Live migration of virtual machines among edge networks via WAN links

Donatella Darsena[1], Giacinto Gelli[2], Antonio Manzalini[3], Fulvio Melito[2], and Francesco Verde[2]
[1]*Parthenope University - DIT, Naples, I-80143, Italy*
Tel: +39 0815476741, Email: darsena@uniparthenope.it
[2]*University Federico II - DIETI, Naples, I-80125, Italy*
Tel: +39 0817683121, Email: [gelli, fulvio.melito, f.verde]@unina.it
[3]*TELECOM ITALIA Strategy - Future Centre, Turin, I-10148, Italy*
Tel: +39 011 2285817, Email: antonio.manzalini@telecomitalia.it

Abstract: This paper addresses the potential impact of emerging technologies, like software defined networking (SDN) and network virtualization (NV), on future network evolution. It is argued that the above mentioned technologies could bring a significant disruption at the edge networks, where it will be possible to develop distributed clouds of virtual resources running on standard hardware. This paper deals with a key technical challenge behind this vision: the capability of dynamically moving virtual machines (VMs), which run network services, functions, and users applications, among edge networks across wide area interconnections. Specifically, we focus on the problem of live migrating the memory state of a single VM between two physical machines, which are located at the edge and are inter-connected by a wide area network (WAN). With reference to a pre-copy mechanism, aimed at iteratively transferring the memory content to the destination machine, we develop a simplified mathematical model that unveils the dependence of the total migration time and downtime of the VM memory transfer on the main WAN parameters, such as capacity, buffering, and propagation delay. Numerical results obtained by Matlab simulations are presented to demonstrate the feasibility of live VM migration across WANs and the directions of RT&D activities.

Keywords: Edge networks, live migration of virtual machines, network function virtualization, software defined networks.

1. Introduction

Information and communications technologies are progressing at an impressive rate: processing is still following Moore's law, doubling in capability roughly every 18 months; storage capacity on a given chip is doubling every 12 months, driving a steady increase in connectivity demand for network access; optical bandwidth is doubling every 9 months, both by increasing the capacity of a single-wavelength fiber, and by transmitting multiple wavelengths on a single fiber. These progresses, and the down-spiralling of costs, are expected to have a dramatic impact on the evolution of network architectures: future networks (Fig. 1) are likely to become less hierarchical and based on optical core infrastructures (with a limited number and types of large nodes), which interconnect (by optical and/or radio links) different local areas, and are populated, at the edge (i.e., in a range of few meters around users) with a sheer number of heterogeneous nodes. The edge, toward which the process of "intelligence" migration is already in act, will become the business area where a new galaxy of ecosystems will be created.

In this scenario, the role of software will represent the true challenge: indeed, future networks will rely more and more on software, which will accelerate the pace of innovation (as it is doing continuously in the computing and storage domains). Already today,

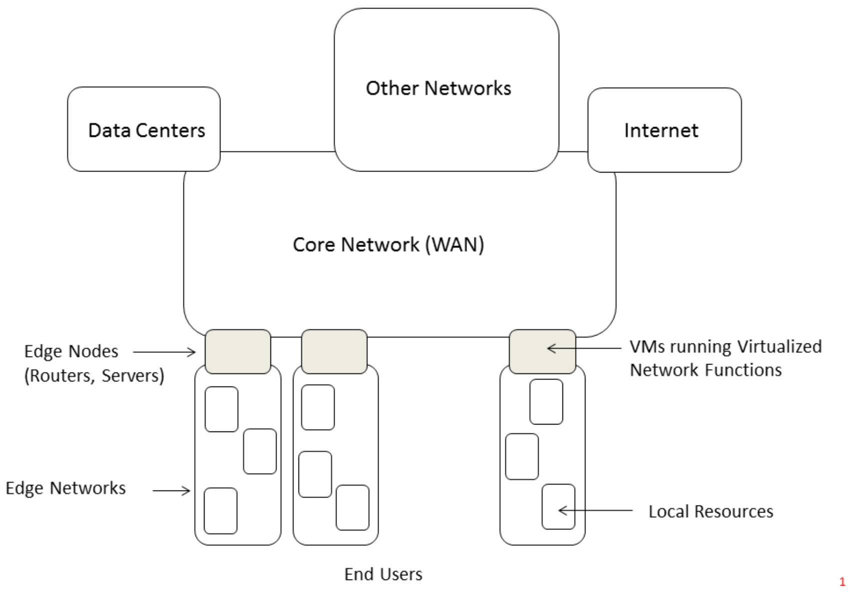

Figure 1: Edge networks scenario.

advances in resource virtualization are driving the deployment, on the same physical infrastructure, of diverse coexisting and isolated virtual networks of resources, which allows one to best fit, in a dynamical manner, a variety of service demands, similarly to having different operating systems (e.g., Windows, IoS, and Linux) on the same laptop. This has multiple advantages: for example, the crash, or the misuse, of a virtual resource is confined within a virtual network (e.g., by applying fault recovery policies enforced by self-healing capabilities), having no impact on other virtual networks; it is possible to implement, in each virtual network, specific procedures and policies (e.g., to optimize the usage of allocated resources according to service level agreements, SLA); the use of physical resources can be optimized, etc.

Software defined networking (SDN) [1, 2] can be seen as a further step in the direction of decoupling hardware from software: in particular, in an SDN architecture, control and data planes are decoupled, so that the network infrastructure is abstracted from business applications. The SDN paradigm should not be confused with *network virtualization* (NV) [3], even if the two concepts could intersect with interesting possibilities: indeed, NV is the second most-important trend allowing the setup of virtual networks by connecting virtual IT and networking resources. The above mentioned technologies are expected to bring about both programmability and flexibility: for example, it will be possible to build, on the same physical infrastructure, multiple overlay networks offering different services. On the other hand, this evolution is also determining an increase in design and management complexity: future networks are likely to exhibit the characteristics of complex systems, consisting of many diverse and autonomous, but interrelated, software and hardware components. Traditional management and control approaches will no longer be applicable: networks should be able to self-adapt and self-configure themselves (with limited human intervention). These capabilities can be achieved by introducing *autonomics* and *cognition* as a transformative software technology.

NV also enables live migration [4], i.e., a whole virtual machine (VM) can be moved between different physical machines (PMs) without disconnecting the client or application. Such a technology has already been proven to be a very effective tool in local area networks (LANs), i.e., when the VMs and the PMs are located within the same data center, to achieve the goals of server consolidation, load balancing, and hotspot mitigation.

In a cloud-based edge scenario, data centers can be spread over wide areas and, thus, the need for migrating VMs over WANs arises. Compared to LAN applications, migration of VMs across WAN links poses [5, 6] three additional challenges: 1) shared storage systems and/or storage area networks for VM disks may not always be available in WAN scenarios and, thus, in addition to random-access memory (RAM) transfer, disk state migration is required in this case (***storage migration issue***); 2) the bandwidth-delay product[1] of a WAN is typically high compared to buffering resources of the network [7], thus adversely impacting on both the total time of the migration and the VM downtime (***high bandwidth-delay issue***); 3) maintaining network level (i.e., IP addresses) reachability after migration is a difficult task in a WAN environment, since moving across different networks forces the VM to get a new IP address and, consequently, breaks existing network connections (***network reconfiguration issue***). To make WAN migration appealing, the requirements in terms of consistency, responsiveness, reliability, and scalability have to be studied thoroughly for cloud-based edge networks.

The paper is organized as follows. Section 2 describes some future network scenarios enabled by the technology advances in processing, storage, networking, and embedded communications. Section 3 provides a mathematical modeling of VM live migration across a WAN, by linking the performance of the RAM migration process to the main network parameters. Monte Carlo simulation results, obtained in a Matlab environment, are presented in Section 4. In Section 5, final remarks elaborate about the lesson learnt and the future steps.

2. Network scenarios

Today, traditional networks are suffering an “ossification”, which is creating several limitations for fast and flexible deployment, as well as adaptation of network functionality, services, and management policies. Launching new services is quite complex and expensive. In order to cope with the growing dynamism of ICT markets, it is becoming increasingly urgent to find technologies and solutions solving the above mentioned limitations in future networks. Moreover, there is the need of reducing operational and capital expenditures (OPEX and CAPEX). OPEX reduction can be achieved by easing human operators (and reducing human mistakes) in managing and automatically configuring equipment and network functionality. CAPEX reduction can be achieved by postponing network resources investments (e.g., optimized use of available resources), for example, by exploiting layer and cross layer “constrained optimizations” (e.g., through several network applications in charge of load balancing, traffic engineering, optimized resource allocations, etc.). Eventually, in contrast to today, where competition exists only at the application level, future network should open new dimensions of business: incentives, cooperation, and competition will boost the long-term value of the network

[1]Roughly speaking, the bandwidth-delay product of a data connection is the round-trip delay times the capacity of the underlying physical link.

– like in ecosystems, where evolution selects the winning species, winning services will succeed, grow, and promote further investments, while losing ideas will fade away.

One of the most promising scenarios is that, in the short-medium term, SDN and NV principles will mature to bring a profound innovation at the edge of traditional networks, where, by the way, "intelligence" is already migrating. In this scenario, technology advances are bringing into the reality the concrete possibility of using tools and the solutions adopted in data centers for managing and orchestrating clouds of virtual resources. This will offering to network operators not only the ability to dynamically instantiate, activate, and re-allocate resources and network functions, but even programming them according to need and policies. Nevertheless this scenario has several challenges [5, 6]: one of these, covered by this paper, is deploying systems and methods capable of seamless migration of ensembles of VMs across WAN connections.

3. Live migration of virtual machines across WANs

The considered scenario is composed of a source physical machine (SPM) and a destination physical machine (DPM) that can be connected to each other by means of a wide-area network (WAN). We assume that the physical resources, i.e., CPU, memory, and input/output (I/O) devices of both source and destination are virtualized, so that multiple virtual machines (VMs), each of them self-contained with its own operating system (OS), can execute specific applications on the physical machine.

Hereinafter, we focus on live migration of a single VM from the source to the destination. Live migration of a VM across a WAN requires [5, 6]: (i) network connection maintenance; (ii) disk state migration; (iii) RAM state migration. In this paper, we restrict our attention to RAM state migration, by assuming that: (a1) a protocol is used to separate the VM's identifier from its topological location, such as, the Locator/Identifier Separation Protocol (LISP) [8]; (a2) either a shared storage system for VM disks exists or a distributed replicated block device (DRBD) disk replication system [5] is employed to migrate storage to the destination.

Many migration techniques use a "pre-copy" mechanism [4] to *iteratively* copy the memory state of a VM from the source to the destination, while the OS continues to run. The OS of the VM has access to its own private, segmented name space; each segment known to the OS is sliced into equal-size units, called *pages*, to facilitate its mapping into the paged main memory. Let $\mathcal{M} \triangleq \{m_1, m_2, \ldots, m_M\}$ be the *set of memory pages* characterizing the OS of the VM, we consider a single source-destination link with capacity[2] of C pages per second, and a first-in first-out (FIFO) buffer of size B pages. Basically, pages are injected into the transport buffer by the upper layer protocol stack at a certain rate and, then, they are removed and transmitted by the physical link. In the sequel C^{-1} is referred to as the *service time*, which is the time needed to transmit a single page over the link. When the buffer is full, any page subsequently arriving is dropped (*buffer overflow*). After receiving a page, the destination is assumed to immediately send back an acknowledgement (ACK); such ACKs are cumulative in the sense that they also indicate the next page expected by the destination. Page drops due to buffer overflow are detected by either the receipt of duplicate ACKs or the expiration of a timer. In order not to complicate the analysis excessively, we assume

[2]To simplify the presentation, we will assume that all the relevant network parameters are measured in units of pages, instead of bytes.

that pages cannot be randomly lost after being transmitted over the physical link.

For the sake of simplicity, we assume that all the M pages experience the same (deterministic) propagation delay, which is denoted by τ, and includes: (i) the time between the transmission of a page from the source and its arrival into the link buffer; (ii) the time between the transmission of the page on the link and its arrival at the destination; (iii) the time between the arrival of the page at the destination and the arrival of the corresponding acknowledgement at the source. Let $T \triangleq \tau + 1/C$ denote the propagation delay plus the service time, which will be referred to as ***round-trip time***, and define the normalized buffer size $B_{\text{norm}} \triangleq B/(C\,T) = B/(C\,\tau + 1)$. Since we are concerned with transmission across a WAN, we restrict our attention to the case when $B_{\text{norm}} \leq 1$ [7], i.e., when the buffer size B on the link is of the same order of magnitude as, or smaller than, the ***bandwidth-delay product*** [7] between the link capacity C and the round-trip time T. On the contrary, in LAN scenarios, it results that $B_{\text{norm}} > 1$.

3.1 Congestion control algorithm

At the transport layer, the source-destination connection is assumed to use a congestion control algorithm, in order to regulate the amount of pages being injected into the WAN. To do this, the source uses a *congestion window* of size W, which sets a limit on the maximum number of pages that can be transmitted over the link before receiving an ACK. The value of W is varied dynamically in response to the network congestion: specifically, it is increased whenever a new page is acknowledged and is decreased whenever a page drop is detected. The maximum size of the congestion window in steady state is given [7] by $W_{\text{max}} \triangleq B + C\,T$, which is achieved when the buffer is fully occupied and there are $C\,T$ pages travelling along the bit pipe. We assume that page drops occur at the window size $W = W_{\text{max}}$ and, to simplify the analysis, we neglect the slow start phase [7]. The considered congestion control algorithm is an oversimplified version of the TCP-reno [9] that can be summarized as follows:

- *Initialization:* set $W = W_{\text{max}}/2$.
- *Congestion avoidance phase:* when an ACK is received, if $W < W_{\text{max}}$, then $W = W + 1/[W]$, where $[W]$ denotes the integer part of W.
- *Page dropping:* when $W = W_{\text{max}}$, a new *cycle* begins with $W = W_{\text{max}}/2$.

The time interval from the end of a congestion avoidance phase to the end of the next one is referred to [7] as a cycle: the evolution of the algorithm is periodic in the sense that successive cycles are identical. To evaluate the *average* page throughput λ of the connection, i.e., the number of pages transmitted per second, we report the main results of the analysis carried out in [7] by using a continuous-time approximation of the congestion control evolution.

Let $\mathrm{d}W/\mathrm{d}t$ denote the rate of window growth with time, $\mathrm{d}W/\mathrm{d}a$ the rate of window growth with arriving ACKs, and $\mathrm{d}a/\mathrm{d}t$ the rate at which the ACKs are arriving. In our case, it results that $\mathrm{d}W/\mathrm{d}a = 1/W$ and $\mathrm{d}a/\mathrm{d}t = \min\{W/T, C\}$, which leads to

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \frac{\mathrm{d}W}{\mathrm{d}a}\,\frac{\mathrm{d}a}{\mathrm{d}t} = \begin{cases} \dfrac{1}{T}, & W \leq C\,T; \\[2ex] \dfrac{C}{W}, & \text{otherwise.} \end{cases} \tag{1}$$

Let $W(t)$ denote the value of the congestion window at time t, with $W(0) = W_{\max}/2$, eq. (1) shows that, when $W \le CT$, one has that $W(t) = W_{\max}/2 + t/T$ for $t \in (0, t_1)$, with

$$t_1 \triangleq T \left(CT - \frac{W_{\max}}{2} \right) . \tag{2}$$

The number of pages transmitted during the interval $t \in (0, t_1)$ is given by

$$n_1 = \int_0^{t_1} \frac{W(t)}{T} \, \mathrm{d}t = \frac{W_{\max}}{2} \frac{t_1}{T} + \frac{t_1^2}{2T^2} . \tag{3}$$

When $W > CT$, it follows from (1) that $W^2(t) = 2C(t - t_1) + (CT)^2$ for $t \in (t_1, t_1 + t_2)$, with

$$t_2 \triangleq \frac{W_{\max}^2 - (CT)^2}{2C} \tag{4}$$

where at the time $t_1 + t_2$ the window size is equal to $W_{\max}$ and a buffer overflow occurs. Since the link is fully utilized during the interval $(t_1, t_1 + t_2)$, the number of pages transmitted is

$$n_2 = C t_2 = \frac{W_{\max}^2 - (CT)^2}{2} . \tag{5}$$

The average page throughput assumes the expression

$$\lambda = \frac{n_1 + n_2}{t_1 + t_2} \tag{6}$$

which depends of the buffer size B, the link capacity C, and the round-trip time τ: hence, the time needed to transmit a single page is $1/\lambda$ on average.

3.2 Migration algorithm

The main drawback of live migrating the VM's memory image stems from the fact that memory pages are continuously modified, or "dirtied", by the OS, and, hence, there is a high probability that those pages, which are frequently updated, may be transferred multiple times during the migration process. Since only the final version of a page is needed at the destination, repeated transfers lead to an unnecessary waste of resources. It has been observed in [4] that, in practice, a certain (possibly large) set of pages is seldom or never be modified by the OS, while the remaining ones are frequently dirtied. In order to account for this fact, it is assumed in [10] that each page m_i is characterized by the probability that it is dirtied at least one time during the transmission of a single page, for $i \in \{1, 2, \ldots, M\}$. In Section 4, we resort to a more general probabilistic model of the rewriting events, which is readily linked to the WAN parameters.

The migration process starts with the VM hypervisor marking all memory page as dirty. Then, the pre-copy algorithm [4] iteratively transfers dirty pages from the source to the destination until the number of pages remaining to be transferred is below a certain threshold $M_{\max}$ or a maximum number of iterations $K_{\max}$ is reached. The hypervisor keeps track of those pages that are modified by using a *dirty page bitmap*, i.e., a memory structure where a bit is set for each dirty page. During live migration, the bitmap is scanned and, if a page is marked as dirty, it is transferred to the destination in the subsequent iteration. Let

$$\phi_0 : \{1, 2, \ldots, M\} \to \{\phi_0^{(1)}, \phi_0^{(2)}, \ldots, \phi_0^{(M)}\} \tag{7}$$

be an *initial page transfer ordering function*, which is assumed to be invertible and its inversion function is denoted by ϕ_0^{-1}, whose choice depends on the dirty page rate, with $\phi_0^{(i)} \in \{1, 2, \ldots, M\}$ and $\phi_0^{(i)} \neq \phi_0^{(j)}$ for $i \neq j$. On the basis of such an ordering, the pages are arranged in the new order $m_{\phi_0^{(1)}}, m_{\phi_0^{(2)}}, \ldots m_{\phi_0^{(M)}}$, wherein the position of page m_i turns out to be given by $\phi_0^{-1}(i)$, for $i \in \{1, 2, \ldots, M\}$. We assume that only the pages belonging to $\mathcal{M}_0 \triangleq \{m_{\phi_0^{(1)}}, m_{\phi_0^{(2)}}, \ldots, m_{\phi_0^{(M_0)}}\}$ are live migrated, with $M_0 \leq M$, whereas the remaining $M - M_0$ ones belonging to $\overline{\mathcal{M}}_0 \triangleq \{m_{\phi_0^{(M_0+1)}}, m_{\phi_0^{(M_0+2)}}, \ldots, m_{\phi_0^{(M)}}\}$ are transferred at the end of the migration process when the VM is stopped.

More precisely, the pre-copy migration process herein studied works as follows:

- At the first iteration ($k = 0$), all the M_0 pages belonging to $\mathcal{M}_0$ are transmitted to the destination in an orderly way (*starting phase*), where $m_{\phi_0^{(1)}}$ is the first page to be transferred, thus spanning a time interval of M_0/λ seconds.
- At the kth iteration, for $k \in \{1, 2, \ldots, k_{\text{end}}\}$, after bitmap scanning, the pages in the subset $\mathcal{D}_k \triangleq \{m_{d_k^{(1)}}, m_{d_k^{(2)}}, \ldots, m_{d_k^{(M_k)}}\} \subseteq \mathcal{M}_0$ result to be marked as dirty, with $M_k \leq M_0$, $d_k^{(i)} \in \{\phi_0^{(1)}, \phi_0^{(2)}, \ldots, \phi_0^{(M_0)}\}$, and $d_k^{(i)} < d_k^{(j)}$ for $i \neq j$, and, hence, if $M_k > M_{\text{max}}$, they have to be retransmitted (*push phase*), thus spanning a time interval of M_k/λ seconds; specifically, let
$$\phi_k : \{1, 2, \ldots, M_k\} \to \{\phi_k^{(1)}, \phi_k^{(2)}, \ldots, \phi_k^{(M_k)}\} \tag{8}$$
be a *kth page transfer ordering function*, which is assumed to be invertible and its inversion function is denoted by ϕ_k^{-1}, whose choice depends on the dirty page rate, with $\phi_k^{(i)} \in \{d_k^{(1)}, d_k^{(2)}, \ldots, d_k^{(M_k)}\}$ and $\phi_k^{(i)} \neq \phi_k^{(j)}$ for $i \neq j$, all the M_k pages belonging to $\mathcal{M}_k \triangleq \{m_{\phi_k^{(1)}}, m_{\phi_k^{(2)}}, \ldots, m_{\phi_k^{(M_0)}}\}$ are live migrated in an orderly way, where $m_{\phi_k^{(1)}}$ is the first page to be transferred.
- at the last iteration $k = k_{\text{end}}+1$, where $k_{\text{end}} = K_{\text{max}}$ if $M_k > M_{\text{max}}$ for each $k \in \{1, 2, \ldots, K_{\text{max}}\}$ or $k_{\text{end}} = k_{\text{f}}$ if $M_{k_{\text{f}}} \leq M_{\text{max}}$ with $k_{\text{f}} \in \{1, 2, \ldots, K_{\text{max}} - 1\}$, the VM is stopped and, let $\mathcal{M}_{k_{\text{end}}+1} \subseteq \mathcal{M}_0$ denote the subset of cardinality $M_{k_{\text{end}}+1} \leq M_0$ collecting the pages that are marked as dirty at the end of the push phase, the pages in $\mathcal{M}_{k_{\text{end}}+1} \cup \overline{\mathcal{M}}_0$ are transmitted in an arbitrary order (*stop-and-copy phase*), which spans a time interval of $M_{k_{\text{end}}+1}/\lambda + (M - M_0)/\lambda$ seconds.

When the complete memory image has been transferred, the VM is resumed at the destination and the live migration process is complete. The time T_{down} required to complete the stop-and-copy phase is referred to as *downtime*, whereas the time T_{tot} needed to collectively finish the starting, push, and stop-and-copy phases is called *total migration time*. It is readily seen that
$$T_{\text{down}} = \frac{M_{k_{\text{end}}+1} + (M - M_0)}{\lambda} \tag{9}$$
$$T_{\text{tot}} = \frac{M_0}{\lambda} + \sum_{k=1}^{k_{\text{end}}} \frac{M_k}{\lambda} + T_{\text{down}} = \frac{M}{\lambda} + \sum_{k=1}^{k_{\text{end}}+1} \frac{M_k}{\lambda} \, . \tag{10}$$
To avoid service interruption, consistency issues, and unpredictable performance, the downtime has to be as small as possible. Moreover, in many scenarios, it is desirable to reduce the total migration time in order to free as fast as possible the VM resources for other use or to avoid waste of network resources, such as bandwidth and power.

4. Numerical performance analysis

Herein, we aim at evaluating the average downtime $\overline{T}_{\text{down}}$ and the average total migration time $\overline{T}_{\text{tot}}$ by Monte Carlo computer simulations carried out in Matlab environment: both $\overline{T}_{\text{down}}$ and $\overline{T}_{\text{tot}}$ are obtained by averaging (9) and (10) over 10^4 Monte Carlo trials.

As first step, we have to specify the probabilistic model of the page rewriting events. For $i \in \{1, 2, \ldots, M\}$, let $N_i(t)$ denote the random number of rewriting events (the count) of page m_i at time t, assuming a zero count at $t = 0$. We assume that the sequence of rewritings in the counting process $N_i(t)$ is a ***Poisson point process*** [11], i.e., rewriting events of page m_i occurs independently of one another, and at a fixed average rate of γ_i rewritings per second, and the event of two rewritings at precisely the same time is impossible, with $N_i(t)$ statistically independent of $N_j(t)$, for $i \neq j$. We refer to γ_i as ***average rewriting rate*** of page m_i. Under these assumptions, the probability $p_i(n)$ that the page m_i is dirtied at least one time during a time interval of duration n/λ, i.e., the average time needed to transmit n consecutive pages, is given by

$$p_i(n) = 1 - e^{-\frac{\gamma_i n}{\lambda}} \,. \tag{11}$$

It is worth noting that, if $\gamma_i \gg \lambda/n$, then $p_i \approx 1$, that is, the page m_i is almost surely dirtied during the transmission of n pages; this is just the case when live transfer of page m_i is wasteful of time, bandwidth, and power. On the other hand, when $\gamma_i \ll \lambda/n$, it follows that $p_i \approx 0$ and, hence, page m_i is not dirtied n/λ seconds almost surely.

At the end of the kth iteration of the migration algorithm, for $k \in \{0, 1, \ldots, K_{\text{max}}\}$, the probability $p_i^{(k)}$ that page m_i is dirtied at least one time is given by

$$p_i^{(k)} = \begin{cases} p_i\left(M_k - \phi_k^{-1}(i) + 1\right), & k = L_i \\[2ex] p_i\left(M_{L_i} - \phi_{L_i}^{-1}(i) + 1 + \displaystyle\sum_{\ell=L_i+1}^{k} M_\ell\right), & k > L_i \end{cases} \tag{12}$$

where $L_i \in \{0, 1, \ldots, k\}$ is the index of the last iteration in which the page m_i has been retransmitted. For each iteration of the migration algorithm, the entries of the dirty page bitmap are randomly generated in each Monte Carlo trial as independent Bernoulli random variables with success (i.e., rewriting) probabilities given by (12).

For the sake of simplicity, we assume in the following example that all the M pages are characterized by the same average rewriting rate, i.e., $\gamma_i = \gamma$ for each $i \in \{1, 2, \ldots, M\}$, and they are migrated in a random order (we used the randperm function of Matlab to ordering the pages at each iteration of the migration algorithm). Regarding the parameters of the migration algorithm, we set $\mathcal{M}_0 = \mathcal{M}$ or,equivalently, $M_0 = M$, and we used as stop conditions $M_{\text{max}} = M/10$ and $K_{\text{max}} = 5$. Moreover, we fixed the WAN parameters as follows: $C = 31250$ pages per second (i.e., the link capacity is 1 Gbps and the size of the pages is 4 kB), $B = 5000$ pages, and $\tau = 0.5$ seconds, which lead to $\lambda = 28710$ pages per second and $B_{\text{norm}} = 0.32$.

Fig. 2 reports the average downtime $\overline{T}_{\text{down}}$ and average total migration time $\overline{T}_{\text{tot}}$ as a function of the ratio γ/λ for different values of the number of pages $M \in \{10, 50, 100\}$. Results show that $\overline{T}_{\text{down}}$ and $\overline{T}_{\text{tot}}$ rapidly increase as the ratio γ/λ raises by (approximatively) saturating to the values M/λ and $(K_{\text{max}} + 1)\,(M/\lambda)$, respectively, which correspond to the case in which all the M pages are dirtied at the end of each iteration

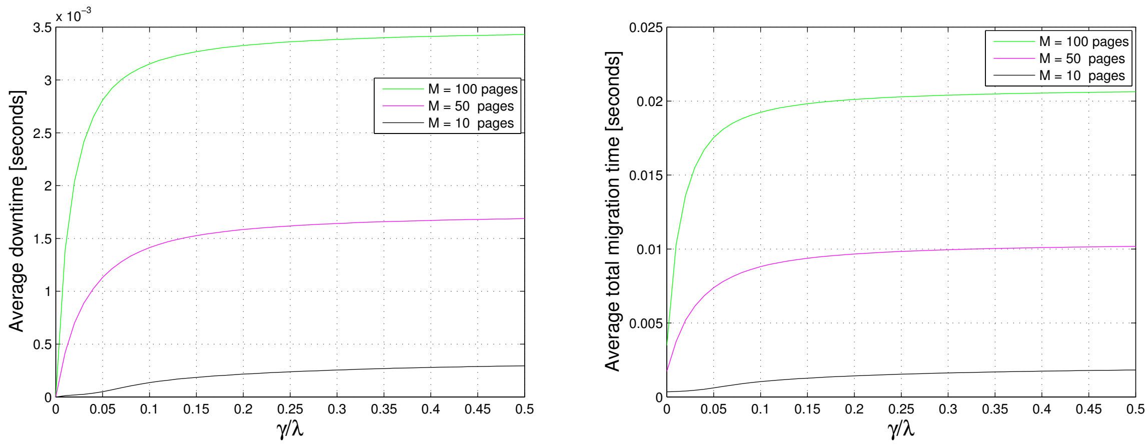

Figure 2: Average downtime and average total migration time versus γ/λ.

of the migration algorithm. It can be argued that, with respect to the case when all the pages are transmitted while the VM is stopped (*non-live migration*), live migration leads to a significantly smaller downtime only when the average rewriting rate γ is a very small fraction of the average page throughput λ. For instance, with reference to the case of $M = 100$ pages, when γ is the 2.5% of λ, the difference between live and non-live migration is about 1.5 ms in terms of downtime. Finally, results not reported here for the sake of brevity show that, as expected, $\overline{T}_{\text{down}}$ and $\overline{T}_{\text{tot}}$ are monotonously decreasing functions of the WAN parameters B and C, whereas they monotonously increase as a function of τ; however, migration performances are influenced more by the link capacity C than by the buffer size B and the propagation delay τ.

5. Conclusions

This paper has argued that, in the short-medium term, SDN and NV principles could bring a profound innovation at the edge of traditional networks. As a matter of fact, we are already witnessing a migration of processing power, storage capability, and embedded communications towards the edge of the network, i.e. towards the end users. This combination of drivers and trends will create the conditions where the end users [both residential and small-to-medium enterprises (SMEs) and large enterprises (LEs)] will “drive the network dynamics” more and more: the edge will become a distributed networking and computing environment, a sort of commodity fabric, capable of offering and providing ICT services through a galaxy of ecosystems. This vision will require overcoming current network ossifications by introducing features for fast and flexible deployment, as well as adaptation of network functionality, services and management policies: one example of these features is the capability of orchestrating ensembles of VMs across multiple edge networks interconnected by WAN links.

In particular, this paper has addressed the development of a mathematical model for performance analysis of live migration of a single VM over a WAN link. The derived model unveils the dependence of the total migration time and downtime of the VM memory transfer on the main WAN parameters, such as capacity, buffering resources, and propagation delay. In particular, the performance of a “pre-copy” migration algorithm with page ordering has been considered. Simulation results have shown that live migration across WANs allows to reduce the downtime only when the average rewriting rate of the pages is a very small fraction of the average page throughput.

Future work will be focused on completing the theoretical analysis of the considered scenario, and validating the obtained result against real data migration figures; moreover, an interesting development is extending the main methodologies proposed in the paper to tackle the more challenging scenario of migrating multiple VMs across WAN links, as well as using the developed framework to improve migration performance through constrained optimization approaches.

References

[1] N. McKeown et al., "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, 2008.

[2] Open Networking Foundation, "Software-defined networking: The new norm for networks," *white paper*, Apr. 2012.

[3] L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization," *IEEE Computer*, vol. 38, pp. 34–41, Apr. 2005.

[4] C. Clark et al., "Live migration of virtual machines," in *Proc. 2nd Symposium on Networked Systems Design and Implementation*, Boston, Massachusetts, USA, May 2005, pp. 273– 286.

[5] K.K. Ramakrishnan, P. Shenoy, and J. Van der Merwe, "Live data center migration across WANs: A robust cooperative context aware approach," in *Proc. 2007 ACM SIGCOMM Workshop on Internet Network Management*, Kyoto, Japan, Aug. 2007, pp. 262– 267.

[6] M. Mishra, A. Das, P. Kulkami, and A. Sahoo, "Dynamic resource management using virtual machine migrations," *IEEE Commun. Mag.*, pp. 34–40, Sep. 2012.

[7] T.V. Lakshman and U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay product and random loss," *IEEE/ACM Trans. Networking*, vol. 5, pp. 336–350, June 1997.

[8] L. Jakab, A. Cabellos-Aparicio, F. Coras, D. Sauces, and O. Bonaventure, "LISP-TREE: A DNS hierarchy to support the LISP mapping system," *IEEE J. Select. Areas Commun.*, vol. 28, pp. 1332 – 1343, Oct. 2010.

[9] S. Floyd and V. Jacobson, "Berkley TCP evolution from 4.3-tahoe to 4.3-reno," in *Proc. 18th Internet Engineering Task Force*, Vancouver, Canada, Aug. 1990.

[10] F. Checconi, T. Cucinotta, and M. Stein, "Real-time issues in live migration of virtual machines," in *Proc. 2009 International Conference on Parallel Processing*, Delft, The Netherlands, Aug. 2009, pp. 454– 466.

[11] W. A. Gardner, *Introduction to Random Processes.* New York: McGraw-Hill, 1990.