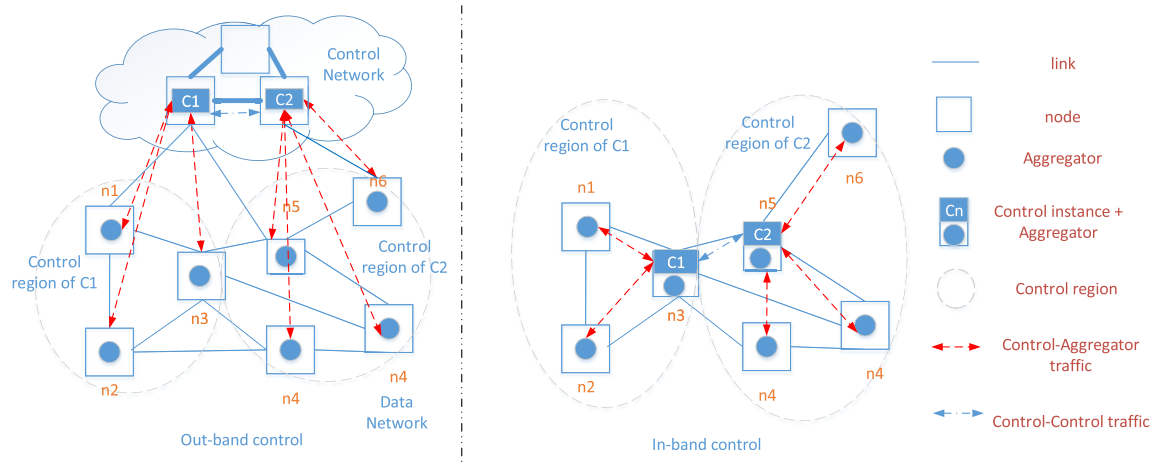


FIGURE 3. Distributed control plane for programmable networks: (left) *out-band* setting, where inter-controller traffic is routed in a dedicated network and (right) *in-band* setting, where control (inter-controller and controller-aggregator) as well as data traffic share the same network. The control of the aggregators in the two depicted examples is distributed between controllers C_1 and C_2 , each responsible for a different subset of the network aggregators.

Coordinating distributed controllers, appearing as one single logical control entity, requires that system events and network state information (such as network topology information) can be shared between the controllers with a certain level of consistency. The behavior of such inter-controller traffic depends on the control application and varies with the size and intensity of information transactions, number of controllers, as well as the communication protocol used for maintaining consistency. Hence, inter-controller traffic can become potentially very expensive in terms of communication overhead in addition to control messages. Different communication/consistency models can be used, synchronous (for strong consistency), or asynchronous (eventual consistency), but the underlying need for one-to-many style communication to update the remaining controller instances remains. In the case of Onix [4], controller coordination events and network states are shared using ZooKeeper [8] and a transactional database called NIB to ensure that information can be distributed with the required consistency levels. As observed in the evaluations of [4], a single update of shared information can generate $4C$ transactions in the control plane, where C is the number of controllers. This finding confirms our intuition behind the required amount of communication: 1) linear in the number of controller instances, and 2) a source of considerable overhead.

Note that from the perspective of a typical SDN setting of today, the inter-control messaging for managing flow tables in OpenFlow switches is relatively modest, varying from a few Mbit/s to a few hundred Mbit/s [2], [6]. However, in the context of the next generation of SDNs, we envision that the inter-controller traffic will vary much more in intensity and size with the deployment of service-specific controller applications, where some control services will generate more inter-controller traffic than others depending on the application and requirements (dynamic control of heterogeneous

wireless networks, service chain coordination, control plane offloading in dense systems, etc.).

Moreover, we cannot always assume that the controllers are deployed in a single data warehouse environment and connected with dedicated ultra-fast networks and homogeneous networking equipment. The diversity of future networks and network applications may require the controllers to reside in highly geo-distributed locations, connected by links of different conditions. Therefore, a deployment strategy of a distributed control plane has to account for the network topology and connectivity in order to ensure robust and reliable inter-controller communication.

B. CHALLENGES IN DISTRIBUTED CONTROL

Control plane deployment here refers to the planning of the controller placement as well as associated control traffic in the distributed control plane. There are two kinds of control traffic: between switches and controllers, and inter-controller traffic [1]–[3]. The traffic routability problem definition depends on applications and QoS requirements. We flexibly address two primary scenarios of the considered problem: 1) with bandwidth and reliability requirements and 2) with bandwidth and reliability plus delay and backlog requirements. Finding a feasible distributed control plane solution is a hard problem mainly due to two major challenges.

First, the control instances must be placed in a way that satisfies the given constraints, such as those related to reliability and scalability. This includes decisions on how many control instances to use, where to place them and how to define their control regions. The controller placement problem in general is NP-hard [9]. Consider a network topology with V nodes. Then, there are V possible choices of the number of controllers to use. When K controllers are used, there are $\binom{V}{K}$ possible ways to map them on the network. For each mapping, there are K^V possible ways of defining control regions.

The size of the entire solution space $\sum_{K=1}^{K=V} \binom{V}{K} K^V$ is huge. To solve the problem, existing work [3], [9]–[15] generally resorts to heuristics to reduce the search space.

Second, it must be verified that the control traffic introduced by a placement solution can be scheduled and routed in the underlying network. The routability problem itself constitutes another major design challenge for the following reasons:

- If we only consider bandwidth constraints, namely whether the flows can be scheduled without overloading any link, such verification can be modeled as a multi-commodity flow problem [16]. Depending on the underlying routing mechanisms of the infrastructure, if flows are splittable [17] the problem can be formulated as a Linear Programming (LP) problem; otherwise, it is a Mixed Integer Linear Programming (MILP) problem [18], which is known to be NP-hard [19]. Moreover, the number of decision variables inside the problem increases exponentially with the networks size. Thus, even if it is an LP problem, it is still challenging to solve it in polynomial time [20].
- If we consider both bandwidth constraints as well as delay and backlog constraints, the problem becomes even more demanding. First, we have to find pertinent ways to model the network elements and flows in order to calculate the delays and backlogs of flows. Second, we need to design an algorithm for finding out a routing plan that satisfies the bandwidth, delay and backlog requirements too. The algorithm must be fast in order to prevent delay and congestion of control traffic and to allow for adapting to real-time network changes (in node and link state, or traffic pattern, for instance).

C. CONTRIBUTIONS

Today, the main shortcoming of existing control plane deployment approaches is the general inability to solve advanced combinatorial problems within reasonable time frames (seconds or minutes). In combination with this drawback, many solutions only consider limited aspects of placement and network performance. Therefore, existing approaches have limited application in practice to network operations and management.

In this article, we propose a novel approach for deploying control plane instances with reliability requirements and routability guarantees covering both bandwidth and delay bounds. Different application scenarios or service providers may have different requirements on routability. In our work we consider two primary scenarios of routability requirements. Scenario 1 only considers the bandwidth limitations, i.e., whether it is possible to route all the flows without exceeding the bandwidth of any link. The corresponding routability problem can be formulated as a multi-commodity flow problem, which is relatively easy to solve. Scenario 2 considers not only the bandwidth limitations, but also QoS guarantees such as end-to-end flow delay bound and backlog

(buffer space) bound. This routability problem is substantially more demanding and time-consuming than scenario 1.

In summary, our contributions are as follows:

- 1) By analyzing the challenges and complexity of the controller placement and traffic routability problem, we introduce a generic black-box optimization process formulated as a feasibility problem, detailing each step of the process along with guiding implementation examples. Unlike existing approaches, our optimization process adds the extra steps needed for quantifying the consequences of deploying a control plane solution fulfilling specified reliability and routability requirements.
- 2) Our proposed optimization process is sufficiently flexible to incorporate network calculus for modeling the network elements and calculating the worst case end-to-end flow delay and backlog requirements.
- 3) We have implemented a fast routability check algorithm *estλ* for scenario 1. The *estλ* algorithm has significantly less time complexity than the original [20] algorithm when used in solving the control plane deployment problem. In our experiments it is faster by **50x** in large networks.
- 4) We have also implemented the CGH algorithm for the scenario 2 routability check. Inspired by the column generation technique, the CGH algorithm simplifies the routing decision by only selecting a small fraction of all the possible paths. According to our experimental results, the CGH algorithm can, for large topologies, reduce the running time to the magnitude of **500x** (or more) while still offering near optimal routing solutions. Because solutions can be obtained within minutes or seconds (instead of hours and days), it is possible to have the algorithm on the critical path of frequent network deployment strategies for adapting to dynamically changing networking conditions. Whereas classic traffic engineering approaches typically account for a fixed number of failures and rely on overprovisioning, our approach can enable less overprovisioning (due to its quick recomputation time), and higher resilience (by increasing the chance of tolerating unforeseen failures).

With the above contributions, we significantly advance our initial approach [21] on solving the control plane deployment problem along with substantial improvements on the methods we use for implementation. We extend our black-box optimization process presented in [21] comprising flow routing under bandwidth constraints, by delay and backlog constraints using network calculus. We detail the design of two routability check algorithms, the refined *estλ* used for bandwidth verification (first introduced in [21]) and a novel algorithm based on CGH for latency verification. We achieve significant reduction in running time with the devised algorithms. In addition, we report on the intuition, demonstrate theoretical proofs, and include evaluation results to show the

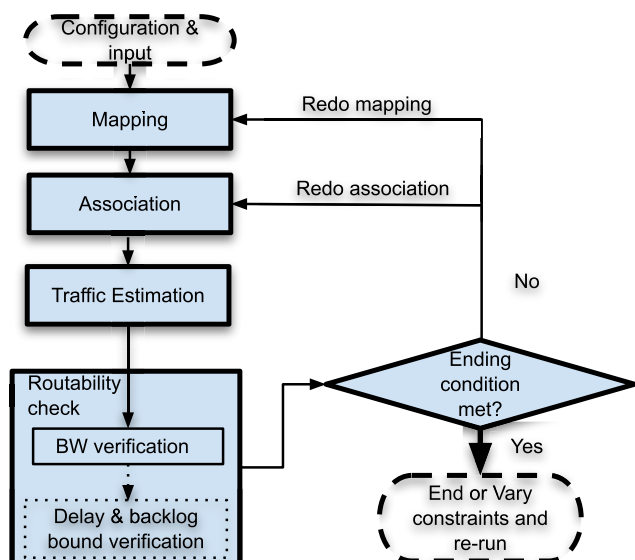


FIGURE 4. The main building blocks of the proposed optimization process: decision on controller instances number and placement (mapping) and control regions (association), estimation of control load (traffic estimation) and verification (routability, reliability and number of iterations).

achieved performance required for practical network operations. Further, we address a basic constraint that we overlooked in our initial approach [21], namely when a node hosts a controller instance and an aggregator, the instance must control the latter. Not respecting such a requirement can unnecessarily inject control traffic, consume energy, decrease reliability, etc.

The remaining of the article is organized as follows. In Section II, we give an overview of our approach. In Section III, we show the prerequisites, assumptions and methods for solving the control plane deployment problem with reliability and bandwidth considerations. In Section IV, we introduce network calculus, and show methods for the control plane deployment problem with reliability, bandwidth, delay and backlog considerations. We show use cases in Section V and evaluation results in Sections VI and VII. Related work and discussions are arranged in Sections VIII and IX, respectively. In Section X we conclude with main take-away messages.

II. THE PROPOSED APPROACH

Our approach for addressing the aforementioned challenges is through an optimization process, which is executed in four steps outlined below and illustrated in Fig. 4.

The **mapping** step places controllers on a given network topology. The input to this first step contains (but is not limited to) network topology and the related link bandwidth as well as the constraints on the placement, such as reliability requirement. The output is a controller location map as well as the quality of the mapping, for instance, the actual reliability. The following **association** step associates aggregators to controllers. The input is the controller location map. The output is an aggregator-controller association plan. The next

traffic estimation step outputs the demand of each control flow according to the input aggregator-controller association plan as well as the demand of each inter-controller flow. The **routability check** step outputs a decision variable, which indicates whether all the control flows can be scheduled or not, given (bandwidth and QoS) requirements. The input consists of network topology properties and control flows. This last step has two sub-steps: bandwidth verification as well as delay and backlog bound verification. Scenario 1 routability check needs to run the bandwidth verification step only, whereas scenario 2 routability check needs to run both.

The process of finding a feasible solution satisfying all conditions (such as reliability, bandwidth, delay and backlog) includes iteration over the four steps until either a feasible solution is found or a limit of iterations is reached (depicted by the ending condition block in Fig. 4).

Note that the process is generic and can be extended to include other (single or multiple) requirements (such as load balancing) by adding proper constraints to the mapping and association steps and end conditions. In other words, the black-box approach offers flexibility to adapting the implementation of each step of the optimization process in line with the practical needs of the network operator. In the following section, we exemplify each step by a possible implementation that addresses the aforementioned challenges and solves a control plane deployment problem.

III. SOLVING THE CONTROL PLANE DEPLOYMENT PROBLEM WITH RELIABILITY AND BANDWIDTH REQUIREMENTS

System reliability is defined as the probability that the system operates without failure in the interval $[0, t]$, given that the system was performing correctly at time 0 [22]. In contrast, service reliability, which we denote by R_{min} , refers to the minimum reliability among all nodes (aggregators). In turn, the reliability of an aggregator is measured by the probability that an operational aggregator is connected to at least one operational controller during the observed interval. Our optimization approach is targeted at *service reliability*. This reliability needs to be guaranteed and above a predefined level called *reliability threshold* and denoted by β , $R_{min} \geq \beta$.

A. PROBLEM FORMULATION

Let $G = \langle V = N \cup M, E \rangle$ be a graph representing a network topology, where V denotes nodes and E links. Moreover, let N denote the set of aggregator nodes and M a candidate set of nodes eligible for hosting controller instances. We model the failure of links, and nodes as i.i.d. random variables. In principle, these probability distributions can be set based on expert knowledge or inferred by learning system performance.

We use binary variables y_i , where $y_i = 1$ if node $i \in M$ hosts a controller, and $y_i = 0$ otherwise. Let $C = \{i | y_i = 1, i \in M\}$ denote the set of deployed controllers and let the binary variable $a_{ij} = 1$ if aggregator $j \in N$ is controlled by the controller in $i \in C$, otherwise $a_{ij} = 0$. Although each aggregator j can only be controlled by one controller

at a time, it can have multiple backup controllers (e.g., with OpenFlow V1.2 protocol [23]). The reliability of node j is represented as $R(G, j, C)$ (among $|C|$ controllers), capturing the probability of node j connecting with at least one of the operational controllers. Solutions satisfying the constraints given topological conditions and reliability threshold β are found by $R_{min} = \min_{j \in N} R(G, j, C) > \beta$.

We can formulate the control traffic routability problem in programmable networks as a multi-commodity flow problem [24] by taking flow splitting into account [17]. Let u_e be the bandwidth on each link $e \in E$ allocated to control plane traffic. Suppose (s_f, t_f) is the (source, sink) of control traffic flow f . Let d_f denote the demand (throughput) of f . Let $F = \{f : (s_f, t_f, d_f)\}$ be the set representing the entire control traffic in the network. Let $F_c \subset F$ be the inter-controller traffic, namely $F_c = \{f : (s_f, t_f, d_f) | s_f \in C, t_f \in C\}$. Let κ_f denote all the possible non-loop paths for $f \in F$, and let $\kappa = \cup_f \kappa_f$. Let variable $X(K)$ denote the reserved guaranteed service rate for a flow along path $K, \forall K \in \kappa$. Then, the reliable control plane deployment problem can be formulated as follows:

$$\begin{aligned} & \text{maximize } 0 \\ & \text{s.t.: } \sum_{i \in C} a_{ij} = 1, \quad \forall j \in N \end{aligned} \quad (1)$$

$$\sum_{i \in M} y_i \geq 1 \quad (2)$$

$$R(G, j, C) \geq \beta, \quad \forall j \in N \quad (3)$$

$$\sum_{K \in \kappa_f} X(K) \geq d_f, \quad \forall f \in F \quad (4)$$

$$\sum_{K: e \in K} X(K) \leq u_e, \quad \forall e \in E \quad (5)$$

$$y_i, a_{ij} \in \{0, 1\} \quad (6)$$

$$X(K) \geq 0, \quad \forall K \in \kappa \quad (7)$$

The above formulation of the control plane deployment problem is general: for $M \subseteq N$, it corresponds to an in-band control plane problem formulation, whereas for $N \cap M = \phi$, it reflects the out-of-band one. Recall that in the latter case, the inter-controller traffic F_c is served by a control network. This is implicitly included in the definition of the set κ_f .

The main difference between this formulation and the traditional reliable controller placement problem [25] is that we model the control plane deployment as a feasibility problem without an optimization objective. The feasibility problem formulation takes into account the constraints on control traffic which, to our knowledge, have not been addressed previously.

This problem is hard in terms of computational complexity for the following reasons. First, constraints (1), (2), (3), (6) constitute a fault tolerant facility location problem. Second, constraints (4), (5), (7) form a multi-commodity flow problem. Third, the computation of the reliability $R(G, j, C)$ can be an NP-hard problem by itself [25]. Fourth, the number

of variables $X(K)$ might be exponential in the number of nodes N and/or edges E .

B. MAPPING

The problem of optimally choosing the number of control instances as well as their location (see Fig. 4) is a combinatorial optimization problem. The simulated annealing (SA) algorithm [26], [27] has been extensively applied for solving combinatorial problems from diverse fields [28]. Further, SA is easy to implement in practice once its constituent parts (such as the cost function and transition probability) are properly defined.

We implement the mapping step of the optimization process following the standard simulated annealing template [26], [27] except that the Simulated Annealing for Mapping (SAM) algorithm that we design generates a new solution and decreases the temperature T when a *redoMapping* signal is received. Such a signal is sent when the reliability verification (executed in the "ending condition" block of Fig. 4) has failed ($R_{min} < \beta$). The temperature T is used to guide the search of the SAM algorithm. The initial number and placement of controllers can be randomly decided or using a heuristic algorithm such as [25]. After initialization (lines 1–4 in Algorithm 1), a new mapping is generated when a *redoMapping* signal is received. In SAM, the *cost_{new}* (a user-defined cost function) of the latest mapping solution C along with the current temperature T is used to decide whether the new mapping plan can replace the previous mapping solution (lines 6–10). The *transition probability function* $P = \min(1.0, \exp(\frac{cost_{new} - cost_{old}}{T}))$ (line 8) defines the probability with which the new mapping will be accepted. The *getNextSolution*($C_{current}$) function generates a new mapping based on the previous mapping ($C_{current}$) by randomly adding, removing or moving (changing the node/location of) a control instance (line 11). Then, the reliability R_{min} of the new mapping is computed (line 12). Finally, the temperature is decreased by a factor of γ (line 13). In our implementation of the simulated annealing algorithm, the mapping aims at maximizing a cost function:

$$cost = \min(0, \log_{10} \frac{1 - R_{min}}{1 - \beta}, \lambda - 1) \quad (8)$$

The λ is calculated in the routability checking step. It is an indicator of whether control traffic is routable ($\lambda \geq 1$) or not ($\lambda < 1$). When both routability and reliability constraints are satisfied (namely, $\lambda \geq 1$ and $R_{min} \geq \beta$), the cost function reaches its maximum value 0.

Since directly computing the reliability $R(G, j, C)$ is NP-hard [25], the approximation method proposed in [25] is applied for computing a lower bound $\hat{R}(G, j, C)$ instead. The approximation method first computes the set of the disjoint paths from a node j to all the controllers C , denoted as κ^j . Given the i.i.d operational probabilities of links and nodes on each disjoint path, the failure probability of each path, denoted by $p_k, k \in \kappa^j$, can be calculated. Then, the lower bound can be computed: $\hat{R}(G, j, C) = 1 - \prod_{k \in \kappa^j} p_k$, [25].

Algorithm 1 The Simulated Annealing Algorithm for Mapping

Input control signal: *RedoMapping* with inputs $C, cost_{new}$

Initialization

- 1: Choose a set C of controllers from the set V
- 2: Calculate $R_{min} = \min_{j \in V} (\overline{R}(G, j, C))$
- 3: $C_{current} = C, T = T_{Initial}$
- 4: **Output** R_{min}, C

- 5: **Upon control signal** $\langle RedoMapping | C, cost_{new} \rangle$ **do**
- 6: **if** $T == T_{Initial}$ **then**
- 7: $cost_{old} = cost_{new}$
- 8: **else if** $P(cost_{old}, cost_{new}, T) \geq random(0, 1)$ **then**
- 9: $C_{current} = C$
- 10: **end if**
- 11: $C = getNextSolution(C_{current}),$
- 12: Calculate $R_{min} = \min_{j \in V} (\overline{R}(G, j, C))$
- 13: $T = \gamma T$
- 14: **Output** R_{min}, C
- 15: **end upon**

C. ASSOCIATION

The association algorithm implements simulated annealing for Association (SAA) and is similar to Algorithm 1. Therefore, instead of repeating the entire algorithm, we outline the main differences:

- During the initialization, each aggregator is assigned to its closest controller. If there is a single controller, the association step just stops after the initialization as there is only one possible association.
- The cost function used is $cost = \min(0, \lambda - 1)$.
- The implementation of the `getNextSolution()` function is shown in Algorithm 2. Its general work flow is: first, a controller is selected randomly (line 2); then, an aggregator from the set of aggregators not currently associated with the selected controller (denoted by *rest* in Algorithm 2) is randomly chosen (line 5). When the distance between the selected controller and aggregator is small, there is a high probability that the aggregator will change its association and will be assigned to the considered controller (lines 6-11).

The association stops if a routable association plan is found (indicated by $cost = 0$), or the temperature used for simulated annealing is below a certain threshold.

D. TRAFFIC ESTIMATION

The demands of aggregator-controller and controller-controller flows have to be estimated. Let (s_f, t_f, d_f) represent the source, sink and demand of a flow f respectively. The objective of this step is to estimate each d_f while s_f and t_f are known from the mapping and association steps.

In principle, since the optimization process treats the model of control traffic as an input variable, any traffic model can be applied for estimating each d_f . For example, we can

Algorithm 2 Procedure of getNextSolution() for Association

Input control signal: The set of controllers C . Current association $\{a_{i,j} | i \in C, j \in N\}$. Number of hops between any pair of nodes $dist(i, j), i \in N, j \in N$. Let $A(c)$ denote the set of aggregators associated to controller c .

- 1: **procedure** getNextSolution($C, \{a_{i,j} | i \in C, j \in N\}$)
- 2: Randomly select a controller $i \in C$, that satisfies $rest = N - A(i) - C \neq \emptyset$, where *rest* denotes the aggregators not associated to i .
- 3: Compute $minDist = \min_{j \in rest} dist(i, j)$.
- 4: **while** True **do**
- 5: Randomly select an aggregator $j \in rest$
- 6: $distInv = 1/(dist(i, j) - minDist + 1)$
- 7: **if** $distInv \geq random(0, 1)$ **then**
- 8: Get the current controller i' of j .
- 9: Assign $a_{i',j} = 0$, assign $a_{i,j} = 1$.
- 10: **return** $\{a_{i,j} | i \in C, j \in N\}$
- 11: **end if**
- 12: **end while**
- 13: **end procedure**

model either average or worst case demands, with either simple linear modeling method or advanced machine learning techniques.

However, as the scope of this paper concerns the generic optimization process, we employ a simple traffic estimation model, assuming that the message sizes of aggregator request and corresponding controller response are $T_{req} = 128$ and $T_{res} = 128$ bytes, respectively. Furthermore, after dealing with a request, the controller instance sends messages of size $T_{state} = 500$ bytes to each of the other $|C| - 1$ control instances notifying them about the network state changes. Note that this traffic model is essentially in line with the ONOS traffic model as described in [2]. The message sizes are here set according to [2], [6], [29], but can be set arbitrarily. With these parameter settings and given the request rate $r_j, j \in N$ of each aggregator, we simply estimate the traffic between aggregator j and its associated controller by $r_j T_{req}$ for aggregator-controller direction and by $r_j T_{res}$ for the controller-aggregator direction. We also use a simple linear model to estimate the outgoing traffic from controller i to any other controller j , which is given by $T_{state} \sum_{j \in N} a_{ij} r_j$.

E. ROUTABILITY CHECK

If only bandwidth constraints are considered, the routability check consists of a bandwidth verification phase. It is a multi-commodity flow feasibility LP problem. Solving this problem means dealing with an undesired exponential number of variables, as indicated by the constraints (4), (5), (7). This issue can be circumvented by formulating a maximum concurrent flow problem [30] (as (9), (10), (11), (12) suggest), which is easier to solve and equivalent to the multi-commodity flow problem.

The fundamental idea of the maximum concurrent flow problem is to keep the capacities of the links fixed while

scaling (adjusting) the injected traffic so that all flows fit into the network. The optimization objective λ reflects the fraction of the traffic that can be routed. When $\lambda \geq 1$, the current traffic is routable, which means that the link utilization is below 100% for all links. In short, more traffic variation can be tolerated with a larger λ .

$$\text{maximize } \lambda \tag{9}$$

$$\text{s.t.: } \sum_{K:e \ni K} X(K) \leq u_e, \quad \forall e \in E \tag{10}$$

$$\sum_{K \in \kappa_f} X(K) \geq \lambda d_f, \quad \forall f \in F \tag{11}$$

$$X(K) \geq 0, \quad \forall K \tag{12}$$

The dual [31] of the above maximum concurrent flow problem has a linear number of variables and an exponential number of constraints, as formulated in (14), (15), (16) (17). This allows for elegantly solving the problem to a desired level of accuracy using a primal-dual algorithm. In particular, we can apply the FAS (Faster Approximation Schemes) algorithm designed by Karakostas [20]. With this algorithm, the near-optimal λ can be obtained, which is guaranteed within the $(1 + \epsilon)$ factor of the optimal and time complexity of $\mathcal{O}(\epsilon^{-2}|E|^2 \log^{\mathcal{O}(1)}|E|)$, according to [20], [30].

$$\text{minimize } D(l) = \sum_{e \in E} u_e l_e \tag{13}$$

$$\text{s.t.: } \sum_{e \in K} l_e \geq z_f, \quad \forall f \in F, \forall K \in \kappa_f \tag{14}$$

$$\sum_{f \in F} d_f z_f \geq 1 \tag{15}$$

$$l_e \geq 0, \quad \forall e \in E \tag{16}$$

$$z_f \geq 0, \quad \forall f \in F \tag{17}$$

Although FAS has been used for solving flow routing problems [24], [32], using it in its original [20] form for verifying control traffic routability can in fact be time consuming and hence not suitable. The control plane traffic routability problem is a special flow routing problem, where every control flow either originates or terminates in a controller, or both (has its origin and destination from the set C of controllers). Inspired by this specific phenomenon, we modified the FAS algorithm, and named the modified algorithm FPTAS (as it belongs to Fully Polynomial Time Approximation Schemes [33]). The resulting FPTAS algorithm runs much faster than FAS in solving the control traffic routability check problem as explained below and demonstrated in Appendix A. This algorithm was initially introduced in our previous work [21]. In the following, we report for the first time the details of the algorithm and its performance.

The FPTAS algorithm consists of a three-layer loop as described in Algorithm 3. We name a round of the outermost layer loop a *phase*, a round of the middle layer loop an *iteration* and a round of the innermost layer loop a *step*. The algorithm works as follows: initially, it computes a value δ that is a function of the desired accuracy level ϵ , and the

Algorithm 3 The FPTAS Algorithm for Computing λ

```

1:  $D(l) \leftarrow 0$ 
2:  $l(e) \leftarrow \delta/u_e$ 
3:  $R_f \leftarrow 0, \forall (s_f, t_f)$ 
4: while  $D(l) < 1$  do ▷ phase loop
5:   for each node  $c \in C$  do ▷ iteration loop
6:      $d'(f) = d_f \forall f \in F^c$ 
7:     while  $D(l) < 1$  and  $d'(f) > 0$  for some  $f \in F^c$  ▷ step loop
8:       do
9:          $P_f^c$ : Shortest path using  $l$  as link weights,  $\forall f \in F^c$  with  $d'(f) > 0$ 
10:         $\rho(e) = \sum_{f:e \in P_f^c} d'(f)/u_e$  is the utilization of  $e \in E$ .
11:         $\sigma = \max(1, \max_{e \in \cup_f P_f^c} \rho(e))$ 
12:        Route  $d_r(f) = d'(f)/\sigma$  amount flow along  $P_f^c, \forall f \in F^c$  with  $d'(f) > 0$ 
13:         $d'(f) = d'(f) - d_r(f), \forall f \in F^c$ 
14:         $R_f = R_f + d_r(f), \forall f \in F^c$ 
15:         $l(e) = l(e)(1 + \epsilon(\sum_{f:e \in P_f^c} \frac{d_r(f)}{u_e})), e \in \{\cup_f P_f^c | \forall f \in F^c\}$ 
16:        Compute  $D(l) = \sum_{e \in E} u_e l(e)$ 
17:     end while
18:   end for
19:  $R_f = R_f / \log_{1+\epsilon} \frac{1+\epsilon}{\delta}, \forall f \in F$ 
20:  $\lambda = \min_{\forall f \in F} (R_f/d_f)$ 

```

Output: λ

number of edges $|E|$. We set $\delta = (1 + \epsilon)^{\frac{\epsilon}{1-\epsilon}} (\frac{1-\epsilon}{|E|})^{1/\epsilon}$ as in [20]. The weight of each edge $e \in E$, is denoted by $l(e)$ and $l(e)$ is initialized to δ/u_e . Then, the algorithm iterates in phases (suggested by the outermost while loop in line 4). Each phase consists of $|C|$ iterations (suggested by the for loop in line 5), and each iteration contains one or several steps (suggested by the innermost while loop in line 7). In each phase, every flow $f \in F$ is routed with d_f amount, distributed on one or several non-loop paths between s_f and t_f . We can route all the flows with $|C|$ iterations in each phase, since every control plane flow has at least one end (source/sink) in C . In each iteration, we select a controller $c \in C$, and deal with a subset of flows F^c that share a common source/sink c ($F^c = F^{cs} \cup F^{ct}$, that $F^{cs} = \{f | s_f = c\}, F^{ct} = \{f | t_f = c, s_f \neq c\}$). The algorithm keeps updating the weight function $l(e)$ in each step. At every step we compute the shortest tree that starts from c or terminates at c using the $l(e)$ link weights. Such a shortest path tree can be computed with Dijkstra's algorithm.

In summary, our FPTAS algorithm follows the same idea and workflow of the FAS algorithm [20]. The modification we introduce is that in each phase, the computation iterates through the controller nodes C , rather than through all the flow source nodes (V in this case). Appendix A explains why this modification reduces the time complexity.

Algorithm 4 The Est λ Algorithm

```

1: calculate  $\lambda_{high}, \lambda_{low}$ 
2: if  $\lambda_{high} < 1.0$  or  $\lambda_{low} > 1.0$  then
3:    $\lambda = (\lambda_{high} + \lambda_{low})/2.0$ 
4: else
5:   compute  $\lambda$  with the FPTAS algorithm described in
   Algorithm 3.
6: end if
Output:  $\lambda$ 

```

To further accelerate the routability verification step, we proposed in [21] a faster algorithm for estimation of λ , which we here name the *est λ* algorithm and for the first time describe in detail in Algorithm 4. Intuitively, we are mainly concerned with knowing whether the estimated traffic is routable ($\lambda > 1$), rather than with the accurate value of λ . The *est λ* algorithm is designed following such an intuition: it uses the bounds of λ to decide on whether $\lambda > 1$ is true or not.

The algorithm *est λ* is based on Algorithm 3, with additional steps for calculating the upper λ_{low} and lower λ_{high} bounds of λ . The algorithm starts with calculating them. If the lower bound is $\lambda_{low} > 1$ (or upper bound is $\lambda_{high} < 1$), the algorithm directly concludes that λ is above (or below) 1 (routable vs not routable). The algorithm only runs the FPTAS algorithm (Algorithm 3) when $\lambda_{low} < 1$ and $\lambda_{high} > 1$ since in such a case, it cannot be concluded whether $\lambda > 1$ is true or not and thus more accurate value of λ is required. Appendix B elaborates on how λ_{low} and λ_{high} are calculated.

In summary, compared to directly using the FAS algorithm [20] for routability check (verifying bandwidth constraints), the *est λ* algorithm has the following advantages:

- It avoids unnecessary calculations for the accurate approximation of λ . The *est λ* algorithm correctly tells whether $\lambda \geq 1$ is true or not, which is enough to make a routability decision and guide the optimization of mapping and association.
- When a more accurate approximation of λ is required, the *est λ* algorithm uses our FPTAS algorithm (Algorithm 3), which is faster than the FAS algorithm in dealing with the control flows as shown in Appendix A, which contains details about time complexity analysis and comparison.

With these advantages, evaluation results suggest that our algorithm can achieve **50x** speedup over FAS in the examined large topologies.

IV. SOLVING THE CONTROL PLANE DEPLOYMENT PROBLEM WITH RELIABILITY, BANDWIDTH, DELAY AND BACKLOG CONSIDERATIONS

Compared to Section III, the control traffic routability check problem addressed in this section is more demanding, since in addition to bandwidth constraints it includes delay and backlog constraints too.

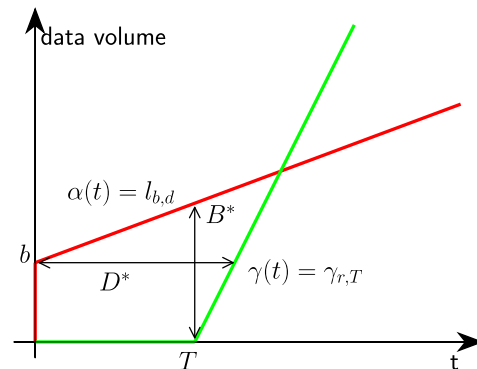


FIGURE 5. Graphical illustration of flow delay bound and backlog bound computation using network calculus concepts. The delay and backlog bounds correspond to the maximum horizontal and vertical deviations between the (aggregated or single flow) arrival $\alpha(t)$ and service $\gamma(t)$ curves, respectively.

To model end-to-end flow delays in a network, we apply Network Calculus (NC) [34]–[37], which is a commonly used approach for analyzing the delay and backlog (buffer space) bounds of flows. We give a brief overview of the NC theory next.

A. NETWORK CALCULUS FUNDAMENTALS

Network calculus [34], [35] is a theory developed for analyzing communication systems. With the models of a flow and the underlying system, three bounds can be calculated with the aid of NC: delay bound, backlog bound, and output flow bound after the flow has passed through the system. Deterministic NC provides deterministic bounds, whereas stochastic NC provides bounds following probabilistic distributions. In this work we consider the former to reduce the computational complexity.

Two key elements in NC theory are arrival curve and service curve. The arrival curve $\alpha(t)$ is defined as the upper bound on the amount of injected data during any time interval. Suppose the total amount of data a flow will send during any time interval $[t_1, t_2]$ is $R(t_2) - R(t_1)$, where $R(t)$ is the cumulative traffic function, which defines the traffic volume coming from the flow within $[0, t]$ time interval. A wide-sense increasing function $\alpha(t)$ is called arrival curve of a flow if for every $t_1, t_2, 0 \leq t_1 \leq t_2$ it satisfies:

$$R(t_2) - R(t_1) \leq \alpha(t_2 - t_1). \quad (18)$$

In practice, the arrival curve $\alpha(t)$ of a flow f is usually modeled with a linear function $l_{b_f, d_f} = d_f t + b_f$, where d_f is the sustainable arrival rate (or demand) and b_f is the burstiness. The interpretation of the linear arrival curve in Fig. 5 is that the flow can send bursts of up to b_f bytes, but its sustainable rate is limited to d_f bytes/s.

The service curve $\gamma(t)$ models a network element (a switch or a channel) and expresses its service capability [38]. The service curve shown in Fig. 5 can be interpreted as the longest time T that a packet of a flow has to wait before being

delivered at a rate of at least r bytes/s. This type of service curve is referred to as a *rate-latency* service curve, $\gamma_{r,T}$.

By modeling a flow and the underlying system with these two curves, the delay and backlog bounds can be calculated. The delay bound corresponds to the maximum horizontal gap between $\alpha(t)$ and $\gamma(t)$, whereas the backlog bound corresponds to the largest vertical gap as shown in Fig. 5. Specifically, the delay bound D^* and backlog bound B^* can be calculated as:

$$D^* = T + b/r, \quad (19)$$

$$B^* = b + Td. \quad (20)$$

Assume a flow traverses two systems with service curves γ_1 and γ_2 , respectively. Then, the equivalent service curve (ESC) of such a concatenated system can be calculated by *min-plus* convolution \otimes between γ_1 and γ_2 :

$$\gamma(t) = (\gamma_1 \otimes \gamma_2)(t) = \inf_{0 \leq s \leq t} \{\gamma_1(t-s) + \gamma_2(s)\}. \quad (21)$$

In particular, if both γ_1 and γ_2 are rate-latency service curves, e.g., $\gamma_1 = \gamma_{r_1, T_1}$ and $\gamma_2 = \gamma_{r_2, T_2}$, then $\gamma = \gamma_1 \otimes \gamma_2 = \gamma_{\min(r_1, r_2), T_1 + T_2}$. We can thus deduce the ESC for a given flow that traverses multiple network elements (such as switches) in a network by applying the concatenation property.

B. ASSUMPTIONS AND REQUIREMENTS

While transmitted along a path, a packet suffers four different kinds of delays: processing, queuing, transmission, and propagation. Propagation delay depends on the link characteristics and the physical distance, and it is assumed to be known. Processing delay depends on the underlying networking hardware, and is usually much smaller than the other delays. We can apply network calculus for calculating the bounds on the queuing and transmission delays. The deterministic end-to-end worst-case delay bound for a flow is calculated as the sum of all the four kinds of delays along the path of a flow.

To apply NC for analyzing control plane flows, we first need to estimate the linear¹ arrival curve $\alpha_f = (b_f, d_f)$ of a flow f and then we need to derive the rate-latency ESC $\gamma_f = (r_f, T_f)$ of the target flow. With these estimates we can derive the delay D_f and buffer B_f bounds of the flow f .

Each node in the network implements some kind of a guaranteed performance service discipline [39], [40] to forward traffic with bounded delay. Suppose for instance that the bandwidth and (propagation plus processing) delay of an output link e are u_e and t_e , whereas the reserved and guaranteed service rate of a flow is r_e , where $r_e < u_e$. In our work we consider the commonly used Weighted Fair Queuing (WFQ) discipline for which the service curve of a flow is given by $\gamma_f^e = (r_e, L_{max}/r_e + L_{max}/u_e)$, where L_{max} denotes the maximum packet size.

¹We assume linear arrival and service curves to reduce computation complexity, see Section IX for further discussion.

C. THE FORMAL PROBLEM

To formulate the control plane deployment problem with network calculus for delay and backlog constraints, we incorporate additional notation to that introduced in Section III-A. Let (u_e, t_e) be the bandwidth capacity and delay of each link $e \in E$. Suppose (s_f, t_f) are the (*source, sink*) of traffic flow f . Let $F = \{f = (s_f, t_f, b_f, d_f)\}$ be the set of the entire control traffic of the deployed infrastructure. We introduce a binary decision variable $\delta^{(K)}, \forall K \in \kappa_f$ to denote whether path K is selected and used for routing flow f . Let variable $X(K)$ denote the reserved guaranteed service rate for flow f along path $K, \forall K \in \kappa$. We assume a flow f can be split and routed on a list of selected paths $\kappa'_f = \{K | \delta^{(K)} = 1, K \in \kappa_f\}$, with each path serving a sub-flow $f^{(K)}, K \in \kappa'_f$. A sub-flow is routed along path K if and only if $\delta^{(K)} = 1$ and $X(K) > 0$. Let $X_f = \{X(K) | K \in \kappa_f\}$ denote the reserved guaranteed service rates on all paths of flow f . Let D_{max} denote the delay bound constraint, and B_{max} the backlog bound constraint.

To calculate the delay D_f and buffer B_f bounds of a flow f , we just need to calculate the delay $D_f^{(K)}$ and backlog $B_f^{(K)}$ bounds of sub-flow $f^{(K)}$, since $D_f = \max\{D_f^{(K)}\}$ and $B_f = \max\{B_f^{(K)}\}$. The burstiness $b_f^{(K)}$ of each sub-flow should be less than or equal to the burstiness of the aggregated flow. Considering the worst case, $b_f^{(K)} = b_f, \forall K \in \kappa_f$. The summation of the arrival rates $d_f^{(K)}$ of all sub-flows $f^{(K)}$ should equal that of the aggregated flow: $\sum_{K \in \kappa'_f} d_f^{(K)} = d_f$.

For each sub-flow $f^{(K)}$, given rate $X(K)$ and path K and the service discipline, we can calculate the rate-latency service curve $\gamma_{f^{(K)}}^e = (X(K), t_e^{(K)})$ at each link $e \in K$. Suppose path K traverses several links. Then its ESC is: $\gamma_{f^{(K)}} = (X(K), t_{s_f}^{(K)}) = \gamma_{f^{(K)}}^{e1} \otimes \gamma_{f^{(K)}}^{e2} \dots \gamma_{f^{(K)}}^{ek}, e1 \dots ek \in K$ by the concatenation property. Here, $X(K)$ is the service rate and t_{s_f} can be understood as the service latency introduced by all the network elements along the entire path K . The delay and backlog bounds of each sub-flow are: $D_f^{(K)} = t_{s_f}^{(K)} + b_f^{(K)}/X(K) + \sum_{e \in K} t_e$ and $B_f^{(K)} = b_f^{(K)} + t_{s_f}^{(K)} d_f^{(K)}$. The delay bounded deployment problem requires for all non-zero sub-flows $D_f^{(K)} < D_{max}, B_f^{(K)} < B_{max}, \forall K \in \kappa_f, \forall f \in F$.

Now, the problem can be formulated as follows:

$$\begin{aligned} & \text{maximize } 0 \\ & \text{s.t.: } \sum_{i \in C} a_{ij} = 1, \quad \forall j \in N \end{aligned} \quad (22)$$

$$\sum_{i \in M} y_i \geq 1 \quad (23)$$

$$R(G, j, C) \geq \beta, \quad \forall j \in N \quad (24)$$

$$y_i, a_{ij} \in \{0, 1\} \quad (25)$$

$$\sum_{K \in \kappa_f} X(K) \delta^{(K)} \geq d_f, \quad \forall f \in F \quad (26)$$

$$\sum_{K: e \in K} X(K) \delta^{(K)} \leq u_e, \quad \forall e \in E \quad (27)$$

$$X(K) \geq 0, \quad \forall K \in \kappa \quad (28)$$

$$\sum_{K \in \kappa_f} d_f^{(K)} = d_f, \quad \forall f \in F \quad (29)$$

$$\delta^{(K)}(ts_f^{(K)}X(K) + b_f^{(K)}) \leq (D_{max} - \sum_{e \in K} t_e)X(K), \quad \forall K \in \kappa_f, \quad \forall f \in F \quad (30)$$

$$\delta^{(K)}(b_f^{(K)} + ts_f^{(K)}d_f^{(K)}) \leq B_{max}, \quad \forall K \in \kappa_f, \quad \forall f \in F \quad (31)$$

$$d_f^{(K)} \leq X(K)\delta^{(K)}, \quad \forall K \in \kappa_f, \quad \forall f \in F \quad (32)$$

$$d_f^{(K)} \geq 0, \quad \forall K \in \kappa_f, \quad \forall f \in F \quad (33)$$

$$\delta^{(K)} \in \{0, 1\}, \quad \forall K \in \kappa \quad (34)$$

We can still apply the optimization process proposed in Section II to solve this new control plane deployment problem. We can also reuse the SA algorithms in sections III-B and III-C for the mapping and association steps. However, to estimate the control traffic (see Section III-D), we need to additionally estimate the flow burstiness. Moreover, for the routability check step, we need a new workflow, which takes constraints on bandwidth as well as delay and backlog into consideration. We name such an extended routability check problem a delay and backlog check problem and we formulate it below.

D. TRAFFIC BURSTINESS ESTIMATION

We assume the burstiness b_f of a flow (Fig. 5) is proportional to its rate (d_f) as in [41], [42]. Therefore, the burstiness of a flow f is estimated as $b_r d_f$, where b_r is a burstiness ratio.

E. THE DELAY AND BACKLOG VERIFICATION PROBLEM FORMULATION

With proper transformation, the delay and backlog check problem is formulated as an optimization problem:

$$\text{maximize } \lambda \quad (35)$$

$$\text{s.t.: } \sum_{K: e \in K} X(K)\delta^{(K)} \leq u_e, \quad \forall e \in E \quad (36)$$

$$X(K) \geq 0, \quad \forall K \in \kappa \quad (37)$$

$$\sum_{K \in \kappa_f} d_f^{(K)} \geq \lambda d_f, \quad \forall f \in F \quad (38)$$

$$\delta^{(K)}(ts_f^{(K)}X(K) + b_f^{(K)}) \leq (D_{max} - \sum_{e \in K} t_e)X(K), \quad \forall K \in \kappa_f, \quad \forall f \in F \quad (39)$$

$$\delta^{(K)}(b_f^{(K)} + ts_f^{(K)}d_f^{(K)}) \leq B_{max}, \quad \forall K \in \kappa_f, \quad \forall f \in F \quad (40)$$

$$d_f^{(K)} \leq X(K)\delta^{(K)}, \quad \forall K \in \kappa_f, \quad \forall f \in F \quad (41)$$

$$d_f^{(K)} \geq 0, \quad \forall K \in \kappa_f, \quad \forall f \in F \quad (42)$$

$$\delta^{(K)} \in \{0, 1\}, \quad \forall K \in \kappa \quad (43)$$

However, the above formulation contains quadratic terms such as the one between the indicator variable $\delta^{(K)}$ and $X(K)$ in (39) (40), which is not efficient for optimization problem

solvers such as CPLEX to solve. Therefore, we use the Big-M method to obtain an equivalent formulation, which eliminates the quadratic terms, see (45)-(53).

$$\text{maximize } \lambda \quad (44)$$

$$\text{s.t.: } \sum_{K: e \in K} X(K) \leq u_e, \quad \forall e \in E \quad (45)$$

$$X(K) \geq 0, \quad \forall K \in \kappa \quad (46)$$

$$\sum_{K \in \kappa_f} d_f^{(K)} \geq \lambda d_f, \quad \forall f \in F \quad (47)$$

$$ts_f^{(K)}X(K) + b_f^{(K)}\delta^{(K)} \leq (D_{max} - \sum_{e \in K} t_e)X(K), \quad \forall K \in \kappa_f, \quad \forall f \in F \quad (48)$$

$$\delta^{(K)} \in \{0, 1\}, \quad \forall K \in \kappa \quad (49)$$

$$b_f^{(K)}\delta^{(K)} + ts_f^{(K)}d_f^{(K)} \leq B_{max}, \quad \forall K \in \kappa_f, \quad \forall f \in F \quad (49)$$

$$X(K) \leq M\delta^{(K)} \quad (50)$$

$$d_f^{(K)} \leq X(K), \quad \forall K \in \kappa_f, \quad \forall f \in F \quad (51)$$

$$d_f^{(K)} \geq 0, \quad \forall K \in \kappa_f, \quad \forall f \in F \quad (52)$$

$$\delta^{(K)} \in \{0, 1\}, \quad \forall K \in \kappa \quad (53)$$

F. IMPLEMENTATION OF THE EXTENDED ROUTABILITY CHECK STEP

We propose a two-phase workflow to perform the extended routability check, which includes:

- **Bandwidth verification:** tests whether all flows can be routed without overloading any link relative to specified bandwidth constraints;
- **Delay and backlog verification:** tests whether the estimated flows can be routed under given flow delay and buffer space requirements.

The reason why we use such a two-phase workflow is two-fold. First, if bandwidth verification fails (some links are overloaded), there is no need to verify the delay and backlog bounds, since the delays of certain flows, in theory, can go to infinity. Second, bandwidth verification is usually less time consuming (more than 10x less) than the delay and backlog verification. Thus, overall it is much more time-efficient to first check the bandwidth bound and use its outcome to decide whether to continue with checking the remaining bounds.

In the following, we show our implementation of the algorithm for performing the delay and backlog verification. Note that in the delay and backlog bound verification problem, defined in (45) – (53), the $ts_f^{(K)}$ in constraints (48) and (49) depends on the choice of the service discipline, which we have assumed to be the commonly used WFQ. According to [39], [40], $ts_f^{(K)} = L_{max}H/X(K) + \sum_{e \in K} L_{max}/u_e$, where H is the number of path hops. By substituting $ts_f^{(K)}$ in (48) (49), we obtain:

$$\begin{aligned} & (\sum_{e \in K} L_{max}/u_e)X(K) + (L_{max}H + b_f^{(K)})\delta^{(K)} \\ & \leq (D_{max} - \sum_{e \in K} t_e)X(K), \quad \forall K \in \kappa_f, \quad \forall f \in F \quad (54) \end{aligned}$$

$$\begin{aligned}
& (b_f^{(K)}X(K) + L_{max}Hd_f^{(K)})\delta^{(K)} \\
& + (\sum_{e \in K} L_{max}/u_e)d_f^{(K)}X(K) \\
& \leq B_{max}X(K), \forall K \in \kappa_f, \quad \forall f \in F \quad (55)
\end{aligned}$$

Because of the quadratic term in (55), the delay and backlog bound verification problem becomes a non-convex Mixed Integer Quadratically Constrained Programming problem (MIQCP). Such non-convex MIQCP problems are hard to solve with existing optimizers, e.g., CPLEX. Thus, we replace (55) with a linear approximation (56):

$$\begin{aligned}
& (b_f^{(K)} + L_{max} * H)\delta^{(K)} + (\sum_{e \in K} L_{max}/u_e)d_f^{(K)} \\
& \leq B_{max}, \forall K \in \kappa_f, \quad \forall f \in F \quad (56)
\end{aligned}$$

It is easy to see that if constraint (56) is satisfied, (55) must be satisfied too, but the reverse is not true. With the replacement, the problem becomes an MILP problem.

We propose a heuristic algorithm based on column generation intuition, which we call CGH, to deal with the formulated optimization problem. Column generation is a well-known technique for solving large-scale linear programming problems. The key idea of column generation is to split the original problem into two problems: a master problem and a subproblem. The master problem is the original problem but with only a subset of the variables being considered. The subproblem is created to find a new variable that could potentially improve the objective function of the master problem. Usually, the dual of the original problem is used in the subproblem with the purpose of identifying new variables. The kernel of the column generation method defines such an iterative process: 1) solving the master problem with a subset of the variables and obtaining the values of the duals, 2) considering the subproblem with the dual values and finding the potential new variable, and 3) repeating step 1) with the new variables that have been added to the subset. The whole process is repeated until no new variables can be found.

Although our problem is not LP, we can still borrow the column generation kernel idea to design a heuristic algorithm. Intuitively, if we could ignore the constraints on delay and backlog, the optimization problem would be reduced to a maximum concurrent flow problem as formulated in (9)–(12). With the column generation method, the master problem is the maximum concurrent flow problem but with a subset of the paths $\kappa^* \subseteq \kappa$. According to [16], [43], the corresponding subproblem can use (14) (in Section III-E) to identify the new potential paths: the potential paths are the ones that do not obey (14).

For our optimization problem, similar to [16], [43], we define the subproblem that is also based on (14) (in Section III-E) for identifying new potential path variables. However, because of delay and backlogs constraints ((54) and (56)), the new potential path variables need to additionally obey $\sum_{e \in K_f} L_{max}/u_e < D_{max} - \sum_{e \in K} t_e$ and $b_f^{(K_f)} + L_{max}H < B_{max}$. Otherwise, constraints (54) and (56)

Algorithm 5 Delay and Backlog Verification Algorithm Based on Column Generation Intuition

Initialization

- 1: Using $L_{max}/u_e + t_e$ as edge weights, compute $\kappa^* = \{K_f | f \in F\}$, where K_f is the shortest path for flow f under the edge weights.
- 2: $iter = 0$
- 3: Relax the binary variable $\delta^{(K)} \in \{0, 1\}$ to $\delta^{(K)} \in [0, 1]$
- 4: Set $\lambda_{old} = 0$
- 5: **while** ($iter < MAXITERATIONS$ and $Neg_f \neq \phi$) **do**
- 6: Solve the optimization problem with the set of paths κ^* , get λ
- 7: **if** $\lambda - \lambda_{old} \leq 0$ **then**
- 8: Break
- 9: **end if**
- 10: $\lambda_{old} = \lambda$
- 11: Calculate the duals $DE = \{l_e | e \in E\}$ of constraint (45).
- 12: Calculate the duals $DF = \{z_f | f \in F\}$ of constraint (47)
- 13: With l_e as the edge weights, compute $P = \{P_f, dist_f | \forall f \in F\}$, where P_f is the shortest path for flow f , and $dist_f$ is the distance of the path.
- 14: $Neg_f = \{\}$
- 15: **for** f in F **do**
- 16: $res = dist_f - z_f$
- 17: **if** $res < 0$ **then**
- 18: **if** $\sum_{e \in K_f} L_{max}/u_e < D_{max} - \sum_{e \in K} t_e$ and $b_f^{(K_f)} + L_{max} * H < B_{max}$ **then**
- 19: Add f to Neg_f
- 20: **end if**
- 21: **end if**
- 22: **end for**
- 23: Select P_f^* that $dist_f = \min(\{dist_f | \forall f \in Neg_f\})$
- 24: Add P_f^* to κ^*
- 25: **end while**
- 26: Change $\delta^{(K)} \in [0, 1]$ to $\delta^{(K)} \in \{0, 1\}$
- 27: Solve the optimization problem with the set of paths κ^* , get λ

Output: λ

will certainly be violated if they are chosen for routing flows ($\delta^{(K)} = 1$). The design of our algorithm is illustrated by Algorithm 5 and described in more detail below.

First, we relax our delay and backlog verification problem to LP by relaxing the binary variable $\delta^{(K)}$ to $[0, 1]$. We initiate the master problem with a set of paths $\kappa^* \subseteq \kappa$ (lines 1–3), Algorithm 5. Then, we begin a repetitive process for adding new paths: 1) solve the master problem² and obtain values of the duals of the constraints imposed on edges and flows (lines 11–12); 2) identify the new paths K_f as the paths that

²In the algorithm implementation, we can use optimization solvers, such as CPLEX, for solving the master problem in line (6) and line (27) of Algorithm 5.

violate dual constraint (14 in Section III-E) (lines 13–17), while satisfying requirements due to delay and backlog constraints (line 18); 3) add the new potential path to the subset κ^* and repeat the process again (line 19–24). The process is repeated until no new paths can be found, or the maximum number of iterations is reached, or the object value does not improve with the newly added path (line 7–8). In the end, we restrict the $\delta^{(K)}$ variable back to integer values 0, 1, and solve the delay and backlog verification master problem with the set of the paths κ^* found by the iterative process.

Naturally, we can use MILP solvers, such as CPLEX, to directly solve the delay and backlog verification problem. We call such an approach a CPLEX-direct method. However, using a CPLEX-direct method is very time consuming, since it needs to consider all the possible flow paths κ , the number of which increases exponentially with the size of the considered topology. In comparison, our CGH algorithm uses much fewer paths. The CGH algorithm is initialized with only one path for each source-destination pair (only $|N|(|N| - 1)$ paths after initialization). The algorithm adds one new path in each iteration (lines 5–24). Usually, even for large topologies, the iteration process terminates within a few hundred rounds. Therefore, the number of paths being considered $|\kappa^*|$ can be thousands of times less when compared to the number of paths $|\kappa|$ used by the CPLEX-direct method, resulting in much shorter running time. Note that our algorithm is a heuristic one, which means that there is no guarantee that it can find the optimal or approximate the optimal result. However, the evaluation results suggest that in practice it performs remarkably well. In particular, it can achieve a **500x** running time reduction over the CPLEX-direct method, while yielding nearly as optimal results as the CPLEX-direct method.

V. USE CASES

To demonstrate the optimization capabilities of the feasibility solver, we discuss two of its use cases below. The performance of the solver under different conditions is evaluated and contrasted to existing solutions in the next sections.

Given certain constraints to be satisfied by the optimization process, one of these constraints can be optimized, given that the remaining ones are hold fixed. In particular, assuming a certain bandwidth constraint, the optimization objective can be to maximize the reliability R_{min} , and vice versa, given a minimum reliability threshold constraint β , the bandwidth u that can guarantee this threshold can be minimized. We apply the binary search method³ [45] to find the optimal value (R_{min} in the former and minimum u in the latter case).

The optimization process is applied to the real-world Internetmci topology (a part of the publicly accessible Internet topologies Zoo (ITZ) [46]). Appendix C lists the topological details of the networks used for experimentation and evaluation. We assume in-band control and the set of nodes eligible

³The binary search method is applicable to a single objective optimization. In multi-objective optimization, hierarchical optimization or trade-off methods [44] can be applied.

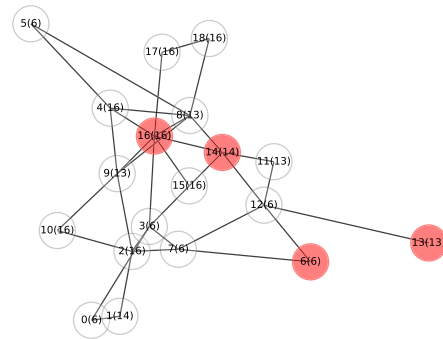


FIGURE 6. The corresponding deployment solution of controllers (red) and the association plan (denoted as Node ID/Associated Controller ID), when the minimum required reserved bandwidth is 35.25 Mbits/s per link, given a reliability threshold $\beta = 0.99999$ and requirement $R_{min} > \beta$, $\lambda = 1.05$.

for hosting controller instances is the entire set of nodes, $M = N$. Each node hosts an aggregator with a 500 requests/s rate [47]. The operational probability of each node, link and controller is set to 0.9999 considering WAN links with four nines of average reliability [25], [48].

A. RELIABILITY AND BANDWIDTH CONSTRAINTS (SCENARIO 1)

We first use the feasibility solver to find bandwidth that satisfies a given reliability constraint β . Then, to find the minimum such bandwidth, we apply the binary search method. The results discussed below are obtained when the optimization process implements SAM and SAA for mapping and association. Assuming equal bandwidth allocation on each link, such that $u_e = u, \forall e \in E$, a minimum bandwidth of 35.25 Mbits/s is needed to ensure $R_{min} > \beta = 0.99999$, see Fig. 6 for an example of a deployment solution.

To exemplify R_{min} maximization, the bandwidth constraint (that is, bandwidth reserved for control traffic) u_e of each link e is set to 24 Mbits/s (considering the modest SDN control plane traffic case with the OpenFlow protocol). The maximum R_{min} achieved under these particular conditions is 0.99989.

Recall that λ acquired by the optimization process under certain deployment and link bandwidth settings can be viewed as a safety margin for tolerating the variations in flow demand. For the deployment plan solution shown in Fig. 6, $\lambda = 1.05$ translates into tolerated flow demand increase of at least 5% (on any flow) when the bandwidth per link is 35.25 Mbits/s. One possible way to attain larger λ is to scale up the bandwidth accordingly. In the first case, for instance, to ensure a $\lambda' = 1.2$ (20 % safety margin) while still satisfying $\beta = 0.99999$, the smallest bandwidth needed becomes $35.25 \frac{\lambda'}{\lambda} = 40.29$ Mbits/s. The deployment solution remains the same as in Fig. 6.

B. RELIABILITY, BANDWIDTH, DELAY, AND BACKLOG CONSTRAINTS (SCENARIO 2)

Backlog and delay are added to the constraints considered previously. Given a reliability $\beta = 0.9999$, bandwidth $u_e = 400$ Mbits/s, and backlog $B_{max} = 80$ Mbits constraints,

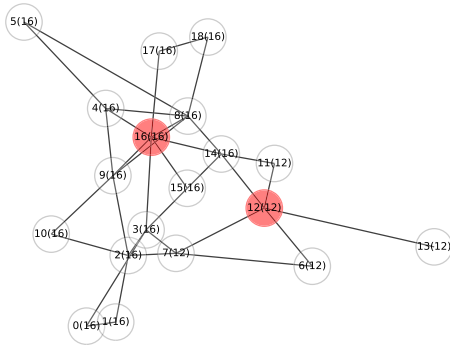


FIGURE 7. The corresponding deployment solution of controller instances (red) and the association plan (denoted as Node ID/Associated Controller ID). The highest reliability achieved by our method is 0.999899, when the bandwidth constraint is $u_e = 24$ Mbits/s. $\lambda = 1.04$ is attained.

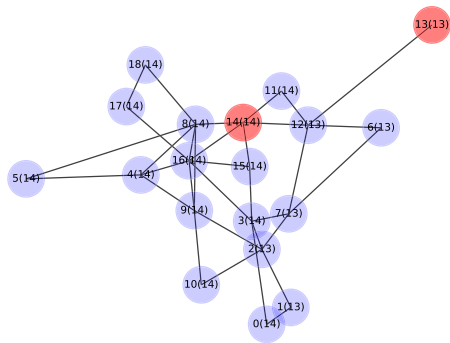


FIGURE 8. The corresponding deployment plan of controller instances (red) and the association plan (denoted as Node ID/Associated Controller ID) that can ensure all flow delays are bounded by 180 ms, when the reliability constraint is $\beta = 0.9999$, bandwidth constraint is $u_e = 400$ Mbits/s and the backlog constraint is $B_{max} = 80$ Mbits.

our approach outputs a controller deployment solution that ensures worst case flow delay of 180 ms. The deployment plan is shown in Fig. 8. Similarly, if the reliability, delay, and backlog constraints are fixed, the bandwidth requirement can be optimized. Under reliability $\beta = 0.9999$, delay $D_{max} = 100$ ms and backlog $B_{max} = 80$ Mbits constraints, the optimization solver reports a minimum bandwidth of 600 Mbits/s needed to satisfy all these constraints.

VI. EVALUATIONS: SCENARIO 1

The goal of this and the following sections is to reveal the capabilities and shortcomings of the devised optimization process by studying parameters such as optimization time (under different implementations), reliability, bandwidth utilization as well as the tradeoff between these optimization objectives.

The choice of values for the parameters used in the experiments is dictated by a simple distributed control service, which manages only flow-tables in OpenFlow switches. The aggregator request rate varies uniformly within $[100, 900]reqs/s$, with an average 500 req/s (in line with OpenFlow traffic characteristics [47]). The operational probability of links and nodes is randomly drawn from a Weibull

TABLE 1. Different implementations of the mapping and association steps used in the evaluations.

Abbreviation	Explanation
EE	Exhaustive search mapping (ESM) Exhaustive association (ESA)
AA	Simulated annealing mapping (SAM) Simulated annealing association (SAA)
FS	Heuristic FTCP mapping [25] Closest aggregator-controller association (CAA)

distribution with parameters 0.9999 and 40000, considering the long tails in the downtime distribution of WAN links with four nines of mean reliability [25], [48]. We plot the failure probability $(1 - R_{min})$ instead of reliability as it is more effective for plotting in log-scale.

We implemented the deployment optimization process in Python. We run the tests on a server equipped with AMD Opteron(tm) processor (8 cores, 3.1GHz) and 128 GB memory. The entire optimization process runs on a single core.

A. MAPPING AND ASSOCIATION IMPLEMENTATIONS

To validate the optimization solver we compare different implementations (summarized in Table 1) of the mapping and association steps (while holding the remaining steps fixed). FTCP (fault tolerant controller placement) algorithm [25] is a heuristic mapping algorithm that aims at placing a minimum number of controllers in a heuristic way for guaranteeing certain reliability.

Four small, three medium and five large topologies [46] are used as test scenarios (see Appendix C for topological details). In all of them the link bandwidth $u_e = u$ varies within $[\mu/2, 3\mu/2]$ randomly drawn from a truncated normal distribution with mean μ and standard deviation $\sigma = 4$. Considering the traffic characteristics of the OpenFlow protocol [47] and the traffic estimation model in Section III-D, a control flow typically ranges from a few Kbits/s to a few Mbits/s, depending on the request rates. Thereby, we set μ to $[8, 24, 48]$ Mbits/s for small, medium and large topologies, respectively. These rates are sufficient for satisfying at least 3-nine reliability, but not for yielding trivial solutions. All the reported results are based on 100 repetitions.

In Fig. 9 the performance of AA and FS for small topologies is shown as a ratio relative to the baseline implementation EE. For medium and large topologies, we only plot the performance ratio between FS and AA, as EE is too slow for obtaining any result. In Fig. 9a the performance is measured in terms of the observed failure probability $(1 - R_{min})$, whereas in Fig. 9b it is in terms of optimization time.

In summary, the results in Fig. 9 provide evidence that the outlined optimization process is capable of providing a tunable control plane management solution that is close to optimal. The choice of the particular method used for mapping and association is a trade-off between the ability to produce close to optimal solutions for different topology

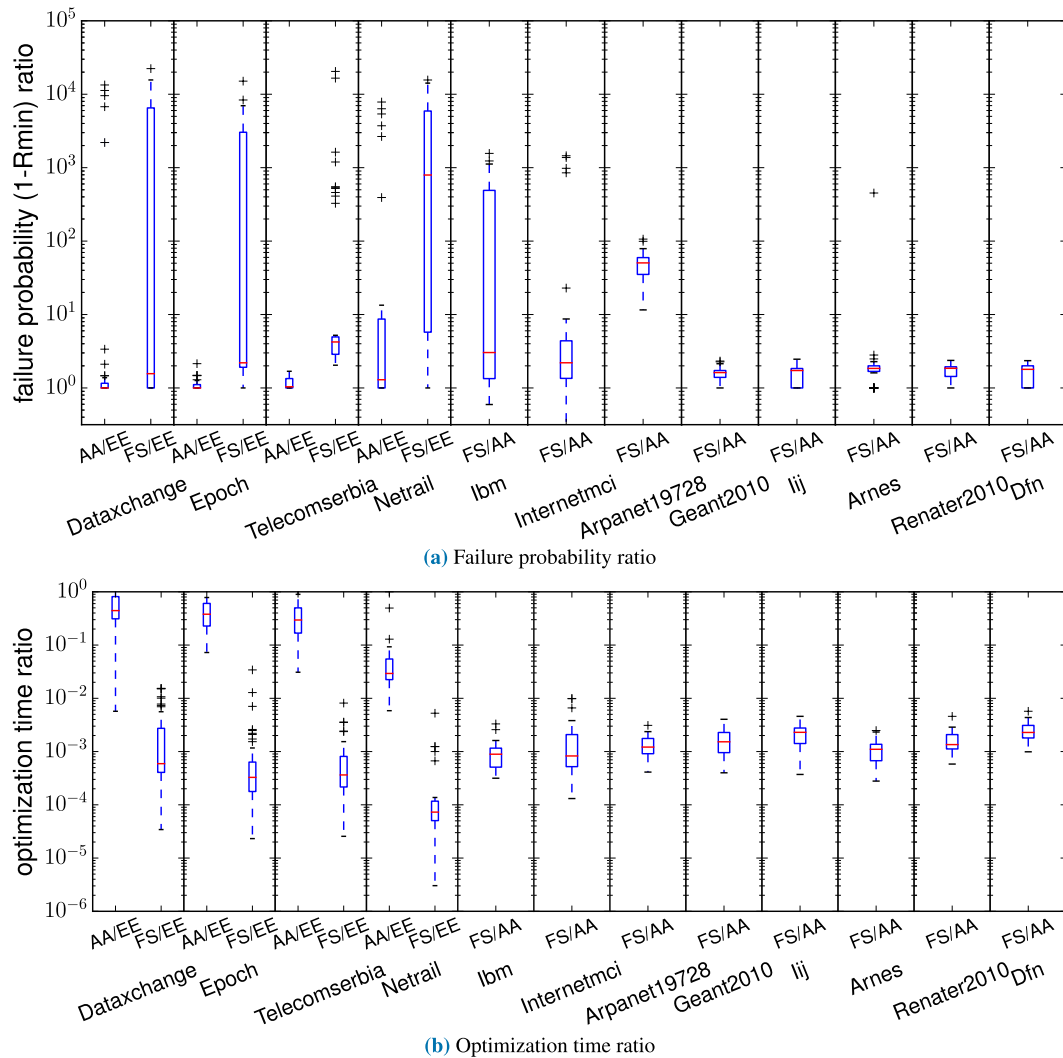


FIGURE 9. In (a) failure probability ratio is studied under different implementations, topologies and variable link bandwidth; (b) optimization time ratio for different topologies as well as implementations is examined. The network topologies vary from small to large (from left to right). AA offers better performance (in terms of lower failure probability under the same link bandwidth), whereas FS is faster.

sizes (Appendix C) and optimization time spent to find them. Specifically, AA offers better performance (in terms of lower failure probability under the same link bandwidth), whereas FS is faster. The same conclusions hold when the optimized parameter is the bandwidth (not depicted).

B. BANDWIDTH VERIFICATION IMPLEMENTATIONS (SCENARIO 1)

We compare two different bandwidth verification implementations, namely the FAS algorithm designed by Karakostas [20] and the $est\lambda$ algorithm designed by us.

Fig 10 shows the total running time when AA is used for mapping and association. The $est\lambda$ algorithm can achieve **50x** running time reduction relative to the FAS algorithm—a reduction *from days to minutes*—as is the case for the “BtEurope” topology. Fig. 11 shows results from the same

experiments⁴ but when, instead of AA, FS is used for mapping and association. The same conclusion—using $est\lambda$ can result in much faster optimization (from minutes to seconds in examined cases) than using the FAS algorithm—is reached.

C. LINK BANDWIDTH AND RELIABILITY SCALING TEST

We systematically quantify the impact of an increased link bandwidth constraint on the achieved maximum R_{min} . Fig. 12 illustrates this effect for the “Internetmci” topology. When scaling up the link bandwidth, the failure probability decreases (equivalently, the reliability R_{min} increases). This phenomenon can intuitively be explained as follows. An increase in bandwidth translates into more control traffic

⁴In addition to the topologies evaluated with the AA implementation, we consider additional network topologies for complementary results (the FS algorithm runs much faster than AA).

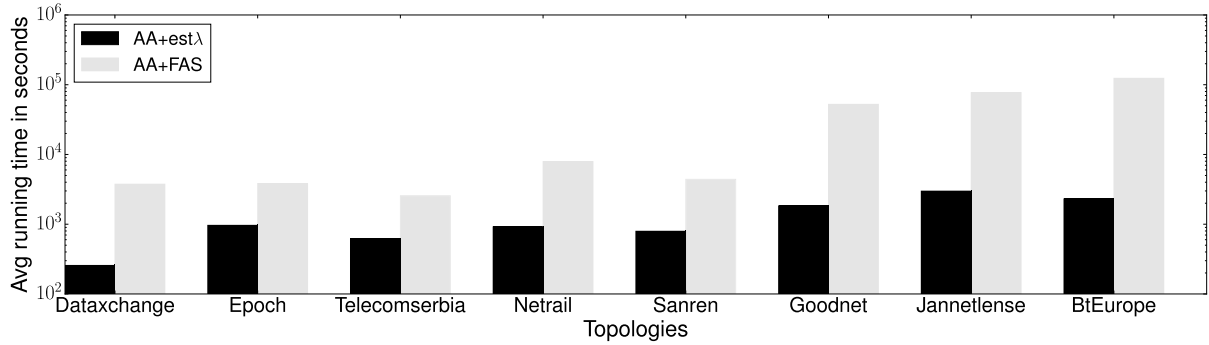


FIGURE 10. The optimization time with different implementations of the bandwidth verification when AA is used for mapping and association. The network topologies depicted on the x-axis are arranged in increasing size order (Appendix C). Clearly, our $est\lambda$ algorithm offers considerable reduction in the running time under all topologies. For larger, medium-sized networks, such as Jannetlense and BtEurope, the running time is reduced from hours to minutes.

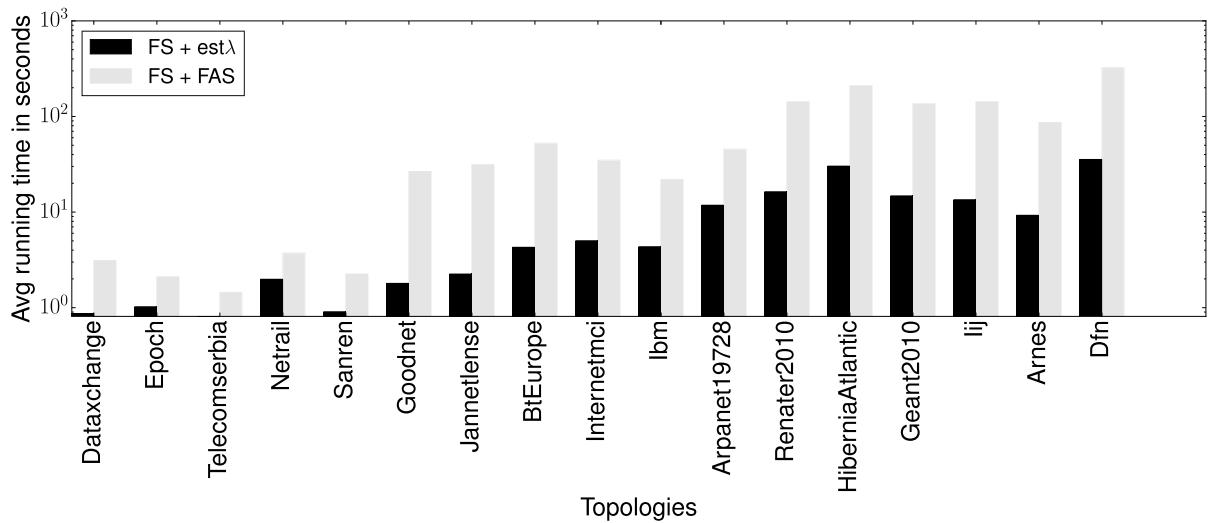


FIGURE 11. The optimization time with different implementations of the bandwidth verification when FS is used for mapping and association. The reduction in running time offered by our $est\lambda$ algorithm is from minutes to seconds.

that can be tolerated, and therefore into a larger number of controllers that can be deployed. A larger number of controller instances increases the probability of aggregators being connected to an operational controller and therefore to enhanced reliability. Another important observation, however, is that the failure probability decrease does not scale linearly with bandwidth increase. A good balance between bandwidth and reliability for the specific scenario studied in Fig. 12 is attained at around 40Mbit/s. Increasing the bandwidth beyond this point leads to insignificant improvement in reliability at the cost of increased bandwidth. In other words, when there are already many controllers in a network and the attained reliability is very high (e.g., 7 nines and above), deploying even more controllers yields only a negligible gains.

Similar reasoning applies when we quantify the impact of increased reliability threshold β , which is a constraint on R_{min} , on the required minimum bandwidth. Note that $(1-\beta)$ is the maximum constraint on the failure probability $(1 - R_{min})$. Naturally, as β increases, more bandwidth is required to

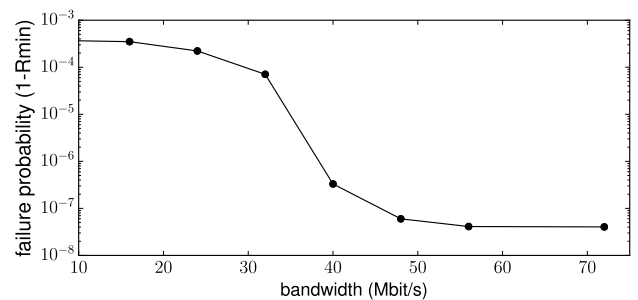


FIGURE 12. Failure probability versus link bandwidth. The plot can be used to determine the optimal trade-off between required reliability and associated bandwidth demands.

guarantee a failure probability below $(1 - \beta)$. Likewise, Fig. 13 shows that when β is increased from 0.9995 to 0.9999, the required bandwidth increases dramatically.

In short, the proposed optimization process can be used by service providers and operators as a practical tool for quantifying the trade-off between bandwidth and reliability gains,

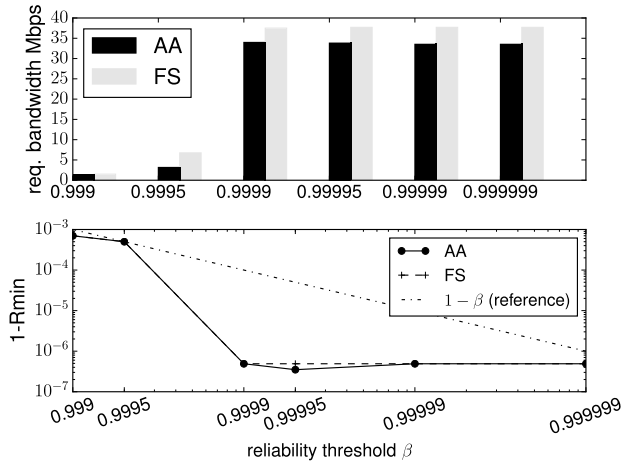


FIGURE 13. (upper) Achieved link bandwidth and (lower) failure probability relative to varying constraints on the minimum reliability. The optimization process always guarantees a failure probability equal to or well below the requirement (the reference line).

TABLE 2. Results of SAS based on 100 runs with $\beta = 0.99999$ and large topology Renater2010.

Reserved BW.	No. Congestions	BUR (Median, Worst case)
50 Mbits	100	(0.0%, 0.0%)
100 Mbits	31	(7.6%, 20.1%)
150 Mbits	0	(36.6%, 50.1%)

enabling development of flexible and fine-tuned controller deployment policies.

D. MEASURING THE IMPACT OF THE ROUTABILITY VERIFICATION STEP

Earlier works did not fully address the control plane deployment problem. In particular, the routability verification is typically ignored to the best of our knowledge. Thereby, it does not seem viable to perform quantitative comprehensive comparisons. Instead, we design an experiment to show the effect of incorporating routability check into the control plane deployment problem.

We use FS (the FTCP method [25] for mapping and CAA for association, see Table 1) i) integrated into the full optimization process (see Fig. 4) and ii) as a stand-alone solution, without traffic estimation and routability check. We abbreviate the latter SAS (Stand-Alone-Solution).

Given a minimum reliability constraint β , our optimization process offers the capability to estimate the control traffic and decide on the bandwidth required to avoid congestion. In contrast, the SAS approach is limited to ensuring a placement satisfying β , but no guidelines on the amount of bandwidth needed to route the control traffic. This limitation of the SAS method yields the dilemma: how to manually reserve sufficient but not excessive bandwidth. We exemplify this dilemma in Table 2 by showing the bandwidth underutilization ratio $BUR = \max(0.0, 1 - BW_{FS}/BW_{SAS})$ for fixed bandwidth reservations with SAS (BW_{SAS}) relative to the estimates produced by FS (BW_{FS}). Note that for the third case

in Table 2, the bandwidth consumption can be largely reduced by at least 36.6% in half of the observed cases without the risk of introducing any congestion when applying the outlined optimization process. Naturally, the average running time of FS is longer (around 8 times for the considered topology) than SAS due to the routability verification step.

VII. EVALUATION: SCENARIO 2

A. DELAY AND BACKLOG VERIFICATION METHODS

We compare our CGH algorithm⁵ with the direct-CPLEX method (that is, with CPLEX when CPLEX is used to directly solve the delay and backlog verification problem) in terms of performance and running time. The time is especially relevant in large-scale networks and when network changes occur frequently since a fast solution in such scenarios can offer the flexibility to respond to variations in conditions accordingly.

We first evaluate the standalone performance of the two different methods when solving randomly generated delay and backlog verification problems. Then we integrate the delay and backlog verification methods into the proposed optimization process, and illustrate the overall optimization time by using different controller deployment implementations.

For the experimental settings, the link propagation delay is extracted from the topology based on physical distance between two nodes. The processing delays are randomly sampled from $[0.1, 1]$ ms [49].

1) PERFORMANCE OF CGH AND CPLEX METHODS AS STANDALONE SOLUTIONS FOR DELAY AND BACKLOG VERIFICATION

We set $L_{max} = 1542$ bytes, which is the maximum Ethernet frame size [50]. In order to comprehensively study the performance of the two methods, two traffic patterns are utilized for evaluation, which are *uniform traffic pattern* and *control traffic pattern*. We randomly generate several sets of bursty flows for each traffic pattern.

For the uniform traffic pattern, the source and destination of a flow are uniformly distributed in the network. The number of flows ranges within $[20, 80]$, whereas the flow rate ranges within $[800, 7200]$ Kbits/s and the burstiness within $[1600, 48000]$ Kbits following a truncated multivariate normal distributions, with means ($\mu_{rate} = 2400, \mu_{burst} = 24000$) and covariance ($\sigma_{rate} = 300, \sigma_{burst} = 3000$) and a correlation parameter $\rho = 0.83$ [42]. The bandwidth of the links is randomly distributed within $[80, 8000]$ Mbits/s. The delay constraints are swept within $[0.5, 1.0]$ s and the backlog constraints within $[120, 240]$ Mbits.

The control traffic pattern emulates the phenomenon that control plane flows tend to concentrate at controllers. The control traffic flows are generated as follow: first, we randomly select $|C| \in [0.1|N|, 0.5|N|]$ nodes as controllers. Second, we sample a probability vector

⁵As mentioned in Section V, the master problem in line (6) and (27) in our CGH algorithm (Algorithm 5) is also solved by using CPLEX.

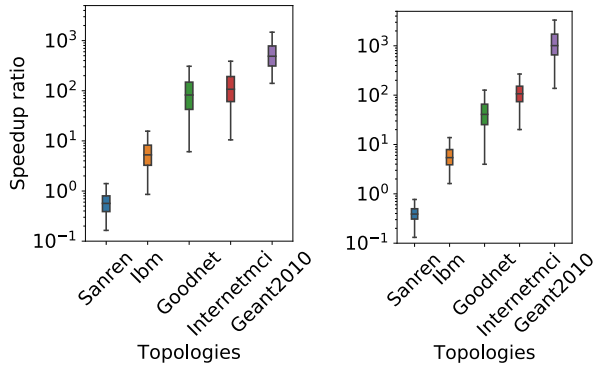


FIGURE 14. Speed-ups by using column generation heuristic algorithm, tested under (left) uniform traffic pattern and (right) control traffic pattern. Compared with the direct-CPLEX method, the CGH algorithm has a speed-up of 500x under uniform traffic pattern and 1000x under control traffic pattern. The size of the examined topologies (Appendix C) ranges from small to large (from left to right).

$p_c = [p_1, p_2, \dots, p_{|C|}]$ from a Dirichlet distribution and assign to each controller a probability value. Then, each aggregator randomly selects one of the controllers for association according to these probabilities p_c . The aggregator request rate varies uniformly within $[100, 900]$ req/s, with an average of 500req/s. The burstiness ratio b_r is randomly sampled from $[0.1, 5.0]$. Then, with the traffic estimation method proposed in Section III-D and Section IV-D, we estimate the rate (demand, d_f) and burstiness (b_f) of a flow f . The bandwidth of the links is randomly distributed within $[8, 25600]$ Mbits/s. The delay constraints are swept within $[0.1, 1.0]$ s and the backlog constraints within $[120, 240]$ Mbits.

The results for each topology (Appendix C) and traffic pattern are obtained over 1000 runs. The cut-off time for the direct CPLEX was set to 10min since if a single routability check runs for longer time, it is infeasible to use in practice. The CPLEX is configured with two threads, which run on two processor cores.⁶

The speed-up ratio of CGH over CPLEX is show in Fig. 14. The approximation ratio (Fig. 15) is defined as $(\lambda_{CGH} + eps)/(\lambda_{CPLEX} + eps)$, where λ_{CGH} and λ_{CPLEX} are the maximization objective value (36) of our CGH and direct-CPLEX, respectively. We set eps to a very small value ($1e - 4$) to avoid numerical issues such as division by zero. The results from the experiments show that overall CGH runs much faster than direct-CPLEX. The larger the topology, the higher the speed-up ratio. In particular, for the largest network topology examined in these set of simulations ('Geant2010'), with uniform traffic pattern and control traffic pattern, we observed a 500x and 1000x reduction in time with our CGH algorithm, respectively. Importantly, this significant reduction in running time is not at the cost of deterioration in optimization performance. As the box plot in Fig. 15 suggests, the two

⁶We observed that parallel running of CPLEX with more than two processor cores (and threads) only leads to insignificant speed-ups. We also observed that an increase in the number of threads will significantly increase the memory consumption, which can lead to memory overflow in certain test cases.

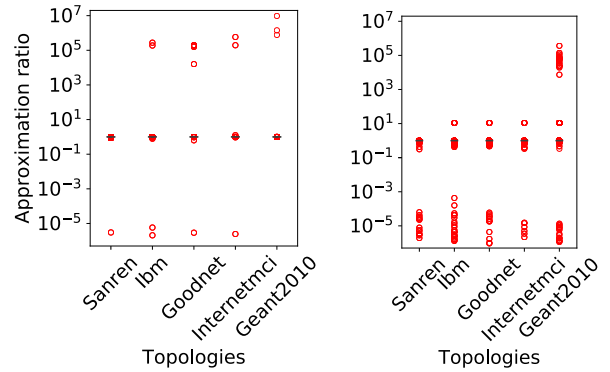


FIGURE 15. Approximation ratio of CGH algorithm to direct-CPLEX method, tested under (left) uniform traffic pattern and (right) control traffic pattern. The box plots in the figure show that minimum, first quartile, median, third quartile, and maximum under both traffic patterns are all 1.0—the CGH algorithm can effectively reach similar performance as the direct-CPLEX method.

TABLE 3. Optimization time comparison of different implementations. Using our CGH for delay and backlog check can reduce the running time from days to seconds.

Implementations	Running time Percentiles (0.5, 0.25, 0.75)
AA +estλ + direct-CPLEX	> 10 ⁵ s
AA +estλ + CGH	(2136.4 s, 1762.9 s, 4468.3 s)
FS +estλ + direct-CPLEX	(1823.8 s, 825.3 s, 4414.0 s)
FS +estλ + CGH	(22.2 s, 10.9 s, 22.3 s)

methods statistically reach the same performance. The first and third quartiles, minimum, median, and maximum values are all 1.0, which means that in general CGH obtains the same (optimal) solution as direct-CPLEX in all the examined test cases. The box plot also shows some outliers. As mentioned earlier, CGH is an heuristic algorithm, therefore in some cases its solution can be worse than that of the direct-CPLEX method (manifested in the box plot through outliers below 1.0). On the other hand, due to the time limit (the cut-off time), the direct-CPLEX method might not be able to find the optimal solution but will instead report the best solution that the method finds during the time limit. In such cases, the solution of CGH might be better (which results in the large valued (above 1.0) outliers).

2) PERFORMANCE EXAMPLE OF THE CONTROL PLANE OPTIMIZATION SOLVER WITH CGH AND CPLEX METHODS FOR DELAY AND BACKLOG VERIFICATION

The time reductions achievable with proposed algorithms and the actual time this may translate to, in terms of running the whole process for solving a controller deployment plan, is exemplified by evaluating a medium-sized topology ("Internetmci", Appendix C) when combining different solver implementations summarised in Table 3 (see also Table 1 for acronyms). In this example, we repeatedly optimized for reliability while keeping bandwidth to 400 Mbps and having the latency requirement be set to a delay between 50 ms to 400 ms.

For the AA + estλ + direct-CPLEX it takes over **50 hours** to complete a deployment plan, which is impractical to use in

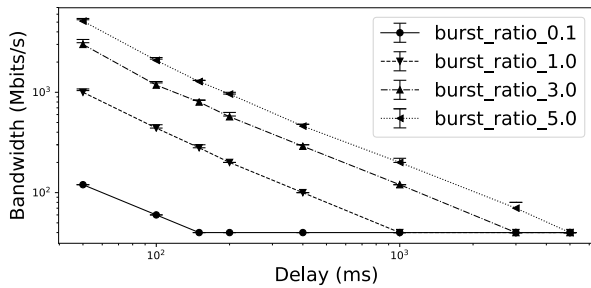


FIGURE 16. Bandwidth vs different delay bound constraints under varying burstiness ratio. The required bandwidth decreases as the delay bound constraint increases. Further, for smaller burstiness ratio the required bandwidth is less.

real implementations. In comparison, the AA + est λ + CGH solver can produce a deployment plan within approximately **35 min** in median. The FS mapping and association offers less optimal deployment plan than AA, but runs much faster as was already indicated by the study in Section VI-A. In our evaluation, the FS + est λ + CGH can yield a deployment plan within about **22 seconds**. Compared with FS + est λ + direct-CPLEX, our FS + est λ + CGH implementation considerably reduced the running time by around **80x**. The observed results demonstrate that our CGH algorithm can greatly accelerate the process of deploying controllers.

B. BANDWIDTH AND DELAY BOUND SCALING TEST

The relationship between bandwidth and reliability was studied for scenario 1. The same conclusions are valid for scenario 2 (we omit the detailed results).

We focus here on studying the relationship between bandwidth and delay in scenario 2 and to this end we use the AA + est λ + CGH solver implementation. We illustrate the results with the Internetmc1 topology. For experiment settings, the rate r_f of a control flow f depends on the request rate of aggregators according to the traffic estimation model in Section III-D. Similar to the settings in Section VI, we vary the request rates of each aggregator randomly within [100, 900]reqs/s. In our tests, the burstiness ratio b_r takes on values within [0.1, 5.0]. Clearly, smaller b_r means lower magnitude of burstiness.

To quantify the influence of the required bandwidth relative to the worst-case flow delay of a deployment plan we proceeded as follows. Given 0.9999 reliability and 80 Mbits backlog constraints fixed, we vary the delay constraint from 50ms to 5s. We observed that as we relax the delay constraint, the required bandwidth for guaranteeing such a delay bound decreases dramatically (see Fig. 16). Furthermore, for guaranteeing the same delay bound, control flows with smaller burstiness require less bandwidth than flows with larger burstiness (as expected). Similarly, in Fig. 17 we quantify the influence of the guaranteed worst case delay relative to the bandwidth limitations. When the bandwidth is varied from 40Mbits/s to 4800Mbits/s, the guaranteed worst-case delay drops dramatically. Moreover, we can conclude that control flows with smaller burstiness can be scheduled with lower

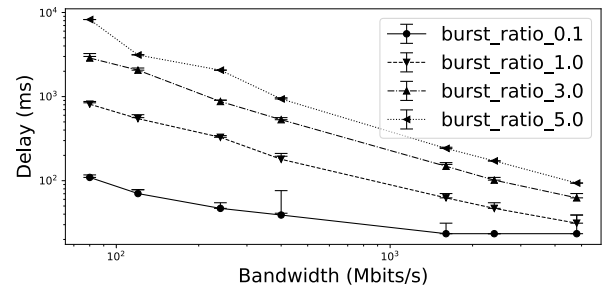


FIGURE 17. Worst case delay versus different bandwidth constraints, under varying burstiness ratio. The worst case delay decreases as the bandwidth constraint increases. Moreover, smaller burstiness ratio corresponds to lower worst case delay.

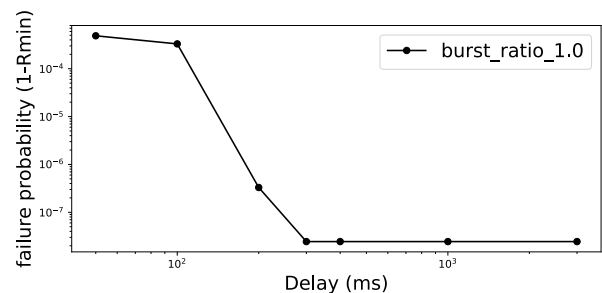


FIGURE 18. Failure probability vs varying delay bound. Relaxing the delay bound results in a decrease of failure probability (increase in reliability).

delay bound than flows with larger burstiness, when the bandwidth constraints are the same. These observations concord with our intuition: the larger the bandwidth, the shorter the worst-case delay, and vice versa.

C. RELIABILITY AND DELAY BOUND SCALING TEST

Three pairs of relationships can be explored with our current implementation of the optimization process: reliability and bandwidth, delay bound and bandwidth, reliability and delay bound (the first of these was studied earlier). We focus on the relationship between reliability and delay bound, given 400 Mbits/s bandwidth and 80 Mbits backlog constraints. When we relax the delay bound constraints, the failure probability decreases, see Fig. 18. When the delay bound is relaxed, less bandwidth is required, thus more controllers can be used in a deployment plan, which results in an increase in reliability.

VIII. RELATED WORK

Control plane deployment has become a hot research topic in recent years. Existing works in this domain focus on aspects such as switch-to-controller or controller-to-controller delay reduction [3], [9], [10], [51]–[53], controller capacity and utilization optimization [54], [55], flow setup time and communication overhead minimization [56], [57], and energy consumption among others.

Existing controller placement works [3], [9], [10], [51]–[53] with delay objectives merely consider packet propagation and processing delay. These works do not

address the control plane traffic and the corresponding control flow routability issues. Queueing and transmission delays are thus typically ignored. Nonetheless, these delays contribute to a large portion of the end-to-end flow delay, which needs to be estimated and scheduled accordingly.

Control plane resilience contributions can be categorized into two major classes: with or without certain theoretical reliability guarantees. Compared to delay or load, reliability (or failure resilience) is hard to compute, often requiring sophisticated modeling. Examples of prior work with reliability models or measurements include [15], [58]–[62], which instead use intuitive objective functions to obtain a placement solution, however, without providing an estimate of the achieved reliability. In contrast, the authors in [25], [63] estimate the network reliability in polynomial time and provide a lower bound of the actual reliability. Additionally, the authors have proposed a heuristic algorithm to decide on the number and location of controllers in order to guarantee a certain reliability requirement.

None of the aforementioned prior approaches address control traffic routability. A scalability optimization model for placement, which has constraints on traffic demand and link bandwidth, has been proposed in [64], albeit in a simplified context by assuming: 1) there is exactly one dedicated one-hop link between each switch and each controller; 2) the control traffic between a switch and a controller is always routed on that dedicated link; and 3) no inter-controller traffic, nor 4) flow delay considerations. In contrast, our approach can be applied to any network topology and can deal with any control traffic pattern, while quantifying reliability, bandwidth and end-to-end flow delay requirements associated with a certain control plane solution. The essential difference between our work and the work reported in [65], which also investigates routing of control flows, is that our solution is proactive—it provides guarantees on routing control flows when a feasible solution is found—whereas the approach in [65] is reactive—it deals with failures and congestion when such events occur.

Traditionally, network flow scheduling has been viewed simply as a multi-commodity flow problem that only considers bandwidth limitation (scenario 1 routability). Works in this area include the classical column generation method [16], [43] and the relatively new polynomial time approximation algorithm [20], which is considered state-of-the-art. We have advanced the latter algorithm [20] for solving scenario 1 control flow routability problems by adapting the general algorithm [20] to the specifics of the aforementioned problem. As a result, the improved algorithm has lower complexity and runs faster than [20]. Importantly, there are only few works that incorporate the entire end-to-end flow delay into the routability (scenario 2 routability) problem definition. Works such as [50], [66] route flows with delay constraints and use network calculus for modeling network elements and calculating flow delays. Nevertheless, they route only new arrival flows, with one flow scheduled at a time, rather than performing a global flow scheduling

and optimization. In contrast to these works, our algorithm optimizes the routes of all existing flows.

In the context of prior art, the essence of our approach is that it is *holistic* as it globally optimizes the routing of all control traffic flows, it is *versatile* as no specific network assumptions are made, and importantly it is *proactive* as it supplies a reliable distributed control plane with performance guarantees. The latter is vital for time-critical services.

IX. DISCUSSION

A. RUNNING TIME

We have generally observed that for medium-sized topologies (Appendix C), if delay and backlog requirements are not considered, using our *estλ* algorithm for bandwidth verification together with the simulated annealing (AA) implementation, needs 5–30 minutes to complete a deployment plan (see also Section VI). In contrast, with the FS implementation (FTCP-CAA), it takes only a few seconds ($< 10s$) to generate a plan, but the plan is farther from optimal. If delay and backlog requirements are added, using our CGH algorithm for delay and backlog verification, *estλ* algorithm for bandwidth verification, together with AA implementation for mapping and association, it takes around 30–120 minutes to accomplish a controller deployment plan. With FS implementation, the estimated time to accomplish a deployment plan is about 0.5–1 minutes (see also Section VII).

Considering practical scenarios, controllers need to be re-deployed when failures occur or in presence of significant variations in the request rates. Failures are rare events, therefore the deployment process will not be invoked frequently. If we only consider hourly variations of the request rates (such as request rate changes due to busy hours and idle periods during the day), the deployment process will run only a few times per day. In both cases, since we only invoke the deployment process infrequently, the AA implementation of the deployment process can satisfy the requirements.

However, in scenarios that require frequent re-deployment or in large-scale networks, where the computational time will be increased due to the network size, we need to consider either accelerating the AA implementation or using the FS implementation. One way to speed-up the execution of the AA implementation is to use the C programming language and parallel computing.

Furthermore, we may cache certain frequently used deployment plans in advance. In particular, we may estimate the busy and idle hours of the day based on historical observations. Then, we can use the deployment process to generate plans for these daily periods in advance and load the pre-calculated plans accordingly.

B. NETWORK CALCULUS

In this article, we use simple affine functions for modeling the arrival and service curves of a flow, since the use of such affine functions can greatly simplify the calculation of the delay and backlog bounds. Theoretically, we may use

complex wide-sense increasing functions to better approximate the arrival and service processes of a flow and hence obtain tighter bounds. However, this often comes at the cost of longer computational time, especially when the curves are non-convex and non-linear. The authors of [35] propose to use a set of piecewise linear functions for better approximations and therefore tighter bounds without complicating the computation too much. However, the effect and relationship between computational complexity and tightness of bounds have not been studied under the context of control plane deployment. This is a potential line for future research.

The deterministic network calculus we applied in our work has one shortcoming: it only considers the worst case. It cannot use the statistical nature of traffic flows, and report on, for instance, the exceeding likelihood of a particular delay bound. Introducing stochastic network calculus into our optimization process, and weighing the pros and cons on aspects such as computational time, network bandwidth requirement and so on, is also another relevant future work.

C. ADVANTAGES AND OTHER POSSIBLE APPLICATIONS

The method proposed in this work can be applied to problems of the same class independently of the specific properties of the underlying network topology and networking conditions, allowing the user to specify traffic models, topological properties and end-conditions arbitrarily. Further, each step of the optimization process allows for a flexible implementation based on any suitable method of choice (heuristic methods, machine learning, exhaustive search, etc.) and is hence adaptive to the system at hand, in terms of computational capacity, platforms and software. We implemented the mapping and association steps by applying the simulated annealing algorithm, but any other algorithm (such as those from the stochastic search category or the extensively used k-means algorithm) pertinent to this type of problems could also be used. Even the implementation of the simulated annealing algorithm is susceptible to different design choices of the cost and transition probability functions (which can have a large impact on the convergence and overall performance of the algorithm).

Furthermore, whereas the optimization process is composed of core steps, its flow (the order of these steps) can be flexibly adapted to accommodate different optimization objectives. Aggregator request rate estimation, for instance, can precede the association step and be used to decide on the control regions so that the load between controllers can be more evenly distributed (that is, balanced).

We believe that with proper implementations, the proposed method may be widely applicable to resource management problems requiring guaranteed performance and reliability in distributed system domain. Examples of applications that may benefit from the proposed method are several:

- Distributed big data computations, where resource allocation and synchronization of partial results require data

transactions under certain performance requirements and reliability guarantees.

- Self-organization of wired and wireless communication infrastructures (e.g., sensor networks) requiring multiple cluster heads selection and association of nodes and scheduling of flows following performance and reliability requirements.
- Scaling of virtualized network functions implementing elastic services, which require performance and reliability guarantees of deployed flows to ensure service availability and quality.
- Organization of wireless access networks, associating access points and base stations as aggregators with distributed controller instances (processing entities) as part of a distributed evolved packet core.

D. SYSTEM INTEGRATION AND CONFIGURABILITY

Our proposed optimization process is generically applicable in the sense that each step can be implemented using suitable methods in line with the needs and requirements of the network operator. In the following, we comment on the practical aspects of integrating and configuring the proposed optimization approach.

1) THE PROCESS AS A NETWORK MANAGEMENT FUNCTION

We envision the optimization process as a functional part of a network management system [67]. In this context, the optimization process can be invoked by auxiliary control and monitoring functions, for triggering adaptation of the deployment plan to increased or decreased network load (i.e., number of requests and expected message sizes), or for mitigating detected failures. The proposed implementation of the optimization process enables fast adaptation to both data volume and topological changes. Parameter tuning and/or alternative implementations of the outlined steps may accelerate the processing speed even further.

2) RUNNING THE OPTIMIZATION FRAMEWORK

In our implementation of the proposed optimization process [67], [68], two types of input is required – a network description file (i.e., graphml-file) and a configuration file specifying the optimization parameters. The former includes an XML-description of the network topology, together with the expected load of each node (i.e., number of requests), the capacity and latency of each link (i.e., the physical transmission delay) and the operational reliability of links and nodes (i.e. failure probabilities). The configuration file specifies the parameter to optimize for (reliability, bandwidth or delay) and the constraints delay, reliability, bandwidth and backlog, as well as the traffic parameters related to inter-traffic between controller and aggregator. The latter requires configuration of three parameters: the message sizes of T_{req} for bidirectional (and possibly asymmetric) communication between controllers and aggregators, as well as T_{state} reflecting the message size for inter-controller communication (see also Section III-D).

3) CONFIGURING THE OPTIMIZATION STEPS

The specific methods used in this paper have been selected and adapted to produce close to optimal results within short time frames, as demonstrated in Section VI and Section VII. In practice, the methods suggested for each step of the optimization process require empirical tuning of few parameters. Specifically, configuration of the applied simulated annealing methods for mapping and association (partially outlined in Algorithm 1 and 2) requires initialization of the initial temperature $T_{initial}$ and the step-factor γ . These two parameters are empirically set by the user as a trade-off between the computation time and level of optimality of the produced result after running the optimization process. Moreover, the FPTAS algorithm (Algorithm 3) implementing the routability check requires only configuration of the ϵ parameter reflecting the level of accuracy. This parameter is user-defined and does not require tuning. For CGH (Algorithm 5), CPLEX is an integral part of the implementation logic. The practitioner would need to ensure the smooth interaction of CPLEX (which solves the master problem) and the remaining part of CGH, which iteratively solves the subproblem by sending input to CPLEX for generating a solution. Knowledge of CPLEX or any other optimization solver is therefore required. The user must also tune the maximum number of iterations allowed before the algorithm stops searching for solution. Theoretically, if this value is very low, CGH might not find a solution. Therefore, setting this parameter requires some empirical work.

X. CONCLUSION

We have proposed a novel optimization approach for flexible deployment of distributed control planes. The approach can automatically decide on the number of controllers, their locations and control regions, and is guaranteed to find a deployment plan that fulfills requirements on control traffic reliability and routability. This feature is especially relevant in the context of future distributed control service applications, where the inter-control traffic required for shared information consistency could potentially become very large with the number of controller instances and when time critical services must be supported by the network. Evaluation results indicate that the approach, involving two novel algorithms for routability, is essentially as effective in finding close to optimal solutions under diverse conditions as other algorithms, but substantially faster, especially for large topologies. The approach can be used as a practical tool for quantifying and predicting the trade-off between bandwidth, delay and reliability, making it suitable for service providers and operators that wish to develop control plane deployment policies. Note that, the approach is applicable not only to the case of deploying instances in controller plane applications, but also to other types of service deployment (such as deployment of back-up service nodes for maintaining service availability).

Our design of two fast algorithms for bandwidth verification as well as delay and backlog verification, makes it possible to satisfy, in extremely fast fashion, different routability requirements. The running time is effectively reduced to the

level of **50x** in the case of bandwidth verification, and by **500x** for delay and backlog verification (compared to CPLEX) for large-sized topologies. In a practical network operations setting, the ability to retrieve a complex deployment plan with many constraints within seconds rather than days, is essential for quickly adapting to changing network conditions (e.g., emergencies where connectivity and communication is vital) and to ensure critical service availability at all times. The possibility to elastically adapt the control plane deployment is also fundamental for effective utilization of networked infrastructure resources. The concepts presented in this article have been filed for patenting [67] and an early implementation of the tool reflecting the features in [21] is also available at <https://github.com/nigsics/dcpmtool>.

APPENDIXES

APPENDIX A

TIME COMPLEXITY ANALYSIS OF THE FPTAS ALGORITHM

The FPTAS algorithm follows the core prime-dual idea and flow of the FAS algorithm [20] as described earlier in Section III-E. Here, we mainly focus on analyzing the running time of our algorithm and answer the question why it is faster than the FAS [20] when routing control plane flows.

We only consider the $\eta \geq 1$ case, where $\eta = \min_l D(l)$ (see (13)), since when $\eta < 1$, we can always scale all the flow demands with a common factor Sc to ensure $\eta' = \eta/Sc \geq 1$. How to find a proper Sc has already been discussed in [20].

For $\eta \geq 1$, we state the lemmas as follow:

Lemma 1: The upper bound on the total number of phases is given by:

$$t = \lceil \frac{\eta}{\epsilon} \log_{1+\epsilon} \frac{|E|}{1-\epsilon} \rceil.$$

The upper bound is the same for both FAS [20] and FPTAS algorithms. For the proof of Lemma 1, the reader is referred to [20].

Now, let us consider the number of iterations in each phase. In comparison to FAS, which executes $|V|$ iterations, each phase in our FPTAS algorithm needs just $|C|$ iterations. Note that each iteration might have different number of steps. Lemma 2 gives a bound on the total number of steps.

Lemma 2: The total number of steps exceeds the total number of iterations by at most $|E| \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$, for $\eta > 1$.

Proof: In each iteration, there are two types of steps: the terminal step (the last step of an iteration) and non-terminal steps. Clearly, the total number of terminal steps equals the total number of iterations. Therefore, we need to prove only that the total number of non-terminal steps is less than $|E| \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$.

If j th step in an iteration is not the last step, there must be at least one saturated edge e with $\rho(e) > 1$ (line 9 of Algorithm 3). Otherwise, $\sigma = 1$ according to line 10 and thus $d'(f) = 0, \forall f \in F^c$, which means there will be no next step. Therefore, according to line 14, for any saturated edge e , its link length $l(e) = (1+\epsilon)l(e)$ is increased. This means that for every non-terminal step in an iteration, the length of at least one edge is increased by a factor of $(1 + \epsilon)$. Furthermore,

we have $D(l) \leq (1 + \epsilon)$ (otherwise the algorithm has already terminated), which requires that $l(e) < (1 + \epsilon)/u(e)$, $\forall e$. Thus, given the initial $l(e) = \delta/u(e)$, $\forall e$, the total number of non-terminal steps is at most $|E| \log_{1+\epsilon} \frac{(1+\epsilon)/u(e)}{\delta/u(e)} = |E| \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$. \square

Therefore, the FPTAS algorithm contains $t|C|$ iterations and hence the total number of steps have an upper bound of $t|C| + |E| \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$. In contrast, the FAS algorithm requires $t|V|$ iterations and the upper bound on the total number of steps is $t|V| + |E| \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$. Considering that in both algorithms each step has the same time complexity $\mathcal{O}(|V| \log |V| + |E|)$ (due to the Dijkstra shortest path algorithm in the inner loop), our FPTAS algorithm outperforms FAS [20] in time complexity. In essence, it requires fewer iterations and steps, and is especially faster when $|C| \ll |V|$.

APPENDIX B

A METHOD FOR CALCULATING THE LOWER AND UPPER BOUNDS OF λ

The lower and upper bounds of λ are calculated based on the following two propositions.

Proposition 1: Let P_f denote the shortest path of flow f with maximum capacity and let cap_f denote the capacity of this path ($cap_f = \min(u_e | e \in p_f)$). Let $r_{min} = \min(cap_f/d_f, \forall f \in F)$. Then, $\lambda \leq |E|r_{min}$, and $\lambda \geq r_{min}/|E|$.

Proof: Suppose we route each flow $f \in F$ with d_f units along its maximum capacity shortest path P_f , considering the utilization of every edge $e \in E$.

$$\rho(e) = \frac{\sum_{f:e \in P_f} d_f}{cap_e}, \forall e \in E \quad (57)$$

$$\leq \sum_{f:e \in P_f} 1/r_{min} \quad (58)$$

$$\leq |E|/r_{min} \quad (59)$$

Since the maximum utilization of any link is smaller than $|E|/r_{min}$, we can always feasibly route each flow $f \in F$ with $d'_f = d_f \frac{r_{min}}{|E|}$. Thus, we have $\lambda = d'_f/d_f \geq \frac{r_{min}}{|E|}$.

Each flow f can have at most $|E|$ different paths ($|\kappa_f| < |E|$). Since cap_f denotes the capacity of the maximum capacity shortest path of flow f , we can at most route $cap_f|E|$ units of each flow. Thus, we have:

$$\lambda \leq \min(cap_f|E|/d_f) \quad (60)$$

$$= |E|r_{min} \quad (61)$$

\square

Proposition 2: Let cap_{out}^n denote the sum of all the outgoing link capacities of arbitrary node $n \in V$ and let d_{out}^n denote the sum of $\{d_f | s_f = n\}$. Similarly, let cap_{in}^n denote the sum of all the incoming link capacities, and d_{in}^n denote the sum of $\{d_f | t_f = n\}$. Then, $\lambda \leq \min(d_{out}^n/cap_{out}^n, d_{in}^n/cap_{in}^n)$.

Proof: We calculate the ratio of the total demand of all the flows originating from a node or ending at a node, over the total input/output bandwidth of the node. Since λ reflects

TABLE 4. Description of used topologies.

Name	No. of nodes	No. of edges
Dataxchange	6	22
Epoch	6	14
Telecomserbia	6	12
Netrail	7	20
Sanren	7	14
Goodnet	17	62
Ibm	18	48
Internetmci	19	66
Jannetlense	20	68
BtEurope	24	74
Arpanet19728	29	64
Arnes	34	92
Iij	37	130
Geant2010	37	112
Renater2010	43	112
HiberniaAtlantic	55	162
Dfn	58	174

the proportion of routable flow demand, it should be smaller than this ratio. \square

By combining Proposition 1 and Proposition 2, we can calculate the upper bound by $\lambda_{high} = \min(|E|r_{min}, d_{out}^n/cap_{out}^n, d_{in}^n/cap_{in}^n)$, and the lower bound by $\lambda_{low} = r_{min}/|E|$.

APPENDIX C

TOPOLOGIES USED

Table 4 provides an overview of the topologies used for the tests in terms of number of nodes and number of edges. In this paper, we consider topologies of small (< 10 nodes), medium (< 30 nodes) and large (≥ 30 nodes) sizes.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers, as well as Prof. Magnus Boman (KTH, Sweden) and the M.Sc. Daniel Felipe Perez-Ramirez (RISE, Sweden), for valuable comments and suggestions received in the preparation of the final version of this manuscript.

REFERENCES

- [1] A. Bianco, P. Giaccone, R. Mashayekhi, M. Ullio, and V. Vercellone, "Scalability of ONOS reactive forwarding applications in ISP networks," *Comput. Commun.*, vol. 102, pp. 130–138, Apr. 2017.
- [2] A. S. Muqaddas, A. Bianco, P. Giaccone, and G. Maier, "Inter-controller traffic in ONOS clusters for SDN networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.
- [3] T. Zhang, A. Bianco, and P. Giaccone, "The role of inter-controller traffic in SDN controllers placement," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2016, pp. 87–92.
- [4] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proc. OSDI*, vol. 10, 2010, pp. 1–6.
- [5] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an open, distributed SDN OS," in *Proc. ACM Hot topics Softw. Defined Netw.*, 2014, pp. 1–6.
- [6] A. S. Muqaddas, A. Bianco, and P. Giaccone. (2016). *Inter-Controller Traffic in ONOS Clusters for SDN Networks*. [Online]. Available: http://onos-cord-eu.create-net.org/wp-content/uploads/2016/09/01-ONOS_CORD_Workshop16-InterClusterTraffic_ONOS-Abridged.pdf
- [7] E. S. Elmallah, "Algorithms for K-terminal reliability problems with node failures," *Networks*, vol. 22, no. 4, pp. 369–384, Jul. 1992.

- [8] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-free coordination for Internet-scale systems," in *Proc. USENIX Ann. Tech. Conf.*, vol. 8, no. 9, 2010, p. 11.
- [9] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proc. ACM SIGCOMM Comp. Comm. Rev.*, 2012, pp. 7–12.
- [10] Y. Jimenez, C. Cervello-Pastor, and A. J. Garcia, "On the controller placement for designing a distributed SDN control layer," in *Proc. IFIP Netw. Conf.*, Jun. 2014, pp. 1–9.
- [11] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan, "Reliability-aware controller placement for software-defined networks," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, 2013, pp. 672–675.
- [12] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "On reliability-optimized controller placement for software-defined networks," *China Commun.*, vol. 11, no. 2, pp. 38–54, Feb. 2014.
- [13] Y. Zhang, N. Beheshti, and M. Tatipamula, "On resilience of split-architecture networks," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Dec. 2011, pp. 1–6.
- [14] Q. Zhong, Y. Wang, W. Li, and X. Qiu, "A min-cover based controller placement approach to build reliable control network in SDN," in *Proc. NOMS - IEEE/IFIP Netw. Oper. Manage. Symp.*, Apr. 2016, pp. 481–487.
- [15] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic approaches to the controller placement problem in large scale SDN networks," *IEEE Trans. Netw. Service Manage.*, vol. 12, no. 1, pp. 4–17, Mar. 2015.
- [16] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, 1993.
- [17] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 2, pp. 51–62, Mar. 2007.
- [18] M. Zhang, C. Yi, B. Liu, and B. Zhang, "GreenTE: Power-aware traffic engineering," in *Proc. 18th IEEE Int. Conf. Netw. Protocols*, Oct. 2010, pp. 21–30.
- [19] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*. Cambridge, U.K.: Cambridge Univ. Press, 2009.
- [20] G. Karakostas, "Faster approximation schemes for fractional multicommodity flow problems," *ACM Trans. Algorithms*, vol. 4, no. 1, pp. 1–17, Mar. 2008.
- [21] S. Liu, R. Steinert, and D. Kostic, "Flexible distributed control plane deployment," in *Proc. NOMS - IEEE/IFIP Netw. Operations Manage. Symp.*, Apr. 2018, pp. 1–7.
- [22] J. W. Rupe, "Reliability of computer systems and networks fault tolerance, analysis, and design," *IEE Trans.*, vol. 35, no. 6, pp. 586–587, Jun. 2003.
- [23] (2011). *Openflow Switch Specification v1.2*. [Online]. Available: <https://www.opennetworking.org/>
- [24] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 2211–2219.
- [25] F. J. Ros and P. M. Ruiz, "On reliable controller placements in software-defined networks," *Comput. Commun.*, vol. 77, pp. 41–51, Mar. 2016.
- [26] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
- [27] A. Corana, M. Marchesi, C. Martini, and S. Ridella, "Minimizing multimodal functions of continuous variables with the 'simulated annealing' algorithm—Corrigenda for this article is available here," *ACM Trans. Math. Softw.*, vol. 13, no. 3, pp. 262–280, 1987.
- [28] D. Bertsimas and J. Tsitsiklis, "Simulated annealing," *Statist. Sci.*, vol. 8, no. 1, pp. 10–15, 1993.
- [29] A. Bianco, P. Giaccone, A. Mahmood, M. Ullio, and V. Vercellone, "Evaluating the SDN control traffic in large ISP networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 5248–5253.
- [30] N. Garg and J. Könemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems," *SIAM J. Comput.*, vol. 37, no. 2, pp. 630–652, Jan. 2007.
- [31] T. H. Cormen, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2009.
- [32] I.-L. Wang, "Multicommodity network flows: A survey, part II: Solution methods," *Int. J. Oper. Res.*, vol. 15, no. 4, pp. 155–173, 2018.
- [33] V. V. Vazirani, *Approximation Algorithms*. Berlin, Germany: Springer-Verlag, 2003.
- [34] R. L. Cruz, "A calculus for network delay. I. Network elements in isolation," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, pp. 114–131, Jan. 1991.
- [35] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, 2001, p. 2050.
- [36] R. Agrawal, R. L. Cruz, C. Okino, and R. Rajan, "Performance bounds for flow control protocols," *IEEE/ACM Trans. Netw.*, vol. 7, no. 3, pp. 310–323, Jun. 1999.
- [37] Y. Jiang and Y. Liu, *Stochastic Network Calculus*, vol. 1. London, U.K.: Springer-Verlag, 2008.
- [38] A. Van Bemten and W. Kellerer, *Network Calculus: A Comprehensive Guide*, Lehrstuhl für Kommunikationsnetze, Technische Universität München, Munich, Germany, 2016.
- [39] Y. Jiang, "Relationship between guaranteed rate server and latency rate server," *Comput. Netw.*, vol. 43, no. 3, pp. 307–315, Oct. 2003.
- [40] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proc. IEEE*, vol. 83, no. 10, pp. 1374–1396, Oct. 1995.
- [41] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 1–12, 2007.
- [42] K.-C. Lan and J. Heidemann, "A measurement study of correlations of Internet flow characteristics," *Comput. Netw.*, vol. 50, no. 1, pp. 46–62, Jan. 2006.
- [43] L. R. Ford and D. R. Fulkerson, "A suggested computation for maximal multi-commodity network flows," *Manage. Sci.*, vol. 5, no. 1, pp. 97–101, Oct. 1958.
- [44] L. S. D. Oliveira and S. F. P. Saramago, "Multiobjective optimization techniques applied to engineering problems," *J. Brazilian Soc. Mech. Sci. Eng.*, vol. 32, no. 1, pp. 94–105, 2010.
- [45] D. E. Knuth, "Searching and sorting," in *The Art of Computer Programming*, vol. 3. Reading, MA, USA: Addison-Wesley, 1973, ch. 6.5.
- [46] (2011). *The Internet Topology Zoo*. [Online]. Available: <http://www.topology-zoo.org/>
- [47] D. Levin, A. Wundsam, A. Feldmann, S. Seetharaman, M. Kobayashi, and G. Parulkar, "A first look at OpenFlow control plane behavior from a test deployment," Fakultät Elektrotechnik Informatik, Technische Universität Berlin, Berlin, Germany, Tech. Rep. 13-2011, 2011. [Online]. Available: <http://www.eecs.tu-berlin.de/menuue/forschung/forschungsberichte/2011>
- [48] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage, "California fault lines: Understanding the causes and impact of network failures," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 315–326, 2010.
- [49] G. P. Katsikas, "NFV service chains at the speed of the underlying commodity hardware," Ph.D. dissertation, KTH Roy. Inst. Technol., Stockholm, Sweden, 2018.
- [50] J. W. Guck, A. Van Bemten, and W. Kellerer, "DetServ: Network models for real-time QoS provisioning in SDN-based industrial environments," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 4, pp. 1003–1017, Dec. 2017.
- [51] L. Liao and V. C. M. Leung, "Genetic algorithms with particle swarm optimization based mutation for distributed controller placement in SDNs," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2017, pp. 1–6.
- [52] B. P. R. Killi, E. A. Reddy, and S. V. Rao, "Cooperative game theory based network partitioning for controller placement in SDN," in *Proc. 10th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2018, pp. 105–112.
- [53] K. S. Sahoo, S. Sahoo, A. Sarkar, B. Sahoo, and R. Dash, "On the placement of controllers for designing a wide area software defined networks," in *Proc. TENCON - IEEE Region 10 Conf.*, Nov. 2017, pp. 3123–3128.
- [54] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 18, no. 8, pp. 1339–1342, Aug. 2014.
- [55] S. Jiugen, Z. Wei, J. Kunying, and X. Ying, "Multi-controller deployment algorithm based on load balance in software defined network," *J. Electron. Inf. Technol.*, vol. 40, no. 2, pp. 455–461, 2018.
- [56] M. He, A. Basta, A. Blenk, and W. Kellerer, "Modeling flow setup time for controller placement in SDN: Evaluation for dynamic flows," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–7.
- [57] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Proc. 9th Int. Conf. Netw. Service Manage. (CNSM)*, Oct. 2013, pp. 18–25.
- [58] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in SDN-based core networks," in *Proc. 25th Int. Teletraffic Congr. (ITC)*, Sep. 2013, pp. 1–9.

- [59] D. Hock, S. Gebert, M. Hartmann, T. Zinner, and P. Tran-Gia, "POCO-framework for Pareto-optimal resilient controller placement in SDN-based core networks," in *Proc. IEEE Netw. Operations Manage. Symp. (NOMS)*, May 2014, pp. 1–2.
- [60] L. F. Muller, R. R. Oliveira, M. C. Luizelli, L. P. Gaspar, and M. P. Barcellos, "Survivor: An enhanced controller placement strategy for improving SDN survivability," in *Proc. IEEE Global Commun. Conf.*, Dec. 2014, pp. 1909–1915.
- [61] D. M. F. Mattos, O. C. M. B. Duarte, and G. Pujolle, "A resilient distributed controller for software defined networking," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.
- [62] P. Vizarreta, C. M. Machuca, and W. Kellerer, "Controller placement strategies for a resilient SDN control plane," in *Proc. 8th Int. Workshop Resilient Netw. Design Model. (RNDM)*, Sep. 2016, pp. 253–259.
- [63] F. J. Ros and P. M. Ruiz, "Five nines of southbound reliability in software-defined networks," in *Proc. 3rd workshop Hot topics Softw. defined Netw. (HotSDN)*, 2014, pp. 31–36.
- [64] A. Sallahi and M. St-Hilaire, "Optimal model for the controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 19, no. 1, pp. 30–33, Jan. 2015.
- [65] B. Gorkemli, S. Tatlicioglu, A. M. Tekalp, S. Civanlar, and E. Lokman, "Dynamic control plane for SDN at scale," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2688–2701, Dec. 2018.
- [66] 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.
- [67] S. Liu, R. Steinert, and D. Kostic, "Dynamic deployment of network applications having performance and reliability guarantees in large computing networks," USPTO Patent 16745477, Jan. 17, 2020.
- [68] D. F. Perez-Ramirez, R. Steinert, N. Vesselinova, and D. Kostic, "Demo abstract: Elastic deployment of robust distributed control planes with performance guarantees," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Jul. 2020, pp. 1–2.



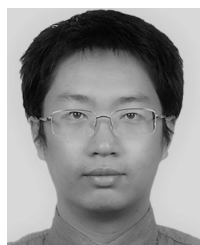
REBECCA STEINERT received the B.Sc. degree in real-time systems, the M.Sc. degree in computer science with emphasis on autonomous systems and machine learning, in 2008, and the Ph.D. degree in probabilistic fault management and performance monitoring in networked systems from the KTH, Royal Institute of Technology, Stockholm, in 2014.

She joined the RISE Research Institutes of Sweden (formerly SICS Swedish Institute of Computer Science), in 2006. She is currently driving research within applied machine learning for intelligent autonomous networked systems, and has been the Leader of the Network Intelligence Research Group, since 2015. Her current research interests include probabilistic modeling, deep learning and combinatorial optimization applied in programmable networks, and distributed systems.



NATALIA VESSELINOVA (Member, IEEE) received the M.Sc. degree in telecommunications engineering from the Technical University of Sofia, and the Nordic Five Tech M.Sc. degree in applied and engineering mathematics from the Chalmers University of Technology and Aalto University, and the Ph.D. degree in telecommunications from the Technical University of Catalonia.

She is currently a Senior Research Scientist with the Research Institutes of Sweden (RISE), where she contributes to the research endeavours of the Network Intelligence Group. At present, she is focused on solving problems from the networking engineering practice with probability, statistics, machine learning, and deep learning theory and tools.



SHAOTENG LIU was born in Xi'an, Shannxi, China, in 1984. He received the B.S. and M.S. degrees in microelectronics from Fudan University, Shanghai, China, in 2010, and the M.S. degree on system-on-chip and the Ph.D. degree in electronics and computer system from the KTH Royal Institute of Technology, Stockholm, Sweden, in 2010 and 2015, respectively.

From November 2013 to June 2014, he was a Ph.D. Intern with the Xilinx Research Laboratory, USA. From 2015 to 2017, he was a Researcher with RISE SICS. Since 2017, he has been a Senior Researcher with the Network Intelligence Group, RISE SICS. He is the author of 12 articles. His research interests include electronic systems, high performance computation, networks-on-chip, distributed systems, optimization and machine learning.

Dr. Liu received the Best Paper Award from the Ninth ACM/IEEE International Symposium on Networks-on-Chip, in 2015.



DEJAN KOSTIĆ received the Ph.D. degree in computer science from Duke University. He spent the last two years of his studies and a brief stay as a Postdoctoral Scholar with the University of California, San Diego, CA, USA. Since 2014, he has been associated with the RISE Research Institutes of Sweden (formerly SICS Swedish Institute of Computer Science). He is currently a Professor of internetworking with the KTH Royal Institute of Technology, where he is the Head of the

Communication Systems Division. His research interests include distributed systems, computer networks, operating systems, and mobile computing.

...