Reliable Virtual Machine Placement and Routing in Clouds

Song Yang, Philipp Wieder, Ramin Yahyapour, Stojan Trajanovski, Xiaoming Fu, *Senior Member, IEEE*

Abstract—In nowadays cloud computing systems, leveraging the virtualization technology, the customer's requested data computing or storing service is accommodated by a set of mutual-communicated Virtual Machines (VM) in a scalable and elastic manner. These VMs are placed in one or more server nodes according to nodes' capacities, failure probabilities, etc. The VM placement availability refers to the probability that at least one set of the whole customer's requested VMs operates during the entire requested lifetime. In this paper, we first study the problem of placing at most H sets of requested k VMs on minimum number of nodes, such that the VM placement availability is no less than δ, and the specified communication delay and connection availability for each VM pair is not violated. We consider this problem with and without Shared-Risk Node Group (SRNG) failures, and prove this problem is NP-hard in both cases. We subsequently propose an exact Integer Nonlinear Program (INLP) and an efficient heuristic to solve this problem. We conduct simulations to compare the proposed algorithms with two existing heuristics in terms of performance. Finally, we study the related reliable routing problem of establishing a connection over at most w link-disjoint paths from a source to a destination, such that the connection availability requirement is satisfied and each path delay is no greater than a given value. We devise both exact algorithm and heuristics to solve this NP-hard problem, and evaluate them via simulations.

Index Terms—Virtual Machine placement, Routing, Availability, Reliability, Cloud computing, Optimization algorithms.

✦

1 INTRODUCTION

C LOUD computing [2] is a distributed computing and storing paradigm, which can provide scalable and reliable service over the Internet for on-demanding data-intensive application (e.g., on-line search or video streaming) and data-intensive computing (e.g., analyzing and processing large volume of scientific data.). The key features of cloud computing including "pay-as-you-go" and "elastic service" attract more and more service providers and customers to deploy their workload from their own infrastructures or platforms to public or private clouds.

Distributed cloud systems are usually composed of distributed inter-connected datacenters, which leverage virtualization technology to provide computing and storage service for each on-demanding requests. Once a request arrives, several Virtual Machines (VM) are created in one or more server nodes (which may locate in either same or different datacenters) in order to accommodate it. However, the server node failures caused by hardware (e.g., hard disk, memory module) failures and software problems (e.g., software bugs, configuration errors) may result in the loss of the VMs hosted on it and hence the whole service cannot be guaranteed. An efficient way to overcome it is to create and place more VM replicas, but this approach should also take the nodes' availabilities into account. For instance, if all the

VMs together with their replicas are placed at nodes with high failure probability, then the proper service cannot be guaranteed. The VM placement availability, a value between 0 and 1, is therefore important and refers to the probability that at least one set of the whole customer's requested VMs are in their operating state during the entire requested lifetime.

Moreover, if two or more VMs are placed on different nodes, we should also guarantee a reliable communication between these VMs. In fact, for a single unprotected path, it will fail if one of its belonged links fails. To increase the availability of transporting data from a source to a destination, path protection (or survivability) are called for. For instance, by allocating a pair of link-disjoint paths from a source to a destination, the data is transported on the primary path. Upon a failure of the primary path, the data can be switched on the backup path. However, the path protection mechanism that does not allow for more than 2 link-disjoint paths may still not be reliable enough and $w > 2$ link-disjoint paths may be needed. Moreover, the link availability should also be take into account. For a connection over at most w link-disjoint paths between a node pair, its availability specifies the probability that at least one path is operational. Connection availability is therefore important for quantitatively measuring the availability of delivering data between VMs located on different nodes for cloud computing.

In this paper, we first study the Reliable VM Placement (RVMP) problem, which is to place at most H sets of requested k VMs on minimum number of nodes, such that the VM placement availability is no less than δ, and the specified communication delay and connection availability for each VM pair is not violated.

Following that, we study the Availability-Based Delay-

- *S. Yang and P. Wieder are with Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG), Göttingen, Germany.*
  E-mail: {S.Yang, P.Wieder}@gwdg.de
- *R. Yahyapour is with GWDG and Institute of Computer Science, University of Göttingen, Göttingen, Germany. E-mail: R.Yahyaour@gwdg.de*
- *S. Trajanovski is with University of Amsterdam and Delft University of Technology, The Netherlands. E-mail: S.Trajanovski@uva.nl*
- *X. Fu is with Institute of Computer Science, University of Göttingen, Göttingen, Germany. E-mail: Fu@cs.uni-goettingen.de*

Constrained Routing (ABDCR) problem, which is to establish a connection over at most w (partially) link-disjoint paths from a source to a destination such that the connection availability is at least η and each path has a delay no greater than D. Our key contributions are as follows:

- We propose a mathematical model to formulate VM placement availability with and without Shared-Risk Node Group failures, and prove that the Reliable VM Placement (RVMP) problem under both cases is NP-hard.
- We propose an Integer Nonlinear Program (INLP) and a heuristic to solve the RVMP problem.
- We compare the proposed algorithms with two existing heuristics in terms of performance via simulations.
- We prove that the ABDCR problem is NP-hard, devise both exact algorithm and two heuristics to solve it, and verify them via simulations as well.

The remainder of this paper is organized as follows: Section 2 and 3 formulate the VM placement availability calculation without and with SRNG failures, respectively. In Section 4, we study the Reliable VM Placement (RVMP) problem and prove it is NP-hard. We propose an exact Integer Nonlinear Program (INLP) and a heuristic to solve the RVMP problem. The proposed algorithms are also evaluated via simulations. In Section 5, we define the Availability-Based Delay-Constrained Routing (ABDCR) problem, prove its NP-hardness, and propose both an exact algorithm and two heuristics to solve it. We also conduct simulations to verify these proposed algorithms as well. Section 6 presents the related work and we conclude in Section 7.

2 VM PLACEMENT AVAILABILITY

The availability of a system is the fraction of time the system is operational during the entire service time. The availability A_j of a network component j (e.g., rack, switch) can be calculated as [3]:

$$A_j = \frac{MTTF}{MTTF + MTTR} \tag{1}$$

where $MTTF$ represents Mean Time To Failure and $MTTR$ denotes Mean Time To Repair. In this paper, we regard the node in the network as a server, and its availability is equal to the product of availabilities of all its components [4] (e.g., hard disk , memory, etc.). We first assume that the node availabilities are uncorrelated/independent.

Suppose there is a set of k requested VMs V, which are represented by $v_1, v_2, \ldots, v_k$. For each requested VM v_i ($1 \leq i \leq k$), placing it on one same node (say n) more than once cannot increase placement availability, since when n fails, all its resided VMs will fail simultaneously. Therefore, we need to place v_i on different nodes to increase the placement availability. Let us use H_i to represent the maximum number of nodes to host VM v_i and denote $H = \max_{i=1}^{k}(H_i)$. Or equivalently, H_i indicates the maximum number of nodes v_i can be placed on. We distinguish and analyze the VM placement availability under two different cases, namely (1) Single Placement: each VM is exactly placed on $H = 1$ node in the network (2) Protected Placement: $\exists v_j \in V$, it can be placed on $H_j > 1$ nodes in the network, i.e., $H > 1$. In the following, we will address the VM placement availability under two node failure scenarios, namely, (1) single node failure scenario: at most one node may encounter failure at one certain time point, and (2) multiple nodes failure: multiple nodes may fail at one certain time point. Without loss of generality, in this paper, we assume multiple node failure scenario.

2.1 Single-node failure

Here it is assumed that all the nodes in the network have very low failure probability (highly reliable). Therefore, we can assume that at one time point, at most one node may encounter failure. In the single placement case, if m nodes with availability A_1, $A_2, \ldots,$ A_m are used for hosting k VMs ($m \leq k$), then the availability of the VM placement is $A_{sn} = \min(A_1, A_2, \ldots, A_m)$. In the protected placement case, if there are another m' ($1 \leq m' \leq k$) nodes which are totally different from the existing m nodes and k VMs are also placed on these m' nodes. In this sense, the availability of placing in total $2k$ VMs on $m + m'$ nodes is 1, since each VM located on one node is fully "protected" by another backup VM located on a different node. We can also see that for each VM, one backup VM placed on a different node is enough, i.e., there is no need to have more than one backup VM. Moreover, when there are less than k backup VMs placed on m' nodes, it indicates that at least one VM does not have its backup. Let us denote the node set $\mathcal{N}_{un}$ as the nodes on which VMs are located and do not have their backups. As a result, the availability of placing g ($k < g < 2k$) VMs on $m + m'$ nodes is $\min_{i \in \mathcal{N}_{un}}(A_i)$. However, this approach only works when all the links are highly reliable. In Appendix A, we will provide an Integer Nonlinear Program (INLP) to solve the Reliable Virtual Machine Placement problem under the single-node failure scenario.

2.2 Multiple nodes failure

It is a more general model where all the nodes may fail simultaneously at one certain time point. In this context, the VM placement availability in the single placement is equal to the product of the availabilities of nodes that host at least one requested VM. For instance, if m nodes with availability A_1, $A_2, \ldots,$ A_m are used for hosting k VMs ($m \leq k$), then the availability (denoted by A_p) of this VM placement is:

$$A_p = A_1 \cdot A_2 \cdots A_m \tag{2}$$

Eq. (2) indicates that since in total k VMs are requested, the availability should take into account the probability that all these k VMs are operational.

In the protected placement case, there exist one or more VMs that can be placed on at most H nodes. Therefore, we regard that a protected placement P is composed of (at most) H single placements. For the ease of clarification, we further term each of H single placements in the protected placement as placement plan p_i, which means the "i−th" placing k VMs on m_i nodes, where $1 \leq i \leq H$ and $1 \leq m_i \leq k$. We regard that p_1 as the primary placement plan. We make no difference between single placement and placement plan. Since different placement plans may place

one or more VMs on the same node, we distinguish the protected placement into two cases, namely (1) *fully protected placement*, for each VM $v \in V$, it is placed by each plan p_i ($1 \le i \le H$) at H different nodes, and (2) *partially protected placement*, $\exists v \in V$, it is placed on less than H nodes, i.e., two or more placement plans place v on the same node.

In the fully protected placement case, the availability can be calculated as:

$$A_{PD}^F = 1 - \prod_{i=1}^{H}(1 - A_{p_i}) = \sum_{i=1}^{H} A_{p_i} - \sum_{0<i<j\le H} A_{p_i} \cdot A_{p_j}$$
$$+ \sum_{0<i<j<u\le H} A_{p_i} \cdot A_{p_j} \cdot A_{p_u} + \cdots + (-1)^{H-1}\prod_{i=1}^{H} A_{p_i} \quad (3)$$

where $A_{p_i} = \prod_{n \in m_i} A_n$ denotes the availability of single VM placement according to Eq. (2). Eq. (3) implies that the availability of H placement plans is equivalent to the probability that at least one single placement (a set of k VMs) is operational in the service-life time.

In the partially protected placement case, if there exists one VM which is placed on less than H nodes, we could regard that it is "sharely" placed by more than one placement plan. For example in Fig. 1, each node is associated with its own availability value and we need to place three VMs (v_1, v_2 and v_3) on it. We set $H_1 = H_2 = 2$ and $H_3 = 1$ for simplicity. We assume that placement plan p_1 to place v_1 and v_2 on node a, and placement plan p_2 to place replica v_1' and replica v_2' on node b, where $v_1' = v_1, v_2' = v_2$. On the other hand, v_3 is only placed on one node. Therefore, we can regard that p_1 and p_2 "sharely" place v_3 on node c.

Placement plan p_1: v_1, v_2, v_3
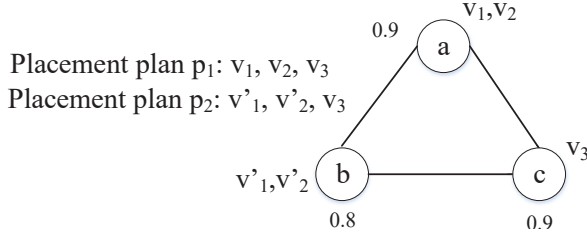Placement plan p_2: v_1', v_2', v_3

Fig. 1: An example of partially VM placement.

However, we cannot directly apply Eq. (3) to calculate its availability, since the availabilities of nodes which hold "shared" VMs will be counted more than once. To amend this, we use a new operator $\circ^1$. Suppose there are m different nodes $n_1, n_2, \ldots, n_m$ with availabilities $A_1, A_2, \ldots, A_m$. For a node n_x with availability A_x, $\circ$ can be defined as follows:

$$A_1 \cdot A_2 \cdots A_m \circ A_x = \begin{cases} \prod_{i=1}^{m} A_i & \text{if } \exists n_i = n_x \\ \prod_{i=1}^{m} A_i \cdot A_x & \text{otherwise} \end{cases} \quad (4)$$

Let $\coprod$ denote the $\circ$ operations of different sets, then the availability (represented by A_{PD}^H) of H partially placement plans can now be represented as:

1. Similarly as we did in [5], [6] to calculate the connection availability of partially link-disjoint paths.

$$A_{PD}^H = 1 - \prod_{i=1}^{H}(1 - A_{p_i})$$
$$= 1 - (1 - A_{p_1}) \circ (1 - A_{p_2}) \circ \circ \circ (1 - A_{p_H}) \quad (5)$$
$$= \sum_{i=1}^{H} A_{p_i} - \sum_{0<i<j\le H} A_{p_i} \circ A_{p_j} +$$
$$\sum_{0<i<j<u\le H} A_{p_i} \circ A_{p_j} \circ A_{p_u} + \cdots + (-1)^{H-1}\prod_{i=1}^{H} A_{p_i}$$

where A_{p_i} denotes the availability of placement plan p_i and can be calculated from Eq. (2).

To better illustrate it, we take Fig. 2 for example. In Fig. 2, it is assumed that each node can host at most one VM for simplicity and its availability value is labeled on itself. Suppose three VMs v_1, v_2, and v_3 are placed on node s, a and b, respectively, then according to Eq. (2), the availability of this VM placement is $0.95 \cdot 0.9 \cdot 0.9 = 0.7695$. If we set $\delta = 0.85$, this VM placement cannot satisfy customer's requirement. We regard the above VM placement plan as p_1. And we add one more plan p_2, where we place one more replica of v_2 (represented by v_2') on node c, one more replica of v_3 (represented by v_3') on d, and regard p_2 and p_1 together sharely place v_1 on node s. Hence, according to Eq. (5), the overall availability of p_1 and p_2 is $1 - (1 - A_s \circ A_a \circ A_b) \circ (1 - A_s \circ A_c \circ A_d) = A_s \circ A_a \circ A_b + A_s \circ A_c \circ A_c - A_s \circ A_a \circ A_b \circ A_s \circ A_c \circ A_d = A_s \cdot A_a \cdot A_b + A_s \cdot A_c \cdot A_c - A_s \cdot A_a \cdot A_b \cdot A_c \cdot A_d = 0.95 \cdot 0.9 \cdot 0.9 + 0.95 \cdot 0.8 \cdot 0.6 - 0.95 \cdot 0.9 \cdot 0.9 \cdot 0.8 \cdot 0.6 = 0.85614 > \delta$.
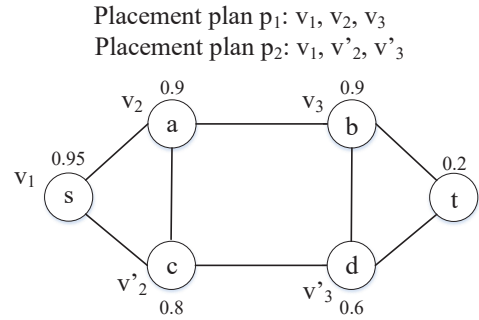

Placement plan p_1: v_1, v_2, v_3
Placement plan p_2: v_1, v_2', v_3'

Fig. 2: An example of partially VM placement availability calculation.

3 SHARED-RISK NODE GROUP

In this section, we assume two types of failures/availabilities, namely Shared-Risk Node Group (SRNG) failures and single node failures/availabilities. A SRNG failure [7] reflects that a certain group of nodes will fail simultaneously, i.e., they have correlated failures. For example, in datacenter networks [8], servers are hosted by different racks, and within the same rack, servers are connected with a Top of Rack (ToR) switch. These ToR switches are further inter-connected through aggregate switches in a tree-like topology. Similarly, these aggregate switches are further connected with several core switches

in the upper layer. In this context, the servers which are hosted by the same rack will fail simultaneously, if their belonged rack fails. Similarly, the failure of a ToR switch(es) will cause the failures of all the racks that are connected with, and hence the failures of their hosted servers. One rack failures then corresponds to one distinct SRNG event. One (server) node can belong to multiple SRNG events (e.g., rack failure, switch failure, etc.). Without loss of generality, we assume there are totally g SRNG events, and the failure probability of SRNG event i is represented by π_i. For the ease of elaboration, we let $\lambda_i = 1 - \pi_i$, which can represent the non-occurring probability of SRNG event i. For each node $n \in \mathcal{N}$, we denote R_n as the set of all the SRNG events it belongs to. The VM placement availability should incorporate the SRNG non-occurring probabilities as well as the node availabilities. As a result, if m nodes with availability $A_1, A_2, \ldots, A_m$ are used for hosting k VMs ($m \leq k$) by a single placement p, then its availability can be calculated as follows:

$$\prod_{i; srng_i \cap p \neq \emptyset} \lambda_i \cdot \prod_{j=1}^{m} A_j \tag{6}$$

Analogously, the protected VM placement (fully and partially) availability can be calculated by substituting Eq. (6) with A_{p_i} in Eq. (5). It is worthy to mention that the operator $\circ$ in Eq. (5) still holds for λ. For example in Fig. 3, each node is assigned with a SRNG event and node availability value, respectively. Suppose $\lambda_1 = 0.999$, $\lambda_2 = 0.99$ and $\lambda_3 = 0.9$. Assume placement p_1 to place v_1 on A and place v_2 on B, placement $p2$ to place v_1 on C and v_2 on B. In this sense, both the SRNG event failure and node availability should only be counted once, especially for $SRNG_2$ and node B (since they both belonged to $p1$ and $p2$). Consequently, the VM placement availability of $p1$ and $p2$ is:

$$1 - (1 - \lambda_1 \circ \lambda_2 \circ A_a \circ A_b) \circ (1 - \lambda_2 \circ \lambda_3 \circ A_b \circ A_c)$$
$$= \lambda_1 \circ \lambda_2 \circ A_a \circ A_b + \lambda_2 \circ \lambda_3 \circ A_b \circ A_c -$$
$$\lambda_1 \circ \lambda_2 \circ A_a \circ A_b \circ \lambda_2 \circ \lambda_3 \circ A_b \circ A_c$$
$$= \lambda_1 \cdot \lambda_2 \cdot A_a \cdot A_b + \lambda_2 \cdot \lambda_3 \cdot A_b \cdot A_c - \lambda_1 \cdot \lambda_2 \cdot A_a \cdot A_b \cdot \lambda_3 \cdot A_c$$
$$= 0.999 \cdot 0.99 \cdot 0.9 \cdot 0.8 + 0.99 \cdot 0.9 \cdot 0.8 \cdot 0.7 -$$
$$0.999 \cdot 0.99 \cdot 0.9 \cdot 0.8 \cdot 0.9 \cdot 0.7 = 0.762432264$$

Placement plan p_1: v_1, v_2
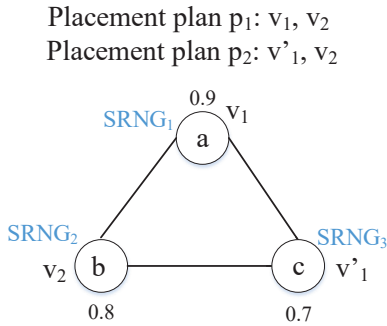Placement plan p_2: v'_1, v_2

Fig. 3: An example of VM placement availability calculation for SRNG.

4 RELIABLE VIRTUAL MACHINE PLACEMENT

4.1 Problem Definition

Formally, the Reliable VM Placement (RVMP) problem is defined as follows:

Definition 1. Given a complete graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ denotes a set of N server nodes and $\mathcal{L}$ represents a set of $L = \frac{N(N-1)}{2}$ links. Each node $N_j \in \mathcal{N}$ has storage upper bound of s_j. For each link $(m, n) \in \mathcal{L}$, a function $F(m, n, \eta, D)$ returns whether m and n can have a connection availability at least η and communication delay at most D. A request is represented by $r(k, c, V, T, A, \delta)$, where k indicates the requested number of VMs V with demanding capacity c_v ($v \in V$), T and A are $k \times k$ matrix, which specifies the delay constraint and connection availability constraint between any two VMs, respectively, and δ denotes requested VM placement availability. The Reliable VM Placement (RVMP) problem is to place at most H sets of k VMs on minimum number of nodes such that:

1) Each node does not exceed its storage limit.
2) Any two VMs i_1 and i_2 under the same placement plan have a communication delay no greater than $T(i_1, i_2)$.
3) Any two VMs i_1 and i_2 under the same placement plan have a connection availability no less than $A(i_1, i_2)$.
4) Its VM placement availability is no less than δ.

In the RVMP problem, we assume that each VM can be placed at up to H different nodes. Moreover, we only consider the server nodes and ignore some other nodes in the network (e.g., router nodes, switch nodes). In fact, the link between each node pair in the RVMP problem actually implies a (set of) path(s) which may traverse some other intermediate nodes. Finding reliable and delay-sensitive paths could be easier within a tree-like datacenter network, but this problem becomes harder when the node pairs are located in different datacenters (a more general network). As we proved in [5], [6], the problem of finding $w \geq 2$ link-disjoint paths for which the connection availability is no less than a given value is already NP-hard and cannot be approximated to arbitrary degree. In this sense, jointly considering the RVMP problem and reliable routing problem will make it even harder to solve. Therefore, we assume $F(m, n, \eta, D)$ is precalcualted by using the algorithms proposed in Section 5, where we will address how to find reliable and delay-sensitive paths by taking link availability and link delay into account.

Theorem 1. The RVMP problem is NP-hard.

  Proof: Let's first introduce the Bin-Packing problem: Given n items with sizes e_1, e_2, ..., e_n, and a set of m bins with capacity c_1, c_2, ...,c_m, the Bin-Packing problem is to pack all the items into minimized number of bins without violating the bin capacity size. If we assume that the $F(m, n, \eta, D) = 1$ for any η and D and all the nodes have availability 1, then the RVMP problem for $H = 1$ turns into the bin-packing problem, which is NP-hard [9]. Next, let's analyze its complexity when the objective of minimizing the number of used nodes is not considered.

- Each node has unlimited storage: In this case, each set of k VMs can be placed on one node and we need to find H nodes in the network to store each set of k VMs. This can be solved in $\binom{N}{H}$ searching when $N > H$ or using N nodes to host N sets of k VMs when $N \leq H$, which is polynomial time solvable.
- Each node has limited storage: Assume $H = 1$ and $A_n = 1 \; \forall n \in \mathcal{N}$. Moreover, assume for a certain D value and $(m, n) \in \mathcal{L}$, $F(m, n, \eta, D)$ keeps the same for any η. That is, the link between each node pair is only assigned with a delay value (connection availability is not taken into account). Under this assumption, Alicherry and Lakshman [10] have proved that the RVMP problem can be reduced to the 3SAT problem, and cannot be approximated to arbitrary degree.

$\square$

The RVMP problem with SRNG failures, is also NP-hard and cannot be approximated to arbitrary degree, when we assume that each node is associated one distinct SRNG event and all the node availabilities are assumed to be 1. In the following, we will devise both exact algorithm and heuristic to solve the RVMP problem.

4.2 Exact Algorithm

In this subsection, we propose an exact Integer Nonlinear Program (INLP) to solve the RVMP problem. We first solve the RVMP problem without SRNG failures and start by explaining the necessary notations and variables:

INLP notations:

$r(k, c, V, T, A, \delta)$: a request r which specifies a set of k VMs V. For each $v \in V$, its demanding capacity is denoted by c_v. T and A indicate the delay and connection availability constraint among different VMs, and δ implies the requested VM placement availability.

$\mathcal{N}, \mathcal{L}$: set of N nodes and set of L links, respectively.

H: The maximum number of times for each VM to be placed in the network.

$F(m, n, \eta, D)$ Returns whether there exists a connection between m and n such that connection availability is at least η and communication delay value is at most D.

λ_i^n: The non-occurring probability of the i-th SRNG if node n belongs to it, otherwise it is 1.

INLP variable:

P_{vn}^h: whether VM v is placed on node n by placement plan h, where $v \in V$, $n \in \mathcal{N}$ and $1 \leq h \leq H$.

Objective:

$$\min \sum_{n \in \mathcal{N}} \left(\max_{1 \leq h \leq H, v \in V} P_{vn}^h \right) \quad (7)$$

Placement constraint:

$$\sum_{n \in \mathcal{N}, 1 \leq h \leq H} P_{vn}^h \geq 1 \quad \forall v \in V \quad (8)$$

Storage constraint:

$$\sum_{v \in V} \left(\max_{h=1}^{H} P_{vn}^h \right) \cdot c_v \leq s_n \quad \forall n \in \mathcal{N} \quad (9)$$

Delay and connection availability constraint:

$$F(m, n, A(m, n), T(m, n)) \cdot P_{am}^h \cdot P_{bn}^h = 1$$
$$\forall 1 \leq h \leq H, (m, n) \in \mathcal{L}, 1 \leq a, b \leq k : a \neq b \quad (10)$$

VM placement availability constraint:

$$\sum_{h=1}^{H} \prod_{n \in \mathcal{N}} \min_{v \in V} \left(1 - P_{vn}^h + P_{vn}^h A_n \right) -$$
$$\sum_{1 \leq h < u \leq H} \prod_{n \in \mathcal{N}} \min \left(\min_{v \in V}(1 - P_{vn}^h + P_{vn}^h A_n), \min_{v \in V}(1 - P_{vn}^u + P_{vn}^u A_n) \right)$$
$$+ \cdots + (-1)^{H-1} \left(\prod_{n \in \mathcal{N}} \min_{1 \leq h \leq H} (\min_{v \in V}(1 - P_{vn}^h + P_{vn}^h A_n)) \right) \geq \delta \quad (11)$$

Eq. (7) minimizes the number of total used nodes. For instance, we first calculate the maximum value of P_{vn}^h for node $n \in \mathcal{N}$, and as long as $P_{vn}^h = 1$ for some $1 \leq h \leq H$ and $v \in V$, it means that node n in use to host VM. After that, we take the sum of $\left(\max_{1 \leq h \leq H, v \in V} P_{vn}^h \right)$ for all the nodes in $\mathcal{N}$ and try to minimize this value. Eq. (8) ensures that each one of k requested VMs must be placed in the network. Eq. (9) ensures that each node does not exceed its storage limit when VMs are place on it. Eq. (10) makes sure that the specified delay constraint and connection availability of any two VMs under the same placement plan is not violated. Eq. (11) ensures that the VM placement availability constraint is obeyed, according to Eq. (5). We note that Eq. (11) can simultaneously calculate the availability of the fully protected placement, partially protected placement, and single placement. For instance when $H = 2$, Eq. (11) becomes:

$$\prod_{n \in \mathcal{N}} \min_{v \in V}(1 - P_{vn}^1 + P_{vn}^1 A_n) + \prod_{n \in \mathcal{N}} \min_{v \in V}(1 - P_{vn}^2 + P_{vn}^2 A_n) -$$
$$\prod_{n \in \mathcal{N}} \min(\min_{v \in V}(1 - P_{vn}^1 + P_{vn}^1 A_n), \min_{v \in V}(1 - P_{vn}^2 + P_{vn}^2 A_n)) \geq \delta \quad (12)$$

When $P_{vn}^1 = P_{vn}^2$ for all $n \in \mathcal{N}$, Eq. (12) becomes

$$\prod_{n \in \mathcal{N}} \min_{v \in V}(1 - P_{vn}^1 + P_{vn}^1 A_n) \geq \delta$$

which is the VM placement availability constraint for the single placement.

To solve the RVMP problem with SRNG failure, we need to rewrite Eq. (11) in Eq. (13) and keep all the objective and other constraints same (Eq. (7)-Eq. (10)).

4.3 Heuristic Algorithm

Our proposed heuristic to solve the RVMP problem, called the Delay-Sensitive and reliable (DSR) placement algorithm, is shown in Algorithm 1. Instead of placing VMs on nodes, the logic of DSR is to assign nodes to VMs until all the VMs are covered by the nodes without violating delay and connection availability constraint. Since we want to use least number of nodes to host VMs to satisfy the availability, we gradually increase the amount of finding placement plans. In what follows, we explain each step of the heuristic algorithm, where the the SRNG failures are first not considered.

$$\sum_{h=1}^{H} \prod_{n\in\mathcal{N}} \min_{v\in V} \left(1 - P_{vn}^h + P_{vn}^h A_n\right) \cdot \prod_{1\leq i\leq g} \min_{v\in V, n\in\mathcal{N}} \left(1 - P_{vn}^h + P_{vn}^h \lambda_i^n\right) -$$

$$\sum_{1\leq h<u\leq H} \prod_{n\in\mathcal{N}} \min\left(\min_{v\in V}(1 - P_{vn}^h + P_{vn}^h A_n), \min_{v\in V}(1 - P_{vn}^h + P_{vn}^h A_n)\right) \cdot \prod_{1\leq i\leq g} \min(\min_{v\in V, n\in\mathcal{N}} \left(1 - P_{vn}^u + P_{vn}^u \lambda_i^n\right), \min_{v\in V, n\in\mathcal{N}} \left(1 - P_{vn}^u + P_{vn}^u \lambda_i^n\right))$$

$$+ \cdots + (-1)^{H-1} \left(\prod_{n\in\mathcal{N}} \min_{1\leq h\leq H} (\min_{v\in V}(1 - P_{vn}^h + P_{vn}^h A_n))\right) \cdot \prod_{1\leq i\leq g} \min_{1\leq h\leq H} \left(\min_{v\in V, n\in\mathcal{N}} \left(1 - P_{vn}^h + P_{vn}^h \lambda_i^n\right)\right) \geq \delta \tag{13}$$

Algorithm 1: DSR($\mathcal{G}(\mathcal{N}, \mathcal{L}), r(k, c, V, T, A, \delta), H, \alpha$)

---

1   $VP[h][v][n] \leftarrow 0 \; \forall 1 \leq h \leq H, |v| = k, |n| = N$
2   **for** $h \leftarrow 1$ **to** H **do**
3     $VP[h] \leftarrow$ DSRPlace($\mathcal{G}(\mathcal{N}, \mathcal{L}), r(k, c, V, T, \delta), H, \alpha$)
4     $\mathcal{N} \leftarrow \mathcal{N}\backslash\mathcal{N}_x$, where $\mathcal{N}_x$ denotes a subset of the used nodes for already found placement plans.
5   **if** $1 - \coprod_{i=1}^{H}(1 - A_{VP[i]}) \geq \delta$ **then**
6   Call PartiallyDSRPlace($\mathcal{G}, VP[h][k][N], r, H$)
7   Return null

Algorithm 2: DSRPlace($\mathcal{G}(\mathcal{N}, \mathcal{L}), r(k, c, V, T, A, \delta), H, \alpha$)

---

1   **foreach** v_m *in* V $(1 \leq m \leq k)$ **do**
2    $v_x \leftarrow v_m, Q \leftarrow \emptyset, \mathcal{G}^m \leftarrow \mathcal{G}, P^m[V][N] \leftarrow 0$
3   **while** Q.Count $< k$ **do**
4   Sort the nodes in $\mathcal{G}^m$ by their availabilities in the decreasing order $n_1, n_2,\ldots,n_N$
5   Find one node n_a with maximum availability to host v_x without violating the delay and connection availability constraint with already placed VMs, such that $s_{n_a} \geq c_{v_x}$
6   **if** *Step 5 succeeds* **then**
7        $P^m[v_x][n_a] \leftarrow 1, s_{n_a} \leftarrow s_{n_a} - c_{v_x},$ $A_{n_a} \leftarrow 1, Q$.Add(v_x)
8   **else**
9   Break;
10     $D \leftarrow +\infty;$
11   **foreach** v_i *in* $V\backslash Q$ **do**
12   **foreach** v_j *in* Q **do**
13   **if** $D > \frac{T(v_i, v_j) \cdot \alpha}{A(v_i, v_j)}$ **then**
14          $D \leftarrow D > \frac{T(v_i, v_j) \cdot \alpha}{A(v_i, v_j)}, v_x \leftarrow v_i$
15   Return P^m with the maximum availability.

Algorithm 3: PartiallyDSRPlace($\mathcal{G}, VP[H][k][N], r, H$)

---

1   Sort the nodes that host VMs in increasing order by nodes' availabilities. Denote this set as $\mathcal{N}_y$.
2   **foreach** $n \in \mathcal{N}_y$ **do**
3     $VB \leftarrow VP$
4     $VB[h][v][n] \leftarrow 0$ for $1 \leq h \leq H, v \in V$ and $n \in \mathcal{N}$
5   **foreach** *placement plan* $h = 1...H$ **do**
6   Try to use its other used nodes to host the VMs that are originally placed by it on n if possible.
7   **if** $1 - \coprod_{i=1}^{H}(1 - A_{VB[i]}) \geq \delta$ **then**
8       $VP \leftarrow VB, VB \leftarrow \emptyset$.
9   Return VP

In Step 1 of Algorithm 1, we first initialize a binary variable $VP[h][v][n]$ representing whether VM $v \in V$ is hosted by node $n \in \mathcal{N}$ under the plan h. After that, for placement plan h, we call Algorithm 2 to place VMs on nodes in Step 3. The purpose of Step 4 is to avoid different plans to have the same placement result. But this will only happen when single node's free capacity is far greater than the VM demanding capacity. That is, all the VMs can be placed on the same node and its remaining free capacity is still large enough so that another set of k VMs can be placed on it. In Step 5, we calculate the availability of $VP[h]$ (containing h placement plans). If availability value of these h placement plans is no less than δ, we call Algorithm 3

trying to return a partially VM placement solution in order to further reduce the number of used nodes. In Algorithm 3, for each node $n \in \mathcal{N}$, we first clear all the VMs resided on it. For each placement plan p, it tries to use its other used nodes to host the VMs that are originally placed by it on n. For simplicity, we apply a greedy approach: for each one of used nodes by placement plan h, it tries to host the VMs originally placed by it from n as many as possible. After that, we calculate whether the whole availability still satisfies δ. If so, we assign this partially placement solution to VP, otherwise we return the original fully placement VP. Next, we will explain the details of Algorithm 2, which is to find single placement.

In Step 1 of Algorithm 2 we start with each $v_m \in V$, assign it to v_x in Step 2. We use a queue Q to store already placed VMs, and initially it is set empty. Besides, we also define variable $P^m[v][n]$ to indicate whether VM $v \in V$ is hosted by node $n \in \mathcal{N}$ corresponding to the placement plan starting with VM v_m. As long as Q's count is less than k, Step 4-Step 9 are going to assign nodes to host unassigned VMs. Step 5 tries to find a node n_a with maximum availability whose capacity should be at least $c(v_x)$. Moreover, if v_x is placed on n_a, it should not violate the delay and connection availability constraint with already hosted VMs. If it succeeds, in Step 7, the capacity of n_a is reduced by c_{v_x}, the availability of n_a is changed to 1, and v_x is added to Q. The reason to change node's availability is that if some nodes have been used to host the existing VM(s), then the availability for these nodes to host other (unassigned) VMs is 1. So we need to change its "availability" after each iteration of covering VM(s). If such node cannot be found in Step 5, it indicates that not all the VMs are covered and we regard this placement plan should "sharely" place uncovered VMs with one of $H - 1$ placement plans found in Algorithm 1. The algorithm then breaks in Step 9. Following

that, Step 10-Step 14 search for an unsigned VM, which has the smallest value of $\frac{T(v_i,v_j)\cdot\alpha}{A(v_i,v_j)}$ to the already placed VMs, where α is a user given value. By doing this, we want to find an unsigned VM that has "smaller" delay constraint and "greater" connection availability with already placed VMs, and assigns it to v_x. The motivation here is that we always first place the VM which has a more critical requirement in terms of delay and connection availability. When Q's count is equal to k, it indicates that all the VMs have been hosted, which means we get a "complete" placement plan. Finally, in Step 15, the algorithm returns a placement plan with the biggest availability from k already determined single placements.

To solve the RVMP problem with SRNG failure, Alg. 1-3 keep the same except:

- In Step 4 of Algorithm 2 and Step 1 of Algorithm 3, we sort the nodes in $\mathcal{G}^m$ by the product of their availabilities and non-occurring probabilities of all their belonged SRNG events in decreasing order and increasing order, respectively.
- In Step 5 of Algorithm 2, we will find one node n_a with maximum node availability multiplied by the non-occurring probabilities of SRNG events set $R_{n_a}\backslash R_x$, where R_x denotes the set of SRGN events that n_a belongs to but has already been considered/counted by the other nodes from Step 4 to Step 14. The reason is that one unique SRGN event can only occur once, we cannot calculate its value under the same placement plan more than once.

The time complexity of Algorithm 2 can be calculated like this: There are k VMs in total in Step 1, and Step 3 has also k iterations. Sorting algorithm for instance like insertion sort in Step 4 takes $O(N\log(N))$ time, and Step 5 has a complexity of $O(N)$. Step 9-Step 13 consume at most $O(k^2)$ time. Therefore, the whole complexity of Algorithm 2 is $O(k^2(N\log(N)+k^2))$. In Algorithm 3, Step 1 consumes $O(N\log(N))$ time via insertion sort and Step 2-8 consume $O(N^2H)$ time, leading to a whole complexity of $O(N(\log N + NH))$. Consequently, the whole time complexity of Algorithm 1 is $O(k^2H(N\log(N)+k^2))$, since it calls at most H times of Algorithm 2.

4.4 Simulation

The simulation is run on a desktop PC with 2.7 GHz and 8 GB memory. We use IBM ILOG CPLEX 12.6 to implement the proposed INLP and C# to implement the heuristics.

We set $\alpha = 1$ for our heuristic DSR. We compare our exact INLP and heuristic DSR with two heuristics, namely (1) Greedy Placement (GP) and (2) Random Placement (RP). These 2 algorithms follow the similar routine with Algorithm 1, except: (1) In Step 6, they directly return the placement result if its availability is satisfied, instead of checking partially placement solution, and (2) they call different heuristics in Step 3 (different from Algorithm 2), which we specify as follows:

- GP: It first selects a node with greatest availability and places as many VMs as possible on it under its storage limit. It then selects the second largest availability node and places remaining VMs as many

as possible, which should also satisfy the delay and connection availability constraints with already placed VMs. This procedure continues until all the VMs are placed or all the nodes have been iterated.
- RP: It first randomly selects a node, and places as many VMs as possible on it without exceeding its capacity. Then it randomly selects the second node, and places remaining VMs as many as possible without violating node's capacity constraint and the delay and connection availability constraints with already placed VMs. This procedure continues until all the VMs are placed or all the nodes have been iterated.

In the following, we first testify the algorithms for the RVMP problem without SRNG failures for both 16-node and 200-node networks, and then evaluate them for the RVMP problem with SRNG failures.

4.4.1 16-node network without SRNG failures

We first conduct simulations on a 16-node network. If we set c (VM demanding capacity) relatively too small, then by placing as many VMs as possible on one node it may return a solution. If we set c relatively too big, then the solution may not exist. Therefore, we let the node's capacity is at most three times of the requested bandwidth of one single VM, by which we want to challenge the algorithm to find the solution. Consequently, the simulation parameters are set like this: the node capacities are randomly distributed between 100 and 200 units, and the node availabilities are randomly distributed among the set of $\{0.99, 0.999, 0.9995, 0.9999\}$. For each request $r(k, c, V, T, A, \delta)$, $k \in [3, 5]$, $c \in [60, 130]$, each element in the delay matrix T is between 15 and 25, each element in the connection availability matrix A is among the set $\{0.999, 0.999\}$, and δ is in the set of $\{0.999, 0.9999, 0.99999, 0.999999\}$. We randomly generate 100 requests for $k = 3, 4, 5$, respectively. With respect to $F(m, n, \eta, D)$, when $\eta = 0.999$ for link (m, n), it returns true when the delay is at most D, where D is randomly chosen between $[10, 20]$, otherwise it returns false; when $\eta = 0.9999$ for link (m, n), it returns true when the delay is at most D, where D is randomly chosen between $[20, 30]$, otherwise it returns false. We set $H = 2$ and 3.

We first evaluate the performance of the algorithms in terms of Acceptance Ratio (AR), which is defined as the number of accepted requests over all the requests. Figs. 4(a) and 5(a) show that the exact INLP always achieves the highest AR. DSR has a close performance with INLP, and it outperforms all the other two heuristics. Besides, we notice that for the same algorithm, it achieves higher AR value when H increases, since more VM replicas are allowed to be placed for a higher H.

Next, we compare the algorithms in terms of Average Number of Used Nodes (ANUN). The ANUN is defined as the total number of nodes consumed by all the accepted requests divided by the number of accepted requests. To have a fair comparison, we only consider the requests accepted by all these four algorithms. We omit the ANUN value for RP when $k = 4, 5$ and $H = 2$, and when $k = 5$ and $H = 3$, since the number of their common accepted requests are too small. From Figs. 4(b) and 5(b), we see that
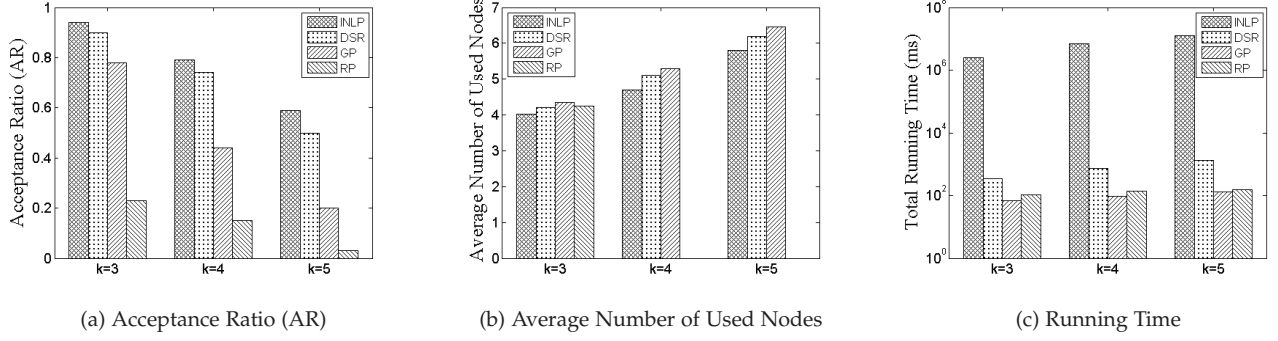
Fig. 4: Simulation results over 100 requests when $H = 2$ for 16-node network: (a) Acceptance Ratio (AR) (b) Average Number of Used Nodes (ANUN) (c) Running Time.
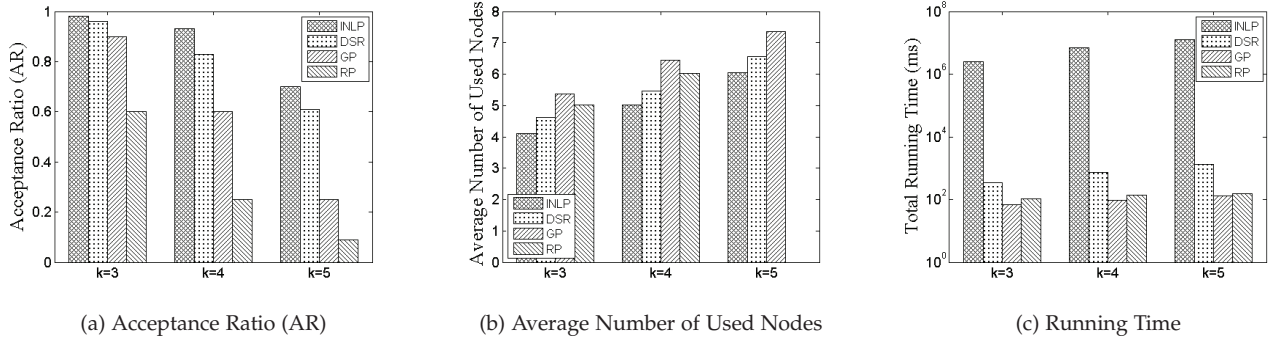

Fig. 5: Simulation results over 100 requests when $H = 3$ for 16-node network: (a) Acceptance Ratio (AR) (b) Average Number of Used Nodes (ANUN) (c) Running Time.
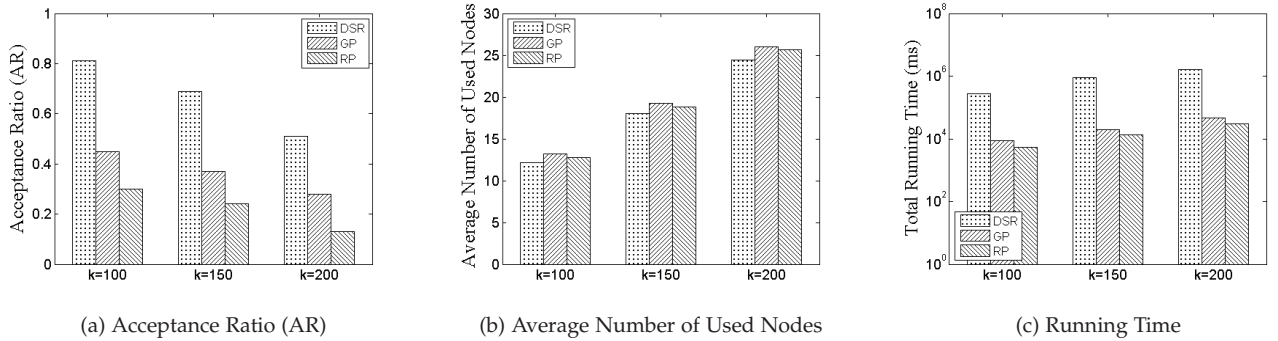

Fig. 6: Simulation results over 100 requests when $H = 2$ for 100-node network: (a) Acceptance Ratio (AR) (b) Average Number of Used Nodes (ANUN) (c) Running Time.

the INLP achieves the minimum value of ANUN, and our proposed DSR obtains the second lowest ANUN value. RP obtains a lower ANUN value than GP when $k = 4$ and $H = 3$, since it is regarded to place more shared VMs. From above, we observe that even under the constrained simulation setup, the exact INLP can always accept most requests and consume least amount of nodes as well, which validates its correctness.

Finally, Figs. 4(c) and 5(c) present the total running time over 100 requests (in log scale). The INLP is significantly more time consuming than all the other 3 heuristics. The DSR, on the other hand, has a slightly higher running time

than the other two heuristics, but it pays off by having a higher AR as shown in Figs. 4(a) and 5(a), and lower ANUN shown in Figs. 4(b) and 5(b).

4.4.2 100-*node network without SRNG failures*

In this subsection, we simulate a 100-node complete graph, where the node capacities are randomly distributed in $[1000, 2000]$. The other simulation setup follows the same with Section 4.4.1, except that $k = 100, 150, 200$ VMs are requested for each 100 requests, respectively. Since the problem sizes increases largely, the INLP becomes very time consuming and it keeps running for at least 1 day without
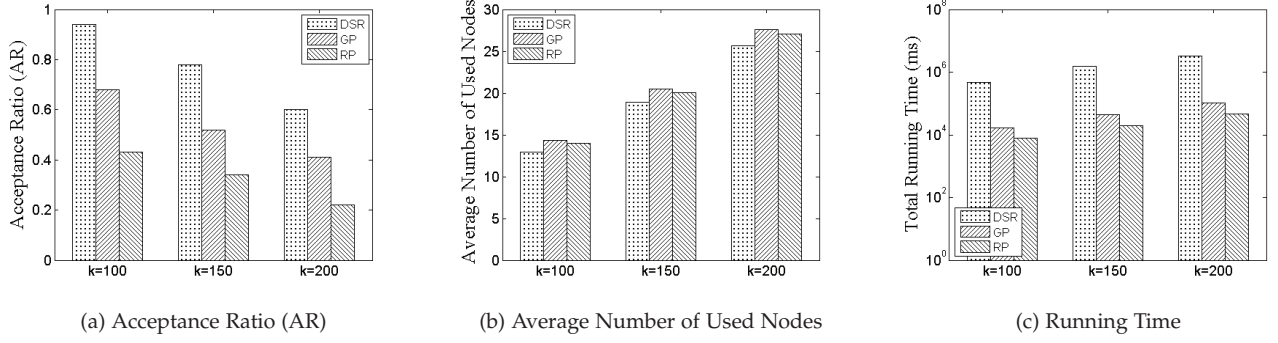
(a) Acceptance Ratio (AR)   (b) Average Number of Used Nodes   (c) Running Time

Fig. 7: Simulation results over 100 requests when $H = 3$ for 100-node network: (a) Acceptance Ratio (AR) (b) Average Number of Used Nodes (ANUN) (c) Running Time.
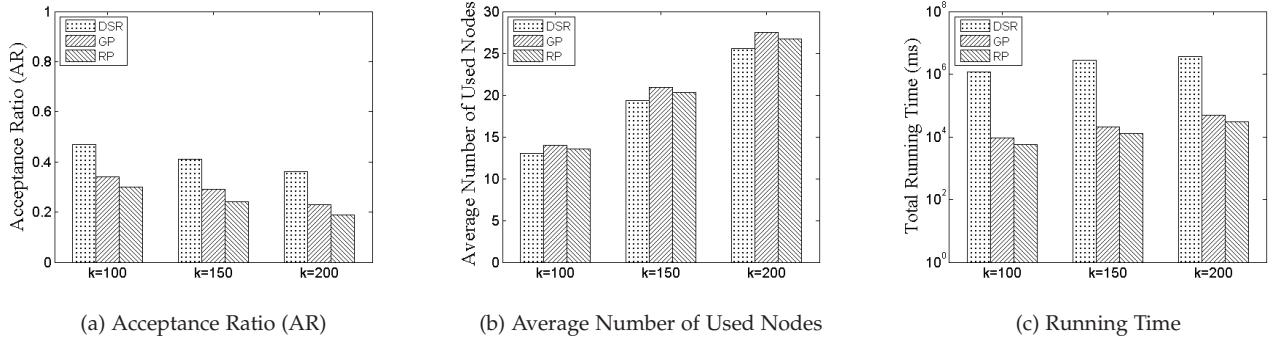

(a) Acceptance Ratio (AR)   (b) Average Number of Used Nodes   (c) Running Time

Fig. 8: Simulation results over 100 requests when $H = 2$ for 100-node network with SRNG failures: (a) Acceptance Ratio (AR) (b) Average Number of Used Nodes (ANUN) (c) Running Time.

returning a feasible solution. We therefore only evaluate the heuristic algorithms. Figs. 6 and 7 depict the AR, ANUN and running time (in log scale) of all these algorithms. Similar to Section 4.4.1, DSR always achieves better performance than the other 2 heuristics in terms of AR (See Figs. 6(a) and 7(a)) and ANUN (See Figs. 6(b) and 7(b)). On the other hand, Figs. 6(c) and 7(c) show that DSR is more time consuming than the other heuristics, but it is still acceptable since it acquires higher AR and lower ANUN value.

4.4.3 Shared-Risk Node Group

It is assumed that there are in total 15 SRNG events, and each SRNG event occurs with the probability in the set of $\{0.000001, 0.000002, 0.000003, 0.000004, 0.000005\}$. Each server node is associated with at most 5 SRNG events. The other simulation setup follows the same with Section 4.4.2. Since more SRNG events are induced for each node, the total VM placement availability for the same set of nodes will decrease according to Eq. (6), causing the optimal solution does not exist for when $\delta > 0.9999$. Due to space limit, we therefore only present the results for $H = 2$ in Fig. 8. Similar to Sections 4.4.1 and 4.4.2, DSR can obtain better performance than the other 2 heuristics in terms of AR (See Fig. 8(a)) and ANUN (See Fig. 8(b)), but this comes at the expense of higher running time as shown in Fig. 8(c) (in log scale). Due to the reason of incurring SRNG events, for all three algorithms, we can see that the achieved AR value in

Fig. 8(a) is lower than Fig. 6(a), and the obtained ANUN value in Fig. 8(b) is higher than Fig. 6(b).

In all, we conclude that the exact INLP can be used as an optimal solution when the computation speed is not a big concern. However, as the problem size increases, its running time will increase exponentially. On the contrary, our proposed DSR is a good compromise between performance and running time, and it is the preferred choice for when VM placement request needs to be computed on-the-fly.

5 AVAILABILITY-BASED DELAY-CONSTRAINED ROUTING PROBLEM

In the RVMP problem, we do not consider the link delay and availability, and assume that the function $F(m, n, \eta, D)$ is given. In this section, we study how to find a connection over at most w link-disjoint paths between a node pair, such that the connection availability is no less than η and each path delay is no greater than D. For completeness, let's first formulate the link failure scenarios and respective connection availability calculation, which is also introduced in [5], [6].

5.1 Link Failure Scenarios

Analogous to the node failure scenarios, we also address two link failure scenarios. For simplicity, suppose there are two (fully) link-disjoint paths ψ_1 and psi_2, and the availability of link l is denoted as $A_l = 1 - f_l$, where

$0 < A_l \le 1$ and f_l is the failure probability of link l. Then their total availability A_{FD}^2 can be computed based on the following scenarios:

- **Single-link failure**: Analogous to the single node failure, here a path ψ's availability (denoted by A_ψ) is equal to its lowest traversed link availability (highest failure probability). Using two disjoint paths will therefore lead to a total connection availability of 1. In Appendix B, we will address the ABDCR problem under the single-link failure scenario.

- **Multiple link failures:** This is a more general scenario where at one certain point in time, several links in the network may fail simultaneously. Hence, for a path ψ, its availability A_ψ should take into account all its links' availabilities, i.e., $A_\psi = \prod_{l \in \psi} A_l$. Consequently, $A_{FD}^2 = 1 - (1 - A_{\psi 1})(1 - A_{\psi 2})$, which indicates the probability that at least one of the two disjoint paths is available. In this paper, we assume multiple link failures may occur.

5.2 Connection Availability and Problem Definition

Similar to the node availability, we assume that the link availability is equal to the product of availabilities of all its components (e.g., amplifiers). If a path p contains the links $l_1, l_2, l_3, \ldots, l_m$, and their corresponding (independent) availabilities are denoted by $A_{l_1}, A_{l_2}, A_{l_3}, \ldots, A_{l_m}$, then the availability of this (unprotected) path (represented by A_ψ) is equal to $A_\psi = A_{l_1} \cdot A_{l_2} \cdot A_{l_3} \cdot \cdots \cdot A_{l_m}$. If we take the $-\log$ of the link availabilities, finding a path with the highest availability turns into a shortest path problem.

When, for a single connection (i.e., a single path), there are $w \ge 2$ paths $\psi_1, \psi_2, \ldots, \psi_w$ with availabilities represented by $A_{\psi_1}, A_{\psi_2}, \ldots, A_{\psi_w}$, the connection availability indicates the probability that at least one path is operational. We consider two cases, namely: (1) fully link-disjoint paths: these w paths have no links in common, and (2) partially link-disjoint paths: at least two of these w paths traverse a common link. In case (1), the availability (represented by A_{FL}^w) can be calculated as follows:

$$A_{FL}^w = 1 - \prod_{i=1}^{w}(1 - A_{\psi_i}) \qquad (14)$$

If we use Eq. (14) to calculate the availability for the partially link-disjoint case, the probability that the overlapping links operate (or the availability of the overlapping links) will be counted more than once. To solve this, we can analogously apply the operators $\circ$ and $\coprod$ introduced in Section 2. Assume there are w partially link-disjoint paths $\psi_1, \psi_2, \ldots, \psi_w$, the availability (represented by A_{PL}^w) of w partially link-disjoint paths can now be represented as:

$$A_{PL}^w = 1 - \coprod_{i=1}^{w}(1 - A_{\psi_i}) \qquad (15)$$

Next, we formally define the Availability-Based Delay-Constrained Routing (ABDCR) problem as follows:

Definition 2. Given a network represented by $G(\mathcal{N}, \mathcal{L})$ where $\mathcal{N}$ represents the set of N nodes and $\mathcal{L}$ denotes the set of L links, and each link l has an availability value A_l and a delay value d_l. For a communication request represented by $r(s, t, \eta, D)$, where s and t denote the source and destination, η $(0 < \eta \le 1)$ represents the connection availability requirement and D indicates the delay constraint, the Availability-Based Delay-Constrained Routing (ABDCR) problem is to establish a connection over at most w (partially) link-disjoint paths, such that the connection availability is at least η and each path has a delay no greater than D.

In the ABDCR problem, we regard that each request corresponds to the communication between each VM pair which are resided on different nodes. When, the delay constraint is not imposed on each path, the ABDCR problem turns into the Availability-Based Path Selection (ABPS) problem [5], [6]. In [5], [6] we have proved that the ABPS problem is NP-hard for $w \ge 2$ and cannot be approximated into arbitrary degree. Therefore, the ABDCR problem for $w \ge 2$ is also NP-hard and cannot be approximated into arbitrary degree. When $w = 1$, the ABDCR problem turns into the Multi-Constrained Routing problem, which is also NP-hard [11]. In all, the ABDCR problem is NP-hard. In the following, we propose an exact algorithm to solve the ABDCR problem.

5.3 Exact Algorithm

To solve the ABDCR problem exactly, we apply a modified Dijkstra's algorithm by letting each node store as many subpaths as possible, which is similar to the exact algorithm for solving the multi-constrained routing problem [11]. We start with some notations used in the algorithm:

$sus[u][m]$: the parent node of node u for its stored m-th subpath from s to it.
$avb[u][m]$: the availability value stored at node u for its stored m-th subpath from s to it.
$delay[u][m]$: the delay value stored at node u for its stored m-th subpath from s to it.
$counter[u]$: the number of stored subpaths of node u.
$sp[u][m]$: node u's stored m-th subpath from s to it.
$adj(u)$: the set of adjacent nodes of node u.

The pseudo code of the exact algorithm for solving the ABDCR problem when $w = 1$ is given in Algorithm 4.
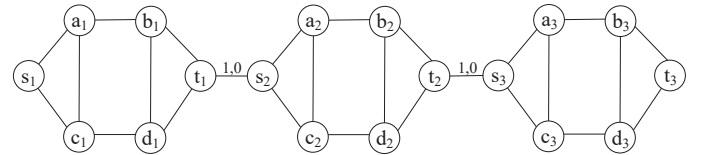

Fig. 9: An example of graph transformation for solving the ABDCR problem when $H = 3$.

When $w > 1$ in the ABDCR problem, we could first duplicate the originate graph $\mathcal{G}$ into w copies $\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^w$. After that, we create link (t_i, s_j) to connect each graph copy with availability 1 and delay 0 for each $1 \le i \le w$, $j = i + 1$ except for $i = w$, where $n \in \mathcal{G}$ and $n_i \in \mathcal{G}^i$. By doing this, we obtain an auxiliary graph $\mathcal{G}^l$, where the source node is s_1

Algorithm 4: ABDCRw1(G, s, t, η, D)

---

1   $Q \leftarrow s$, $avb[s][1] \leftarrow 1$, $delay[s][1] \leftarrow 0$,
    $avb[i][m] \leftarrow +\infty$, $sus[i][m] \leftarrow i$, $counter[s] \leftarrow 1$,
    $counter[i] \leftarrow 0$, $\forall i \in \mathcal{N} \backslash \{s\}$

2   **while** $Q \neq \emptyset$ **do**

3      $u[m] \leftarrow$ Extract-min(Q)

4   **if** $u == t$ *and* $avb[u][m] \geq \eta$ **then**

5   Return the path $sp[u][m]$

6   **else**

7   **foreach** $v \in adj(u)$ **do**

8   **if** $delay[u][m] + d_{uv} \leq D$ **then**

9              $counter(v) = counter(v) + 1$

10   Assign the availability of subpath
              $sp[u][m]$-(u, v) to $avb[v][counter(v)]$;

11             $delay[v][counter(v)] \leftarrow delay[u][m] + d_{uv}$

12             $sus[v][counter(v)] \leftarrow u$

13   Insert $(Q, v, counter(v))$

---

and the destination node(s) is t_i. In this context, if we find a path ψ from s_1 to t_i in $\mathcal{G}^l$, we can map ψ into the original graph $\mathcal{G}$, where we can get i (link-disjoint) paths from s to t. For example, Fig. 9 depicts the resulted auxiliary graph after transformation from the topology shown in Fig. 2 when $H = 3$. Suppose we find a path ψ_1: s_1-a_1-b_1-t_1-s_2-c_2-d_2-t_2, then it is equivalently to say that we find two link-disjoint paths s-a-b-t and s-c-d-t. On the other hand, if there is a path ψ_2: s_1-a_1-b_1-t_1-s_2-a_2-b_2-t_2, then it can be mapped to one same path s-a-b-t in the original graph. In this context, we regard the availability of ψ_2 in $\mathcal{G}^l$ the same with s-a-b-t in $\mathcal{G}$.

As a result, we could slightly modify Algorithm 4 to solve the ABDCR problem for $w > 1$ as follows ($1 \leq i \leq w$):

- In Step 4, the condition should only be **If** $u == t_i$ and the availability of i link-disjoint paths (after mapping them to the original graph) is greater than or equal to η.
- In Step 10, for each node v_i, its avb value is calculated based on i link-disjoint paths after mapped to $\mathcal{G}$ together with the subpath from s_i to v_i, according to Eq. (5). For example, suppose in Step 3 node a_2 is extracted and its stored subpath is s_1-a_1-b_1-t_1-s_2-a_2. Now in Step 7, suppose the neighbor node c_2 is selected to update its avb value according to node a_2. It can be seen that the subpath from s_1 to c_2 is s_1-a_1-b_1-t_1-s_2-a_2-c_2, which are paths s-a-b-t and s-a-c after mapped to the original graph.
- In step 11, the selected node s_i does not update its $delay$ value and sets $delay[s_i][counter(s_i)] = 0$.

It is worthy to mention that our proposed exact algorithm can also solve the ABDCR problem in Shared-Risk Link Group (SRLG) networks. Similar to SRNG, the links in the same SRLG will fail simultaneously if their corresponding group fails. For example, in optical networks [12], several fibers may reside in the same duct and a cut of the duct would cut all fibers in it. One duct in this context corresponds to one distinct SRLG. To solve it, we only need to change the connection availability calculation in Step 4 of

Algorithm 4. More details of the connection availability in SRLG networks can be found in [5], [6].

The time complexity of algorithm 4 can be computed as follows. Let $Z_{\max}$ denote the maximum number of subpaths for each node to store, then in Step 2, Q contains at most $Z_{\max}N$ subpaths. According to [13], $Z_{\max} \leq \lfloor e(N - 2)! \rfloor$, where $e \approx 2.718$. When using a Fibonacci heap to structure the heap, selecting the minimum cost path has a time complexity of $O(\log(Z_{\max}N))$ [14] in Step 3. Step 7-Step 13 take at most $O(Z_{\max})$ time for each link to be iterated and hence result in $O(Z_{\max}L)$ time; because for a fixed link, the steps within the inner loop (Steps 8-12) all cost $O(1)$ time. Hence, the overall time complexity of Algorithm 4 is $O(Z_{\max}N \log(Z_{\max}N) + Z_{\max}L)$. Accordingly, when $w > 1$ in the exact algorithm, the overall time complexity will be $O(Z_{\max}wN \log(Z_{\max}wN) + Z_{\max}wL)$.

5.4 Heuristic algorithms

We propose two heuristic algorithms to solve the NP-hard ABDCR problem. The first one is called SeqTAMCRA: it leverages on TAMCRA [11], which is a heuristic to solve the multi-constrained routing problem. The procedure of SeqTAMCRA is like this: it iteratively runs TAMCRA, so in each iteration we may obtain a path with the biggest availability and delay no greater than D. After each iteration, the traversed links will be pruned. This procedure continues until the connection availability is satisfied or the number of paths is greater than w.

The second heuristic is called TADRA, Tunable Availability-based Delay-constrained Routing Algorithm, and it is identical to the exact algorithm except that the number (variable $counter$) of stored paths for each node should not exceed a given value (say M). For instance, Step 8 of Algorithm 4 should be rewritten as:
if $counter(v) \leq M \&\&delay[u][m] + d_{uv} \leq D$ **then**.

5.5 Simulation

In order to verify the proposed algorithms, we conduct simulations on two networks[2]: USANet, displayed in Fig. 10, which is a realistic carrier backbone network, and GÉANT, shown in Fig. 11, which is a pan-European communications infrastructure. The link availability is in the set of $\{0.99, 0.999, 0.9999\}$, and the link delay is set between 10 and 25. We randomly generate 1000 requests, and for each request $r(s, t, \eta, D)$, s and t are randomly generated, η is among the set $\{0.9995, 0.9996, 0.9997, 0.9998, 0.9999\}$, and $D \in [15, 25]$. We set $w = 1, 2, 3$. For both SeqTAMCRA and TADRA, the maximum number of stored paths is set to wN, where w is the number of maximum link-disjoint paths and N is the number of nodes in the network.

Figs. 12 and 13 depict the Acceptance Ratio (AR) of the 3 algorithms, which is defined by the number of accepted requests divided by the total number of requests. The exact algorithm can always achieve the highest AR value, which also verifies its correctness. TADRA obtains close to optimal performance when $w = 2$ and $w = 3$, while SeqTAMCRA

---

2. Since most of typical datacenter network's topologies are tree-like (e.g., Fat-Tree, BCube), the number of link-disjoint paths between node pairs are limited. Hence, to examine the algorithms more thoroughly, we choose well-connected backbone networks for the evaluation.
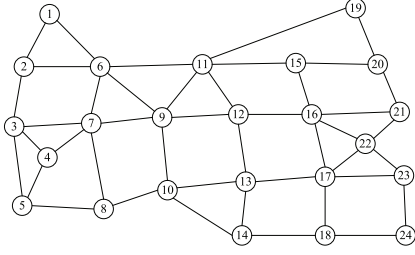
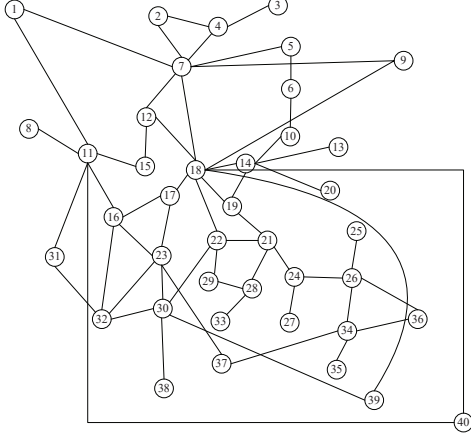Fig. 10: USA carrier backbone network.

Fig. 11: GÉANT pan-European research network.

performs well when $w = 1$. The reason is that when $w = 1$, leveraging the TAMCRA, SeqTAMCRA dynamically maintain wN best paths for each node compared to the TADRA, so it can achieve a better performance. When $w > 1$, after finding a feasible path by SeqTAMCRA, pruning the used links will prevent it to find link-disjoint paths in some cases, leading to a worse performance. However, dynamically maintaining fixed number of best paths only works for when $w = 1$, since the connection availability calculation is non-linear when $w \geq 2$. Therefore, the exact algorithm and TADRA cannot adopt this technique to improve the efficiency for finding feasible paths. Nevertheless, we could jointly use TADRA when $w > 1$ and SeqTAMCRA when $w = 1$ as a heuristic combination.

When the solution does not exist, the exact algorithm needs much longer time (several hours or more in some
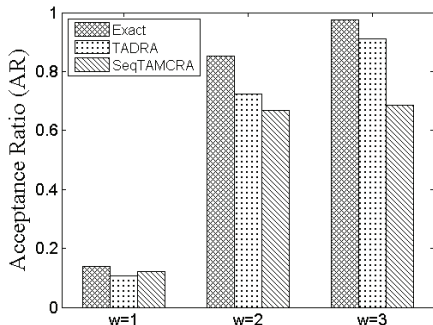

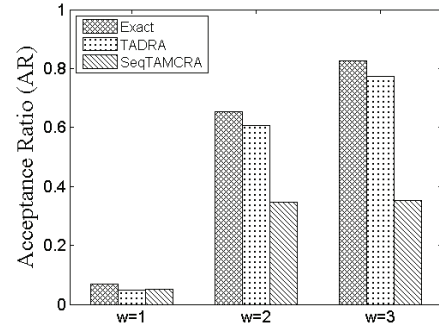Fig. 12: Acceptance ratio for the algorithms in USANet.

Fig. 13: Acceptance ratio for the algorithms in GÉANT.

cases) to return, we therefore only show the algorithms' running times (in log scale) for all their accepted requests in Fig. 14. Even this, we see that the exact algorithm is still more time consuming than the others. TADRA consumes higher running time than SeqTAMCRA when $w > 1$, since its graph (input) size increases w times than $w = 1$.
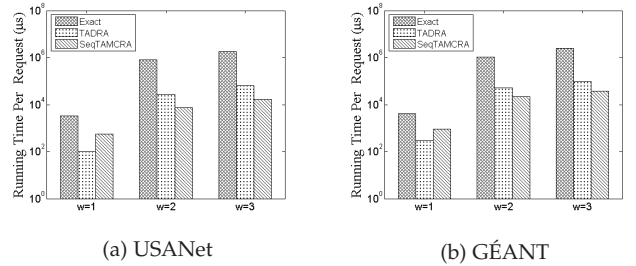

(a) USANet   (b) GÉANT

Fig. 14: Running time per request in two networks: (a) USANet (b) GÉANT .

6   RELATED WORK

A high-level comprehensive survey about VM placement can be found in [15] [16].

6.1   Network-Aware VM Placement

Alicherry and Lakshman [17] deal with 3 problems of placing VMs in geo-distributed clouds. They first study how to place requested VMs on distributed datacenter nodes such that the maximized length (e.g., delay) of placed VM pairs is minimized. A 2-approximation algorithm is proposed to solve this problem when a triangle link length is assumed. They subsequently study how to place VMs on physical machines (racks and servers) within a data center in order to minimize inter-rack communication. Assuming the topology of the data center is a tree, they devise an exact algorithm to solve this problem. Finally, they propose a heuristic for partitioning VMs into disjoint sets (e.g., rack) such that communication between VMs belonging to different partition is minimized.

Biran *et al.* [18] address the VM placement problem by minimizing the min-cut ratio in the network, which is defined as the used capacity of the cut links consumed by the communication of VMs divided by the total capacity

of the cut links. They prove this problem is NP-hard and propose two efficient heuristics to solve it. Jiang *et al.* [19] jointly consider VM placement and routing problem within one data center network. They propose an approximation on-line algorithm leveraging the technique of Markov approximation.

Meng *et al.* [20] address the problem of assigning VMs to slots (CPU/memory on a host) within a datacenter network in order to minimize total network costs. A one to one mapping function is defined as $\pi(x) \rightarrow y$, indicating whether VM x is placed at slot y, where $x, y \in [1, \ldots, k]$ and k denotes the total number of requested VMs and slots. In this context, the network cost is calculated as $\sum D_{ij} C_{\pi(i)\pi(j)}$, where D_{ij} denotes the traffic rate between VMs i and j, and $C_{\pi(i)\pi(j)}$ refers to the communication cost between slot $\pi(i)$ and $\pi(j)$. They prove the problem is NP-hard and propose a heuristic that tries to assign VMs with large mutual rate requirement close to each other.

6.2 Reliable VM Placement

Israel and Raz [21] study the Virtual Machine (VM) Recovery Problem (VMRP). The VMRP is to place the backup VMs for their corresponding servicing VMs on either active or inactive host, which needs to strike a balance between the (active) machine maintenance cost and VM recovery Service Level Agreement (e.g., recovery time). They show that the VMRP is NP-hard, and they propose a bicriteria approximation algorithm and an efficient heuristic to solve it.

Bin *et al.* [22] tackle the VM placement problem by considering k-resiliency constraint to guarantee high availability goals. A VM is marked as k-resilient, if its current host fails and there are up to $k - 1$ additional host failures, then it can be guaranteed to relocate to a non-failed host. In this sense, a placement is said to be k-resilient if it satisfies the k-resiliency requirements of all its VMs. They first formulate this problem as a second order optimization statement and then transform it to a generic constraint program in polynomial time.

Zhu *et al.* [23] address the Reliable Resource Allocation (RRA) problem. In this problem, each node has a capacity limit of storing VMs and each link is associated an availability value (between 0 and 1). The request is arriving in an on-line fashion, which specifies the requested number of VMs and availability value. The problem is to find a star of a given network, such that the node capacity limit is obeyed and the availability of the star is at least δ. They prove that the RRA problem is NP-hard and propose an exact algorithm as well as a heuristic to solve it. However, the defined problems in [23] do not consider the node's availability and also it restricts to find a star instead of arbitrary subgraph.

Besides, Li and Qian [24] assume that the VM reliability requirement as the maximum fraction of VMs of a same function can be placed in a rack. Yang *et al.* [25] develop a variance-based metric to measure the risk of violating the VM placement availability requirement, but none of them take VM replicas/backups into account. Nevertheless, none of above papers quantitatively (e.g., Sections 2 and 3) model the availability of VM placement (and solve the respective reliable VM placement problem), as we do in this paper.

6.3 Availability-Aware Routing

Song *et al.* [26] propose an availability-guaranteed routing algorithm, where different protection types are allowed. They define a new cost function for computing a backup path when the unprotected path fails to satisfy the availability requirement. She *et al.* [27] prove the problem of finding two link-disjoint paths with maximal reliability (availability) is NP-hard. They also propose two heuristics for that problem. Luo *et al.* [28] address the problem of finding one unprotected path or a pair of link-disjoint paths, such that the cost of the entire path(s) is minimized and the reliability requirement is satisfied. To solve it, they propose an exact ILP as well as two approximation algorithms. However, the reliability (availability) calculation in [28] is different from the aforementioned papers, and assumes a single-link failure model. Assuming each link in the network has a failure probability (=1-availability), Lee *et al.* [29] minimize the total failure probability of unprotected, partially link-disjoint and fully link-disjoint paths by establishing INLPs. They further transform the proposed INLPs to ILPs by using linear approximations. Yang *et al.* [5], [6] study the availability-based path selection problem, which is to find at most w (partially) link-disjoint paths and for which the total availability is no less than η. They prove that this problem is NP-hard and cannot be approximated to arbitrary when $w \geq 2$. They propose an exact INLP and a heuristic to solve this problem.

7 CONCLUSION

In this paper, we have first studied the RVMP problem, which is to place at most H sets of requested k VMs on minimum number of nodes, such that the VM placement availability is no less than δ, and the specified communication delay and connection availability for each VM pair is not violated. We have shown that the RVMP problem is NP-hard, and cannot be approximated to arbitrary degree. To solve it, we have proposed an exact INLP as well as an efficient heuristic and compare these algorithms with another two heuristic modifications. The simulation results reveal that, our proposed heuristic can always achieve a better performance in terms of acceptance ratio and the number of used nodes than the modified heuristics, although it consumes an (acceptably) higher running time. On the other hand, the exact INLP can always achieve the best performance, but its running time is significantly larger than all the heuristics.

Following that, we have studied the Availability-Based Delay-Constrained Routing (ABDCR) problem, which is to establish a connection over at most w (partially) link-disjoint paths, such that the connection availability is at least η and each path has a delay no greater than D. We have shown that the ABDCR problem is NP-hard and have proposed both a concrete exact algorithm and two heuristics to solve it. Finally, we have test these 3 algorithms via simulations on two networks. The simulation results indicate that the exact algorithm can always achieve the highest acceptance ratio, but this comes at the expense of much higher running time. Meanwhile, SeqTAMCRA and TADRA return close to optimal result in a shorter time for $w = 1$ and $w > 1$,

REFERENCES

[1] S. Yang, P. Wieder, and R. Yahyapour, "Reliable virtual machine placement in distributed clouds," in *Proc. of 8th IEEE/IFIP International Workshop on Reliable Networks Design and Modeling (RNDM)*, 2016, pp. 1–7.

[2] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proc. of IEEE Grid Computing Environments Workshop*, 2008, pp. 1–10.

[3] W. Zou, M. Janić, R. Kooij, and F. Kuipers, "On the availability of networks," *Proc. of BroadBand Europe*, 2007.

[4] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proc. of the 1st ACM symposium on Cloud computing*, 2010, pp. 193–204.

[5] S. Yang, S. Trajanovski, and F. Kuipers, "Availability-based path selection," in *Proc. of 6th IEEE International Workshop on Reliable Networks Design and Modeling (RNDM)*, 2014, pp. 39–46.

[6] S. Yang, S. Trajanovski, and F. A. Kuipers, "Availability-based path selection and network vulnerability assessment," *Networks*, vol. 66, no. 4, pp. 306–319, 2015.

[7] P. Datta and A. K. Somani, "Graph transformation approaches for diverse routing in shared risk resource group (srrg) failures," *Computer Networks*, vol. 52, no. 12, pp. 2381–2394, 2008.

[8] C. Kachris, K. Bergman, and I. Tomkos, *Optical interconnects for future data center networks*. Springer Science & Business Media, 2012.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman & Co., 1979.

[10] M. Alicherry and T. Lakshman, "Optimizing data access latencies in cloud systems by intelligent virtual machine placement," in *IEEE INFOCOM*, 2013, pp. 647–655.

[11] P. Van Mieghem and F. A. Kuipers, "Concepts of exact qos routing algorithms," *IEEE/ACM Transactions on Networking*, vol. 12, no. 5, pp. 851–864.

[12] B. Mukherjee, *Optical WDM Networks*. Springer Science & Business Media, 2006.

[13] P. Van Mieghem, "Paths in the simple random graph and the waxman graph," *Probability in the Engineering and Informational Sciences*, vol. 15, no. 04, pp. 535–555, 2001.

[14] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, 2001.

[15] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, pp. 1–53, 2014.

[16] Z. Á. Mann, "Allocation of virtual machines in cloud data centers—-a survey of problem models and optimization algorithms," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, p. 11, 2015.

[17] M. Alicherry and T. Lakshman, "Network aware resource allocation in distributed clouds," in *Proc. of IEEE INFOCOM*, 2012, pp. 963–971.

[18] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera, "A stable network-aware vm placement for cloud systems," in *Proc. of 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2012, pp. 498–506.

[19] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint vm placement and routing for data center traffic engineering," in *Proc. of IEEE INFOCOM*, 2012, pp. 2876–2880.

[20] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. of IEEE INFOCOM*, 2010, pp. 1–9.

[21] A. Israel and D. Raz, "Cost aware fault recovery in clouds," in *Proc. of IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2013, pp. 9–17.

[22] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz, "Guaranteeing high availability goals for virtual machine placement," in *Proc. of 31st IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2011, pp. 700–709.

[23] Y. Zhu, Y. Liang, Q. Zhang, X. Wang, P. Palacharla, and M. Sekiya, "Reliable resource allocation for optically interconnected distributed clouds," in *Proc. of IEEE International Conference on Communications (ICC)*, 2014, pp. 3301–3306.

[24] X. Li and C. Qian, "Traffic and failure aware vm placement for multi-tenant cloud computing," in *IEEE 23rd International Symposium on Quality of Service (IWQoS)*, 2015, pp. 41–50.

[25] Z. Yang, L. Liu, C. Qiao, S. Das, R. Ramesh, and A. Y. Du, "Availability-aware energy-efficient virtual machine placement," in *IEEE International Conference on Communications (ICC)*, 2015, pp. 5853–5858.

[26] L. Song, J. Zhang, and B. Mukherjee, "Dynamic provisioning with availability guarantee for differentiated services in survivable mesh networks," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 3, pp. 35–43, 2007.

[27] Q. She, X. Huang, and J. Jue, "How reliable can two-path protection be?" *IEEE/ACM Transactions on Networking*, vol. 18, no. 3, pp. 922–933, 2010.

[28] H. Luo, L. Li, and H. Yu, "Routing connections with differentiated reliability requirements in WDM mesh networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 1, pp. 253–266, 2009.

[29] H.-W. Lee, E. Modiano, and K. Lee, "Diverse routing in networks with probabilistic failures," *IEEE/ACM Transactions on Networking*, vol. 18, no. 6, pp. 1895–1907, 2010.

[30] C.-L. Li, S. T. McCormick, and D. Simchi-Levi, "The complexity of finding two disjoint paths with min-max objective function," *Discrete Applied Mathematics*, vol. 26, no. 1, pp. 105–115, 1990.

APPENDIX A
RVMP PROBLEM UNDER THE SINGLE-NODE FAILURE

In this appendix, we provide an INLP for the RVMP problem under the single-node failure scenario. The notations and variables follow the same with the ones in Section 4.2. Slightly different from Eqs. (7)-(11), Eq. (17) ensures that the primary placement plan (P_{vn}^1) to put all the VMs on the network. Eq. (18) ensures that the primary placement plan does not place the same VM on the same node whose availability is less than δ with the backup placement plan (P_{vn}^2). Eq. (21) accounts for the placement availability. By setting $\beta = \frac{1}{\min_{j \in \mathcal{N}}(A_j)}$, as long as one VM $v \in V$ is placed on two nodes, the regarded "availability" value is returned as greater than 1, which is also greater than δ.

Objective:

$$\min \sum_{n \in \mathcal{N}} \max_{v \in V} \left(P_{vn}^1, P_{vn}^2 \right) \tag{16}$$

Placement constraint:

$$\sum_{n \in \mathcal{N}} P_{vn}^1 \geq 1 \quad \forall v \in V \tag{17}$$

$$P_{vn}^1 \neq P_{vn}^2 \quad \forall v \in V, n \in \mathcal{N} : A_n < \delta \tag{18}$$

Storage constraint:

$$\sum_{v \in V} \left(\max(P_{vn}^1, P_{vn}^2) \cdot c_v \right) \leq s_n \quad \forall n \in \mathcal{N} \tag{19}$$

Delay and connection availability constraint:

$$F(m, n, A(m, n), T(m, n)) \cdot P_{am}^{h_1} \cdot P_{bn}^{h_2} = 1 \tag{20}$$
$$\forall h_1, h_2 = 1, 2, (m, n) \in \mathcal{L}, a, b \in V : a \neq b$$

VM place availability constraint:

$$\min_{n \in \mathcal{N}}(1 - P_{vn}^1 + P_{vn}^1 A_n) + \max_{n \in \mathcal{N}}(P_{vn}^2 \cdot A_n) \cdot \beta \geq \delta$$

$$\forall v \in V, \ \beta = \frac{1}{\min_{n \in \mathcal{N}}(A_n)} \quad (21)$$

APPENDIX B
ABDCR PROBLEM UNDER THE SINGLE-LINK FAILURE

In this section, it is assumed that at one certain time, at most one link may fail. For a certain request $r(s, t, \eta, D)$, when $w = 1$ in the ABDCR problem, we provide a polynomial-time algorithm to solve it. First, we remove the links whose availability is no greater than η. After that, we run shortest path algorithm from s to t to obtain a minimum delay path. If there is a feasible path and its delay is less than D, then it is the optimal solution, otherwise there is no solution.

When $k \geq 2$, if the optimal solution consists of k fully link-disjoint paths, then 2 fully link-disjoint paths also exist and have availability 1 under the single-link failure scenario, which is optimal. When the optimal solution consists of k partially link-disjoint paths, then $w = 2$ (partially) link-disjoint paths are also enough. The reason is that the availability of partially link-disjoint paths is decided by one unprotected link (say l). Hence, it suffices to find $w = 2$ link-disjoint paths. The proof for $w > 2$ follows analogously from the proof for $w = 2$. Consequently, we have the following theorem:

Theorem 2. The ABDCR problem for $w = 2$ under the single-node failure scenario is NP-hard.

  Proof: **Fully link-disjoint paths:** We regard that the link delay as the link weight in the graph. Hence, any two fully link-disjoint paths have connection availability 1, which can always satisfy η. Now, the ABDCR problem is equivalent to the decision version of the disjoint min-max problem, which is to find two link-disjoint paths from a source to a destination, such that the maximum path weight is minimized. Since the disjoint min-max problem is NP-hard [30], our proof is therefore complete.
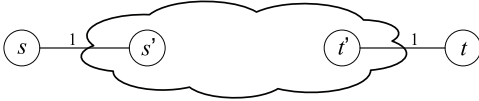

Fig. 15: Reduction of ABPS problem from partially link disjoint to fully link disjoint.

Partially link-disjoint paths: The case for partially link-disjoint paths can be reduced to the case of fully link-disjoint paths by a transformation such as in Fig. 15. More specifically, if we assume any link in Fig. 15, except for (s, s') and (t', t), has availability less than δ, then no link, except for (s, s') and (t', t), can be the unprotected link in the solution of the ABDCR problem for the partially link disjoint case from s to t. In this context, for such η, solving the fully link-disjoint ABDCR problem from s' to t' is equivalent to solving the partially link-disjoint ABDCR problem from s to t.

□
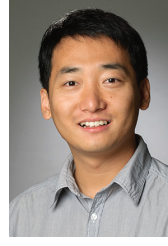
Song Yang received the B.S. degree in software engineering and the M.S. degree in computer science from the Dalian University of Technology, Dalian, Liaoning, China, in 2008 and 2010, respectively, and the Ph.D. degree from Delft University of Technology, The Netherlands, in 2015. He is currently a postdoc researcher in GWDG. His research interests focus on network optimization algorithms in optical networks, stochastic networks and datacenter networks.

Philipp Wieder is deputy leader of data center of the GWDG at the University of Göttingen, Germany. He received his doctoral degree from TU Dortmund in Germany. He is active in the research areas on clouds, grid and service oriented infrastructures for several years. His research interest lies in distributed system, service level agreements and resource scheduling. He has been actively involved in the FP7 IP PaaSage, SLA@SOI and SLA4D-Grid projects.

Ramin Yahyapour is full professor at the Georg-August University of Göttingen. He is also managing director of the GWDG, a joint compute and IT competence center of the university and the Max Planck Society. Dr. Yahyapour holds a doctoral degree in Electrical Engineering and his research interest lies in the area of efficient resource management in its application to service-oriented infrastructures, clouds, and data management. He is especially interested in data and computing services for eScience. He gives lectures on parallel processing systems, service computing, distributed systems, cloud computing, and grid technologies. He was and is active in several national and international research projects. Ramin Yahyapour serves regularly as reviewer for funding agencies and consultant for IT organizations. He is organizer and program committee member of conferences and workshops as well as reviewer for journals.

Stojan Trajanovski is a postdoctoral researcher at the University of Amsterdam and a visiting researcher at Delft University of Technology in The Netherlands. He received his PhD degree (*cum laude*, 2014) from Delft University of Technology and his master degree in Advanced Computer Science (*with distinction*, 2011) from the University of Cambridge, United Kingdom. He also holds an MSc degree in Software Engineering (2010) and a Dipl. Engineering degree (*summa cum laude*, 2008) from Ss. Cyril and Methodius University in Skopje. He successfully participated at international science olympiads, winning a bronze medal at the International Mathematical Olympiad (IMO) in 2003.
His main research interests include network science, network robustness, complex networks, game theory, and optimization algorithms.
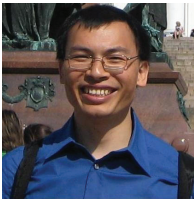
Xiaoming Fu received his Ph.D. in computer science from Tsinghua University, Beijing, China in 2000. He was then a research staff at the Technical University Berlin until joining the University of Göttingen, Germany in 2002, where he has been a professor in computer science and heading the Computer Networks Group since 2007. He has spent research visits at universities of Cambridge, Uppsala, UPMC, Columbia, UCLA, Tsinghua, Nanjing, Fudan, and PolyU of Hong Kong. Prof. Fu's research interests include network architectures, protocols, and applications. He is currently an editorial board member of IEEE Communications Magazine, IEEE Transactions on Network and Service Management, and Elsevier Computer Communications, and has served on the organization or program committees of leading conferences such as INFOCOM, ICNP, ICDCS, MOBICOM, MOBIHOC, CoNEXT, ICN and COSN. He has served as secretary (2008-2010) and vice chair (2010-2012) of the IEEE Communications Society Technical Committee on Computer Communications (TCCC), and chair (2011-2013) of the Internet Technical Committee (ITC) of the IEEE Communications Society and the Internet Society. He has been involved in EU FP6 ENABLE, VIDIOS, Daidalos-II and MING-T projects and is the coordinator of FP7 GreenICN, MobileCloud and CleanSky projects. Prof. Fu is a recipient of the ACM ICN 2014 Best Paper Award, IEEE LANMAN 2013 Best Paper Award and the 2005 University of Göttingen Foundation Award for Exceptional Publications by Young Scholars. He is an IEEE Senior Member and an IEEE Communications Society Distinguished Lecturer.