Offering Resilient and Bandwidth Guaranteed Services in Multi-Tenant Cloud Networks:
Harnessing the Sharing Opportunities

Hyame Assem Alameddine, Sara Ayoubi, Chadi Assi
{hy_alame, sa_ayou, assi} @encs.concordia.ca
Concordia University, Montreal, Canada

Abstract—The sharing of computing and networking resources in the cloud is challenged by several obstacles, such as providing bandwidth guarantees for a predictable performance of the hosted applications, as well as maintaining the availability of their services following outages. Therefore, the wide scale adoption of this emerging computing paradigm remains highly dependent on overcoming these challenges. In fact, a lack of bandwidth guarantees extends the completion time for jobs, thus increasing expenses for clients paying for their time of use. In addition, outages in data centers may result in severe revenue losses for both, the cloud operators and their clients alike. To overcome these challenges, cloud operators should be empowered with a strategic design plan that is able to guarantee resilient and predictable performance for hosted applications. Such a plan consists of provisioning additional backup resources (e.g., virtual machines, bandwidth) while ensuring efficient network bandwidth utilization. In this work, we study the design of various facets of such a plan. Namely, we exploit several bandwidth sharing opportunities in multi-tenant cloud networks while offering resilient and bandwidth guaranteed services. In contrast to previous works which target cloud clients satisfaction, we focus on optimizing network bandwidth utilization in order to increase the cloud operators revenues while maintaining such bandwidth allocation transparent to the clients. Through several motivational examples, and numerical studies, we highlight the sharing opportunities and show that they are able to increase cloud operators revenues by an average of 21.4% while providing up to 50% of bandwidth gain in the network.

I. INTRODUCTION

Today, the cloud computing paradigm is gaining increasing interests, within academia and industry alike, for its capability of offering elastic and on-demand computing resources. Owing to its efficient pay-as-you-use charging model, this paradigm enabled cloud providers to lease their computing resources in the form of virtual machines (VMs) with isolated performance on CPU and memory. Thus, cloud computing will substantially reduce enterprises’ IT expenses, which may cost around 10$ million to 25$ million per year to build and manage their dedicated data centers (DCs) [1].

The essence of cloud computing is to enable the sharing of the underlying network resources among the hosted cloud applications, thereby increasing the revenues of cloud operators. Such sharing of resources may however come at the expense of decreasing the popularity of the cloud, since the reputation of cloud computing highly depends on its ability to provide bandwidth guarantees and high service availability to its clients’ applications [2], [3].

In fact, the bandwidth needed for the communication between the VMs of a cloud client (tenant) fluctuates significantly due to the best-effort sharing at the flow level of the Transmission Control Protocol (TCP) used in today’s DCs [2]. Given the bandwidth sharing in the network, competing applications may interfere with each others, resulting in unpredictable performance. Such variable applications performance may result in important revenue losses for tenants due to the uncertainty of their jobs execution times. Another performance attribute which is equally important, is to provide high service availability for the cloud clients, especially those running critical applications, such as banking, retail systems, etc [4].

A single element failure can cause severe revenue losses for those tenants. A recent survey [4] estimated the cost of one hour downtime of such applications to vary between 25,000$ and 150,000$. Thus, cloud providers seek to offer guaranteed performance and high service availability to incentivize enterprises to move their services to the cloud.

Many work in the literature discussed either one of both problems; bandwidth guarantee ([2], [5], [6], [7], [8], [9]) and service survivability ([4], [10], [11], [12], [13]), but only few tackled the trade-off that exists between them ([14], [3], [15], [16], [17]). In fact, it was observed that colocating VMs of a particular application reduces the bandwidth to guarantee for their communication, allowing cloud providers to admit more tenants in the network. However, such allocation degrades the application’s fault tolerance. Thus, cloud providers are interested in realizing the trade-off between providing high survivability while guaranteeing predictable applications performance and efficiently utilizing their DCs network resources. Previous research focused on guaranteeing bandwidth in the cloud along with high survivability to respond to clients requirements and satisfy them. In contrast, we focus in this work on the cloud providers requirements in increasing their revenues while respecting tenants’ demands. This can be accomplished through the exploration of several efficient network resources utilization opportunities. In this work, we consider a single element failure and we guarantee 100% service availability through the provisioning of redundant computing and network resources (backup VMs, backup bandwidth) in the form of a protection plan [14]. Given the provisioned

This work is supported by NSERC Discovery Grant.

978-0-9883045-1-2/16 $31.00 © 2016 ITC
protection plans, we seek at efficiently using cloud network resources through the sharing of tenants’ backup bandwidths. We observe that: 1) the provisioned backup bandwidths remain idle until the occurrence of a failure; 2) considering a single element failure, some tenants may not fail at the same time while others may without contending for the same protection resources, upon a failure. Therefore, those tenants may be able to share their backup bandwidths. Such sharing remains transparent to the tenants but beneficial to the cloud providers. We show through illustrative examples that up to 50% bandwidth gain can be attained by exploiting the sharing opportunities. To the best of our knowledge, sharing bandwidth between tenants has not been proposed or discussed in the literature before. Thus, we present and formulate the Tenants Bandwidth Share Design (TBS-Design), a novel approach to decide on the optimal sharing strategies that will achieve the highest bandwidth gain. We show that TBS-Design is an NP-complete problem and we propose an efficient heuristic for solving it. Our numerical results show that bandwidth sharing between tenants increases the tenants’ admission rate in a cloud DC and allows cloud providers to generate more revenues.

The remaining of this paper is organized as follows: In Section II, we provide a literature review of the bandwidth guarantee and high survivability problems in a cloud DC. Section III explains a bandwidth allocation approach in a cloud DC and depicts the protection plan design problem. Section IV presents efficient bandwidth utilization opportunities. Section V provides a definition and a formulation of the TBS-Design problem. Section VI explains the Tenants Bandwidth Share Design-A heuristic (TBSH-Design) that solves the TBS-Design problem. Our numerical evaluation is exposed in Section VII. We conclude in Section VIII.

II. RELATED WORK

Recently, there has been much effort for guaranteeing network performance and providing high survivability for cloud applications. In order to reduce the bandwidth usage in the core of the network, Oktopus [6] developed a VM embedding heuristic that collocates VMs of the same tenant under the smallest sub-tree while guaranteeing bandwidth based on the hose model [6], [5], [2]. The hose model is an abstraction model that allows tenants to express their resources requirements (VMs, bandwidth) to the cloud provider independently from the underlying infrastructure. However, Oktopus overlooked the fact that collocation decreases fault tolerance.

Alternatively, the CloudMirror team [15], [16] proposed the Tenant Application Graph (TAG) that reflects the structure of the tenant’s application, to guarantee bandwidth. Such structure is unknown by the tenant which makes the TAG model not practical. It also considered the Worst Case Survivability (WCS) requested by the client to provide fault tolerance. WCS being the smallest number of VMs that should remain functional during a failure of a single sub-tree, causes service degradation in case of failure. Bodik et al. [3] also employed the WCS as a measure of fault tolerance. They proposed the K-way cut algorithm to provide an initial embedding for the VMs while minimizing the bandwidth at the core of the network. They improved this initial allocation by realizing multiple moves of the embedded VMs in order to achieve fault tolerance. They assumed that a physical server can only host one VM of the same virtual data center (VDC) which extensively spreads the tenant’s VMs and leads to a higher bandwidth usage.

The work in [13] provided 1-redundant and k-redundant approaches to support the failure of critical nodes. In order to minimize the incurred bandwidth cost, they implement bandwidth sharing techniques known as cross-sharing and backup-sharing. To reduce the idle backup bandwidth, the study in [11] provided two heuristics; the first one solves the virtual node embedding and the link embedding problems separately. It chooses the virtual node embedding solution that minimizes the reserved backup bandwidth. The other heuristic solves both problems jointly by adopting a link packing approach. Both work [13], [11] guarantees VM to VM bandwidth by supposing that a dedicated link exists between each pair of communicating VMs. This assumption is unrealistic because links are shared among multiple VMs. In addition, VMs communication dependencies change over time. The pipe model [9], [16] approach refers to such assumption.

The work in [17] used the anycast routing principle to provide resilience against failure while reducing the backup footprint. Anycast routing is based on choosing, out of a set of candidate destinations, a destination for a given service request originating from a known source node. They considered backup path sharing upon a link failure but did not examine bandwidth sharing between tenants upon any node failure.

Our work differs from those in the literature as we do not focus on solving the bandwidth guarantee and the survivability problems. We rather use the solutions provided for those to further decrease the backup footprint through the sharing of backup bandwidth between tenants. Our work can be addressed with any of the above techniques.

III. PRELIMINARIES

A. Bandwidth Provisioning in cloud DCs

To provision bandwidth for tenants, a cloud provider requires the knowledge of their requirements (in terms of VMs and bandwidth required for their communication). In addition, tenants need a simple and intuitive interface to express these requirements independently of the underlying physical infrastructure of cloud DCs. Such interface is known as “Abstraction Model”. In fact, many abstraction models were discussed in the literature. The pipe model [9], [2] guarantees host-to-host connectivity; however, it is not practical because it assumes the knowledge of the communication matrix between VMs which is hard to determine or to estimate by the tenant. The hose model [9], [2], [6] guarantees the minimum bandwidth required by each VM. The tenant application graph (TAG) proposed by [15], [16] relies on the tenant’s knowledge of the application structure in order to determine the bandwidth to guarantee. Without loss of generality, we use in this work the hose model to guarantee bandwidth between VMs due to its simplicity in expressing the tenant’s requirements.

Consider a tenant request $<N, B>$ of $N$ primary VMs and $B$ bandwidth to be guaranteed for the communication between
those VMs. The hose model interconnects the $N$ VMs to a central switch of $N+B$ bandwidth (Fig.1(a)). This ensures $N+B$ bandwidth as a maximum communication rate between those VMs. Because multiple VMs are likely to communicate at the same time with a single destination that can only receive data at rate $B$, the hose model provisions the minimum bandwidth needed by each VM.

For instance, consider a tenant request $<6, B>$ of 6 primary VMs, and $B$, the bandwidth to be guaranteed for the communication between those VMs. Such request is embedded as shown in Fig.1(b). We observe that the link of interest (thick line in Fig 1(b)) divides the network into 2 component $C1$ of $m = 5$ VMs and $C2$ of $N-5 = 1$ VM. Hence, the bandwidth to be guaranteed on this link, based on the hose model, is $\min(m, N-m) \times B$. We refer to the hose interconnecting the primary VMs as the pre-failure hose.

**B. Protection Plan Design**

Guaranteeing bandwidth for a tenant in a cloud DC, depends on the number and the placement of VMs provisioned for the tenant. In fact, today, cloud DCs are prone to several failures causing the disruption of one or more cloud clients. Thus, tenants demand a certain level of service availability. In order to respond to those availability requirements, a cloud provider should devise a protection plan for its clients through the provisioning of backup resources. Backup resources correspond to the backup VMs which become active only upon a failure, and the backup network bandwidth used by the post-failure hose. A post-failure hose is the interconnection between all VMs (primary and backup) of the tenant, that are operational following any failure affecting the tenant.

Designing a protection plan for a hosted tenant is a hard problem that involves the following steps: 1) identifying the required number of backup VMs to provision, 2) deciding on their placement, 3) determining the primary-to-backup VMs correspondence; that is, which backup VMs will protect which primary VMs upon a failure, 4) establishing the backup bandwidth to reserve. Depending on the approaches used to perform each of the previous steps, multiple protection plans can be designed.

In [14], we designed a protection plan that aimed at providing 100% service availability upon a single node failure. In order to ensure efficient utilization of resources (VMs, bandwidth) while providing 100% service continuity, we provisioned a number of backup VMs equal to the maximum number of primary VMs of a tenant hosted on the same physical server. Such number represents the minimum backup VMs footprint required to protect all the tenant’s primary VMs. Embedding those backup VMs while collocating them with the primary ones reduces the backup bandwidth footprint [6]. Such footprint is also affected by the primary-to-backup VMs correspondence plan. The best correspondence is the one that minimizes the backup bandwidth to reserve. It can be determined by the protection plan design (PPD) model discussed in [14]. Now, considering the knowledge of the primary and backup VMs embedding, in addition to the primary-to-backup VMs correspondence, determining the backup bandwidth to reserve on a link can be accomplished by evaluating the backup bandwidth to provision upon the sequential failure of each physical server hosting primary VMs of the specified tenant. Let $l$ denote a link in the network and $b_l$ be the backup bandwidth that must be provisioned on this link. $b_l = \max(b_i)$; where $b_i$ is the backup bandwidth required on link $l$ to assume the failure of the VMs hosted on server $S_i$. $b_i$ is determined according to the correspondence between primary VMs and backup VMs. The total bandwidth to reserve on link $l$ to assume the failure of the VMs hosted on server $S_i$ becomes: $b_l = b_i + b_k$, where $b_k$ refers to the bandwidth required on link $l$ for the pre-failure hose.

To illustrate the backup bandwidth provisioning process, we consider the primary and backup VMs embedding of a tenant $<6, B>$ as shown in Fig.2(a). This tenant requires 6 primary VMs, hosted on servers $S1$, $S2$, and $S3$. The total bandwidth to provision for the communication between those primary VMs is $8B$ as depicted by the specified pre-failure hose on that same figure. We determine a protection plan for this tenant through the provisioning of 4 backup VMs (Fig.2(a)) and we define the primary-to-backup VMs correspondence as follows: the backup VMs hosted on server $S2$ and $S3$ protect the primary VMs hosted on $S1$, backup VMs hosted on $S1$ and $S3$ take care of the primary VMs hosted on $S2$. Similarly, the primary VMs hosted on $S3$ is protected by the backup VMs embedded on $S1$. Hence, by considering a single node failure of the physical servers $S1$ (Fig.2(b)), $S2$ (Fig.2(c)) and $S3$ (Fig.2(d)) and determining the backup bandwidth needed upon each failure, we can determine the backup bandwidth to be reserved on each link as the maximum bandwidth of all those provisioned by all the defined post-failure hoses. It can be easily verified that this backup bandwidth depicted in Fig.2(e), is sufficient to ensure service continuity upon any single node failure.

**IV. EFFICIENT BANDWIDTH UTILIZATION OPPORTUNITIES**

A good protection plan design entails an effective network utilization. In the following, we uncover several strategies to better make efficient use of the network resources.

**A. Bandwidth reuse**

Upon any failure of a physical server affecting a tenant, the primary bandwidth reserved for the communication of the primary VMs (hosted on the failed server) is released, and thus, it can be reused by the post-failure hose of the same tenant. Consequently, instead of reserving the sum of primary and backup bandwidths on each link ($b_l = b_k + b_i$) (Fig.2(f)), one can provision the maximum of both as shown in Fig.2(g). By considering such bandwidth reuse, we can save
7B by reserving 11B (Fig.2(g)) instead of 18B (Fig.2(f)) (39% bandwidth saving) [14].

B. Bandwidth sharing between multiple tenants

Since bandwidth backup is only used following a failure, it can be shared between multiple tenants that will not require it simultaneously. In fact, upon considering a single node failure, we can identify two cases in which tenants can share their backup bandwidths:

1. Non-concurrent failure of tenants

By considering a single node failure, tenants that do not have primary VMs hosted on the same physical servers will not be vulnerable to a simultaneous service disruption. Hence, they can share their protection bandwidth on the common links along their routes (on their protection plans).

Fig. 2: Backup Bandwidth reservation procedure.

In the example of Fig.3, we depict the embedding of 2 tenants: tenant 1 <3, B1>, tenant 2 <2, B2>. Since the primary VMs of both tenants are hosted on different physical servers, any single node failure will result in the service disruption of only one of the tenants at a time. This suggests that the backup VMs of tenant 1 and tenant 2 will not be used at the same time. Thus, those tenants can share the backup bandwidth that is needed for their communication. Such bandwidth sharing requires the reservation of the maximum backup bandwidth needed by each of them on their shared links (thick lines in Fig.3) (Fig.3(b)). By comparing Fig.3(a) and (b), one can notice the importance of bandwidth sharing between tenants, which results in the saving of 3B2 (when B1=B2=3B, a saving of 12.5% is obtained). A key observation is that sharing between tenants is only possible on those links where no bandwidth reuse (Section IV-A) of the same tenant is considered between its primary and backup bandwidth. In fact, even though the dashed links in Fig.3 are common for both tenants, no bandwidth sharing is possible on such links because the primary bandwidth of each tenant is reused by its post-failure hose on those links. Thus, we make the following observation:

Observation 1. Bandwidth sharing on a link l is allowed between tenants whose protection plans do not reuse their primary bandwidths on l.

2. Simultaneous failure of tenants

While in the previous example (Fig.3) we have shown that services that do not fail simultaneously can share their backup bandwidths, we illustrate in Fig.4 the cases where tenants who are vulnerable to a simultaneous failure may also share their backup bandwidths on the same links traversed by their corresponding post-failure hoses. We consider a network of two tenants: tenant 1 of <3, B1> and tenant 2 <2, B2> embedded with their backup VMs as presented in Fig.4(a). Tenant 1 primary VMs hosted on S1 are protected by its backup VMs hosted on S5 and S7 while the primary VM hosted on S4 is protected by its backup VM embedded on S6. Both tenants have primary VMs hosted on server S4, thus they can fail simultaneously if S4 fails.

In order to determine the bandwidth which needs to be reserved for each of the two tenants, we consider the failure of each of the servers S1, S3 and S4 hosting the primary VMs of both tenants. When S1 fails (Fig.4(b)), only tenant 1 fails, requiring bandwidth B1 on links l1, l2 and l3. When server S3

Fig. 3: Bandwidth sharing between tenants.
fails (Fig. 4(c)), only tenant 2 fails, demanding bandwidth $B_2$ on links $l_1$, $l_2$ and $l_3$. However, if we consider the failure of S4 (Fig. 4(d)), both tenants fail since the two of them have primary VMs hosted on this server. Tenant 1 demands bandwidth $B_1$ to be reserved on links $l_1$ and $l_2$ for its service restoration, while tenant 2 requires bandwidth $B_2$ to be reserved on link $l_3$ (in addition to those already reserved on links $l_1$ and $l_2$). Since tenant 1 and tenant 2 require backup bandwidth to be reserved simultaneously on links $l_1$ and $l_2$, upon the failure of S4, they can not share their backup bandwidth on those links. Thus, we make the following observation:

**Observation 2.** Two tenants having primary VMs hosted on the same server S and their post-failure hoses go through the same link $l$, upon the failure of $S$, they can not share their backup bandwidth on $l$.

Now, following any failure on any server hosting primary VMs of tenant 1 and tenant 2 (S1, S3 or S4), one of the two tenants requires bandwidth on link $l_3$ at a time. Hence, both tenants can share bandwidth on this link. Thus, instead of reserving $B_1 + B_2$ on $l_3$, we can reserve $\max(B_1, B_2)$. In Fig. 4(e), we represent the total primary and backup bandwidth ($10B_1 + 7B_2$) needed for both tenants while considering bandwidth reuse on the dashed links. As mentioned previously, no bandwidth sharing is possible on links where bandwidth reuse is considered. Fig. 4(f) depicts that $10B_1 + 6B_2$ is to be reserved for the communication of tenant 1 and tenant 2 while considering bandwidth reuse and bandwidth sharing between them, saving one $B_2$ through sharing (when $B_1 = B_2 = B$, a saving of 6% is obtained).

V. TENANTS BANDWIDTH SHARE DESIGN (TBS-DESIGN)

Given a substrate network, a set of hosted and protected tenants, we seek an optimal use of the network capacity by reducing the amount of the overall reserved network bandwidth. This can be accomplished either through reusing the primary bandwidth as backup bandwidth upon a failure (Section IV-A) or by sharing the backup bandwidth between multiple tenants (Section IV-B). In this Section, we explore the bandwidth sharing problem. We start by providing a definition and a formulation of the optimal Tenants Bandwidth Share Design (TBS-DESIGN) problem.

**A. Problem definition**

Given that: 1) network links have limited capacity and are shared between multiple tenants; 2) tenants require a guaranteed and predictable performance through fixed dedicated network bandwidth; 3) cloud providers are interested in serving the largest number of clients to generate more revenue; we seek to explore the opportunities for sharing bandwidth between tenants while meeting their network requirements.

Based on the motivational examples presented in Section IV-B, we know that two tenants are able to share their backup bandwidth on a certain link if they do not use it simultaneously to route their backup traffic upon any single server failure. In this case, only the maximum backup bandwidth required by these two tenants can be reserved instead of their sum. Clearly, the best sharing approach that can be achieved on any given link $l$ is to reserve the maximum backup bandwidth required by any tenant whose backup traffic is routed through $l$ (all tenants using $l$ are able to share their backup bandwidth on $l$); that is: $b_l = \max \{b_1, b_2, b_3, ..., b_n\}$ where $b_1, b_2, b_3, ..., b_n$ are the backup bandwidths required by tenants $t_1, t_2, t_3, ..., t_n$ on $l$ respectively. In contrast, the worst case scenario is when all the tenants whose backup paths traverse $l$ fail at the same time, and thus, are not able to share their backup bandwidth on $l$ (Observation 2). Hence, the backup bandwidth to be reserved on $l$ becomes equal to the sum of the backup bandwidths needed by each of the tenants: $b_l = b_1 + b_2 + b_3 + ... + b_n$.

In fact, some of the tenants may be able to share their backup bandwidth on $l$, when others may not. Thus, we can group those tenants into several subsets which we define as independent sharing sets. An independent sharing set is a subset of one or more tenants $t \in T$ that may share their backup bandwidths on $l$ (an independent set is a subset of nodes of a graph $G$, such that no two of them are adjacent.
We denote $T$ as the set of tenants requiring backup bandwidth on $l$ and not having any bandwidth reuse on this link (Observation 1). Note that each tenant $t \in T$ can only be an element of exactly one sharing set of $l$. Accordingly, we define for each link $l$ in a network the set $S_l^l$ of sharing sets $s_l^l$ ($i \in 1..m$); $S_l^l = s_l^1 \cup s_l^2 \cup ... \cup s_l^m$ such that $s_l^1 \cap s_l^2 \cap ... \cap s_l^m = \emptyset$. There exists multiple combinations of sharing sets for a single link $l$. Consider the example where a tenant $t_1$ can share its backup bandwidth with $t_2$ and $t_3$; however, $t_2$ and $t_3$ cannot share their bandwidth on $l$, consequently, they cannot be in the same sharing set. Thus, we can form 3 combinations of sharing sets:

1. $s_1^1 = \{t_1\}; s_2^1 = \{t_2\}; s_3^1 = \{t_3\};$
2. $s_1^2 = \{t_1, t_2\}; s_2^2 = \{t_3\};$
3. $s_1^3 = \{t_1, t_3\}; s_2^3 = \{t_2\};$

Since our objective is to save bandwidth on $l$, the combination of sets that yield the best sharing is the one that minimizes the total backup bandwidth $b_l$ to reserve on $l$, $b_l$ being equal to the sum of backup bandwidth ($b_l^i$) to be reserved for each sharing set $s_l^i$: $b_l = \sum_{i=1}^{m} b_l^i$, where $b_l^i = \max \{b_i^j\}$ ($b_l^i$ being the backup bandwidth required by each tenant in $s_l^i$ on $l$).

Therefore, the TBS-Design problem consists of determining for each link in a network, the independent sharing sets that maximize its saved bandwidth. Thus, we provide the following definition for the problem:

**Definition 1.** Given a set of tenants ($t \in T$), each requiring a backup bandwidth $b_l^i$, on a link $l$, find the independent sharing sets combination that minimize the bandwidth to reserve on $l$.

**Theorem 1.** The optimal TBS-Design problem is NP-complete.

**Proof.** We prove that the TBS-Design problem is NP-Complete by a reduction from the graph coloring problem, known as NP-Complete. For completeness, we present a formal definition of the graph coloring decision problem.

**Definition 2.** “Let $G = (V, E)$ be an undirected graph. Is there a $k$-coloring of $V$, such that no two adjacent vertices have the same color? ”[19].

Given a substrate link $l$, and a set $T$ of tenants requiring backup bandwidth on $l$ and not incurring any bandwidth reuse on it (observation 1); we construct a conflict graph $G_p = (V_p, E_p)$, where each vertex $v_i \in V_p$ corresponds to a tenant $t_i \in T$ ($|V_p| = |T|$). $E_p$ is the set of edges in the conflict graph, where an edge $e$ is added between two vertices $v_i, v_j \in V_p$ corresponding tenants, $t_i, t_j \in T$, can not share their backup bandwidth on $l$ (observation 2). Subsequently, we can reformulate the TBS-Design decision problem as follows: “Given a link $l$, a set of tenants $t_1, t_2, ..., t_n \in T$ require a uniform backup bandwidth $b_l^i = B$ on $l$. In this case, defining the MINIMUM number $w$ of independent sharing sets will solve our TBS-Design problem. This is true, because $b_l = \sum_{i=1}^{w} b_l^i = \sum_{i=1}^{w} \max \{b_l^i\} = \sum_{i=1}^{m} B = w B$ where $w$ is the number of independent sharing sets and $b_l^1, b_l^2, ..., b_l^w$ are defined earlier.”

**B. Problem formulation**

In this section, we provide a mathematical formulation of the TBS-Design problem. This work targets a single path tree topology, however our work can be easily extended to handle any other network topologies if the needed inputs are known and valid. Let $G(V, E)$ be the constructed auxiliary graph as described in section V-A. The TBS-Design problem can be formulated as follows:

**Parameters**

$q_{t \ell'} \in \{0, 1\}$: indicates whether tenants $t$ and $t'$ can not share their backup bandwidth on the link $l$ (1) (or can (0)).

$b_l$: defines the backup bandwidth required for tenant $t$ on $l$.

**Decision Variables**

$y_i^l \in \{0, 1\}$: specifies if tenant $t$ is part of the sharing set $i$ (1) (or not (0)).

$b_l^i$: the backup bandwidth that needs to be reserved for the sharing set $i$.

$M$: set of sharing sets, to be defined. In the worst case, non
of the tenant will be able to share its bandwidth, thus, \( m = |M| = |V| \) sharing sets.

**Model**

Minimize \( \sum_{i=1}^{m} b_i \) \hfill (1)

\[ y_i^d + y_i^d' \leq 1 \quad \forall i \in M; \quad \forall (l', l) \in E : \; q_{l,l'} = 1 \] \hfill (2)

\[ \sum_{i=1}^{m} y_i^d = 1 \quad \forall i \in V \] \hfill (3)

\[ b_i = \max \{ b_i y_i^d \} \quad \forall i \in M; \forall i \in V \] \hfill (4)

The objective function (Eq.1) consists of defining independent sharing sets that minimize the backup bandwidth to reserve on a given link \( l \). Constraint 2 specifies that two conflicting tenants that can not share their backup bandwidths, only one of them can be part of a specified set \( i \). Constraint 3 depicts that a tenant should be part of one and only one set. Constraint 4 determines the bandwidth which need to be reserved for the tenants in a set \( i \) which is equal to the maximum backup bandwidth of all the tenants in the set. This constraint can be converted to a linear programming format as follows:

\[ b_i \geq b_i M_i^d \quad \forall i \in M; \forall i \in V \] \hfill (5)

Since the objective is to minimize \( b_i \), the model will set \( b_i \) to the maximum backup bandwidth required by all the tenants in the set \( \{ b_i y_i^d \} \). The TBS-Design model is a Mixed integer linear problem which is complex. Next, we present, the **Tenants Bandwidth Share Design - A Heuristic (TBSH-Design)** to solve it.

**VI. TENANTS BANDWIDTH SHARE DESIGN - A HEURISTIC (TBSH-DESIGN)**

The tenants bandwidth share design problem consists of determining the independent sharing sets of tenants for every link in the network. In addition, it specifies the bandwidth to be reserved on each link for the defined sets. To solve this problem we developed the **Tenants Bandwidth Share Design heuristic (TBSH-Design)** depicted in Algorithm 1.

Our methodology for solving the TBSH-Design problem for each link \( l \) in the network consists of selecting the tenants that are eligible to share their backup bandwidths on \( l \). An eligible tenant is a tenant that is not reusing his primary bandwidth as backup bandwidth on \( l \) (Observation 1). Because the bandwidth to reserve for a sharing set is equal to the maximum backup bandwidth of all the tenants in the set (Section V-A), trying to place the eligible tenants who request the biggest amount of backup bandwidth on \( l \) in the same set, may reduce the total bandwidth to reserve on it. Motivated by this intuition, our approach sorts the eligible tenants in decreasing order of their backup bandwidth requirements on \( l \). It also stores them in an array, that we denote \( sortedRequests \). Given the \( sortedRequests \) array, the TBSH-Design recursively builds the sharing sets of \( l \). The heuristic starts by creating a sharing set \( s \) for \( l \) (line 4) and adds to it the first tenant in the \( sortedRequests \) array (lines 5-6). This tenant will be the one with the highest backup bandwidth demands on \( l \). Thus, the algorithm sets the bandwidth to reserve for \( s \) equal to the backup bandwidth demands of this tenant (line 7). This latter is then removed from the \( sortedRequests \) array (line 8). Afterwards, the algorithm loops over the remaining requests in the array (line 9), and checks if each one of them is able to share its backup bandwidth with all the requests that belong to \( s \), through the call of \( canShareBw(s) \) function (line 11). The \( canShareBw(s) \) function verifies that the post-failure hoses of the request of interest do not go through \( l \) upon the service disruption of any of the requests in \( s \) (Observation 2). If the request of interest can share its backup bandwidth on \( l \) with all the requests in \( s \), it will be added to the set \( s \) (line 12) and removed from the \( sortedRequests \) array (line 13). After evaluating all the requests in the \( sortedRequests \) array and adding those who can share their bandwidth to the set \( s \), the heuristic will try to build a new sharing set with the remaining requests (the requests that are not part of \( s \)). This is performed by calling the TBSH algorithm again and passing to it the updated \( sortedRequests \) array (line 17). The code will keep on calling the TBSH heuristic until all the eligible requests become part of a sharing set of \( l \). The bandwidth reserved on \( l \) will be updated to consider the bandwidth to reserve for each defined sharing set. If the \( sortedRequests \) array was of size \( N \), looping over the \( N \) requests will take \( O(n^2) \). In addition, if none of the requests was able to share its bandwidth, the TBSH-Design will be called recursively \( N \) times. Hence, the worst case complexity of the TBSH-Design algorithm is \( O(n^2) \).

**Algorithm 1 TBSH (Array sortedRequests)**

1: Given:
2: \( l \): link on which we are solving the TBSH-Design problem
3: \( s \): newSharingSet();
4: \( r \): sortedRequests.get(0);
5: \( s \).requests.add(r);
6: \( s \).bandwidthToReserve = r.backupBandwidth[l];
7: \( sortedRequests \).remove(0);
8: for (i = 0; i < sortedRequests.size(); i ++) do
9: \( r \) = sortedRequests.get(i);
10: if (r.canShareBw(s)) then
11: \( s \).requests.add(r);
12: \( sortedRequests \).remove(i);
13: end if
14: end for
15: if (sortedRequests.size() > 0) then
16: \( TBSH(sortedRequests) \);
17: end if

**VII. NUMERICAL RESULTS**

We carry out an extensive empirical study to evaluate the performance of our TBSH-Design against our TBS-Design model and a no bandwidth share approach. The no bandwidth share method consists of embedding the requests and protecting them without performing any bandwidth share between the admitted tenants. The three methods previously mentioned use the same primary embedding algorithm which consists of collocating the VMs of a request in the smallest sub-tree
to reduce the bandwidth use in the network ([6], [15], [16]). They also provide 100% reliability for the admitted requests by designing their protection plan based on the approach specified in Section III-B.

We simulate a three-level fat tree topology with no path diversity. We assume that all VMs are of homogeneous CPU and memory capacity. Additionally, we consider that each physical server has a capacity of \( \theta \) VM slots. We perform our simulations over a network of 128 physical servers with \( \theta = 6 \). We set the capacity of the links interconnecting the switches to 10 Gbps. We randomly generate sets of 100 requests each, of varying VMs ([5-25] VMs) and network ([100-500] Mbps) requirements. All our numerical evaluations are conducted using Cplex version 12.4 to solve the optimization problem on an Intel core i7-4790 CPU at 3.60 GHZ with 16 GB RAM. We use two different approaches in our tests:

### A. Offline Approach

In order to evaluate the performance and the scalability of the TBSH-Design vs the TBS-Design, we run our tests using an offline approach where tenants are known a priori and do not leave the network once embedded. This increases the sharing probability, since a bigger number of requests are occupying the network.

#### 1- Execution time and optimality gap

We consider a single link and vary the number of requests using it. We randomly generate the backup bandwidth to reserve for each request on this link to be between [100-500] Mbps. In addition, we build a 2-dimensions array which randomly specifies if each pair of the generated requests can/cannot share their backup bandwidth on this link. We use the TBSH-Design and the TBS-Design to share bandwidth between those requests. The results depicted in Table I clearly prove that the TBSH-Design is much more scalable than the TBS-Design model. TBSH-Design is able to share bandwidth between 5000 requests in only 65 ms, while the TBS-Design runs for approximately 3 hours to decide on the sharing sets for 20 requests only. The optimality gap between the TBSH-Design and the TBS-Design is 1.8% for 10 requests and 13.6% for 20 requests. It is clear that, this optimality gap increases with the increase of the number of requests. Further investigation is needed for improving the performance of the TBSH-Design.

#### 2- Bandwidth gain per link type

Since most DCs are usually oversubscribed at the upper level links of a network (TOR and Aggregation level) [15], cloud providers seek at increasing the available bandwidth on these links. To this end, Fig. 5 presents the average bandwidth gain per link type (Physical server to TOR switch, TOR to Aggregate switch, Aggregate to Core switch) obtained through a single run of 100 requests. The bandwidth gain is calculated as depicted in Eq. (6) where \( bw'Ts \) is the total backup bandwidth that can be shared between tenants and \( sBw \) represents the total bandwidth reserved for the tenants after sharing.

\[
\text{BandwidthGain} = \left( \frac{bw'Ts - sBw}{bw'Ts} \right) \times 100 \quad (6)
\]

Fig. 5 reveals that sharing bandwidth between tenants is more profitable on the upper level links than the physical server to TOR links. Since we are using collocation as an embedding technique, physical server to TOR links are more likely to be used for the communication between the VMs of the same tenant. However, the number of tenants using the upper level links is greater than those using the physical server to TOR links, which increases the sharing probability on those latter. One can also note that the TBSH-Design provides similar average bandwidth gain as the TBS-Design.

#### B. Online Approach

We conduct our tests using an online approach. We consider a Poisson traffic arrival of requests. We alter the load by varying the arrival rate \( (\lambda) \) while fixing the average service time \( (\mu) \) of the requests \( (load = \lambda/\mu) \). Our numerical results are depicted in Fig. 6.

##### 1- Bandwidth Gain Over Time: Sharing bandwidth between the admitted tenants increases the amount of available network resources and can provide up to 50% of bandwidth gain as depicted by Fig. 6(a). This figure presents the bandwidth gain (Eq. (6)) over time obtained by the TBSH-Design and the TBS-Design for a single run over a load \( = 6 \).

One can clearly notice that at a certain point in time \( t \), the TBSH-Design may provide more bandwidth gain than the TBS-Design, and vice versa. This can be explained by the fact that the sharing sets built by each of those two methods at a time \( t' < t \) are different. Thus, the bandwidth gain provided by each of them may be spread over different links. This affects the embedding of every new request which arrives and gets admitted at time \( t \). Thus, the bandwidth gain provided at time \( t \) by each of the two methods is different.

##### 2- Rejection Rate: The rejection rate is an important metric to look at, especially that cloud providers are interested in admitting more tenants in their DCs. Here, we compare the rejection rate with the no bandwidth share method against the TBSH-Design and the TBS-Design. Our results presented

<table>
<thead>
<tr>
<th>Nb. of requests</th>
<th>TBS-Design</th>
<th>TBSH-Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

**Table I: TBS-Design and TBSH-Design comparison over varying number of requests on a single link.**
The rejection rate is calculated as the ratio of the number of rejected requests (\(\text{RejectedNb}\)) over the total number of requests (\(\text{TotalNb}\)) (Eq.(7)).

\[
\text{RejectionRate} = \frac{\text{RejectedNb}}{\text{TotalNb}} \times 100 
\tag{7}
\]

Since sharing bandwidth between tenants increases the available bandwidth in the network, one can directly guess that the rejection rate should decrease. This is true, given that the requests that were rejected because of lack of bandwidth in the network using the no bandwidth share approach, are more probable to get admitted using the TBSH-Design and the TBS-Design methods. This is clearly depicted in Fig.6(b), which shows that the TBS-Design can decrease the rejection rate by an average of 30.5% over the load, while the rejection rate is decreased by 21.6% using the TBSH-Design. This decrease is calculated in comparison with the no bandwidth share approach results. Alternatively, the average rejection rate gap between TBSH-Design and the TBS-Design is 11%.

3. Revenue Over Time: Admitting more tenants in the network yield an important factor to increase cloud providers’ revenue. Fig.6(c) presents the revenue over time obtained by a single run of 100 requests for a load = 6. The revenue is calculated as shown in Eq.(8) where \(m\) is the number of requests \((<N_i, B_i>)\) admitted in the network, \(c_{vm}\) and \(c_{bw}\) are the costs of leasing one unit of VM and one unit of bandwidth respectively. We consider that \(c_{vm} > c_{bw}\) in our tests.

\[
\text{Revenue} = \sum_{i=1}^{m} N_i c_{vm} + \sum_{i=1}^{m} B_i c_{bw} 
\tag{8}
\]

Fig.6(c) shows that the TBS-Design can increase the cloud providers revenue by an average of 21.4% over time, while the TBSH-Design gives a similar average increase of returns approximated to 18.97% in comparison with the no bandwidth share approach.

VIII. Conclusion

This paper exploits several bandwidth sharing techniques between tenants, making efficient use of cloud DC networks. Given an embedding and a protection plan design for each tenant in the DC, we formulate the TBS-Design model to solve to optimality the tenants bandwidth share problem. We proved that the optimal TBS-Design problem is NP-complete. Thus, we developed the TBSH-Design, a heuristic, shown to be much more scalable than the TBS-Design model. Through extensive simulations, we confirmed that our bandwidth sharing techniques are able to increase cloud operators’ revenue by an average of 21.4% over time while reducing the rejection rate by an average of 30.5%. Our sharing techniques increase the bandwidth gain in the network up to 50% and can be applied to any network topology. However, we do believe that studying the advantages of these sharing techniques using different type of embedding and protection plan designs is indeed a relevant and interesting problem that we leave for future work.

REFERENCES