PAPER Special Section on Information and Communication System Security

Rethinking Business Model in Cloud Computing: Concept and Example

Ping DU[†a)] *and* **Akihiro NAKAO**[††], ***Members***

SUMMARY In cloud computing, a cloud user pays proportionally to the amount of the consumed resources (bandwidth, memory, and CPU cycles etc.). We posit that such a cloud computing system is vulnerable to DDoS (Distributed Denial-of-Service) attacks against quota. Attackers can force a cloud user to pay more and more money by exhausting its quota without crippling its execution system or congesting links. In this paper, we address this issue and claim that cloud should enable users to pay only for their admitted traffic. We design and prototype such a charging model in a CoreLab testbed infrastructure and show an example application.

key words: *cloud computing, denial-of-quota attacks, pay-as-you-admit*

1. Introduction

Cloud Computing [1]–[3] has the potential to bring major changes in the networking and IT ecosystem. In conventional IT industry, application service providers are required to define the hardware infrastructure and setup application execution platform. Cloud computing provides on-demand scalability of computing resources, thereby application service providers can focus on innovative service design without defining hardware infrastructure and setting up application execution platform. A cloud user can pay proportionally to the amount of the consumed computing resources on a short-term basis as needed.

In making the decision about whether hosting a service in a cloud, we must examine the availability of our application service. Besides the accidental outrages of the outrage, we should check whether our service can survive under attacks. Distributed Denial of Service (DDoS) [5] attacks aimed at hosted services have some new features. As opposed to the traditional servers that can use additional firewalls to filter unwanted traffic, a hosted service on a cloud cannot refuse unwanted packets before their arrival. Those unwanted packets will consume the hosted service's resources (at least incoming bandwidth). Often, a hosted service pays proportionally to the amount of the time and resources (bandwidth, memory, and CPU cycles etc.) it uses. Attackers can launch DDoS attacks to a hosted service by exhausting its quota without crippling its execution system or congesting links. Those attacks could be UDP and ICMP flooding to consume the incoming bandwidth or TCP SYN flooding and HTTP GET flooding to consume memory and CPU cycles. If attackers can force a hosted service to pay more and more money, that might be a better attack than disrupting his availability.

In this paper, we posit that the current cloud computing system charging model is vulnerable to DDoS attacks against quota and improper charge for unwanted traffic. To address these problems, we propose a new charging model, called *pay-as-you-admit*, to enable a hosted service to pay only for its admitted usage [6], where a hosted service uses sessions for admission control. Our insight into this proposal is based on the observation that a hosted service may not be able to refuse unwanted packets but has ability to reject unwanted sessions before session establishment. Here, the term "session" refers to a duration of continuous usage of a service from an end-client [7].

Furthermore, if a cloud is immune to Denial-of-Quota attacks, we can rent its service and use the cloud infrastructure as a shield between the clients and traditional servers so that we can shift the DDoS problem from the end-hosts to the more powerful cloud infrastructure. As an example, we design and implement such a cloud-based attack defense system, which is running on cloud infrastructures as a hosted service to protect Web servers.

The main contribution of this paper is that we implement a prototype pay-as-you-admit system and show that cannot only liberate hosted services from DDoS against quota and improper charge of unwanted traffic but also motivate the cloud provider to filter unwanted traffic.

The rest of the paper is organized as follows. Sections 2 points out the security challenges of cloud computing. Section 3 models the cloud computing system and propose pay-as-you-admit. Section 4 prototypes such a charging system. Section 5 analyzes this charging model. Section 6 gives an example application. Finally, Section 7 concludes this paper.

2. About Cloud Computing

2.1 What Cloud Computing Changes

Before Cloud Computing, application-level Software-as-a-Service (Saas) (Hotmail, Gmail, Google Docs etc.) has made a great success. They provide users with online personal storage and application service. The developing technologies of virtualization and the growth of the Internet had low-level of computing resources as a utility fulfilled. These

---

Manuscript received February 1, 2011.

Manuscript revised May 23, 2011.

[†] The author is with the National Institute of Information and Communications Technology, Tokyo, 113–0001 Japan.

[††] The author is with The University of Tokyo, Tokyo, 113–0033 Japan.

a) E-mail: duping@nict.go.jp

DOI: 10.1587/transinf.E94.D.2119

low-level packages of computing resources can be a whole virtual machine instance or a programmable execution environment. They are known as Infrastructure-as-a-Service (Iaas) and Platform-as-a-Service (Paas).

Amazon.com has been at the forefront of the cloud computing trends and its Amazon Elastic Compute Cloud (EC2) [1] presents a completely controlled virtual computing environment. Google App Engine (GAE) [2] lets clients run their own Web applications on Google's cloud infrastructure. Different from EC2, GAE applications run in a secure sandbox environment that only allows Web traffic to enter and leave each application. Microsoft's Azure [3] is based on its .NET libraries and providing a language-independent execution environment, which is an intermediate solution between application frameworks (GAE) and hardware virtual machines (EC2). IBM is also reported to provide their cloud services — SmartCloud [4].

2.2 What Benefits from Cloud Computing

(1) Elasticity

With Cloud Computing, cloud users can assume that there are infinite computing resources available on demand without knowing the details behind the provision of the computing resources. Cloud Computing enables the users to pay only for their usage on a short-term basis without paying far ahead for provisioning the equipments of large-sale networks. The risks of underutilization and saturation have been shifted to cloud computing. Developers with innovative ideas for new Internet services no longer require the large capital outlays in hardware and the human expense to operate it.

(2) Cost-Efficiency

All these cloud computing systems provide *elastic*—the more resources a hosted service uses, the more it gets charged—platforms to run hosted services. Such an elastic charging model, named pay-as-you-go, is appreciated for enabling a hosted service to pay for the use of computing resources on a short-term basis as needed.

2.3 What are Security Challenges

A cloud infrastructure (such as Google), which consists of a large number of geographically-distributed machines, has an aggregate capacity exceeding the firepower of most of botnets. From this aspect, a cloud infrastructure is much more robust than a traditional server. With the exception of rare outages[†], you can pretty much trust that whatever computing resources you need will be there when you need it.

Due to the open feature of the current IP infrastructure, where anyone can send anything to anyone at anytime without permission in advance, a hosted service on a cloud cannot refuse unwanted packets before their arrival. Those unwanted packets will consume the hosted service's resources (at least incoming bandwidth). Actually, the security of a hosted service is not about availability, but about money. We posit that a cloud vendor with pay-as-you-go charging model has the following two vulnerabilities.

First, a hosted service is vulnerable to DDoS attacks against quota that limits the usage of network resources (bandwidth, memory, and CPU cycles etc.). To provide protection against sudden large charge caused by unwanted access to the service, a cloud provider usually allows a hosted service to set quota to the maximum consumption of resources for a unit of time. As a result, attackers can launch DDoS attacks by exhausting its quota without crippling its execution system or congesting links. After its quota has been exhausted, the hosted service is frozen until the refresh of quota, which is usually as long as one day.

Second, even a hosted service can survive DDoS attacks; it is required to pay for the attack traffic. As opposed to the traditional servers that can use additional firewalls to filter unwanted traffic, the hosted service can only rely on the cloud provider to do this. Because the attack traffic can be paid by the hosted service, the cloud provider may have few economic incentives to filter it.

We are aware that most of commercial cloud vendors have already been equipped with anti-DDoS products. However, at the time of writing, many commercial cloud services are still vulnerable to denial-of-quota attacks. Due to legal concern, we couldn't introduce our experiments on them in the paper. Some cloud vendors have noticed this problem. For example, Google encourages customers to set a big quota and promises to provide a mechanism for the reimbursement of the proven attack traffic. However, a hosted service may risk of charging of undetected attack traffic such as that from botnets. Also, it is laborious and just hard to prove it is really an attack. Charging model for hosting services is not often discussed but poses significant operational problems regarding security and customer satisfaction, which sometimes leads to a court case [9].

3. Pay-as-You-Admit

To address these security challenges, we propose a new charging model for cloud computing, named *pay-as-you-admit*—the more usage a hosted service admits, the more it gets charged.

3.1 Modeling Cloud Computing

Before we introduce our proposal, we first analyze the interactions in a cloud computing system (IaaS or PaaS), which consists of a cloud provider, a hosted service (also cloud user), and end-clients.

[†] Who is responsible for loss of revenue/profits from a significant cloud computing outage is another open issue. For example, Web-based code hosting service Bitbucket experienced more than 19 hours of downtime over the weekend after an apparent DDoS attack on the sky-high compute infrastructure it rents from Amazon.com on 4 Oct 2009.

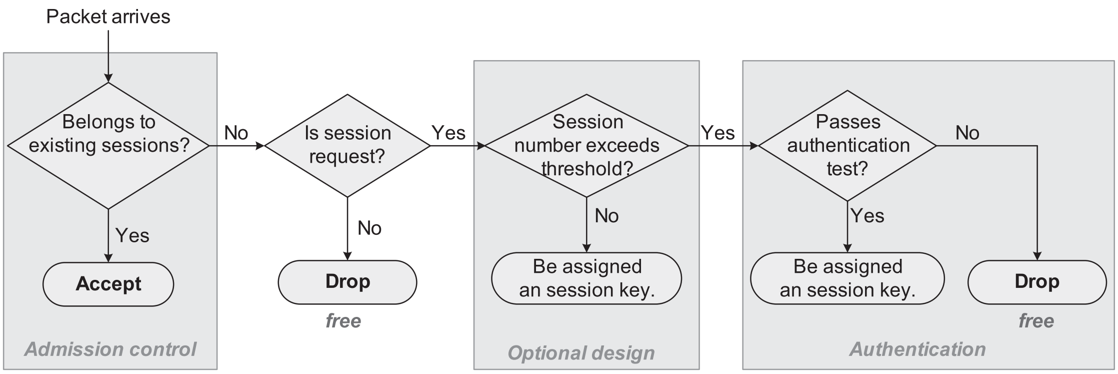

Packet arrives
Belongs to existing sessions?
No
Yes
Accept
Admission control
Is session request?
Yes
No
Drop
free
Session number exceeds threshold?
Yes
No
Be assigned an session key.
Optional design
Passes authentication test?
No
Yes
Be assigned an session key.
Drop
free
Authentication

Fig. 2 Design of pay-as-you-admit charging system.

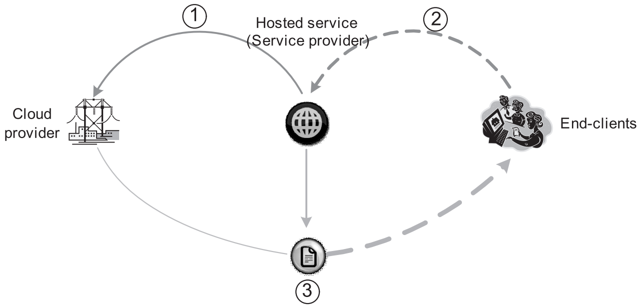


Fig. 1 Interactions in cloud computing system, where (1) the hosted service pays the cloud provider for the network resources consumed by the end-clients; (2) the hosted service gains income by delivering contents to the end clients; and (3) the content is delivered from the hosted service to the end-clients through the network resources.

Figure 1 illustrates the relationship among the entities involved in the cloud computing. Unlike the traditional two-party market in telecommunications and networking, the cloud computing is featured with three-party interactions. First, the end-clients are not aware of the charging model since they are not charged directly, but only indirectly through content from the hosted service. Second, the cloud provider recovers its cost of consumed network resources and makes profit by charging each hosted service for the usage of resources. Third, the hosted service recovers cost from the content provision. Hosted service takes both its expenditure and revenue into account.

The strategy of charging hosted services is an impacting factor in cloud computing, since the hosted services adjusts its behavior according to the charging strategy and the adjustment affects both the cloud provider and the group of end-clients. Since charging model for cloud services is a novel area that hasn't been well explored, it is instructive to gain insights into developing one based on the existing efforts on the charging in traditional telecommunications and networking. Among these efforts, Expected Capacity Charging [10] model is popularly applied in mobile markets, where each user pays for the expected capacity profile based on her general usage. Paris-Metro Charging [11], [12] allows ISPs to auction their service when the Internet is in congestion. Edge Pricing [13] proposes to charge Internet at the access point. In [14], K. Hosanagar et al. suggests that providing volume discounts to content providers may motivate the usage of Content Delivery Networks (CDN). A more detailed introduction about Internet pricing could be found in survey paper [15].

If we directly apply these pricing and charging models to cloud computing, they are not sufficient because they do not address the problem of DDoS attacks against quota. They all follow the strategy of *pay-as-you-go*, which improperly charges unwanted traffic towards hosted services and discourages the cloud provider from filtering unwanted traffic.

It is difficult for cloud providers to completely distinguish legitimate traffic from malicious traffic. Also, cloud providers and cloud users (hosted services) could have different understanding about the legitimacy of the received traffic. This observation leads to our motivation for the idea of *pay-as-you-admit*. In our proposed pay-as-you-admit charging model, cloud providers don't need to categorize traffic into legitimate and malicious. Instead, it is hosted service that categorizes traffic into admitted and unwanted traffic.

Considering that a hosted service cannot rule out unwanted packets before their arrival but has an ability to reject session requests before session establishment, we use session as the unit of admitted usage. Each hosted service is provided with authentication and admission control schemes to distinguish and reject unwanted session requests. After the hosted service admits a session, the pay-as-you-admit charging scheme may come into play. In this paper, the term "session" refers to a duration of continuous usage of a service from an end-client.

3.2 Design

In order to enable the *pay-as-you-admit* charging model, components for session-based pricing, admission control and authentication need to be added into the cloud computing environment. A design of the system is demonstrated in Fig. 2.

Session-based pricing: *Pay-as-you-go* charges all the traffics involving a hosted service, including the traffics from attackers. For example, although GAE applications can drop unwanted HTTP requests, these requests are still accounted as incoming traffic volume that is included in the hosted services' charge. The severer the attacks are, the more the hosted service pays to the cloud provider. *Pay-as-you-admit* addresses this unfairness and argues that the unwanted traffic should be filtered by the provider rather than being paid by hosted services. For the hosted services, they need to pay only for their admitted usage. The admission of traffic is controlled by the hosted service in the granularity of session. The hosted service pays only for admitted sessions while the other dropped packets (shown in Fig. 2) are not included in the charge.

After a session is admitted, the *pay-as-you-admit* mechanism can charge it by the metered usage of network resources or at flat rate.

Admission control: The admission control is to prevent unadmitted clients from accessing the hosted service. It is realized through issuing active session keys, which are maintained in a *session table*. In real implementation, the session table could be combined with the existing forwarding tables or ruling tables at the gateway so that it will not increase the operational cost obviously. A valid session key allows a client to access the hosted service for a specific period. For example, a client can access a hosted Web service with a session key storing it in her cookie. The session key can be generated with a private Hash function of the client IP address and the expiration time.

To prevent the quota from being exhausted due to overly aggressive admitted clients, the amount of resources allocated to each client should also be limited. For a public Web service, we can rate limit the http requests from each client.

Authentication: The authentication is to distinguish admitted clients from unadmitted ones. The hosted service could choose its own authentication approach (e.g., strong cryptographic verification [16] or graphical Turing tests [17]) based on its provided service or even waive authentication test (See following *Optional design*). A client who has failed authentication tests after a few trials will be blocked and their requests will be dropped. We can also add overly aggressive clients to a blacklist. On the other hand, a client who has passed authentication tests will be assigned an authenticated session key.

Optional design: Requiring all clients to pass authentication tests may discourage normal clients from accessing and introduces extra latency to the service. For example, you cannot ask every casual visitor to a newspaper Website to pass an authentication test. To address this issue, an *optional* design is to waive the authentication when the hosted service is lightly loaded. The maximum number of allowed unauthenticated clients can adapt the workload of the hosted service. When the number of the concurrent sessions is less than a threshold, a new request can access the hosted service without doing authentication tests. Otherwise, a new client must pass authentication tests before accessing the hosted service.

Our proposed pay-as-you-admit is not to substitute the existing anti-DDoS products. Instead, it should be combined with them. The pay-as-you-admit provides the hosted service with control schemes to distinguish admitted traffic from unwanted traffic. How to block the unwanted traffic relies on the traditional anti-DDoS products on the cloud provider.

4. Prototype

In this section, we show a prototype implementation of the components of pay-as-you-admit in CoreLab [19], [20], [22]. As shown in Fig. 3, CoreLab is a testbed that hosts multiple virtual machines (VM) on a physical infrastructure, the VMs are connected through virtual openflow switch (vOFS) [21], [23]. Through vOFS, the CoreLab supports generic flow-space isolation. We design a virtual service gateway (vSGW) for each VM, each flow to/from a VM is filtered by the vSGW. The VM can add/remove forwarding rule through a NOX [24]. Since each flow in the vSGW is a duration of continuous usage of a service from an end-client, which can be regarded as a session. In the following, we just denote the "flow" of vSGW as "session".

Through NOX, we can measure the usage of each session through openflow switch. Figure 4 shows such a session counting process. In Step 1, the vSGW forwards the re-

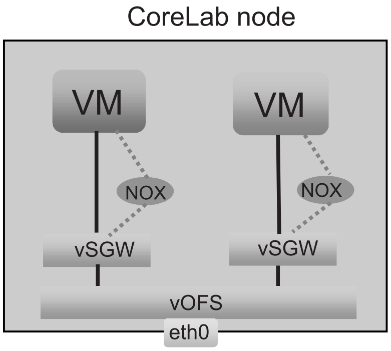


Fig. 3 Prototype of pay-as-you-admit charging model.

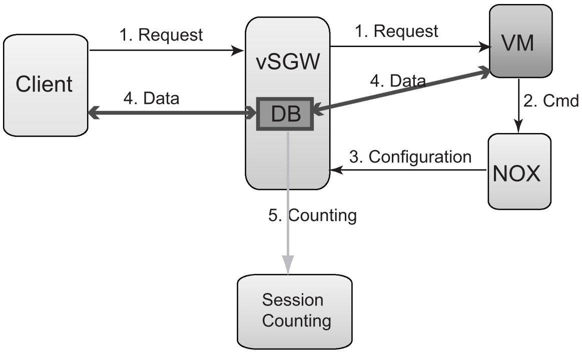


Fig. 4 Session counting process.

```
1   <Rule>
2     <VM> cl_1 </VM>
3     <Hard_timeout> 1800 </Hard_timeout>
4     <Idle_timeout> 60 </Idle_timeout>
5     <Cmd> Accept </Cmd>
6     <Fields>
7       <ServicePort> 80 </ServicePort>
8       <ClientIP> 10.0.0.1 </ClientIP>
9       <ClientPort> 5432 </ClientPort>
10    </Fields>
11  </Rule>
```

(a) Setting up a rule of admitting a session

```
1   <Rule>
2     <VM> cl_1 </VM>
3     <Hard_timeout> 60 </Hard_timeout>
4     <Idle_timeout>  0  </Idle_timeout>
5     <Cmd> Reject </Cmd>
6     <Fields>
7       <ServicePort> 80 </ServicePort>
8       <ClientIP> 10.0.0.2 </ClientIP>
9     </Fields>
10  </Rule>
```

(b) Setting up a rule of rejecting a session

Fig. 5 Examples of command from VM to NOX.

quest packets to the VM; other packets are denied by vSGW by default. VM does authentication. If the authentication successes, VM sends a command to the NOX to setting up a flow rule. After the forwarding rule has been set up, the client can communicate with VM and the usage of each session is recorded in built-in database (DB) of vSGW.

Figure 5 shows an example of forwarding rule, which denotes the forwarding rule that enabling a session for virtual machine cl_1. In lines 2-3, if both idle_timeout and hard_timeout are set, the session will timeout after idle_timeout seconds with no traffic, or hard_timeout seconds, whichever comes first. Lines 6-10 indicate flow space. Line 5 indicates the command for the flow space. When the client passes the authentication, the command field is filled with Accept; otherwise it is filled with Reject.

5. Analyses

In this section, we analyze how the cloud provider, the hosted service and end-clients behave under different charging models.

5.1 Assumptions

Our analysis focuses on a monopoly cloud provider without price war with other counterparts and its resource is enough to support infinite demand on resource. For simplicity, we use the traffic volume to represent the amount of usage-based network resource consumption. We have the following assumptions.

A1. For the cloud provider, we assume the average cost for transferring the traffic is proportional to the traffic volume.

A2. For a hosted service, we assume the maximum quota for traffic volume in a unit of time T_0 is w. let y_i denote the consumed (incoming and outgoing) traffic volume of client i with an identical average rate γ, which represents average usage of resource paid by the hosted service for each client's access; therefore it represents the quality that a client obtained from the hosted service.

A3. We assume a client's arrival process $\{x(t)\}$ is with a rate λ. Since the client's behavior is affected by the service quality, we assume λ is a function of γ.

A4. We assume unwanted packets arrive with average rate λ_a and average length l_a; the cumulative unwanted traffic volume is $A(t)$ in [0, t].

5.2 Charging in Cloud Provider

In pay-as-you-go, each hosted service pays for the total consumed traffic volume of their clients, we define $z_t(t)$ as the expenditure in time [0, t]; thus $z_t(t) = \psi\sum_{i=1}^{x(t)} y_i + \psi A(t)$, where ψ is the price per unit of traffic volume [†].

If $z_t(T)=w$ $(T < T_0)$, the hosted service's quota will be exhausted at time T and its service will be suspended until the next T_0 period. Otherwise, the hosted service can provide continuous service but may be charged for unwanted traffic. Let E_{zt} denote the ensemble average of expenditure of the hosted service for the time unit T_0; it can be calculated as

$$\begin{aligned} E_{zt} &= E[z_t(T_0)] = \psi E[\sum_{i=1}^{x(T_0)} y_i + A(T_0)] \\ &= \psi E[x(T_0)]E[y_i] + \psi E[A(T_0)] \\ &= \psi\lambda\gamma T_0 + \psi\lambda_a l_a T_0. \end{aligned} \tag{1}$$

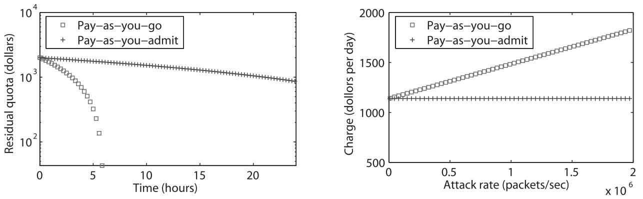
Similarly to Eq. (1), we can calculate the average cost of the cloud provider for transferring the traffic volume for the hosted service for the time unit T_0 by $E_{ce}=\chi(\lambda\gamma T_0+\lambda_a l_a T_0)$, where χ is the cost per unit of traffic volume for the cloud provider (Assumption *A1*). The average net profit E_{ct} of the cloud provider for the time unit T_0 is denoted as

$$E_{ct} = E_{zt} - E_{ce} = (\psi - \chi)(\lambda\gamma T_0 + \lambda_a l_a T_0). \tag{2}$$

Eq. (2) shows that the cloud provider can make profit through unwanted traffic under pay-as-you-go charging model so that the cloud provider has no economic incentive to filter unwanted traffic. However, under current pay-as-you-go charging model, the hosted services are vulnerable to denial-of-quota attacks. And eventually, the cloud providers will loss the customers in cloud market. In Eq. (2),

---

[†] If there exists free quota such as in GAE, the new expenditure $z'_t(t)=z_t(t)-\psi Y_0$ when $z_t(t) > \psi Y_0$ else $z'_t(t)=0$, where Y_0 is the free quota.



(a) Under heavy attacks (b) Under light attacks

Fig. 6 Comparisons of pay-as-you-go and pay-as-you-admit charging models under (a) heavy and (b) light DDoS attacks, where heavy attacks refer to those that can exhaust the hosted service's quota and light attacks refer to those that cannot.

when λ decreases, the net profit of cloud provider will decrease. And finally, they have the economic incentives to introduce the pay-as-you-admit charging model to attract customers.

In our analyses of pay-as-you-admit charging system, we assume each hosted service is provided with some existing authentication and admission control schemes [16]–[18], [25], [26], which are out of the scope of this paper. We assume these schemes can detect and reject all unwanted session requests. Since whether accepting an access or not is controlled by the hosted service, the cloud provider is liberated from the responsibility of rejecting valid accesses.

Let $z_t^*(t)$ denote the host service's expenditure in time $[0, t]$; thus $z_t^*(t) = \psi \sum_{i=1}^{x(t)} y_i$, its ensemble average E_{zt}^* for the time unit T_0 can be calculated as $E_{zt}^* = E[z_t^*(T_0)] = \psi\lambda\gamma T_0$.

Finally, the average net profit E_{ct}^* in time unit T_0 under the metered pricing scheme can be calculated as

$$E_{ct}^* = E_{zt}^* - E_{ce} = \psi\lambda\gamma T_0 - \chi(\lambda\gamma T_0 + \lambda_a l_a T_0). \tag{3}$$

In contrast with pay-as-you-go in Eq. (2), Eq. (3) shows that the cloud provider has economic incentives to filter unwanted traffic since unwanted traffic will reduce its profit under pay-as-you-admit charging model.

In summary, we have a conclusion: compared with pay-as-you-go, pay-as-you-admit can liberate the hosted service from unwanted traffic and force the cloud provider to filter unwanted traffic.

Numerical Analyses: we give some numerical results for our proposed pay-as-you-admit charging model.

Suppose the content revenue is ϕ= 0.01 (dollars) per end-client. In pay-as-you-go, the cloud provider charges the hosted service ψ=0.1 (dollars) per gigabyte traffic volume. The cost per gigabyte of the cloud provider is χ=0.01 (dollars). In the assumption A2, we have assumed that γ represents the average usage of resource paid by the hosted service for each client's access; therefore it represents the quality that a client obtained from the hosted service. A hosted service with higher quality can attract more clients until it becomes saturated gradually. As an instance for numerical analysis, we suppose the client average arrival rate $\lambda=10(1 - e^{-0.02\gamma})$ sessions per second, where γ (Mbytes) is the average consumed traffic volume per client.

Figure 6 compares pay-as-you-go and pay-as-you-admit perform under DDoS attacks. Here, we set γ=40 (Mbytes). The attack packets arrive in Poisson distribution with identical length of 40 bytes. The quota for traffic volume is set to 2000 (dollars) per day. In Fig. 6 (a), a heavy attack with a rate of 2×10^7 packets/sec can freeze the hosted service by exhausting its quota in pay-as-you-go charging system while the attack is inefficient under pay-as-you-admit charging models. The result in Fig. 6 (b) shows that the low-rate attacks in pay-as-you-go can increase the charge even though they may not cripple the hosted service. As a comparison, the charge in both pay-as-you-admit charging models are flat since the amount of admitted traffic volume stays the same. In summary, the results in Fig. 6 show that a pay-as-you-admit charging system can defend DDoS attacks against quota and avoid improper charging of unwanted traffic.

5.3 Charging in Service Provider

In this subsection, we assume there is no attack or the service provider is immune to the attack; thus $E_{zt} = E_{zt}^*$. We assume the service provider charges the end-client at a flat fee ϕ. Usually, a service provider can charge end-clients more for better quality of service. So, ϕ is a function of γ. Let $r(t)$ denote the revenue of a hosted service in $[0, t]$; thus $r(t)=\phi x(t)$, where ϕ is the average revenue that the hosted service can gain from each arrival. Then the ensemble average of revenue E_r for the time unit T_0 can be denoted as

$$E_r = E[r(T_0)] = \phi E[x(T_0)] = \phi\lambda T_0. \tag{4}$$

We denote E_{pt} as its net profit for the time unit T_0; thus $E_{pt}=E_r - E_{zt}^*$.

$$E_{pt} = (\phi - \psi\gamma)\lambda T_0. \tag{5}$$

The first result is $\phi - \psi\gamma > 0$; otherwise the cloud provider will opt out of the market. According to the assumption *A3*, the hosted service can adjust its service through adjusting the average usage γ of each client. The first-order differentiation of E_{pt} with respect to γ can be calculated by

$$\frac{dE_{pt}}{d\gamma} = (\phi T_0 - \psi\gamma T_0)\frac{d\lambda}{d\gamma} + (\phi' - \psi)\lambda T_0. \tag{6}$$

From Eq. (6), we can see that a hosted service has economic incentives to improve its service through adjusting γ only when $\frac{dE_{pt}}{d\gamma}$ >0, thus

$$\frac{d\lambda}{d\gamma} > \frac{(\psi - \phi')\lambda}{\phi - \psi\gamma}. \tag{7}$$

Otherwise, it is discouraged from improving its service. Intuitively, a hosted service will lose incentives to improve its service when $\frac{d\lambda}{d\gamma} \sim 0$, which means the improvement cannot attract much more clients.

When ϕ is a constant, the service provider may be discouraged from providing better quality of service when it reaches an optimal value. Currently, most online service with light traffic volume such as a newspaper web site makes profit from advertisement delivery can be looked as this kind.

When $\psi < \phi'$, we always have $\frac{d\lambda}{d\gamma} > 0$. In this case, the service provider always has the incentive to improve its service. This is one reason why the market of online video is discussing to charge content instead of recovering cost from the income of advertisement delivery.

6. Example Application: CLAD

If a cloud is immune to Denial-of-Quota attacks, we can rent its service and use the cloud infrastructure as a shield between the clients and traditional servers.

Distributed Denial of Service (DDoS) attacks have become one of the most serious threats to the Internet, but widely-deployable solution is still absent [28]. Most of current DDoS defense approaches [5], [18], [28], [29] require changing the intermediate routers and suggest to be implemented by end-hosts or ISPs, which result in high development costs that are beyond the affordability of small companies and individuals.

With cloud computing, we can shift the DDoS problem from the end-hosts to the more powerful cloud infrastructure, who is likely to have already DDoS protection as a core competency to absorb most DDoS attacks. As an example, we design and implement such a CLoud-based Attack Defense system, named *CLAD* [27], which is running on the cloud infrastructure as a hosted service to protect traditional Web servers from both network-layer attacks (such as UDP flooding, ICMP flooding, TCP SYN flooding) and application-layer attacks (HTTP flooding).

Comparing with most of current DDoS defense approaches, CLAD is more cost-efficient since the users can pay only for their usage on a short-term basis without paying far ahead for provisioning the equipments of large-sale networks and also save the human expense to operate it.

For cloud providers, CLAD could be a new kind of service. They can charge it at a suitable price. Obviously, CLAD will be deployed on a cloud only when the cloud provider's profit is than the cost. We think it is possible. Due to the effect of economies of scale, the cloud provider can offer a more competitive price (less than 1/5 [8]) than what individuals or even medium-scale companies would have to pay for the same hardware and network bandwidth. Moreover, the defense system could be shared among multiple of services to increase the system utilization.

6.1 Architecture

Figure 7 depicts the gist of CLAD, which is a *network service* running on cloud infrastructures. Each CLAD node can be viewed as a virtual machine or an application running Web proxy and control mechanisms. The protected server, which can be a single server or a server farm, is hidden from Internet and only accepts requests from the CLAD system. Usually, a client must consult DNS server for the IP address of the protected server before accessing especially when the DNS mapping changes frequently. For public Internet, name resolution for the protected server will resolve into IP addresses of the CLAD. And only the CLAD knows the exact concurrent IP address of the protected server. All traffic to the protected server is forwarded through at least one CLAD node, which verifies the clients and relays the requests.

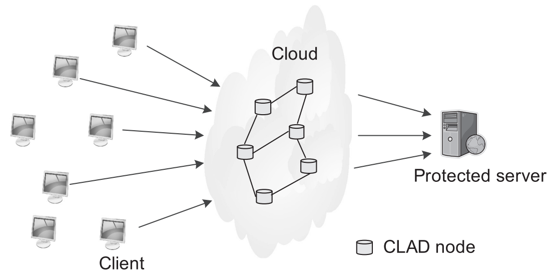


Fig. 7 Architecture of CLAD, which is a *network service* running on cloud infrastructures. A CLAD node can be viewed as a virtual machine or an application running Web proxy and control mechanisms. The protected server is hidden from Internet and only accepts requests from the CLAD system.

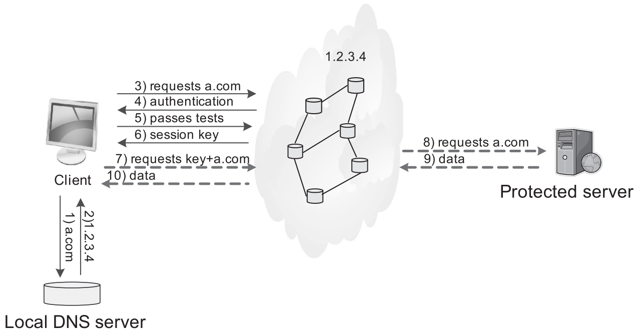


Fig. 8 Basic protocol of CLAD, where the solid lines represent the connection setup processes and the dashed lines represent the data transfer processes.

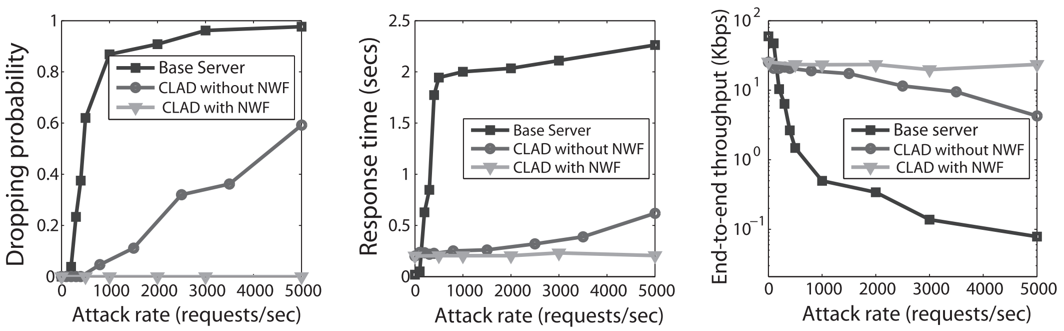
6.2 Protocol

Figure 8 shows the basic protocol of a CLAD system. It works as follows.

Steps 1-2: When a client wants to connect the protected server a.com, she first sends a request to the local DNS server to query the server's IP address. The DNS server returns the IP address (e.g., 1.2.3.4) of a CLAD node that is selected based on each CLAD node's load or healthy status. The name resolution queries from different stub networks can be resolved into the IP addresses of different CLAD nodes.

Steps 3-6: When the client requests a.com to the CLAD node 1.2.3.4, the CLAD node responds her with a graphical Turing tests page to ask for authentication. If the client can pass the tests, the CLAD node will assign her a session key.

Steps 7-10: The client can access a.com with the session key through 1.2.3.4. After the CLAD node has val-



(a) Request dropping probability (b) Request response time (c) Throughput

Fig. 10 Comparisons of (a) dropping probabilities, (b) response time, and (c) average end-to-end throughput for legitimate request as a function of attack rate, where "NWF" represents network-layer filtering.

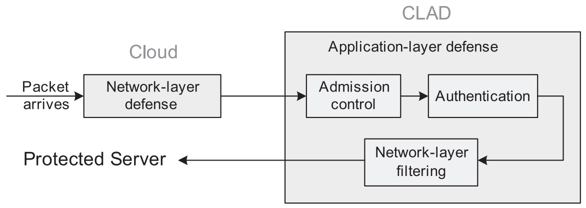


Fig. 9 Flow chart of CLAD.

idated the session key, it relays the request to a.com, and then relays the responded data from a.com to the client.

6.3 DDoS Defense with CLAD

In this section, we will present the flow chart of CLAD in detail, which is shown in Fig. 9. Each CLAD node can be a virtual machine or an application. We describe the network-layer defense in CLAD first.

6.3.1 Network-Layer Defense

How to defend against network-layer attacks depends on the execution environment of the cloud infrastructure. One case is that each CLAD node running on a completely controlled virtual machine such as Amazon EC2 [1] instance. Here, all packets (including SYN flooding, UDP flooding) will reach a CLAD node. The CLAD node should distinguish unwanted packets and configure the firewall to ask the cloud infrastructure to filter unwanted packets.

It is desirable to run CLAD on another more transparent execution environment such as Google's GAE [2], where each CLAD node running on a secure sandbox with only application-layer access allowed. When a new connection request arrives, the cloud infrastructure will first check whether it is an HTTP request. Since the cloud infrastructure only allows Web traffic to pass it, only HTTP requests can reach the CLAD system and other non-HTTP traffic such as network-layer attack (SYN flooding, UDP flooding, ICMP flooding) packets will be dropped by the cloud infrastructure. The network-layer attack defense process is transparent to both the protected Web servers and the CLAD system.

6.3.2 Application-Layer Defense

(1) Admission Control

When a client wants to access the protected server, she should attempt to obtain a session key *k* from CLAD first. After that, the client can access the protected Web server with the key embedded in the URL. A HTTP request with an invalid session key in its URL will be dropped by CLAD or redirected to an authentication page.

(2) Authentication

To distinguish human clients from robots, we adopt reCAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) [17], which is reported to be more secure than conventional CAPTCHAs. Each client that has passed reCAPTCHA test will be granted an authenticated session with a period. A client who has failed graphical tests after a few trials will be blocked and their IP addresses will be added to the blacklist.

(3) Network-Layer Filtering

In pay-as-you-admit charging model, CLAD is liberated from improper charge of unwanted traffic while the cloud provider is motivated to provide a desirable function—*network-layer filtering*—to filter attack packets at network-layer. Network-layer filtering can enable CLAD to filter unwanted requests at the ingress of cloud computing. As a result, these unwanted requests can not reach the CLAD node so that they will not be included in the costs.

We implement CLAD with Google App Engine (GAE) and run it on Google's cloud infrastructure, which reportedly consists of more than one-million servers distributed around the world. Each CLAD node works as a GAE application running in a secure sandbox environment that only allows other computers to communicate with it through Web requests. All network-layer DDoS attacks such as TCP SYN flooding and UDP flooding to the protected Web servers are filtered by Google. To evaluate CLAD under attacks, we have also implemented CLAD on CoreLab [19] and measured the performance with attacks launched from one hundred nodes on PlanetLab [30]. The detailed implementation and evaluation can be found in [27].

Figure 10 (a) compares the probability of dropping requests under attacks. The first result we have obtained is that our proposed CLAD with network-layer filtering can tolerate an attack rate of 5000 requests/sec while directly accessed base server ("Base server" in the figure) is completely denied of service at a request dropping rate of 99.5%. The second result is that a CLAD system with network-layer filtering is better than the one without it. This is because when CLAD is configured with network-layer filtering, all HTTP requests from banned clients will be rejected without arriving the CLAD system. Otherwise, although the HTTP requests from attackers may be dropped, they have already established TCP connections and consumed the resources such as CPU. Therefore, we conclude that network-layer filtering is a desirable function to CLAD design.

Figure 10 (b) compares the average response time of each successful HTTP request. We notice that even a CLAD system without network-layer filtering has a better performance of average response time than that of directly accessed base server. This is because the bad request will not reach the protected server. However, the response time will also increase with the increase of attack rate since those bad HTTP requests also consume the processing capacity of CLAD nodes. When the bad requests are filtered at network-layer, the average response time will decrease dramatically.

Figure 10 (c) compares the average end-to-end throughput of a legitimate client who sends one request per second to download a file of 100 Kbytes. We can see that a legitimate client of CLAD system with network-layer filtering achieves the best performance, followed by CLAD without network-filtering, then by the base server because the client of directly accessed base server will suffer from high request dropping probability and large response time.

7. Conclusion and Future Work

We propose pay-as-you-admit charging model that serves as an alternative charging scheme for cloud computing to defend DDoS attacks against quota and to avoid improper charge of unwanted traffic. This paper addresses the tussle between the cloud provider, the hosted service and the end-clients. How to maximize a cloud provider's revenue in a saturated cloud computing market in a price war among cloud providers is our future work.

We have also designed DDoS defense as a network service of cloud computing system called CLAD. Most of current DDoS defenses suggest to be implemented by end-hosts or ISPs, which are not so cost-efficient especially for small companies. We hope our design to be offered as a new network service by cloud computing so that even small companies can enjoy this service at a reasonable prize. And the competitiveness of this market will grant the clients more choices.

References

[1] "Amazon ec2," http://aws.amazon.com/ec2/

[2] "Google app engine," http://code.google.com/appengine/

[3] "Azure," http://www.microsoft.com/azure/

[4] "IBMSmartCloud," http://www.ibm.com/cloud-computing/

[5] A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," ACM SIGCOMM, 2003.

[6] P. Du, M. Chen, and A. Nakao, "Pay-as-you-admit: A new charging model of cloud computing," ICOIN, 2010.

[7] N. Kausar, B. Briscoe, and J. Crowcroft, "A charging model for sessions on the Internet," IEEE ISCC, 1999.

[8] "Above the clouds: A berkeley view of cloud computing," http://d1smfj0g31qzek.cloudfront.net/abovetheclouds.pdf

[9] "Example of court case," http://www.huffingtonpost.com/aaron-greenspan/why-i-sued-google-and-won_b_172403.html

[10] J. Cushnie, D. Hutchison, and H. Oliver, "Evolution of charging and billing models for gsm and future mobile internet services," LNCS, vol.1922, pp.321–323, 2000.

[11] A. Odlyzko, "Paris metro pricing for the internet," ACM conference Electronic Commerce, 1999.

[12] C.K. Chau, Q. Wang, and D.M. Chiu, "On the viability of paris metro pricing for communication and service networks," IEEE INFOCOM, 2010.

[13] S. Shenker, D. Clark, D. Estrin, and S. Herzog, "Pricing in computer networks: Reshaping the research agenda," ACM Computer Communication Review, vol.26, pp.19–43, 1996.

[14] K. Hosanagar, R. Krishnan, M. Smith, and J. Chuang, "Optimal pricing of content delivery network (CDN) services," 37th Annual Hawaii International Conference on System Sciences, 2004.

[15] M. Falkner, M. Devetsikiotis, and I. Lambadaris, "An overview of pricing concepts for broadband ip networks," IEEE Communications Surveys and Tutorials, vol.3, pp.2–13, 2000.

[16] A.D. Keromytis, V. Misra, and D. Rubenstein, "Sos: Secure overlay services," ACM SIGCOMM, 2002.

[17] L. Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum, "recaptcha: Human-based character recognition via web security measures," Science, vol.321, pp.1465 – 1468, 2008.

[18] P. Du and A. Nakao, "Mantlet trilogy: ddos defense deployable with innovative anti-spoofing, attack detection and mitigation," IEEE ICCCN, 2010.

[19] "Corelab," http://www.corelab.jp

[20] A. Nakao, R. Ozaki, and Y. Nishida, "Corelab: An emerging network testbed employing hosted virtual machine monitor," ROADS, 2008.

[21] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, and J. Rexford, "Openflow: enabling innovation in campus networks," SIGCOMM Comput. Commun. Rev., vol.38, pp.69–74, 2008.

[22] P. Du, M.K. Chen, and A. Nakao, "Port-space isolation for multiplexing a single IP address through open vswitch," TridentCom, 2010.

[23] P. Du, M.K. Chen, and A. Nakao, "OFIAS: A platform for exploring in-network processing," TridentCom, 2011.

[24] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: Towards an operating system for networks," SIGCOMM Computer Communication Review, vol.38, pp.105–110, 2008.

[25] D.G. Andersen, "Mayday: Distributed filtering for internet services," 4th USENIX Symposium on Internet Technologies and Systems (USITS), 2003.

[26] S. Kandula, D. Katabi, M. Jacob, and A.W. Berger, "Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds," USENIX NSDI, 2005.

[27] P. Du and A. Nakao, "Ddos defense as a network service,"

IEEE/IFIP NOMS, Poster Session, 2010.

[28] P. Du and A. Nakao, "Overcourt: ddos mitigation through credit-based traffic segregation and path migration," Computer Communications, Elsevier, vol.33, pp.2164–2175, 2010.

[29] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y. Hu, "Portcullis: protecting connection setup from denial-of-capability attacks," ACM SIGCOMM, 2007.

[30] "Planetlab," http://www.planet-lab.org/

Ping Du received B.E and M.E degree from University of Science and Technology of China in 2000 and 2003, respectively. He received a Ph.D. from the Graduate University for Advanced Studies in Japan in 2007. Now, he works as a researcher at the National Institute of Information and Communications Technology (NICT) of Japan. His research interests include optical network, network security, network virtualization etc.

Akihiro Nakao received B.S. (1991) in Physics, M.E. (1994) in Information Engineering from the University of Tokyo. He was at IBM Yamato Laboratory/at Tokyo Research Laboratory/at IBM Texas Austin from 1994 till 2005. He received M.S. (2001) and Ph.D. (2005) in Computer Science from Princeton University. He has been teaching as an Associate Professor in Applied Computer Science, at Interfaculty Initiative in Information Studies, Graduate School of Interdisciplinary Information Studies, the University of Tokyo since 2005. (He has also been an expert visiting scholar/a project leader at National Institute of Information and Communications Technology (NICT) since 2007.)