# Scalable Traffic-Aware Virtual Machine Management for Cloud Data Centers

Fung Po Tso*, Konstantinos Oikonomou†, Eleni Kavvadia†, Dimitrios P. Pezaros*
*School of Computing Science, University of Glasgow, G12 8QQ, UK
†Department of Informatics, Ionian University, 49100 Corfu, Greece
Email: {posco.tso, dimitrios.pezaros}@glasgow.ac.uk, {okon, ekavvadia}@ionio.gr

***Abstract*—Virtual Machine (VM) management is a powerful mechanism for providing elastic services over Cloud Data Centers (DC)s. At the same time, the resulting network congestion has been repeatedly reported as the main bottleneck in DCs, even when the overall resource utilization of the infrastructure remains low. However, most current VM management strategies are traffic-agnostic, while the few that are traffic-aware only concern a static initial allocation, ignore bandwidth oversubscription, or do not scale.**

**In this paper we present S-CORE, a scalable VM migration algorithm to dynamically reallocate VMs to servers while minimizing the overall communication footprint of active traffic flows. We formulate the aggregate VM communication as an optimization problem and we then define a novel distributed migration scheme that iteratively adapts to dynamic traffic changes. Through extensive simulation and implementation results, we show that S-CORE achieves significant (up to 87%) communication cost reduction while incurring minimal overhead and downtime.**

***Index Terms*—Virtual Machine, Migration, Consolidation, Communication Cost, Scalable, Traffic-Aware, Data Center Network**

## I. Introduction

Elastic resource provisioning in Cloud Data Centers (DC)s is managed by a number of control loops at the routing [1][2], transport [3], and Virtual Machine (VM) [4][5] layers. Unlike routing and transport mechanisms that typically concern the efficient utilization of the underlying network, VM management through initial placement [6], consolidation [7], or migration of live VMs [8] has been employed to optimize a range of diverse objectives, such as, e.g., server resource (CPU, RAM, net I/O) usage [9] and energy efficiency [10], that are often in direct conflict with each other. In particular, virtualization itself has significant impact on network congestion [11][12] especially at the core layers of DC topologies which in turn becomes the main bottleneck throughout the infrastructure [1], hindering efficient resource usage and Cloud providers' revenue [13].

A limited number of studies have proposed traffic-aware VM management schemes that try to minimize the impact of virtualization on the DC network. However, the proposed algorithms are either centralized and therefore do not scale well to the full size of today's DCs [14][15], concern the initial placement of VMs and do not deal with maintaining steady-state throughout the system's evolution [16], or they only consider bandwidth allocation at the lower host-to-network layers [17][16] overlooking congestion that happens in a significant fraction of the core links [18] even when parts of the DC infrastructure remain underutilized [19].

In this paper, we present S-CORE, a *Scalable communication COst REduction* scheme for intra-DC workloads that dynamically re-allocates VMs through live migration and minimizes the communication cost incurred by the resulting traffic dynamics in an always-on manner, while adhering to server-side resource capacity boundaries. By assigning distinct link weights at the different layers of the DC infrastructure and accounting for the temporal traffic routed over these links, a function of the network-wide communication cost is defined that can then be minimized in terms of the contributing pairwise aggregate VM traffic load. S-CORE adopts a distributed approach based on information available locally at each VM to inform migration decisions rather than using in-network statistics, a property that makes it scalable and realistically implementable over large-scale DC infrastructures. It deals with the dynamic evolution of DC workloads by iteratively localizing pairwise VM traffic to lower-layer links where bandwidth is not as over-subscribed as in the core, and where interconnection switches are cheaper to upgrade [3]. We have formulated the distributed algorithm and implemented S-CORE on the Xen hypervisor. Through simulation and testbed experiments we show that S-CORE can significantly reduce traffic over the high-cost links at the core of the topology that are prone to congestion [19][11] while incurring minimal migration overhead and system downtime. S-CORE can achieve an overall communication cost reduction of as high as 87% of the optimal allocation, as this is approximated by centralized algorithms that assume global traffic knowledge but are prohibitively expensive to implement in practice.

The remainder of this paper is structured as follows. Section II provides the formulation of communication cost and Section III derives the network-wide optimal VM allocation and its complexity. Section IV introduces *S-CORE*, a distributed VM migration scheme that reduces the overall communication cost of the topology based on information available locally at each VM. Section V discusses the implementation of S-CORE on Xen with two alternative migration prioritization policies. An extensive evaluation of S-CORE over simulated and testbed environments over diverse topologies and under realistic DC traffic patterns is presented in Section VI. Section VII outlines related work, and Section VIII concludes the paper.

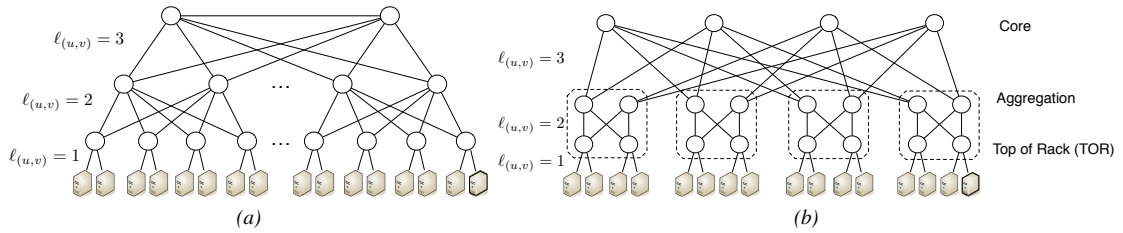



*Fig. 1:* (a) Canonical tree and (b) Fat-tree DC topologies

## II. System Definitions

Let $\mathbb{V}$ be the set of VMs in the DC hosted by the set of all servers $\mathbb{S}$, such that every VM $u \in \mathbb{V}$ and every server $\hat{x} \in \mathbb{S}$. Each VM $u$ in the DC is unique and it is assigned a unique identifier $\texttt{ID}_u$. Furthermore, each VM is hosted by a particular server and let $\mathcal{A}$ denote an *allocation* of VMs to servers within the DC. Let $\hat{\sigma}^{\mathcal{A}}(u)$ be the server that hosts VM $u$ for allocation $\mathcal{A}$, $u \in \mathbb{V}$ and $\hat{\sigma}^{\mathcal{A}}(u) \in \mathbb{S}$. Let $\mathbb{V}_u$ denote the set of VMs that exchange data with VM $u$.

DCs typically employ a layered tree topology with multiple redundant paths that try to increase bisection bandwidth between any pair of servers. However, due to equipment and energy costs bandwidth is typically oversubscribed in the higher layers of such topologies [13]. Without loss of generality, we assume three distinct communication layers, Top-of-Rack (ToR), aggregation, and core, as depicted in Fig 1. Network links that connect servers to ToR switches are referred to as 1-*level links*, those between ToR and aggregation switches as 2-*level links*, etc.

Let $\texttt{h}(\hat{x}, \hat{y}) > 0$ denote the *number of hops* between server $\hat{x}$ and server $\hat{y}$ along a shortest path. Let $\ell^{\mathcal{A}}(u, v)$ denote the *communication level* between VM $u$ and VM $v$ for a given allocation $\mathcal{A}$. Obviously, if the servers hosting VMs $u$ and $v$ are collocated, then $\ell^{\mathcal{A}}(u, v) = 0$. If communication is established over 1-level links, then $\ell^{\mathcal{A}}(u, v) = 1$, etc. In general, $\ell^{\mathcal{A}}(u, v) = \frac{\texttt{h}(\hat{\sigma}^{\mathcal{A}}(u), \hat{\sigma}^{\mathcal{A}}(v))}{2}$. Let $\ell^{\mathcal{A}}(u)$ denote the *highest communication level* for VM $u$ for allocation $\mathcal{A}$, or $\ell^{\mathcal{A}}(u) = \max_{\forall v \in \mathbb{V}_u} \ell^{\mathcal{A}}(u, v)$.

Not all DC links are equal, and their cost depends on the particular layer they interconnect. For example, high-speed router interfaces are much more expensive than lower-level ToR switches. Therefore, in order to accommodate a large number of VMs in the DC and at the same time keep investment costs low from a provider's perspective, utilization of the "lower cost" switch links is preferable to utilization of the "more expensive" router links. In order to reflect the increasing cost of high-density, high-speed (10 Gb/s) switches and links at the upper layers of the DC tree topologies, and their increased over-subscription ratio, we assign a representative link weight $\texttt{c}_i$ for an $i$-level link per data unit (e.g., byte). Without loss of generality, in this case $\texttt{c}_1 < \texttt{c}_2 < \texttt{c}_3$. However, in the general case, link weight assignment can be based on DC operator policy to reflect diverse metrics, such as, e.g., energy consumption, performance, fault tolerance, etc.

## III. Communication Cost Analysis

Link utilization is dictated by the intensity of pairwise traffic between VMs and the objective here is to utilize links of small weights when possible. Let $\lambda(u, v)$ denote the *traffic load* (or rate) per time unit exchanged between VMs $u$ and $v$ (incoming and outgoing). Note that the traffic load is expected to vary in such highly dynamic environments. Therefore, $\lambda(u, v)$ will denote the average rate over a certain time window, suitable to capture the dynamism of the environment.

The focus of this work is on communication levels $\ell^{\mathcal{A}}(u, v) > 0$, i.e., traffic routed over at least one ToR switch. For communication level $\ell^{\mathcal{A}}(u, v) = 1$, data are exchanged over two links (i.e., $\texttt{h}(\hat{\sigma}^{\mathcal{A}}(u), \hat{\sigma}^{\mathcal{A}}(v)) = 2$); the corresponding link weight for using each link being $\texttt{c}_1$. For each of the links, the product $\lambda(u, v)\texttt{c}_1$ corresponds to a (weighted) communication cost for utilizing the particular 1-level link. If the communication is through level 2 of the hierarchy (i.e., $\ell^{\mathcal{A}}(u, v) = 2$), data exchanges take place over four links, two being 2-level (weight $\texttt{c}_2$) and two 1-level (weight $\texttt{c}_1$) links. Eventually, the communication cost in this case corresponds to $2\lambda(u, v)(\texttt{c}_1 + \texttt{c}_2)$. In general, when the communication among two VMs $u$ and $v$ is of level $\ell^{\mathcal{A}}(u, v)$, the communication cost corresponds to $2\lambda(u, v) \sum_{i=1}^{\ell^{\mathcal{A}}(u,v)} \texttt{c}_i$.

Given that any VM $u$ communicates with all VMs in set $\mathbb{V}_u$, there is a *communication cost*, denoted by $C^{\mathcal{A}}(u)$, attributed to VM $u$, for allocation $\mathcal{A}$,

$$C^{\mathcal{A}}(u) = 2 \sum_{\forall v \in \mathbb{V}_u} \lambda(u, v) \sum_{i=1}^{\ell^{\mathcal{A}}(u,v)} \texttt{c}_i. \tag{1}$$

It is now possible to derive an expression with respect to the *overall communication cost* for all VM-to-VM communication over the DC. Let $C^{\mathcal{A}}$ denote this cost for allocation $\mathcal{A}$. Obviously, $C^{\mathcal{A}} = \frac{1}{2} \sum_{\forall u \in \mathbb{V}} C^{\mathcal{A}}(u)$ ($\frac{1}{2}$ is inserted since each link is accounted for twice). Eventually,

$$C^{\mathcal{A}} = \sum_{\forall u \in \mathbb{V}} \sum_{\forall v \in \mathbb{V}_u} \lambda(u, v) \sum_{i=1}^{\ell^{\mathcal{A}}(u,v)} \texttt{c}_i. \tag{2}$$

Note that Eq. (2) does not take into account traffic in or out of the DC. For this case, any shortest path is along ToR, aggregation and core switches for any allocation $\mathcal{A}$.

From Eq. (2) it becomes evident that different allocations $\mathcal{A}$ correspond to different overall communication costs. The objective here is to derive a particular allocation for which the overall communication cost is minimized (i.e., optimal). Let $\mathcal{A}_{opt}$ denote an *optimal allocation*, such that $C^{\mathcal{A}_{opt}} \leq C^{\mathcal{A}}$, for any possible $\mathcal{A}$ (note that there might be more than one allocations that minimize the overall communication cost). The objective is to derive the optimal allocation for a given DC environment and most importantly, to be able to adapt to any

dynamic changes in this environment. In special cases, the optimal allocation can be easily derived. For example, if all active VMs can be accommodated within a single rack, then this allocation minimizes the overall communication cost. This observation is confirmed by Eq. (2), however, it is a reduced case since DCs are built to support a large number of VMs that are initially allocated either at random or in a load-balanced manner. In the general case, deriving the optimal allocation is a hard optimization problem due to (i) its high complexity (given the number of permutations that must be considered in an exhaustive search approach) and (ii) the global knowledge required in a highly dynamic environment like a DC. Every time the traffic dynamics change, there is a need to gather that information and recompute the optimal values in short timescales. Obviously, such a centralized approach does not scale with the number of VMs and the size of current DC topologies. In fact, optimal VM allocation does not have a polynomial time solution. We have shown in appendix that the NP-complete Graph Partitioning (GP) problem with vertex weight 1 can be reduced to the optimal VM allocation, and therefore the latter is also NP-complete [20]. These observations motivate S-CORE, a distributed approach under which a VM unilaterally decides whether to migrate based on information available locally.

## IV. Scalable Communication COst REduction (S-CORE)

The approach and analysis presented in the sequel assumes the presence of a *token* in the network and that the VM holding the token at a given time is the one that decides whether to migrate or not. Then, the token is passed on to another VM according to the adopted *token policy*. Token policies are discussed in Section V.

Let *migration* of a VM $u$ from its current location (server $\hat{\sigma}^{\mathcal{A}}(u)$) to some other server $\hat{x}$ be denoted by $u \rightarrow \hat{x}$ for allocation $\mathcal{A}$. If migration takes place, then allocation $\mathcal{A}$ changes; let $\mathcal{A}_{u\rightarrow\hat{x}}$ denote the new allocation. Assuming that migration $u \rightarrow \hat{x}$ did take place, there is a new communication cost $C^{\mathcal{A}_{u\rightarrow\hat{x}}}(u)$ corresponding to allocation $\mathcal{A}_{u\rightarrow\hat{x}}$. Let $\Delta C^{\mathcal{A}}_{u\rightarrow\hat{x}}(u) = C^{\mathcal{A}}(u) - C^{\mathcal{A}_{u\rightarrow\hat{x}}}(u)$ denote the *communication cost difference* that is attributed to migration $u \rightarrow \hat{x}$. The aim next is to determine the condition under which $\Delta C^{\mathcal{A}}_{u\rightarrow\hat{x}}(u) > 0$ is satisfied.

*Lemma 1:* Given migration $u \rightarrow \hat{x}$, there is a communication cost difference,

$$\Delta C^{\mathcal{A}}_{u\rightarrow\hat{x}}(u) = \sum_{\forall z\in\mathbb{V}_u} C^{\mathcal{A}}(z) - C^{\mathcal{A}_{u\rightarrow\hat{x}}}(z). \tag{3}$$

*Proof:* Migration $u \rightarrow \hat{x}$ affects the communication of all VMs $z \in \mathbb{V}_u$, in addition to that of VM $u$. The rest of the VMs (i.e., $\mathbb{V} \setminus \mathbb{V}_u \cup \{u\}$) are not affected and therefore, there is no change in their corresponding communication costs. For any $z \in \mathbb{V}_u$, the difference $C^{\mathcal{A}}(z) - C^{\mathcal{A}_{u\rightarrow\hat{x}}}(z)$ corresponds to the contribution of this particular VM $z$ to the communication cost difference $\Delta C^{\mathcal{A}}_{u\rightarrow\hat{x}}(u)$. The lemma is proved by summing up $C^{\mathcal{A}}(z) - C^{\mathcal{A}_{u\rightarrow\hat{x}}}(z)$, $\forall z \in \mathbb{V}_u$. ∎

*Lemma 2:* Given migration $u \rightarrow \hat{x}$, there is a communication cost difference,

$$\Delta C^{\mathcal{A}}_{u\rightarrow\hat{x}}(u) = 2\sum_{\forall z\in\mathbb{V}_u} \lambda(z,u)\left(\sum_{i=1}^{\ell^{\mathcal{A}}(z,u)} \mathsf{c}_i - \sum_{i=1}^{\ell^{\mathcal{A}_{u\rightarrow\hat{x}}}(z,u)} \mathsf{c}_i\right). \tag{4}$$

*Proof:* As stated above, migration $u \rightarrow \hat{x}$ affects the communication of all VMs $z \in \mathbb{V}_u$. Given allocation $\mathcal{A}$ (before migration), then according to Eq. (1), for any $z \in \mathbb{V}_u$, $C^{\mathcal{A}}(z) = 2\sum_{\forall v\in\mathbb{V}_z} \lambda(z,v) \sum_{i=1}^{\ell^{\mathcal{A}}(z,v)} \mathsf{c}_i$, which can be written as $C^{\mathcal{A}}(z) = 2\sum_{\forall v\in\mathbb{V}_z\setminus\{u\}} \lambda(z,v) \sum_{i=1}^{\ell^{\mathcal{A}}(z,v)} \mathsf{c}_i + 2\lambda(z,u)\sum_{i=1}^{\ell^{\mathcal{A}}(z,u)} \mathsf{c}_i$.

Suppose that migration $u \rightarrow \hat{x}$ does take place. Considering the new allocation $\mathcal{A}_{u\rightarrow\hat{x}}$, the corresponding communication cost for any VM $z \in \mathbb{V}_u$ can be written as follows (similarly as before), $C^{\mathcal{A}_{u\rightarrow\hat{x}}}(z) = 2\sum_{\forall v\in\mathbb{V}_z\setminus\{u\}} \lambda(z,v) \sum_{i=1}^{\ell^{\mathcal{A}_{u\rightarrow\hat{x}}}(z,v)} \mathsf{c}_i + 2\lambda(z,u)\sum_{i=1}^{\ell^{\mathcal{A}_{u\rightarrow\hat{x}}}(z,u)} \mathsf{c}_i$.

At the same time, migration $u \rightarrow \hat{x}$ does not affect VMs $z \in \mathbb{V}_z \setminus \{u\}$. Consequently, $\ell^{\mathcal{A}}(z,v) = \ell^{\mathcal{A}_{u\rightarrow\hat{x}}}(z,v)$, $\forall z \in \mathbb{V}_z \setminus \{u\}$. Eventually, $C^{\mathcal{A}}(z) - C^{\mathcal{A}_{u\rightarrow\hat{x}}}(z) = 2\lambda(z,u)\left(\sum_{i=1}^{\ell^{\mathcal{A}}(z,u)} \mathsf{c}_i - \sum_{i=1}^{\ell^{\mathcal{A}_{u\rightarrow\hat{x}}}(z,u)} \mathsf{c}_i\right)$, for any $z \in \mathbb{V}_u$. Based on Lemma 1, by summing up Eq. (3) $\forall z \in \mathbb{V}_u$, the lemma is proved. ∎

The following lemma derives an expression with respect to the *overall communication cost difference* $C^{\mathcal{A}} - C^{\mathcal{A}_{u\rightarrow\hat{x}}}$, denoted by $\Delta C^{\mathcal{A}}_{u\rightarrow\hat{x}}$.

*Lemma 3:* Given a migration $u \rightarrow \hat{x}$, the overall communication cost difference is given by,

$$\Delta C^{\mathcal{A}}_{u\rightarrow\hat{x}} = 2\sum_{\forall z\in\mathbb{V}_u} \lambda(z,u)\left(\sum_{i=1}^{\ell^{\mathcal{A}}(z,u)} \mathsf{c}_i - \sum_{i=1}^{\ell^{\mathcal{A}_{u\rightarrow\hat{x}}}(z,u)} \mathsf{c}_i\right). \tag{5}$$

*Proof:* Given that the overall communication cost $C^{\mathcal{A}}$ can be expressed as $C^{\mathcal{A}} = \frac{1}{2}\sum_{\forall z\in\mathbb{V}} C^{\mathcal{A}}(z)$, it can also be written as, $C^{\mathcal{A}} = \frac{1}{2}\sum_{\forall z\in\mathbb{V}\setminus\mathbb{V}_u\cup\{u\}} C^{\mathcal{A}}(z) + \frac{1}{2}\sum_{\forall z\in\mathbb{V}_u} C^{\mathcal{A}}(z) + \frac{1}{2}C^{\mathcal{A}}(u)$. Similarly, when migration $u \rightarrow \hat{x}$ takes place, $C^{\mathcal{A}_{u\rightarrow\hat{x}}} = \frac{1}{2}\sum_{\forall z\in\mathbb{V}\setminus\mathbb{V}_u\cup\{u\}} C^{\mathcal{A}_{u\rightarrow\hat{x}}}(z) + \frac{1}{2}\sum_{\forall z\in\mathbb{V}_u} C^{\mathcal{A}_{u\rightarrow\hat{x}}}(z) + \frac{1}{2}C^{\mathcal{A}_{u\rightarrow\hat{x}}}(u)$.

Since migration $u \rightarrow \hat{x}$ does not affect the communication of VMs $v \in \mathbb{V} \setminus \mathbb{V}_u \cup \{u\}$, there is no change in the communication level or communication costs for these VMs, and therefore $C^{\mathcal{A}}(z) = C^{\mathcal{A}_{u\rightarrow\hat{x}}}(z)$, $\forall v \in \mathbb{V} \setminus \mathbb{V}_u \cup \{u\}$.

Consequently, $\Delta C^{\mathcal{A}}_{u\rightarrow\hat{x}}$ can be expressed by $C^{\mathcal{A}} - C^{\mathcal{A}_{u\rightarrow\hat{x}}} = \frac{1}{2}\sum_{\forall z\in\mathbb{V}_u} C^{\mathcal{A}}(z) + \frac{1}{2}C^{\mathcal{A}}(u) - \frac{1}{2}\sum_{\forall z\in\mathbb{V}_u} C^{\mathcal{A}_{u\rightarrow\hat{x}}}(z) - \frac{1}{2}C^{\mathcal{A}_{u\rightarrow\hat{x}}}(u) = \frac{1}{2}\left(\sum_{\forall z\in\mathbb{V}_u} C^{\mathcal{A}}(z) - \sum_{\forall z\in\mathbb{V}_u} C^{\mathcal{A}_{u\rightarrow\hat{x}}}(z)\right) + \frac{1}{2}\left(C^{\mathcal{A}}(u) - C^{\mathcal{A}_{u\rightarrow\hat{x}}}(u)\right)$. It is derived that $\Delta C^{\mathcal{A}}_{u\rightarrow\hat{x}} = \frac{1}{2}\left(\sum_{\forall z\in\mathbb{V}_u} C^{\mathcal{A}}(z) - C^{\mathcal{A}_{u\rightarrow\hat{x}}}(z)\right) + \frac{1}{2}\left(C^{\mathcal{A}}(u) - C^{\mathcal{A}_{u\rightarrow\hat{x}}}(u)\right)$. Based on Eq. (3), and (4), $C^{\mathcal{A}} - C^{\mathcal{A}_{u\rightarrow\hat{x}}} = 2\sum_{\forall z\in\mathbb{V}_u} \lambda(z,u)\left(\sum_{i=1}^{\ell^{\mathcal{A}}(z,u)} \mathsf{c}_i - \sum_{i=1}^{\ell^{\mathcal{A}_{u\rightarrow\hat{x}}}(z,u)} \mathsf{c}_i\right)$, and the lemma is proved. ∎

Live migration of VMs can itself incur data transfer and configuration overheads for DC operators. We capture such

overheads in the *migration cost*, $c_m$, which should be compensated by the gain of the network-wide communication cost reduction, i.e., $\Delta C^{\mathcal{A}}_{u\to\hat{x}} > c_m$. In the next section, we present an implementation of S-CORE that keeps such overheads as well as VM downtime low. The following theorem provides the fundamental condition that needs to be satisfied for a migration $u \to \hat{x}$ to take place and its proof is straight forward from Eq. (5).

*Theorem 1:* When migration $u \to \hat{x}$ takes place, the overall communication cost compensates for the migration cost $c_m$, if and only if, $2\sum_{\forall z\in\mathbb{V}_u}\lambda(z,u)\left(\sum_{i=1}^{\ell^{\mathcal{A}}(z,u)}\mathtt{c}_i - \sum_{i=1}^{\ell^{\mathcal{A}_{u\to\hat{x}}}(z,u)}\mathtt{c}_i\right) > c_m$.

Based on the condition of Theorem 1, the following migration policy for virtual machines is proposed.

*The S-CORE Migration Policy*: A VM $u$ migrates from server $\hat{\sigma}^{\mathcal{A}}(u)$ to another server $\hat{x}$, provided that $2\sum_{\forall z\in\mathbb{V}_u}\lambda(z,u)\left(\sum_{i=1}^{\ell^{\mathcal{A}}(z,u)}\mathtt{c}_i - \sum_{i=1}^{\ell^{\mathcal{A}_{u\to\hat{x}}}(z,u)}\mathtt{c}_i\right) > c_m$ is satisfied.

Apart from the token policy that will be discussed in the following section, there are some important features of VM migration that need to be highlighted. In particular, the condition of Theorem 1 relies on information that is available locally at a given VM $u$. First, communication level $\ell^{\mathcal{A}}(z,u)$ for $z \in \mathbb{V}_u$ requires knowledge of the physical location of $u$ and any VM $z \in \mathbb{V}_u$ exchanging data with it. This is achieved by assigning servers IP addresses from a subnet associated with each rack. A VM $u$ can then use a combination of static topology information and active probing to identify the number of hops to any other VM. Link weights $\mathtt{c}_i$ and migration cost $c_m$ can be readily available locally to each VM. Traffic load $\lambda(u,v)$ can be captured dynamically by monitoring incoming and outgoing traffic between VMs $u$ and $v$, averaged over a given time interval. Even though subject to the accuracy of periodical estimations, this approach can adapt to the dynamic changes of DC traffic depending on the size of the temporal window. At the same time, the measurement interval needs to capture steady-state and avoid reacting to instantaneous fluctuations in traffic dynamics. To address this, the size of the time window can be set on the order of minutes to hours over which traffic load is averaged before a VM is migrated.

If a VM migrates, Theorem 1 ensures that the overall communication cost will be reduced. This local decision that requires local information and eventually reduces a global cost metric, is a scalable alternative to the centralized approach presented in the previous section.

## V. Implementation

### A. Token Policies

An important part for the coordination and execution of the distributed migration algorithm is the order in which VMs make a unilateral decision on whether to migrate to a different physical host. This is achieved through maintaining and passing a token that contains information for all VMs. This consists of a VM ID and a communication level value, identifying the communication cost incurred by the VM's current traffic load. A token is a message formed as an array of entries.capable of representing over 4 billion IDs before recycling, and an 8-bit communication level. Entries are stored in ascending order by VM ID. The size of the message is of the order of the number of VMs (i.e., $|\mathbb{V}|$) in the network. VM ID allocation is handled by a centralized VM instance placement manager, which is part of the underlying DC network fabric. A VM currently holding the token can identify the next VM that the token should be passed onto based on some policy. We have implemented a number of distinct token passing policies in [21]. Due to space limitations, in this paper we focus on two policies which are straight-forward to implement and incur the least instrumentation and real-time measurement overhead.

*1) Round-Robin (RR):* As a first approach for a token policy, a basic round-robin token passing mechanism is employed. The *round-robin token policy* passes the token among VMs based on their IDs in an ascending order, assuming each VM has a unique and totally ordered identifier. In particular, starting from the VM with lowest ID, denoted as $v_0$, the token then passes to VM $u$ such that $\mathtt{ID}_{v_0} < \mathtt{ID}_u < \mathtt{ID}_z$, for any $z \in \mathbb{V} \setminus \{v_0, u\}$. Let $u \leftarrow v \oplus 1$ denote that VM $u$ is the one that follows VM $v$, or that there is no other VM $x$ such that $\mathtt{ID}_u > \mathtt{ID}_x > \mathtt{ID}_v$.

*2) Highest-Level First (HLF):* Although *RR* is trivial to implement, it can be wasteful since not all VMs will need to migrate at any given time nor will all VM migrations equally reduce the network-wide communication cost. A token passing policy that can prioritize VMs which are likely to migrate and substantially improve communication cost would therefore be more efficient. We have defined a highest-level first policy that passes the token amongst VMs whose traffic load is routed through the highest-layer links of the network topology. Links are most costly and more likely to experience congestion at this level, and is therefore reasonable to assume that migration is likely to take place, greatly reducing the overall communication cost. The highest communication level is initialized at zero for all VMs. When the token is held by VM $u$ then $\mathtt{l}_u$ can be updated since VM $u$ is aware of its own highest communication level $\ell^{\mathcal{A}}_u$. VM $u$ is also aware of the communication level of those VMs it communicates with (i.e., $v \in \mathbb{V}_u$). Therefore, it can update the corresponding entries $\mathtt{l}_v \leftarrow \ell^{\mathcal{A}}(u,v)$ accordingly. This update takes place only if the existing estimation $\mathtt{l}_v$ is smaller than the new value $\ell^{\mathcal{A}}(u,v)$. The token is passed on to the next VM at this communication level, otherwise the token is passed to a VM at the next lowest level. If no VM suitable for migration is found, the policy restarts from the VM belonging to the highest communication level with the lowest ID. The policy uses this simple heuristic and exploits the partial distributed state stored at each VM instead of requiring global state which would make prioritization more efficient albeit with significant overhead. The details of the HLF token policy are presented in Algorithm 1.

**Algorithm 1** HLF Token Policy

$\texttt{cl} \leftarrow \texttt{l}_u$ ▷ cl maintains the current value of $\texttt{l}_u$
$found \leftarrow$ FALSE ▷ Flag regarding next VM

**for** $\forall v \in \mathbb{V}_u$ **do** ▷ Update VMs connected to $u$
  **if** $\texttt{l}_v < \ell^{\mathcal{A}}(u, v)$ **then**
    $\texttt{l}_v \leftarrow \ell^{\mathcal{A}}(u, v)$

$z \leftarrow u \oplus 1$ ▷ Pick the next VM after $u$
**while** $\texttt{cl} \geq 0$ && $!found$ **do**
  **while** $\texttt{l}_z \neq \texttt{cl}$ **do**
    $z \leftarrow z \oplus 1$ ▷ Pick the following VM
  **if** $\texttt{l}_z \leftarrow \texttt{cl}$ **then**
    $found \leftarrow$ TRUE ▷ Next node is found
  **else** ▷ Next node is not found at this level
    $\texttt{cl} \leftarrow \texttt{cl} - 1$ ▷ Go to a lower level
    $z \leftarrow v_0$ ▷ and start from the beginning

**if** $!found$ **then** ▷ No unchecked VMs are left
  Pick VM $z : \min_{\texttt{ID}_x}\{\forall x \in \mathbb{V} : \texttt{l}_x = \max_{\forall v \in \mathbb{V}}(\texttt{l}_v)\}$
Send $token$ to VM $z$

### B. S-CORE Implementation on Xen

S-CORE relies on VMs passing the token amongst themselves and the token holder making a unilateral migration decision. In practical deployment however, system virtualization is transparent; virtual host are not aware that they run inside a VM and do not interact explicitly with the hypervisor. We have implemented the distributed S-CORE migration algorithm, the token policies and the required traffic measurement modules within dom0 of the Xen hypervisor [22].Ubuntu 12.04 was used for dom0 (domain zero, the initial domain started by Xen on boot), and the Python-based *xm* [22] was used as the management interface. In order to better integrate S-CORE with *xm*, we have implemented S-CORE in Python. We have also enabled Open vSwitch in Xen as it provides flow-level access for monitoring at the hypervisor level for all local VMs, rather than on a per-VM basis. Although S-CORE solely considers communication cost initially, we note that it can be easily extended to add more constraints such an individual host's CPU, RAM, and bandwidth availability.

*1) Flow Monitoring:* In order for VMs to maintain flow-level statistics, we have implemented our own flow table supporting the following operations: *fast addition of new flows*; *updating existing flows*; *retrieval of a subset of flows, by IP address*; *access to the number of bytes transmitted per flow*; *access to flow duration, for calculation of throughput*. The flow table is periodically updated through polling Open vSwitch for datapath statistics allowing for the storage of flows for as long as it is required. Flows are stored from when they start and until a migration decision is made for a VM.

*2) Token Passing:* We have used the IPv4 address of a VM as the 32-bit VM ID carried in each token. As all VMs must have a unique IP address, this provides a unique identifier simplifying the token passing process, as the token can be sent directly to the IP address of the next VM. To efficiently pack the token for network transmission, it is stored and transmitted as a block of 32-bit unsigned integers. Similarly, for the *HLF* token policy that requires an additional highest communication level entry, we have specified an 8-bit value that follows the VM ID. To facilitate token passing, a token listening server runs on a known port in dom0 of each hypervisor. For the token server to receive the token, a NAT redirect is installed in dom0's *iptables*, redirecting messages for a particular port to dom0 itself. When dom0 holds the token for a VM it hosts, it is then able to conduct the migration decision process on behalf of the VM, before passing the token.

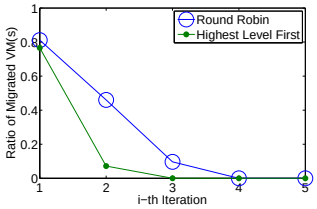



*Fig. 2:* Ratio of migrated VMs in 5 consecutive iterations.

*3) Throughput Calculation:* When dom0 receives the token for a colocated VM, the first step is to calculate the aggregate load between that VM and all the neighbors it communicates with. This is achieved by looking up S-CORE's flow table for the source and destination flows associated with that IP address, and calculating the total number of bytes transmitted. As each flow stores a timestamp of when it was started, these timestamps can be used to deduce the length of time for which the flow statistics have been gathered since last being cleared, allowing calculation of the aggregate throughput in the form of bytes-per-second.

*4) Location Identification:* To compute the communication cost we have to first determine the communication level. As we store a flow table of the IP addresses each VM communicates with, we can probe neighboring VMs to find out the IP address of their dom0. Similar to the token passing method, we can send a custom *location request* to the IP address of each communicating VM. A NAT redirect in dom0 of each hypervisor will then pass it to dom0, which can send a *location response* containing dom0's static address back to the VM. With that information, the dom0 currently holding the token can make a lookup into a precomputed location cost mapping with its own IP address and the IP address of the underlying dom0 of each communicating VMs.

*5) Migration Location Identification:* Since we now have the IP addresses of each hypervisor, after probing for the communication cost, we can rank neighboring VMs from highest to lowest communication levels and probe each server to see if it is able to host the current VM. A *capacity request* packet is sent to the hypervisor of the neighboring VM with the highest communication cost, which responds with a *capacity response* packet, detailing how many more VMs it is able to host and the amount of RAM it has available (to account for

VMs with heterogeneous RAM requirements).

### C. Load Balancing Considerations

The S-CORE algorithm takes both topology and link load into consideration. For topology-load awareness, S-CORE ultimately reduces congestion through traffic localization. Operators often oversubscribe their network (even with fat-tree topology [2]) in order to lower the total cost of the design. As a result, the oversubscription ratio increases dramatically from edge to core layers, limiting host bandwidth to less-than-line-rate for distant VM communication. Nevertheless, this essentially implies that the edge (lower) part of the topologies where hosts attached to rack switches have significantly more provisioned bandwidth between one another than they do with hosts in other racks. For example, when communicating VMs are in the same host servers or racks, they can communicate at full NIC speed.

On the other hand, S-CORE can detect VMs that exchange large flows, which are often the root cause of network congestion and impair short flows, and migrate them to exploit highest rack-level bandwidth. Cloud DC traffic patterns have been shown to exhibit a long tail distribution. Mice flows dominate the DC workload, yet most bytes are transferred across the network in a relatively small set of very large flows (elephants). [18][1][23][19]. One important feature of S-CORE is that it measures the number of bytes exchanged between communicating VMs periodically. Since large flows will contribute significantly to the average number of bytes per unit time per communicating VM pair, they will be picked up by S-CORE which will subsequently migrate VMs that emit these large flows to the same server or rack to leverage the highest possible available bandwidth provided by the edge layer, and avoid congesting aggregation/core layers where network bandwidth is sparse and congestion happens most often [1][2].

S-CORE also incorporates link load thresholds in its decision-making algorithm. As mentioned in Section VI, a heuristic is used that explicitly considers end-host resource limitations which include network bandwidth. Hence, if the target host does not have sufficient bandwidth to accommodate the requesting VM, the next best choice with adequate bandwidth will be considered. The actual bandwidth threshold used can be set according to overall network utilization policies over the topology.

## VI. Evaluation

We have conducted extensive simulation and testbed experiments to evaluate S-CORE's scale properties and implementation overhead. We have used *ns-3* to construct representative canonical (2560 physical hosts; 128 ToR switches; 20 hosts per rack) and fat-tree ($k$ =16; 1024 hosts) DC topologies (cf. Fig. 1). Each host can accommodate up to 16 VMs to model a typical DC server's capacity. A single VM is modeled as a socket application which communicates with one or more other VMs in the network. Similar to actual virtualization, each server has a VM hypervisor network application to manage a collective number of VMs, supporting in-migration (when one or more VMs move into a server) as well as out-migration (when one or more VMs move out of a server).

We have built a DC traffic generator to evaluate S-CORE under realistic DC load patterns at increasing intensities, as these have been reported in a number of DC measurement studies [18][1][23][19]. The sample of a 10s Traffic Matrix (TM) of all ToR switches is given in Fig.3a, exhibiting properties in accordance to those unveiled in [23]. Similar with traffic patterns used in previous works on traffic-aware virtual machine management [16][17][14][15], the TM is sparse and only a handful of ToRs become hotspots. Still, a significant fraction of traffic is routed over the upper layers of the topology hierarchy, resulting in episodes of congestion and incurring high communication cost. We have scaled the initial TM by a factor of 10 and 50, respectively, in order to evaluate S-CORE under increased load stress.

We have considered the maximum number of VMs that can be accommodated in a single physical host, and therefore a VM migrates only when Theorem 1 is satisfied and the target host has sufficient system resources (e.g., residual CPU, memory and host bandwidth) available. We have set the link weight cost, $\mathsf{c}_i$, to grow exponentially for each layer, hence $\mathsf{c}_1 = e^0$, $\mathsf{c}_2 = e^1$, $\mathsf{c}_3 = e^3$, etc. Migration (overhead) cost $c_m$ was initially set to zero to allow for a fair comparison between a centralized approximate-optimal approach and S-CORE. However, since a DC operator may wish to limit, e.g., the number of MV migrations over a temporal interval, we have also experimented with different $c_m$ values which are presented later.

### A. Computation of Centralized Optimal Values

As discussed in section III, centrally calculating the optimal VM allocation is computationally infeasible. In order to evaluate the performance of the distributed S-CORE VM migration, we have employed a tractable heuristic-search alternative, using a genetic algorithm (GA). Our GA has been implemented as part of the ns-3 library to compute approximate values for different simulation scenarios. The GA starts with a population of $1,000$ individuals representing densely-packed VM distributions, each of which may or may not be an optimal solution (of VM assignments). The crossover operator has been implemented using edge assembly crossover (EAX), and the replacement of individuals is based on tournament selection. Mutation happens by swapping a random number of VMs between racks. The GA stops when there is no significant improvement in communication cost reduction ($< 1\%$) in 10 consecutive generations. Execution time over a medium-load simulation setup is circa 12 hours using a 2.66 GHz, 8GB RAM quad-core system. In the following, we assume results achieved by GA approximation are optimal.

### B. Simulation Results

Based on the TM depicted from Fig. 3a, Fig. 2 shows that, when using S-CORE, the ratio of migrated VMs plummets after the second token-passing iteration. This demonstrates

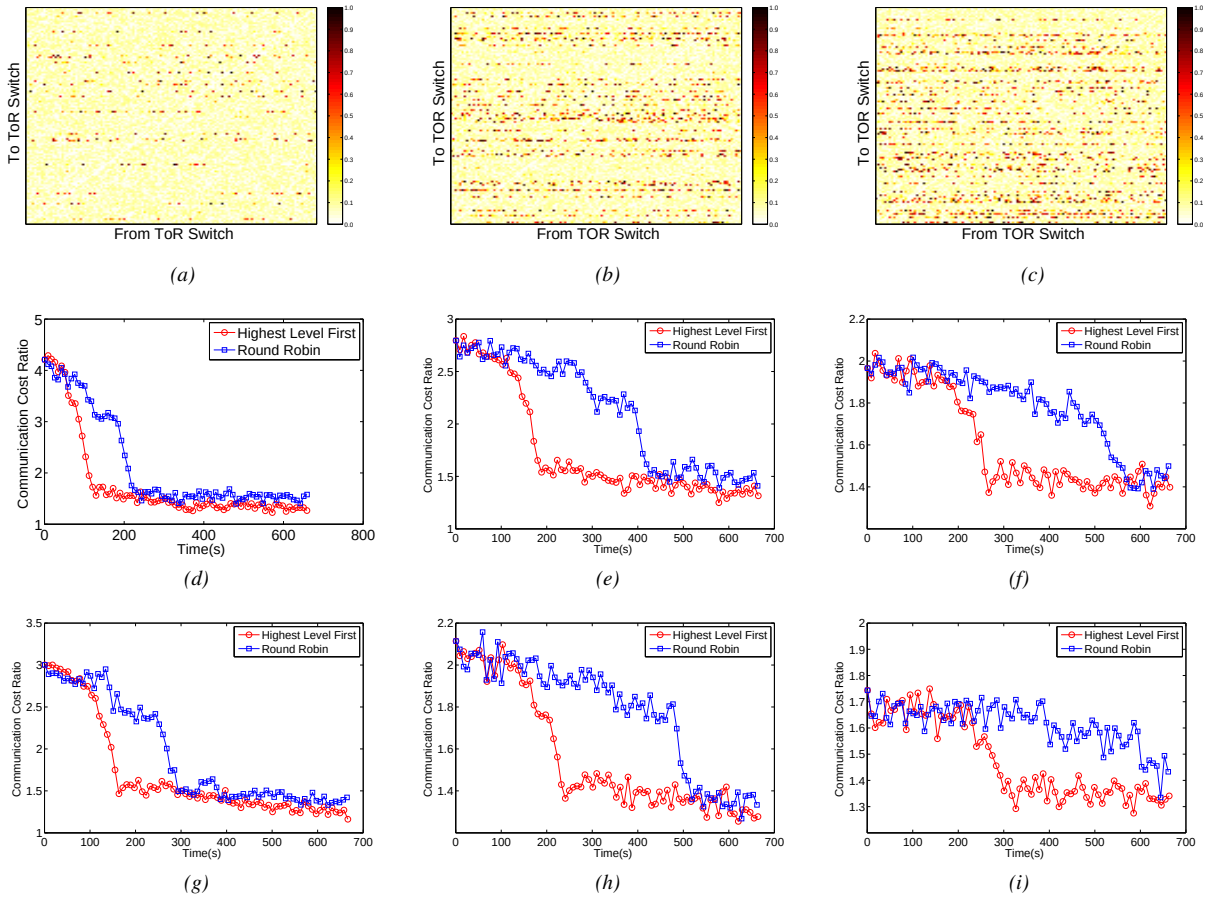
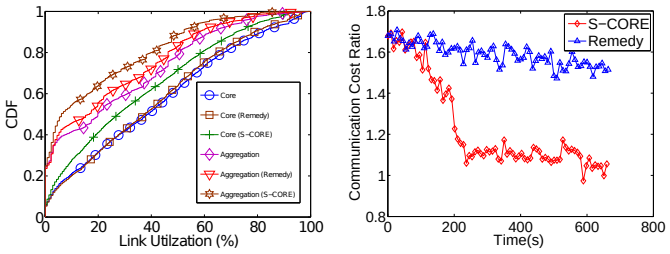



*Fig. 3:* Communication cost reduction ratio over GA-optimal approximation for different token policies under (a) sparse, (b) medium, and (c) dense Traffic Matrices for canonical tree ((d), (e), (f)) and fat-tree ((g), (h), (i)) topologies, respectively.

that S-CORE quickly converges to a stable VM distribution within two token-passing iterations and very few VMs need to migrate after that. S-CORE reduces communication cost by migrating VMs to a resulting allocation that contains pairwise traffic flows within the lower layers of the network. Fig. 3 shows the ratios of communication cost achieved by S-CORE and GA-optimal respectively. Clearly we can see that S-CORE significantly reduces communication cost by as much as 72%-87% of the GA-optimal in all scenarios, using only VM-local load information. This figure also reveals that the ratio of network-wide communication cost reduction (with respect to the optimal cost) achieved under sparse, medium, and dense TMs for both topologies is rapid and substantial. In all scenarios, using the *Highest-Level First* token passing policy exhibits better performance than *Round-Robin* in terms of communication cost reduction speed and proximity to the optimal cost regardless of the topology. It is hence evident that maintaining the additional (communication level) state required by HLF makes a significant difference in the algorithm's efficiency.

Comparing results from canonical (Fig. 3d, 3e, 3f) and fat-tree topologies (Fig. 3g, 3h, 3i), we see that while S-CORE achieves similar proximity to the GA-optimal allocation, there is a smaller reduction ratio for fat-tree. This difference is due to fat-tree's increased path diversity at the lower layers, which alleviates the need for using core layer links. Furthermore, S-CORE deviates more from the GA-optimal as TM becomes denser. Nevertheless, deviation from the GA-optimal merely increases from 13% to 28% even when TM density is drastically increased by a factor of 50. These results demonstrate that S-CORE is a topology-neutral scheme and that it can efficiently optimize network usage through reshaping traffic condition over the network.

VM stability is crucial for dynamic VM migration algorithms since oscillations can potentially have a big impact on the network and the servers. While the possibility of VM oscillations cannot be completely eliminated, S-CORE is insensitive to short-term oscillations for two reasons. First, the algorithm uses average pairwise traffic rates over a certain temporal interval, which can be set suitably long to match the dynamism of the environment while not responding to instantaneous traffic bursts. Moreover, existing DC measurement studies suggest that DC traffic exhibits fixed-set hotspots that change slowly over time. We have shown that S-CORE




*(a)* *(b)*

*Fig. 4:* (a) Comparison of resulting link utilization at core, aggregation and ToR layers between S-CORE and Remedy at stable state; (b) Comparison of communication cost reduction.

captures these characteristics and cluster VMs accordingly. So long as hot destination servers do not change rapidly over time, VMs do not oscillate under S-CORE. Second, VMs do not consider migrating arbitrarily nor do they measure individual flow arrivals and completion. Rather, they only consider migration periodically, when they receive the migration token, and their computation is based on aggregate traffic load over that period. Therefore, the short-term effects of sudden arrivals of mice flows are canceled out when averaged over one iteration of the algorithm. Fig. 2 also shows that when VMs are densely clustered into racks, there are no more migrations, even at the onset of bursty dynamic flows.

To better reflect the performance of S-CORE, it is important to compare it with existing network-aware VM management works. Among these works, [16][17] only consider initial VM placement and host-level bandwidth constraint. Work in [14] describes a VM location optimization problem based in the topological network bandwidth, CPU, and memory resources. Their formulation is complex, taking 3000 seconds to compute VM allocations for 343 hosts. Obviously, their approach does not scale to the size of data centers which often consists of hundreds of thousands of servers. We have directly compared S-CORE with *Remedy* [15], a network-aware VM management sharing some common characteristics. Even though S-CORE is the first distributed formalization of traffic-aware VM migration, both systems monitor and collect link traffic statistics and try to adapt to link conditions through migrating VMs and take pairwise VM traffic into account for reducing communication cost. In addition, they both model and consider communication cost associated with each individual migration. Nevertheless, the most important feature is that both approaches take topological network capacity into account.

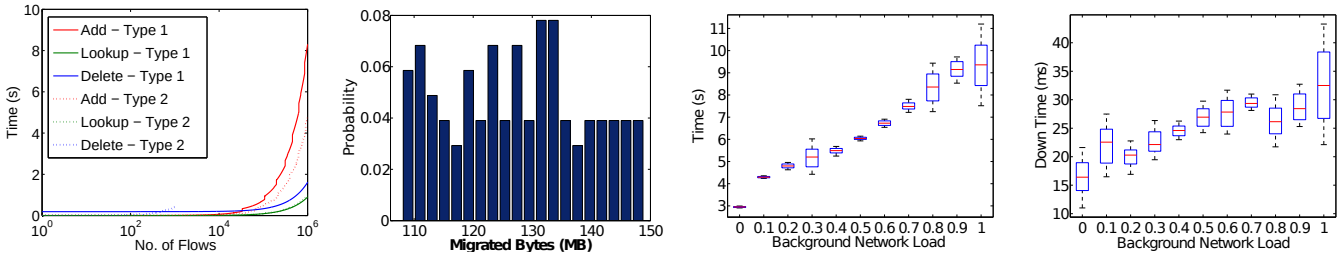
We have implemented Remedy alongside S-CORE in ns-3 and used a sparse TM under which Remedy achieves best results [15]. For a fair comparison, we have used Remedy's migration cost model which estimates the number of migrated bytes as a function of page dirty rate, and set S-CORE's $c_m$ accordingly. Evaluation results are shown in Fig. 4. Fig. 4a demonstrates that S-CORE greatly reduces link utilization on core and aggregation links, whereas Remedy marginally alleviates core link utilization and slightly reduces aggregation link utilization. This is because Remedy tries to balance network traffic as much as possible while S-CORE takes the topology into consideration and explicitly avoids links in higher layers which are often oversubscribed. In addition to link utilization, Fig. 4b shows that while S-CORE significantly improves overall communication cost by 40%, Remedy only reduces it by 10%. This means that Remedy's momentary load balancing approach does not optimize topological capacity utilization. These comparisons demonstrate that S-CORE is more efficient in network resource management with greater potential for providing the operators with increased network capacity headroom in the long run. Remedy, on the other hand, due to its centralized global link monitoring, is more responsive to transient network congestion.

### C. Testbed Results

We evaluated S-CORE implementation over a testbed environment to assess the algorithm's footprint and the performance of actual VM migrations. We have used Intel's P4 3GHz servers with 2GB RAM running Xen hypervisor ver. 4.1 with Ubuntu server 12.04 as dom0. VMs are ubuntu 10.04 with 196MB RAM allocated. In the experiments, initially we started two VMs on each server. Each VM hosts a HTTP server as well as an *iperf* server and client. We have also set-up a Network File System (NFS) server since live migration requires VM images to reside on shared storage. Hosting VM images on shared storage is a commonly used set up in Cloud DC as it has one prominent benefit - only transferring of memory state is needed while keeping the actual file system intact to reduce network usage. Typical VM images are hundreds of MB to tens of GB big in size depends services that they run. Migrating these VM images over the network can impose significant amount of traffic overhead in the DC network and hence should be avoided.

A key module in S-CORE is the flow table, which maintains state for adding, updating, retrieving and removing flows. In order to stress-test the resource consumption of adding flows to the flow table, experiments were conducted for up to 1 million simultaneous flows. We defined two different sets of flows: The first set is 1 million flows with all source IP addresses being unique (type 1). The other set is 1 million unique flows, where groups of 1000 flows share the same source IP address (type 2). Fig.5a shows that flow addition, lookup and deletion operations all require less time on a flow table with a type 2 flow set. Nevertheless, addition, lookup and deletion operations will not need more than 100ms for a realistic DC production workload of 100 concurrent flows. These results evidently demonstrate that S-CORE can run efficiently inside individual machine without incurring computation slowdown.

We have measured over 100 actual migrations and illustrated in Fig. 5b the probability distribution of the number of migrated bytes for each VM migration. The spread appears flat and wide due to the highly varying memory dirty rate at the time when a VM is being migrated. However, the VM memory size to migrate are all below 150MB. The mean and standard deviation of migrated bytes are 127MB and 11MB




*(a)* *(b)* *(c)* *(d)*

*Fig. 5:* (a) Flow table operations for Type 1 and Type2 flows (b) Distribution of migrated bytes per migration; (c) Virtual machine migration time and (d) Down-time under increasing traffic load. Background network traffic is the ratio of 1Gb/s CBR.

respectively. Given today's Cloud DC networks, this additional control load is negligible (1-second's worth of transmission time over a 1 Gb/s link).

Next, we examined whether migration over busier links will impact machine down-time. We set up an experiment where S-CORE operates under background CBR traffic of increasing intensity. We have measured the total migration time, and we used (*fping*) with 1ms interval to determine the VM's actual down-time, as shown in Fig.5c and Fig.5d, respectively. The mean total migration time increases from 2.94s for no background traffic to 4.29s with 100Mb/s of background traffic. With background traffic approaching 100% of link capacity, migration time increases sub-linearly from 4.29s to 9.34s. Arguably, a more important measure is server's down-time, the period during which the VM is unable to service user requests. This happens in the stop-and-copy stage [8] of the live migration process when a VM is suspended, and its CPU state and any remaining inconsistent memory pages are transferred to another server. As shown in Fig. 5d, down-time is an order of magnitude smaller and stays well below 50 ms when link utilization approaches 100%. Hence, although higher link load does have an impact on VM down-time, S-CORE remains efficient even under adverse load conditions.

## VII. Related Work

Virtual Machine migration [24][8], placement and consolidation [4][5] have been considered mainly to improve server resource usage and power consumption. Consolidation has also been suggested for reducing the number of network switches that must be power on at any given time [10]. A limited number of studies have focused on traffic-aware initial VM placement [16] or dynamic migration to satisfy traffic demands [14][25] or to meet SLA requirements [26]. Initial placement studies do not take the current state of the network into consideration, while the rest assume a pre-computed Traffic Matrix that reduces the dynamism and adaptivity of their approaches.

Most relevant to our work, *Remedy* [15] ranks VMs viable for migration based on the network cost of migrating and temporal VM traffic load. Remedy uses aggregate traffic statistics collected from network switches through a centralized OpenFlow-based algorithm, while S-CORE is fully distributed and allows VMs to make migration decisions unilaterally using data collected locally. We have compared the two systems in section VI-B.

## VIII. Conclusion

VM management is a powerful control loop for the provisioning of elastic services over Cloud DC infrastructures that can capture the dynamism and exploit the recourse redundancy of such environments. However, the traffic-agnostic placement of VMs to servers has been reported to result in sub-optimal dynamics that lead to congestion in the core and hinder the overall efficiency of resource usage.

In this paper, we have presented S-CORE: a scalable, traffic-aware VM migration scheme that reduces the topology-wide communication cost in a fully distributed manner, using information available locally at each VM. Through explicitly assigning cost weights to links at different layers of tree topologies, S-CORE can contain traffic at the lower layers, significantly revealing higher-level links that are significantly oversubscribed and prone to congestion. At the same time, through the assignment of different cost weights, the algorithm can be exploited to optimise different performance objectives according to DC operator policy. We have implemented S-CORE and evaluated it extensively over simulated and experimental testbed environments. We have shown that it can achieve up to $87\%$ communication cost reduction compared to an approximate optimal approach and that it is equally applicable to diverse DC network architectures under realistic DC load conditions. S-CORE outperforms centralized VM migration algorithms, while incurring a maximum of circa 40ms VM downtime at extreme load conditions.

## References


[1] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible data center network," in *Proc. ACM SIGCOMM'09*, 2009, pp. 51–62.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM'08*, 2008, pp. 63–74.

[3] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM'11*, 2011, pp. 266–277.

[4] G. Jung, M. Hiltunen, K. Joshi, R. Schlichting, and C. Pu, "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," in *IEEE Int. Conf. on Distributed Computing Systems (ICDCS'10)*, June 2010, pp. 62–73.

[5] A. Verma, P. Ahuja, and A. Neogi, "pMapper: power and migration cost aware application placement in virtualized systems," in *Proc. ACM/IFIP/USENIX Int. Conf. on Middleware*, 2008, pp. 243–264.

[6] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible, "Improving performance and availability of services hosted on IaaS clouds with structural constraint-aware virtual machine placement," in *IEEE Int. Conf. on Services Computing (SCC'11)*, July 2011, pp. 72–79.
[7] S. Mehta and A. Neogi, "ReCon: A tool to recommend dynamic server consolidation in multi-cluster data centers," in *IEEE Network Operations and Management Symp. (NOMS'08)*, Apr. 2008, pp. 363–370.
[8] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *USENIX NSDI'05*, 2005, pp. 273–286.
[9] J. Xu and J. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *IEEE/ACM GreenCom'10*, Dec. 2010, pp. 179–188.
[10] V. Mann, A. Kumar, P. Dutta, and S. Kalyanaraman, "VMFlow: Leveraging VM mobility to reduce network power costs in data centers," in *Proc. IFIP TC 6 Networking Conf.*, ser. LNCS, 2011, vol. 6640, pp. 198–211.
[11] G. Wang and T. Ng, "The impact of virtualization on network performance of Amazon EC2 data center," in *Proc. IEEE INFOCOM'10*, Mar. 2010, pp. 1–9.
[12] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: comparing public cloud providers," in *Proc. ACM SIGCOMM Internet Measurement Conf. (IMC'10)*, 2010, pp. 1–14.
[13] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, Dec. 2008.
[14] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera, "A stable network-aware VM placement for cloud systems," *Proc. IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (CCGRID '12)*, pp. 498–506, 2012.
[15] V. Mann, A. Gupta, P. Dutta, A. Vishnoi, P. Bhattacharya, R. Poddar, and A. Iyer, "Remedy: Network-aware steady state VM management for data centers," in *Proc. IFIP TC 6 Networking Conf.*, ser. LNCS, 2012, vol. 7289, pp. 190–204.
[16] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. IEEE INFOCOM'10*, Mar. 2010, pp. 1–9.
[17] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *Proc. IEEE INFOCOM'11*, Apr. 2011, pp. 71 –75.
[18] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc. ACM SIGCOMM Internet Measurement Conference (IMC'09)*, 2009, pp. 202–208.
[19] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM SIGCOMM Internet Measurement Conf. (IMC'10)*, 2010, pp. 267–280.
[20] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.
[21] F. P. Tso, K. Oikonomou, E. Kavvadia, G. Hamilton, and D. P. Pezaros, "S-CORE: Scalable communication cost reduction in data center environments," School of Computing Science, University of Glasgow, Tech. Rep. TR-2013-338, February 2013.
[22] "Xen hypervisor." [Online]. Available: http://xen.org/
[23] S. Kandula, J. Padhye, and P. Bahi, "Flyways to de-congest data center networks." Proc. ACM HotNets, 2009.
[24] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *USENIX NSDI'07*, 2007.
[25] P. Bodík, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica, "Surviving failures in bandwidth-constrained datacenters," 2012, pp. 431–442.
[26] A. Stage and T. Setzer, "Network-aware migration control and scheduling of differentiated virtual machine workloads," in *Proc. ICSE CLOUD'09*, 2009, pp. 9–14.
[27] C. M. Papadimitriou, *Computational complexity*. Reading, Massachusetts: Addison-Wesley, 1994.

## APPENDIX

We show that the Optimal VM Allocation ($OVMA$) problem at hand does not have a polynomial time solution. $OVMA$ is not a decision problem but rather a typical optimization one, where the optimization goal is to compute whether there is a quantity $\mathcal{A}$ so that Eq. 2 is less than or equal to a target value $J$. We simplify the problem by considering only one communication link with cost $c_1$. In the sequel, we will show that $OVMA \in NP$ and then we reduce a known NP-Complete problem to $OVMA$ in polynomial time [20] (or in logarithmic space [27]).

In order to show that an optimization problem is in $NP$, the traditional way is to show that the following property is satisfied: for each "yes" instance there exists a "proof" or "certificate" of polynomial size, whereas "no" instances have no polynomial "certificates". $OVMA$ has this property since the certificate is an allocation $\mathcal{A}$ which is polynomial in the size of the input and it exists if and only if this allocation achieves the goal $J$.

The next step is to reduce a known NP-complete problem to $OVMA$. Note that a problem $X$ is at least as hard as problem $Y$, if $Y$ reduces to $X$, [20], [27]. We will consider the Graph Partitioning ($GP$) problem [20] which will be reduced to $OVMA$. For completeness, $GP$ is stated below:

INSTANCE: Graph $G = (V, E)$, weights $w(v) \in \mathbb{Z}^+$ for each $v \in V$ and $l(e) \in \mathbb{Z}^+$ for each $e \in E$, positive integers $K$ and $J$.

QUESTION: Is there a partition of $V$ into disjoint sets $V_1, V_2, \ldots, V_m$ such that

$$\sum_{v \in V_i} w(v) \leq K$$

for $1 \leq i \leq m$ and such that if $E' \subseteq E$ is *the set of edges that have their two endpoints in two different sets* $V_i$, then

$$\sum_{e \in E'} l(e) \leq J \quad ?$$

In our reduction we shall use the version of $GP$ *with vertex weight 1*, which is still NP-Complete for $K \geq 3$ (can be solved in polynomial time when $K = 2$ by matching [20]). Consider the following straightforward reduction:

- the set of VMs $\mathbb{V}$ is $V$, i.e., $\mathbb{V} = V = \{v_1, v_2, \ldots, v_n\}$,
- the traffic load $\lambda(u, v)$ between VMs $u$ and $v$ is defined as follows: $\lambda(v_i, v_j) = l(e)$, if in the undirected graph $G$ there exists an edge $e$ between $u$ and $v$ and is taken to be 0 if there is no edge between $u$ and $v$ in $G$,
- $K \in \mathbb{Z}^+$ is the rack capacity, i.e., how many virtual machines a rack may accommodate,
- the fact that the vertex weights are taken to be 1 satisfies the assumption that all VMs are equivalent in weight,
- the goal $J \in \mathbb{Z}^+$ for $OVMA$ is precisely the goal $J$ of the $GP$, and
- the original question whether there is a partition of $V$ into disjoint sets $V_1, V_2, \ldots, V_m$ now becomes the question whether there is an allocation of virtual machines to racks $r_1, r_2, \ldots, r_m$.

The above reduction is trivial and can be carried in polynomial time. Therefore, $GP$ with vertex weight 1 reduces polynomially to $OVMA$, which completes the proof that $OVMA$ is NP-Complete.