SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2014; **7**:492–502
Published online 28 February 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.740

RESEARCH ARTICLE

Verifying cloud service-level agreement by a third-party auditor

Hongli Zhang[1], Lin Ye[1], Jiantao Shi[1], Xiaojiang Du[2]* and Mohsen Guizani[3]

[1] School of Computer Science and Technology, Harbin Institute of Technology, Harbin, 150001, China
[2] Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, U.S.A.
[3] Qatar University, Doha, Qatar

ABSTRACT

In this paper, we study the important issue of verifying service-level agreement (SLA) with an untrusted cloud and present an SLA verification framework that utilizes a third-party auditor (TPA). A cloud provides users with elastic computing and storage resources in a pay-as-you-go way. An SLA between the cloud and a user is a contract that specifies the computing resources and performances that the cloud should provide to the user. A cloud service provider (CSP) has incentives to cheat on the SLA, for example, providing a user with less central processing unit and memory resources than specified in the SLA, which allows the CSP to support more users and make more profits. A malicious CSP can easily disrupt the existing SLA monitoring/verification techniques by interfering with the monitoring/measurement process. A TPA resolves the trust dilemma between a CSP and its users. Under the TPA framework and the untrusted-cloud threat model, we design two effective testing algorithms that can detect an SLA violation of the virtual machine memory size. Using real experiments, we demonstrate that our algorithms can detect cloud cheating on a virtual machine's memory size (i.e., SLA violations). Furthermore, we show that our testing algorithms can defend various attacks from a malicious CSP, which tries to hide an SLA violation. Copyright © 2013 John Wiley & Sons, Ltd.

KEYWORDS

service-level agreement; verification; security; cloud computing

***Correspondence**

Xiaojiang Du, Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, U.S.A.
E-mail: dxj@ieee.org

1. INTRODUCTION

As an emerging and promising scheme, cloud computing has been used in many fields, such as mobile Internet protocol television systems [1] and digital television platforms [2]. Cloud computing offers dynamically provisioned resources as a service over the Internet to users. A cloud is client/mission oriented, formed by service-level agreements (SLAs) between a cloud service provider (CSP) and paid clients. Depending on the type of resources provided by the cloud, cloud services can be classified into Infrastructure as a Service, Platform as a Service, and Software as a Service. Amazon's Elastic Compute Cloud (EC2) [3] is a prominent example for an Infrastructure as a Service, which provides basic infrastructure components such as central processing units (CPUs), memory, and storage. The Google App Engine [4] is an example for a Platform as a Service, which enables us to deploy Python-based and Java-based Web applications.

Recently, the cloud computing paradigm is gaining increasing attention because it could bring many economic benefits to users. The main benefit is to reduce capital expenditures. This includes hardware costs and software license costs. Cloud computing also significantly reduces the operational expenditures such as costs of hiring information technology personnel. In addition, users will have universal data access and data storage at multiple independent geographical locations [5]. A number of major information technology companies (such as Amazon [3], Google [4], IBM [6], and Microsoft [7]) have started offering cloud computing services. Meanwhile, an increasing number of enterprises are migrating their computing to the cloud environment.

A cloud provides users with elastic computing and storage resources in a pay-as-you-go way. Before a service starts, a user will negotiate and sign an SLA with a CSP. The SLA is a contract between the two parties, which specifies the details of the computing/storage resources

and performances that the cloud should provide to the user. The typical SLA metrics include memory size, CPU speed/frequency, CPU type (e.g., CPU or graphics processing unit (GPU)), storage size, network bandwidth, response time, system uptime, and packet loss. For example, a small instance of the Amazon EC2 has the following configuration [8]: 1.7-GB memory, one 1.0–1.2-GHz Opteron or Xeon processor, and 160-GB instance storage.

An SLA serves as the basis for the expected level of service from the CSP. The price that a user pays to the CSP is closely related with the SLA. The more memory, the faster the CPU, and the larger the storage space, then the more the user will pay. An important question arises here. How does a user know if he/she obtains the memory size or CPU speed specified in the SLA? A CSP is a profit-based company, and hence, it has incentives to cheat on the SLA. For example, a CSP may provide less memory and/or CPU resources to a user, and this allows the CSP to support more users and make more profits. We need some way to verify the SLA.

Obviously, we cannot rely on CSPs to verify the SLA. Currently, Amazon EC2 puts the burden of verifying SLAs on users. However, it is very difficult for a user to verify the SLA because the CSP has complete control of its resources, including all the physical machines, hypervisors, virtual machines (VMs), and routers within the cloud. In summary, neither the CSP nor the user is suitable for SLA verification. Hence, a third party should be used to perform the cloud SLA verification.

Much work (e.g., [9,10]) has been carried out on SLA monitoring/verification in the traditional Internet. Some of the work uses a third party. However, SLA verification in the cloud is much more difficult than that in the traditional Internet and computer networks. An untrusted CSP can easily defeat the existing Internet SLA monitoring/verification techniques by interfering with the monitoring/measurement process.

In this paper, we present an SLA verification framework that utilizes a third-party auditor (TPA). A TPA resolves the trust dilemma between a CSP and its users. We believe that the TPA will become a standard service for cloud computing in the near future as it is critical to audit and evaluate the performance of CSPs, which will ensure the benefits of cloud users. The TPA would be similar to today's public key certification service provided by VeriSign.

There are several benefits of our TPA-based SLA verification framework. First, the TPA framework is flexible and scalable. It supports various types of tests targeting at different SLA metrics (e.g., memory or CPU). Second, it supports testing from multiple (including a large number of) users. This feature could significantly enhance the capability of testing a cloud. Third, the TPA framework also relieves users from the testing burden.

With the third-party auditing function, we can either prove to cloud users that the CSP indeed satisfies the SLA (which will in turn build trust between users and the CSP) or detect and report an SLA violation to users (which will protect users' benefit and also deter CSP from cheating in the future). To sum up, a third-party auditing function is very important, and it will promote the healthy growth of cloud computing.

In the following, we summarize our contributions in this paper:

- We propose a flexible and scalable framework that utilizes a TPA for cloud SLA verification. The framework supports various types of SLA tests.
- We design two effective testing algorithms that can detect SLA violations on the memory size of a VM.
- Using real experiments, we demonstrate that our algorithms can detect cloud cheating on VM memory size (i.e., SLA violation).
- We show that our testing algorithms can defend various attacks from a malicious CSP, which tries to hide an SLA violation.

The rest of this paper is organized as follows. Section 2 describes related work on SLAs. Section 3 discusses our assumptions and the threat model. Section 4 presents the TPA framework and each component in the framework. Section 5 discusses the two testing algorithms for detecting SLA violations on VM memory size. Section 6 concludes our paper.

2. RELATED WORK

The SLAs have been widely used by Internet and telecommunication service providers to define certain performance guarantees for users. First, we give some background information on SLAs. An SLA is an agreement that defines performance guarantees made by a service provider. An ideal SLA should contain the following [11]:

- A set of services the provider will deliver
- A complete, specific definition of each service
- The responsibilities of the provider and the user
- A set of metrics to determine whether the provider is delivering the service as promised, such as throughput, reliability, durability, elasticity, linearity, automation, and user service response times
- An auditing mechanism to monitor the service
- The remedies available to the user and provider if the terms of the SLA are not met
- How the SLA will change over time

There are two types of SLAs: off-the-shelf agreements and customized, negotiated agreements. The latter one is preferred if a user has critical or special demands, which cannot be satisfied by an off-the-shelf agreement.

The authors in [9,10] discussed SLA issues in Internet protocol networks. The authors in [12,13] proposed a layered cloud architecture to model the bottom-up propagation of failures and use it to detect SLA violations by mapping resource metrics to SLA parameters. In [14–18], the authors presented several approaches for SLA

assessment with a focus on accurately measuring or estimating quality-of-service parameters. For example, Sommers *et al.* [14] proposed a new active measurement methodology to monitor whether measured network path characteristics are in compliance with performance targets specified in SLAs. Wang *et al.* [18] presented a quantitative study of the end-to-end networking performance among Amazon EC2 from users' perspective and concluded that virtualization can cause significant throughput instability and abnormal delay variations. Serral-Gracià *et al.* [16] proposed a novel passive traffic analysis approach for online SLAs assessment, which reduces both the need for measuring quality-of-service metrics and the interactions between the ingress and egress nodes in the network. Li *et al.* [19] compared the performance and cost of several major cloud providers (Amazon, Microsoft, Google, and Rackspace). Some recent work (e.g., [29]) has studied cloud security.

However, none of the preceding work considers an untrusted/malicious cloud that can actively interfere with the measurement/monitoring from the users. For example, an untrusted cloud could adversarially modify, delay, drop, inject, or preferentially treat packets to disrupt the measurement. In [15], Goldberg *et al.* did consider the presence of adversaries. However, the threat model in [15] considers an adversary in the middle of a path, which is different from the threat model in our study, where the adversary (the cloud) is at the end of a path. In addition, [15] mainly considers networking SLAs (such as packet loss rate and delay). In this paper, we study a different SLA parameter—the memory size of a VM.

3. ASSUMPTIONS AND THREAT MODEL

In our SLA verification framework, we assume that cloud users trust the TPA. We expect that the TPA will become a standard service for cloud computing in the near future, similar to today's public key certification service provided by VeriSign and other companies. Our assumptions are given in the following:

(1) Cloud users allow the TPA to perform auditing functions using their accounts. For example, a user may delegate his account to the TPA for a short period, during which the TPA may perform the auditing. This assumption is reasonable and sometimes necessary because certain SLA aspects must be verified from the users' machine. For example, to verify if the response time satisfies the SLA, the test has to be performed from the users' machine.
(2) One thing we can do to prevent a CSP from cheating on users is to require the CSP to provide the source code of its hypervisor to the TPA. The TPA can examine the hypervisor source code and ensure that there is no "malicious" code.
(3) However, a secure static code does not guarantee a secure running instance because the CSP may run a different version of the hypervisor. Hence, the TPA should be able to verify the integrity of a hypervisor during its run time. This can be achieved by using some existing techniques, such as the HyperSentry [20]. The details are given in Section 4.
(4) The CSP allows the TPA to monitor its hypervisor, which ensures that the hypervisor does not have a "malicious" code to perform the following two tasks: (i) to detect if there is a TPA test on the cloud; and (ii) after a TPA test is detected, to change the resource allocation of a VM such that the SLA is satisfied. For example, a user leases a VM with a 2-GB memory. The CSP only sets the maximum memory value of the VM to 1 GB. When the CSP detects a test from a TPA, it changes the value back to 2 GB. Then, the TPA will not be able to detect any SLA violation (which actually happened).

Our threat model is given as follows:

1 The cloud has complete control of its own resources, including all the physical machines, hypervisors, VMs, and routers within the cloud.
2 The CSP is able to know any security material (such as an encryption key) used by a VM because the material is stored in the physical memory and/or the hard drive, which the hypervisor has access to. Hence, the standard crypto schemes do not work. In addition, a CSP is able to modify any message that is sent by a VM running in the cloud without being detected. For example, if a VM runs a test program and creates a timestamp after the program is completed, the cloud could change the timestamp without being detected, even if a message authentication code (MAC) is used because the hypervisor knows the key for the MAC and it can create a new valid MAC after changing the message.
3 A cloud will only perform the cheating if the monetary cost of doing so is less than C, where C is a parameter. Otherwise, the cloud does not have the incentive to cheat. For example, if the CSP needs to reserve a GPU such that it is not detected by a GPU/CPU test, then the cost of cheating may be too large for the CSP, and the CSP is not interested in doing so.

To support more users and hence achieve more financial gains, a CSP may intentionally assign fewer resources (than specified in the SLA) to cloud users. This will degrade the performance of users' applications, which directly hurts users' benefit. Our main objective is to develop an auditing solution that can examine whether the CSP satisfies the user SLA or not. Given an untrusted CSP, we propose a TPA-based framework and design effective testing algorithms that can verify if the user SLA is satisfied. The details are given in Sections 4 and 5.

Security Comm. Networks 2014; **7**:492–502 © 2013 John Wiley & Sons, Ltd.
DOI: 10.1002/sec

4. SERVICE-LEVEL AGREEMENT VERIFICATION FRAMEWORK

Our SLA verification framework consists of three components as shown in Figure 1: (i) a third-party auditing module (TPAM) running in the cloud machine; (ii) the SLA testing programs (testers); and (iii) the TPA server, which is located outside the cloud. In the following, we discuss each component in more detail.

Third-party auditing module. The TPAM is a software module implemented by the TPA to monitor the integrity and correctness of a running hypervisor in the cloud. Given the hypervisor's source code, first, the TPA will carefully examine the source code by static program analysis and other techniques to find any malicious/illegal operations. Then, using compilation parameters and other parameters supplied by the cloud, the TPA can generate a correct version of the hypervisor exe code. The job of the TPAM is to check the consistency between the running instance and the correct exe code. To securely test the running hypervisor, first, we need to ensure that the TPAM is trustworthy. That is, the execution of the TPAM should not be modified or interrupted by the hypervisor, nor should the TPAM's measurement be modified. The TPAM may be protected by using the HyperSentry [20] technique. HyperSentry has the following properties:

(1) HyperSentry provides a framework to enable an agent to measure the integrity of the highest privileged software (e.g., the hypervisor).
(2) HyperSentry can be invoked without alerting the hypervisor (i.e., without the hypervisor knowing).
(3) The measurement output can be securely conveyed to a remote verifier. The hypervisor is not able to alter or forge the measurement output.

The aforementioned properties make HyperSentry a good candidate to perform our task, that is, to check if a running hypervisor is the same as the correct exe code.

Because the TPAM runs in the cloud machine, it may be modified or compromised by a malicious hypervisor. We make similar assumptions as in [20] and assume that the hardware of the cloud machine is trusted. By using a trusted boot hardware of the Trusted Computing Group [21], the TPAM is examined step by step until all components of the TPAM are executed. During the trusted boot procedure, the TPAM code and data are copied into the SMRAM (a designated and lockable memory) and kept from being accessed or modified, regardless of the process' privilege level. Therefore, no software can modify the TPAM code and data, and the trust in it can be maintained.

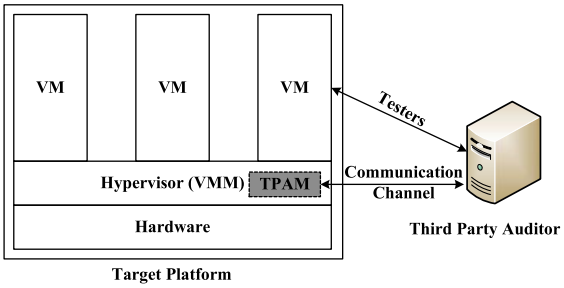

Figure 1. The service-level agreement verification framework. TPAM, third-party auditing module; VM, virtual machine; VMM, virtual machine manager.

The SLA verification may be activated by interrupts from the Intelligent Platform Management Interface [22] via an out-of-band channel, which is triggered by the TPA. The Intelligent Platform Management Interface is a server-oriented platform management interface directly implemented in hardware and firmware. Hardware features can be used to differentiate between interrupts generated by the out-of-band channel and other methods. Only an interrupt generated by the out-of-band channel may trigger the TPAM. After that, the TPAM can work securely.

The actual checking of the running hypervisor is straightforward. After being activated by the outside TPA, the TPAM computes a hash of the running hypervisor, securely signs the hash, and then sends the signed hash to the outside TPA for verification.

Service-level agreement testing programs. Traditional benchmark programs are used to test the performance of certain hardware (such as CPU and memory), and they run in a trusted environment. However, in our case, the VMs run in machines that are controlled by the CSP. In addition, the CSP controls hypervisors and other resources. A malicious CSP may change the measurement result or complete a testing task at a different (more powerful) machine, without being detected. Hence, the existing benchmarks cannot be used for our purpose. Similarly, the existing SLA measurement techniques cannot be used to verify SLA in an untrusted cloud because the cloud may modify the measurement results and hence make SLA violations undetectable. In our work, we design special testing programs (also called testers) that can defend various attacks from an untrusted cloud while still being able to detect SLA violations.

A CSP is very powerful and has complete control of its resources, including all the physical machines, hypervisors, VMs, and routers within the cloud. Hence, it is a challenging task to detect SLA violations by an untrusted cloud. To address the challenge, we propose two novel ideas. The first idea is to exploit the time difference between some fundamental operations, on which a CSP cannot cheat. For example, we may utilize the difference of the access time to physical memory and hard drive. The physical memory access time ranges from 2 to 70 ns. On the other hand, the hard drive access time is about 3 to 5 ms (for a 10 000-rpm hard drive) [23], which is in the order of 10^6 larger than the physical memory access time. This is the basic idea of our access-time-based (ATB) algorithm (in Section 5.1).

The second idea is to suddenly increase the workload on the VM, which makes the required resource close to

the SLA parameter, and then we measure the response time of the workload. For example, suppose the available physical memory of a VM should be 2 GB, and the VM runs a Web application. We can suddenly increase the number of Web requests, with a total memory requirement close to 2 GB. If less memory is allocated by the cloud, then the response time will be much longer. The measurement result is compared with a benchmark result that is obtained under a similar environment but with the specified SLA parameters (e.g., 2-GB memory). If the result measured in the cloud is similar to the benchmark result, then we consider the SLA satisfied. If the measurement result is notably worse (e.g., much longer response time) than the benchmark result, then the cloud must be cheating on some SLAs. Our second algorithm (in Section 5.2) is based on this idea.

The third-party auditor server. The TPA server plays a centric role in the auditing framework. It controls the SLA verification process. When the TPA server wants to test a cloud, it triggers the TPAM, which checks the integrity and correctness of the hypervisor. Then, the TPA server runs the SLA testing programs. After finishing the tests, it will record and analyze the time difference to give a proof on whether the SLA has been violated.

5. SERVICE-LEVEL AGREEMENT VERIFICATION ON MEMORY SIZE

In this section, we present two effective algorithms that can verify (test) the available memory size of a VM.

5.1. The access-time-based memory testing

For any SLA violation on VM resources to be detected, the key is to design effective testing programs (testers) for different resources (such as CPU and memory) according to their usage characteristics. An SLA between a cloud and a user may have several aspects, such as memory size, CPU speed/frequency, storage size, network bandwidth, response time, system uptime, and packet loss.

In our current work, we study one of the important SLA aspects—memory size. We design effective algorithms that can detect if the cloud actually allocates a VM the memory size as specified in the SLA. It is normal for a cloud to dynamically allocate physical memory to a VM because this is the way the cloud operates. For example, if a user leases a VM (say VM1) with a 2-GB memory size, it does not mean that the cloud has to allocate a 2-GB memory to VM1 all the time. The cloud may allocate only the amount of memory (may be less than 2 GB) needed by VM1. However, when the applications in VM1 require more memory, the cloud should allocate more memory to VM1 immediately, up to the maximum value (e.g., 2 GB in this example). The preceding cloud behavior is considered normal, and the SLA is satisfied.

We consider it an SLA violation if the following happens: in the SLA, VM1 is specified with a 2-GB memory. When VM1 is running, the hypervisor also tells VM1 (and the user) that its maximum memory is 2 GB. However, the hypervisor sets the actual maximum memory of VM1 to 1.5 GB. This means that VM1 will never obtain more than the 1.5-GB physical memory, no matter how many processes are running in VM1. Hence, when the workload in VM1 increases, the computations in VM1 will need more time than should be. That is, all the computations in VM1 suffer from performance degradations. Our goal is to detect such cheating by the cloud.

Note that we should consider the usable memory size in a VM, that is, it excludes the memory used by the VM and other system software. As is well known, the usable memory size is very important for application performance because less usable physical memory will cause more memory page faults, which means more swapping operations between the physical memory and the hard disk. And this significantly increases the access/computation time because the hard drive access time (3 to 5 ms) is much larger than the physical memory access time (2 to 70 ns). We design our Algorithm #1 on the basis of the aforementioned access time difference, and we refer to our Algorithm #1 as the ATB algorithm.

Denote M as the maximum memory size that is usable by user applications. From our experiments in Amazon EC2, we observe that in many cases the size of memory used by all the applications in a VM is less than M. We will not be able to detect if there is any SLA violation on memory size if the memory usage is not close to the value of M. From the preceding observation, we design the ATB algorithm such that

- first, it tries to use a memory size of (or close to) M;
- second, to defeat any cheating from the hypervisor, the algorithm must have computations based on actual access (e.g., read) to the physical memory;
- the computation result should not be predictable by the hypervisor (unpredictable); otherwise, the hypervisor could precompute the result;
- the computation result should be verifiable by the outside TPA (verifiable).

From the preceding requirements, we design an effective testing algorithm, as presented in the following:

(1) The TPA server creates an array R of size M.
(2) The TPA server sets the value of each element of R (a simple case is $R[i] = i$).
(3) The TPA server uploads the array R to VM1. This means that VM1 creates the array R and sets the same value for each element.
(4) When the TPA server wants to test the cloud, it generates a random number r and sends r to VM1.
(5) Immediately after finishing the sending, the TPA server records the time t_1.
(6) VM1 randomly selects N array elements (details given in the following) and computes the sum of

DOI: 10.1002/sec

the N elements. This means that VM1 needs to read the N elements from the physical memory. If the cloud does not allocate sufficient physical memory (e.g., M) to VM1, then some of the array elements will not be stored in the physical memory (but rather in the hard disk). This will cause a much longer access time.

(7) As soon as the computation is done, VM1 returns the result (i.e., the sum) to the TPA server. The TPA server verifies if the result is correct.
(8) The TPA server records the time t_2 when it receives the result.
(9) From the times t_1 and t_2, the TPA can figure out the computation time in VM1 (details given in the following), and then the TPA can determine if the cloud actually allocates a size of M physical memory to VM1.

In the preceding algorithm, there are two issues that need to be further explained: the selection of N random elements (Step 6) and how to figure out the computation time in VM1 (Step 9).

First, we give two trivial approaches to generate the random indexes. In Approach #1, when the TPA wants to test the cloud, it randomly generates N indexes and sends the N indexes to the TPAM in the cloud. However, the number of indexes may be large (e.g., could be 50 million). This approach may introduce a large communication overhead. Hence, we will not use Approach #1.

In Approach #2, when the TPA wants to test the cloud, it randomly generates a seed and sends it to the TPAM. The TPAM uses the seed and a prestored function to generate N random indexes of array R. However, the hypervisor is able to see the function (stored in physical memory or hard drive), and it can compute the N random indexes when it sees the random seed. Then, the hypervisor can move all the N array elements (corresponding to the N random indexes) into the physical memory before the computation starts. The aforementioned cheating makes the computation fast enough, and the TPA will not be able to find out any SLA violation. Hence, Approach #2 does not work.

Next, we present our solution to this problem. To select N elements randomly and efficiently, we use the random number r as the index of the first element and determine the next index from the value of the current element. In general, the nth index is based on the value of the $(n-1)$th element $R[n-1]$. Specifically, to achieve random memory testing, we determine the *nth* index by

$$n\text{th index} = \{(n-1)\text{th index} + \text{lowest } k\text{-bit of } R[n-1]\} \bmod (M)$$

where k satisfies $2^{k-1} \le M < 2^k - 1$ and M is the size of the array. The lowest k-bit of the $(n-1)$th element $R[n-1]$ may be considered as a "random" number between 0 and $2^k - 1$.

In the preceding method, one has to read the value of the $(n-1)$th element to generate the nth index. The testing program in VM needs to read the value anyway. However, it would be very costly for the hypervisor to generate all the indexes using the preceding method because the method requires the hypervisor to read all the N elements, which is equivalent to our test. And if some of the N elements are not in the physical memory (i.e., in hard drive), it will take the hypervisor a much longer time to complete the index generation, which makes any cheating of the hypervisor detectable.

Next, we discuss the second issue (Step 9). The test time $\Delta t = t_2 - t_1$ includes the round-trip time (RTT) between the TPA server and VM1, plus the computation time of the test in VM1. As we know, the RTT may have some variations, and this is a noise to our measurement. We use the following approach to reduce the effect of the RTT noise: if the computation time is much larger than the RTT, then the variation of RTT will have little effect on the measurement result. In [19], the authors measured the RTT of four major CSPs (Amazon, Google, Microsoft, and Rackspace) and found out that the RTT is less than 200 ms. If the computation time is in the order of several seconds, then the variation of the RTT does not affect the correctness of our decision (i.e., if there is an SLA violation) at all. We will show this by our experimental data in the following.

We run real experiments to evaluate the effectiveness of the aforementioned ATB algorithm. The experiments were conducted in a small cloud at the Harbin Institute of Technology. XEN [24] was used as the hypervisor. In the experiments, we run two test scenarios: (i) the cloud allocates the exact memory size as specified in the SLA; and (ii) the cloud allocates less memory than specified in the SLA, which means some VM memory operations will have to access the hard disk. The results are reported in Table I, where the sample rate refers to the rate of sampling the array R (for example, a sample rate of 1/10 means the algorithm randomly selects 1/10 elements from the array). The sample rate is used to control the number of elements to be accessed, which in turn controls how large the computation time will be. Δt_{full} is the test time (Δt) recorded under Scenario 1 (i.e., full-memory access); and $\Delta t_{\text{partial}}$ is the test time recorded under Scenario 2 (i.e., with 50% memory access and 50% hard drive access).

Table I. Comparison of execution time with full and partial-memory access.

Sample rate	Δt_{full} (s)	$\Delta t_{\text{partial}}$ (s)	$\Delta t_{\text{partial}}/\Delta t_{\text{full}}$
1/10	0.622	7.648	12.296
1/20	0.314	3.824	12.178
1/30	0.213	2.554	11.991
1/40	0.160	1.925	12.031
1/50	0.129	1.552	12.031
1/100	0.068	0.778	11.441
1/200	0.036	0.389	10.806
1/500	0.017	0.156	9.176
1/1000	0.011	0.079	7.182

We run each test five times to eliminate the random influences from other applications in the VM. From Table I, we can see the big difference of the execution time between the full-memory access and the partial-memory access. The partial-memory access time is about 7 to 12 times of the full-memory access time. Table I also shows that we can control the length of the computation time by varying the sample rate. When the sample rate is 1/50, the difference between Δt_{full} and $\Delta t_{\text{partial}}$ is larger than 1 s. Given that RTT is less than 200 ms, all the tests with sample rate higher than 1/50 are able to detect if full physical memory is available (i.e., if SLA is satisfied). The actual RTT in our experiments is less than 1 ms.

In general, if there is a big difference between the two times, Δt_{full} and $\Delta t_{\text{partial}}$, then our ATB algorithm is able to detect if the cloud allocates sufficient memory to the VM.

We also study the effectiveness of our algorithm when a cloud performs different levels of cheating. The level of cheating refers to the percentage of memory that is not provided to the VM. A small percentage of memory cheating may not be easily detected because it only causes a small performance degradation for the user. We run experiments for various percentages of memory cheating, and we report the results in Table II, where the sample rate is 1/100.

Table II gives the execution time for different memory cheating percentages, varying from 50% to 10%. Columns 2 and 3 list the execution time under full-memory access and partial-memory access, respectively. As one can expect, the smaller the cheating percentage, the less is the difference between the two access times. For example, when the memory cheating percentage is 50%, the partial-memory access time is about 12 times of the full-memory access time. However, when the memory cheating percentage is 10%, the difference becomes smaller, only 2.33 times or 106 ms.

By using a high sample rate, our ATB algorithm is able to detect even a small percentage of memory cheating. In Table III, we present the results when the sample rate is 1/10. As we can see, even for the 10% memory cheating, the difference between Δt_{full} and $\Delta t_{\text{partial}}$ is more than 1 s, which is large enough for us to tell a memory cheating (an SLA violation).

5.2. The resource exhaustion testing

In this subsection, we present another approach that can test the size of VM memory allocated by the cloud. Many users lease cloud to run Web services. This approach is designed for Web services running on cloud. The basic idea is to estimate the VM memory size from the number of simultaneous connections and the corresponding response times. When a Web server receives a user request, it consumes certain system resource (such as memory) to process the request. As the number of simultaneous requests increases, more system resource will be consumed. When the number of simultaneous requests reaches a threshold (denoted as r_t) at which the system resource is exhausted, new user requests will be put into a queue or even dropped. As a result, we will observe much longer response times for the new requests and/or dropped requests. Basically, we want to find out the relationship between the available memory size and the number of simultaneously supported requests. Then, we can estimate the available VM memory size by the number of simultaneously supported requests.

Table II. Comparison of execution time for different percentages of memory cheating (sample rate = 1/100).

Percentage of memory cheating	Δt_{full} (s)	$\Delta t_{\text{partial}}$ (s)	$\Delta t_{\text{partial}}/\Delta t_{\text{full}}$
50	0.068	0.778	11.441
40	0.089	0.440	4.944
30	0.053	0.324	6.113
20	0.083	0.279	3.361
10	0.080	0.186	2.325

Table III. Comparison of execution time for different percentages of memory cheating (sample rate = 1/10).

Percentage of memory cheating	Δt_{full} (s)	$\Delta t_{\text{partial}}$ (s)	$\Delta t_{\text{partial}}/\Delta t_{\text{full}}$
50	0.624	7.683	12.313
40	0.889	4.387	4.935
30	0.517	3.693	7.143
20	0.827	2.765	3.343
10	0.790	1.853	2.346

The threshold r_t is an important parameter that affects the accuracy of the test. r_t may be obtained as follows: In our own computer, we set up a VM with the same configuration as the one that will run in the cloud. Then, we vary the number of simultaneous requests and record the number of successful connections and the response times. We can also use a system tool to monitor the usage of the system resource (e.g., memory). From the preceding data, we will be able to determine the value of r_t.

The main advantage of the resource exhaustion test is that the test is stealthy to the cloud. That is, the cloud does not know that it is being tested because the number of user requests varies all the time. It is very natural to have many user requests at the same time. This feature makes the test very effective.

We design real experiments to evaluate the performance of the resource exhaustion test. We set up an Apache Web server (version 2.2.3) [25] to run Web applications and response to user requests. To handle many simultaneous requests, we set the "ServerLimit" of the Apache server to a large number—1000—which means that the Apache parallel connection limit is 1000. This value is large enough such that Apache will not limit the number of connections before the memory does. To test memory size, we use a dynamic Web page (PHP page) instead of a static page (html only) because the latter does not consume much VM memory. In our experiments, the PHP page declares an array and consumes about 8-MB memory when it is successfully processed.

Security Comm. Networks 2014; **7**:492–502 © 2013 John Wiley & Sons, Ltd.
DOI: 10.1002/sec

Table IV. Comparison of connection time with 1-GB memory.

Connections			Connection time (ms)			Memory exhausted
Number	Rate	Success (%)	Minimum	Maximum	Average	
100	100	100	152.1	3 919.7	2 405.8	No
110	110	100	86.5	4 233.0	2 622.9	No
120	120	100	110.2	4 677.3	2 910.9	No
130	130	100	246.0	6 165.2	3 454.0	No
140	140	100	88.0	34 349.2	11 268.0	Yes
150	150	100	62.9	67 378.3	19 343.1	Yes
160	160	100	154.3	59 729.8	16 464.1	Yes

We use httperf [26] as the client-side tool, which is a scalable client workload generator to send concurrent hypertext transfer protocol requests to the Web server. Because each PHP page consumes about 8-MB memory, httperf does not need to generate a large number of requests at the same time. For example, a test with 200 parallel requests will consume about 1.6-GB memory.

To study the impact of different memory sizes on Web service performance, we run tests under two scenarios, that is, a VM with 1-GB memory and a VM with 512-MB memory. Table IV gives experimental results using a VM with 1-GB memory. In Table IV, from left (column) to right, the results include the number of connections requested, the request rate (per second), the success rate, the connection time (i.e., the Transmission Control Protocol connection lifetime), including the minimum, maximum, and average values, and whether the memory is exhausted (yes or no). The success rate means the percentage of requests that have been successfully processed. The requests are sent at a rate of 100 to 160 requests per second.

In this set of experiments, we allow the test cycle (i.e., the response time) to be as long as it could be, that is, we do not set a limit for the response time. Because of this, all the user requests were successfully processed.

Table IV shows that all the requests are processed (all the success rates are 100%), which indicates that no connection is discarded because of Apache or httperf connection limits. The connection time is basically the response time (the http lifetime). Hence, in the rest of the paper, we use the term connection time instead of response time. When the memory is not exhausted (i.e., when the number of connections are no more than 130), the average connection time is less than 3.5 s. When the memory is exhausted (i.e., the number of connections is larger than or equal to 140), the average connection time dramatically increases (larger than 11 s), and the maximum connection times are larger than 34 s. We also plot the average and maximum connection times in Figure 2, where the *x*-axis is the number of connections.

The reason for this large increase is given in the following. When more requests come in, more memory is needed by the Apache server to handle the requests. Before the VM memory is exhausted, the requests can be successfully handled in a short time because there is still physical memory available to the VM. However, when the memory is exhausted, the Apache server is not able to process any more request, and it has to put the new requests in a queue. Only after some existing requests have been processed and the corresponding memory have been released, the Apache server can start processing the new requests. This causes a significant increase of average (and maximum) connection time.

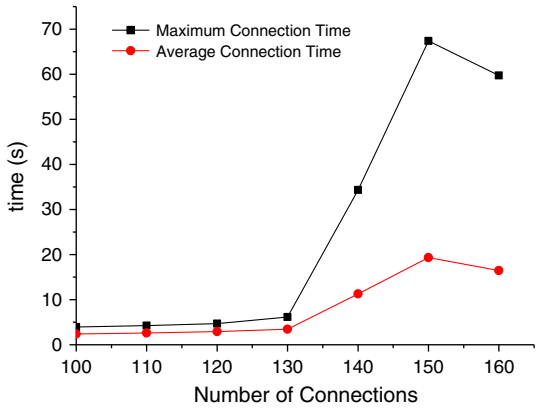

Figure 2. Comparison of connection time for a virtual machine with 1-GB memory.

Note that the minimum connection time is not a good base for determining the available VM memory size because the client requests are handled one by one at the Apache server side, even if the requests come in at the same time. Hence, the first client request can always be processed quickly, no matter how many parallel requests are sent.

In a different set of experiments, we set the maximum test cycle (i.e., the response time) to a given value, and we record the success rate. The results are reported in Table V. As we can see, when there are 140 user connections requested

Table V. Success rate for different maximum test cycles with 1-GB memory.

Number of connections	Maximum test cycle (s)			
	10 (%)	20 (%)	30 (%)	40 (%)
130	100.00	100.00	100.00	100.00
140	62.14	74.29	85.00	100.00

Table VI. Comparison of connection time with 512-MB memory.

Connections			Time (ms)			
Number	Rate	Success (%)	Minimum	Maximum	Average	Memory exhausted
30	30	100	45.4	731.3	258.2	No
40	40	100	39.7	1 425.3	510.0	No
50	50	100	45.1	2 154.3	758.0	No
60	60	100	45.3	2 732.1	1 093.5	No
70	70	100	46.2	53 139.3	20 658.7	Yes
80	80	100	45.4	82 959.3	39 964.8	Yes
90	90	100	58.6	90 308.4	28 290.9	Yes
100	100	100	55.9	142 034.4	54 297.7	Yes
110	110	100	72.2	117 761.1	34 881.8	Yes
120	120	100	63.4	303 454	163 314.7	Yes
130	130	100	46.9	212 886.9	104 220.7	Yes
140	140	100	68.3	257 316.2	113 667	Yes
150	150	100	87.9	440 457.6	225 931.6	Yes
160	160	100	141.9	425 082.6	211 940.9	Yes

Table VII. Success rate for different maximum test cycle with 512-MB memory.

Number ofconnections	Maximum test cycle (s)			
	10 (%)	20 (%)	30 (%)	40 (%)
60	100.00	100.00	100.00	100.00
70	52.86	57.14	62.86	68.57

(VM memory is exhausted), the success rate varies from 62% to 100%, depending on the maximum test cycle. The shorter the test cycle, then the smaller is the connection success rate. This means if the application has a limit on the response time, then some user requests will be dropped when the memory is not sufficiently large. Hence, the connection success rate may also be used to detect SLA violations on memory size.

We want to see if the preceding results apply to different memory sizes. Hence, in another experiment, we measure the connection time in a VM with a 512-MB memory. The results are reported in Table VI. When the number of parallel connections is larger than 70, the VM memory is exhausted. We observe similar results as in Table IV. That is, when the VM memory is exhausted, the average (and maximum) connection time dramatically increases. In a different set of experiments, we set the maximum test cycle to a given value and compare the success rate. The results are reported in Table VII, which are similar to those in Table V. That is, the success rate decreases when the maximum test cycle becomes smaller.

Figure 3 plots the cumulative distribution function (CDF) of the connection time of a VM with 1-GB memory, for 130 and 140 connections. As we can see, the CDFs under the two cases are quite different. Figure 3 shows the following: (i) when the number of connections is 130, 100% of the connections are successfully processed within

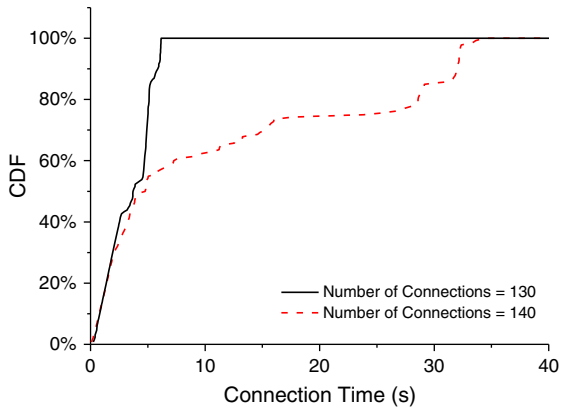

Figure 3. The distribution of connection time with 1-GB memory. CDF, cumulative distribution function.

7 s; and (ii) when the number of connections is 140, only 58.6% of the connections are processed within 7 s.

Figure 4 plots the CDF of the connection time of a VM with 512-M memory, for 60 and 70 connections. Figure 4 shows similar results as in Figure 3: (i) when the number of connections is 60, 100% of the connections are successfully processed within 4 s; and (ii) on the other hand, when the number of connections is 70, only 38.5% of the connections are processed within 4 s.

The preceding experimental results and discussions show that the resource exhaustion test is very effective in detecting an SLA violation on VM memory size.

5.3. Security analysis

For the ATB memory testing algorithm (in Section 5.1), one cheating that a CSP could do is to migrate the VM to another machine that has sufficient physical memory when

Security Comm. Networks 2014; **7**:492–502 © 2013 John Wiley & Sons, Ltd.
DOI: 10.1002/sec

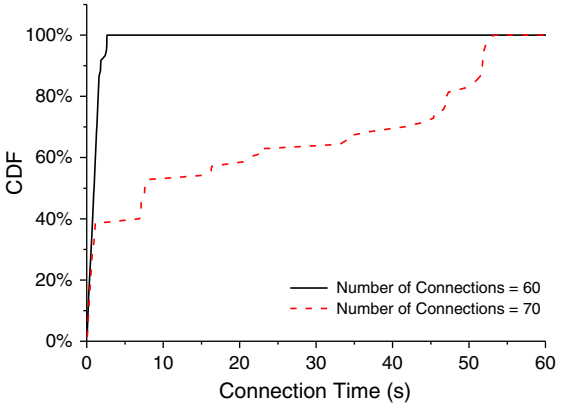

Figure 4. The distribution of connection time with 512-MB memory. CDF, cumulative distribution function.

the CSP detects such a test. However, according to [27] and [28], even at a fast bandwidth, a VM migration takes about 5–10 s, with a downtime of about 0.3–1.5 s. Recall that our ATB algorithm takes less than 0.9 s to complete (for full-memory access). Hence, a VM migration will be detected because of the large delay. Note that a cloud is allowed to do normal VM migrations when some resource at a machine is running out. However, if a cloud always migrates a VM when the VM is being tested by the TPA, then it is a strong indication of cloud cheating.

A malicious cloud may launch other attacks on the ATB algorithm. For example, if the cloud does not allocate size M of physical memory to VM1, during a test, it may use only array elements in the physical memory for the computation. However, the computation result will not be correct because the outside TPA server knows which array elements should be used for computation and it knows the correct result. Hence, this attack can be easily defeated.

It is harder for a cloud to launch attacks on the resource exhaustion test (in Section 5.2) because the test is similar to normal usages of the Web application.

To sum up, our memory testing algorithms can defeat various cheating/attacks from an untrusted cloud.

6. CONCLUSIONS

In this paper, we proposed a flexible and scalable framework that utilizes a TPA for cloud SLA verification. Under the framework, we designed two effective testing algorithms that can detect an SLA violation on VM memory size. The ATB memory testing algorithm utilizes the difference of the access time to physical memory and hard drive. The resource exhaustion test is to suddenly increase the number of user requests to the VM, which makes the required memory close to the SLA parameter. Then, we can determine if there is an SLA violation from the response times of the user requests. Using real experiments, we demonstrated that the two algorithms can effectively detect cloud SLA violations on VM memory size. Also, we showed that our algorithms can defend various attacks from a malicious cloud.

ACKNOWLEDGEMENT

This research was supported in part by the China National Basic Research Program (973 program) under grants 2011CB302605 and 2007CB311101, the China National High Technology Research and Development Program (863 program) under grants 2010AA012504 and 2011AA010705, and the US National Science Foundation under grants CNS-0963578, CNS-1002974, CNS-1022552, and CNS-1065444, as well as the US Army Research Office under grant W911NF-08-1-0334.

REFERENCES

1. Lai Y, Lai C, Hu C, Chao H, Huang Y. A personalized mobile IPTV system with seamless video reconstruction algorithm in cloud networks. *International Journal of Communication Systems* 2011; **24**(10):1375–1387.
2. Lai C, Chang J, Hu C, Huang Y, Chao H. CPRS: a cloud-based program recommendation system for digital TV platform. *Future Generation Computer Systems* 2011; **27**(6):823–835.
3. Amazon EC2, http://aws.amazon.com/ec2/
4. Google App Engine, www.google.com/enterprise/appengine/
5. Opencrowd cloud taxonomy, www.opencrowd.com/views/cloud.php
6. IBM Cloud, www.ibm.com/cloud-computing/us/en/
7. Microsoft Azure, www.microsoft.com/windowsazure/
8. Amazon EC2 Instance Types, aws.amazon.com/ec2/instance-types/
9. Shaikh A, Greenberg A. Operations and management of IP networks: what researchers should know. *ACM SIGCOMM Tutorial Session*, 2005
10. Martin J, Nilsson A. On service level agreements for IP networks. *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, 2002; 855–863.
11. Cloud computing use cases whitepaper, version 4.0, July 2, 2010, http://opencloudmanifesto.org/resources.htm
12. Haeberlen A. A case for the accountable cloud. *Proceedings of the 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware*, 2009; 52–57.
13. Brandic I, Emeakaroha VC, Maurer M *et al*. LAYSI: a layered approach for SLA-violation propagation in self-manageable cloud infrastructures. *Proceedings of 34th Annual IEEE Computer Software and Applications Conference Workshops*, 2010; 366–370.

14. Sommers J, Barford P, Duffield N, Ron A. Multi-objective monitoring for SLA compliance. *IEEE-ACM Transactions on Networking* 2010; **18**(2):652–665.
15. Goldburg S, Xiao D, Tromer E, Barak B, Rexford J. Path-quality monitoring in the presence of adversaries. *Proceedings of the 2008 ACM SIGMETRICS on Measurement and Modeling of Computer Systems*, 2008; 193–204.
16. Serral-Gracià R, Yannuzzi M, Labit Y, Owezarski P, Masip-Bruin X. An efficient and lightweight method for Service Level Agreement assessment. *Computer Networks* 2010; **54**(17):3144–3158.
17. Gill P, Ganjali Y, Wong B, Lie D. Dude, where's that IP? Circumventing measurement-based IP geolocation. *Proceedings of the 19th USENIX Conference on Security*, 2010; 241–256.
18. Wang GH, Eugene Ng TS. The impact of virtualization on network performance of Amazon EC2 data center. *Proceedings of the 29th IEEE Conference on Computer Communications*, 2010; 1163–1171.
19. Li A, Yang X, Kandula S, Zang M. CloudCmp: comparing public cloud providers. *Proceedings of the 10th Internet Measurement Conference*, 2010; 1–14.
20. Azab AM, Ning P, Wang Z, Jiang X, Zhang X, Skalsky NC. HyperSentry: enabling stealthy in-context measurement of hypervisor integrity. *Proceedings of the 17th ACM Conference on Computer and Communications Security*, 2010; 38–49.
21. Trusted Computing Group, www.trustedcomputinggroup.org/
22. IPMI—Intelligent Platform Management interface specification v2.0. http://download.intel.com/design/servers/ipmi/IPMIv2_0rev1_0.pdf
23. http://en.wikipedia.org/wiki/Hard_disk_drive#Access_time
24. Xen Hypervisor, www.xen.org
25. Apache, httpd.apache.org/
26. httperf, www.hpl.hp.com/research/linux/httperf/
27. Akoush S, Sohan R, Rice A, Moore AW, Hopper A. Predicting the performance of virtual machine migration. *Proceedings of the 18th Annual Meeting of the IEEE International Symposium on Modeling Analysis and Simulation of Computer and Telecommunication Systems*, 2010; 37–46.
28. Zhao M, Figueiredo RJ. Experimental study of virtual machine migration in support of reservation of cluster resources. *Proceedings of the 2nd International Workshop on Virtualization Technologies in Distributed Computing*, 2007; 51–58.
29. Feng Z, Bai B, Zhao B, Su J. Redball: throttling shrew attack in cloud data center networks. *Journal of Internet Technology* 2012; **13**(4):667–680.