# A New Form of DOS Attack in a Cloud and Its Avoidance Mechanism

Huan Liu
Accenture Technology Labs
50 W. San Fernando St., Suite 1200
San Jose, CA 95113
huan.liu@accenture.com

## ABSTRACT

Data center networks are typically grossly under-provisioned. This is not a problem in a corporate data center, but it could be a problem in a shared infrastructure, such as a co-location facility or a cloud infrastructure. If an application is deployed in such an infrastructure, the application owners need to take into account the infrastructure limitations. They need to build in counter-measures to ensure that the application is secure and it meets its performance requirements. In this paper, we describe a new form of DOS attack, which exploits the network under-provisioning in a cloud infrastructure. We have verified that such an attack could be carried out in practice in one cloud infrastructure. We also describe a mechanism to detect and avoid this new form of attack.

## Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Network topology*

## General Terms

Design, Security

## Keywords

DOS attack, bandwidth estimation

## 1. INTRODUCTION

Data center networks are typically grossly under-provisioned. Typical designs are under-provisioned by a factor of 2.5:1 to 8:1 [22]. An 8:1 under-provisioning ratio means that a host can only send at 1/8 of its interface speed for some communication patterns. The reasons for this level of under-provisioning are two-folds. First, it is prohibitively expensive to build a network with full 1:1 bi-section bandwidth, even for a modest-size data center. Second, although the current network architecture supports multi-path routing with ECMP [8], the number of paths supported is typically small. Fig. 1 shows a typical data center network architecture. There are several tiers of routers, each aggregates traffic from the local tier. There are two routers at each tier providing both path diversity and fault tolerance. Since the number of paths is limited, the full bisection bandwidth of a network cannot exceed a small multiple (in Fig. 1, only twice) of a single link's capacity. Although proposals to change the network architecture exist [1] [20] [6] [7] [5]. , they often require router changes and it will take years to swap out the existing routers, even if they are adopted by the industry.

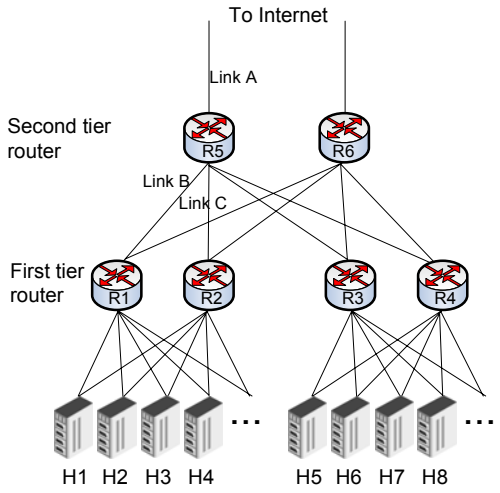


**Figure 1: A typical data center network architecture.**

Under-provisioning is typically not a problem in a traditional corporate data center. The standard practice is to co-locate servers for an application (e.g., web servers, app servers and database servers for a multi-tier application) in the same subnet; thereby localize the bulk of the communication. Since data center managers have the full control over the infrastructure, they can perform the optimizations necessary to avoid the worst-case communication patterns. In addition, because of the full control, data center managers can track down offending applications or put in counter-measures if and when the worst-case communication pattern happens.


*CCSW'10,* October 8, 2010, Chicago, Illinois, USA.

However, under-provisioning could be a problem in a cloud data center or a hosting company's data center. The problem arises because a cloud data center is different from a corporate data center in several regards. First, a cloud data center is typically much bigger than most corporate data centers. For example, at the end of 2007, Rackspace has 36,692 servers, hosted in 114,749 square feet of data center space [25]. Because of the limit on the number of multi-paths in the current network architecture, a cloud data center' network could be dramatically under-provisioned. For example, our analysis of a cloud company's network shows that it is under-provisioned by at least a factor of 45:1 [1].

Second, a cloud data center could be used by many people from many organizations. This opens doors for adversaries to attack other applications hosted in the same cloud data center. In Sec. 2, we describe how an adversary can exploit the under-provisioned network and perform a DOS (Denial Of Service) attack on other applications hosted in the same data center.

Third, an application owner has no or little control over the underlying network in a cloud data center. In a corporate data center, an application owner at least has an indirect access to the underlying network, so that she can put in counter-measures in the network if needed. But, application owners do not own the cloud data center, and they have a very limited visibility into and control on the underlying network in the cloud.

Solving the DOS attack that we describe in Sec. 2 is difficult in a traditional data center without human intervention. Even if the application can detect that an attack is underway, it has no ability to dynamically provision the infrastructure to react. Since manual intervention is required, it often takes days to react to a DOS attack, long after the damage is incurred. However, when data center is virtualized and when a self-service infrastructure management capability (such as the ability to start new virtual servers on demand) is built-in (e.g., in a cloud data center), an application owner can leverage the new provisioning capability to build a highly dynamic application which can automatically detect and avoid these kinds of DOS attacks.

We have verified that the new form of DOS attack could be effectively launched in one cloud provider's data center in practice [2]. Because the data center manager was not aware of this new type of DOS attack, no measure was in place to detect and prevent. A hacker could potentially bring down many cloud users at the same time with a minimal cost. We have also verified that our solution, described in Sec. 3, can effectively detect and avoid such attacks. Even though we have only verified the problem in one cloud data center, we believe the under-provisioning problem is widespread. Furthermore, we believe that the DOS attack and our avoidance mechanism are applicable to many other cloud data centers.

The contributions of this paper include:

- Identify a new form of DOS attack in a cloud data center, and verify that such an attack could be carried out in a real cloud data center. We further show that this attack could be carried out even if the cloud vendor locks down its network completely, for example, by disabling TTL expiration notices. We demonstrate that the topology information needed for attack could still be obtained by a network probing technique.

- Propose and evaluate a new mechanism for applications to dynamically relocate to a different infrastructure when the desired Quality of Service (QoS) could not be met. This mechanism is applicable not only when an application is under a DOS attack, but it could also be used when QoS degrades due to other legitimate bandwidth-hungry applications.

- Propose and evaluate a new available bandwidth detection technique which can accurately determine the available bandwidth in a high speed network.

[1]Analysis omitted due to confidentiality agreement

[2]Data center name not released due to confidentiality agreement

## 2. A NEW FORM OF DOS ATTACK

The gross under-provisioning and the public nature of a cloud data center open a potential venue for exploit. An adversary can simply saturate the limited network bandwidth to perform a DOS attack against other applications in the same network. To perform such an attack, an adversary must first identify a bottleneck link in the network to attack.

In a data center network today, a router is typically connected to a large number of hosts (or other routers), and the aggregate capacity of these hosts greatly exceeds the uplink capacity. In Fig. 1, Link A, B, and C are the uplinks of router R5, R1 and R2 respectively. We refer to the hosts connected to a router (e.g., host H1..H4 for router R1 or R2) as part of the *subnet* of the router. We use the term *subnet* loosely. Even if the hosts are connected to a layer 2 switch, we still refer to them as the subnet of the switch.

Since the uplink capacity is much smaller than the aggregate subnet capacity, an uplink is an easy attack target. To attack, an adversary simply transmits enough traffic from a few hosts in the subnet to hosts in other subnets. By symmetry, the traffic would saturate the uplink in both directions. As an example, let us consider Link B as a target, assuming Link B is the active link and Link C is a fail-over link. To saturate Link B, an adversary needs to send traffic from a host in R1's subnet (e.g., H1) to another host in a different subnet (e.g., H5). Due to under-provisioning, a small number of hosts in R1's subnet are sufficient to saturate link B. In a commodity data center, link B and C typically have 1Gbps of bandwidth and the hosts also have 1Gbps interface speed; thus, one host is sufficient. When Link B is saturated, no other legitimate traffic could reach other hosts in R1's subnet, effectively denying services to all hosts in R1's subnet.

We distinguish between an *untargeted* and a *targeted* attack. In an untargeted attack, an adversary just want to have a critical mass (enough hosts) in any subnet. In a targeted attack, an adversary wants to gain a critical mass in a specific subnet, for example, to attack a specific application hosted in that subnet.

### 2.1 Topology identification

Taking topology information into account (which hosts are under which subnets) is important to launch an effective attack. A naive approach is to gain access to a number of hosts in a cloud data center, then blindly send traffic to each other at the maximum rate. Unfortunately, such

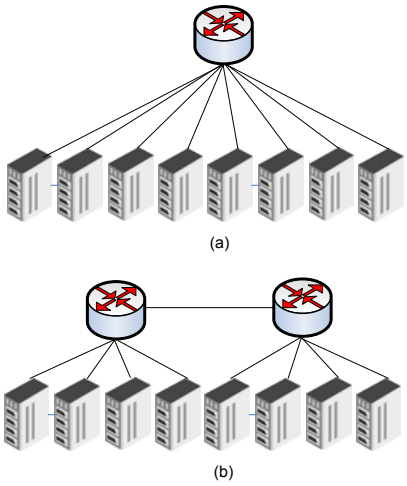


**Figure 2: Gaining topology information is important to maximize attack effectiveness. (a) sending traffic at full speed would not saturate network. (b) sending traffic across the link between the two routers can saturate the network.**

a brute-force attack may not be effective at all. Consider the example shown in Fig. 2 where an adversary is able to gain access to eight hosts. For simplicity, we assume all links have 1Gbps capacity. The topology could be as shown in Fig. 2(a), where all hosts are connected to the same switch. An adversary could blindly send traffic, but no traffic pattern would saturate the network since the switch can support 100% throughput. However, if the adversary knows that the topology is as shown in Fig. 2(b), he can easily identify the link between the two routers (an uplink for both routers) as a bottleneck and he can send traffic from just one host on the left to another host on the right to saturate the network.

To carry out an attack, an adversary would first gain access to a set of hosts (e.g., by launching virtual machines using the cloud API), then learn the topology, and determine whether there is a bottleneck link to attack. If none found, the adversary can continue to gain access to more hosts and repeat the steps above. In this section, we describe two approaches one can use to identify the topology given a set of hosts.

In the first approach, one can gain a good insight into the network topology by simply running Traceroute from the end hosts. By running Traceroute among all pairs of nodes, one can map out all routers connecting the hosts, and learn what hosts are in the same subnet. For ease of setting up and management, data center networks typically follow a regular structure and the IP addresses are typically assigned based on a set naming convention. Running Traceroute from a few hosts is often enough to infer the overall IP layer topology.

There are two disadvantages of using the Traceroute approach. First, one cannot see layer 2 switches through the Traceroute command, but only the layer 3 routers. Second, Traceroute requires router support, i.e., routers must respond to TTL (Time To Live) expiration. Such router support could be disabled, rendering Traceroute useless. Today, most cloud networks support Traceroute and there are

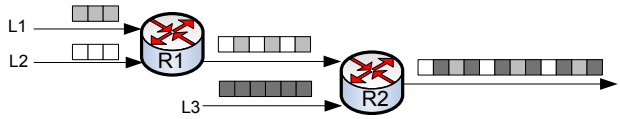


**Figure 3: Routers multiplex incoming packets.**

good reasons for it. First, it is a valuable debugging tool for both the administrators and the application owners. Second, it requires router change to disable TTL expiration, which cannot readily be done by a network administrator. Even though difficult, an administrator may eventually disable TTL as the DOS attack we describe in this paper becomes more common.

In addition to the first approach using Traceroute, we propose another approach to detect the network topology. The second approach exploits the multiplexing nature of a router. As shown in Fig. 3, packets from the two incoming links (link L1 and L2) of router R1 are multiplexed before they are sent out on the outgoing link. At the downstream router R2, the packets are again multiplexed with other packets from the other incoming link (link L3). As a result, at the final output link, packets from L1 and L2 receive half of the bandwidth compared to that of L3. In general, the further away a link is from the output link, the less bandwidth it will receive.

Given a set of hosts, to probe for the network topology among them, we choose one host as the sink and the rest of hosts as sources. From each source, we send a sequence of packets back-to-back to the sink at the same time. At the sink, we measure the number of packets that we received from each source. Based on the received traffic rate, we can derive how many switches are between a source and the sink; therefore, we can derive the topology from the sink host's perspective. To build a complete topology, we need to choose all hosts as the sink hosts and construct the view from each host's perspective. Although we suspect that a much smaller number of sink hosts are needed, we leave that optimization as an enhancement for future work.

We should note that our detected topology may be different from the actual topology due to a *compression* effect. For example, in Fig. 4, the detected topology (Fig. 4(b)) is different from the real topology (Fig. 4(a)). This is because there are two routers on the same path, and packets traverse through them unaltered. Since we are not able to inject traffic in between the two routers, we are not able to observe the existence of the extra router. This inaccurate view is not a problem for us, since we are only interested in determining if we have enough critical mass in a router's subnet to launch an attack.

Since each node will send traffic at its maximum interface speed during probing, the load to the network could be very high. To minimize impact, we limit the probing length, e.g., to a few milliseconds. We denote the probing length – the time to continuously send packets from a host – to be $L$. The probe length $L$ varies as the number of hosts involved. The more hosts, the longer the probe length needs to be to maintain good resolution. The probe length $L$ also needs to account for the network latency incurred when the master node sends out commands to all hosts to start sending probing packets at the same time. At the sink host, we start the measurement period after we have received at least one

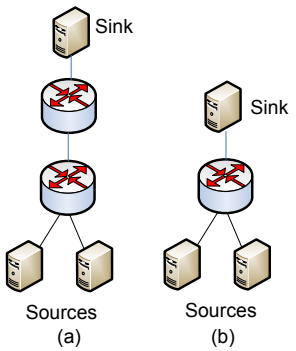


**Figure 4: The *compression* effect. (a) The real topology. (b) The detected topology. One of the routers is invisible because we are not able inject traffic in betweem.**

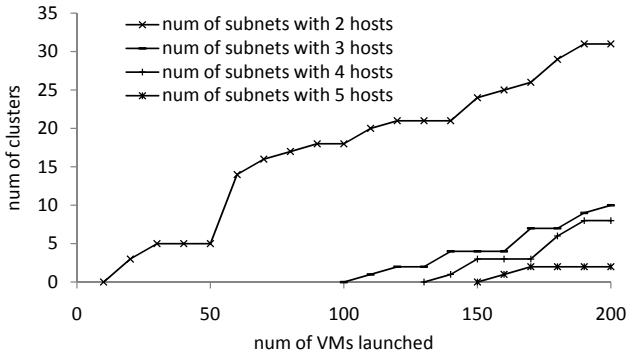


**Figure 5: Number of clusters formed, where each cluster is in a single subnet. Cluster size varies from 2 to 5.**

packet from each source, and we continue packet collection as long as possible until a point where at least one packet from each source is still in the queue. This guarantees that we capture the period where packets from all sources are being multiplexed.

The bandwidth probing technique described above only works well when all the links have the same capacity, e.g., 1Gbps. This is true for all three different cloud providers that we investigated. The reason is that cloud is built on commodity hardware and 10Gbps links and routers are still very expensive. If a router's uplink capacity is higher than the incoming link's capacity, two streams of probes from two incoming links would retain the same data rate after passing the router. Since the probing traffic's rate is not reduced, we would not observe this router in our probing. Although the probing technique would not accurately depict the network topology if an uplink has a higher capacity (e.g., 10Gbps), those links are unlikely to be a bottleneck link because of their higher capacity; thus, they are less interesting from a DOS attack perspective.

We have done a preliminary evaluation of the second proposed approach of topology detection in one cloud vendor. Instead of evaluating whether the detected topology is exactly the same as the real topology (e.g., as detected by the first traceroute approach), we check whether we can accurately find the router whose subnet contains the most number of hosts. In the first experiment, we launch 10 hosts in the cloud vendor and compare the two approaches for topology detection. In both cases, we are able to identify the router with the most number of hosts in its subnet (which has 2 hosts). We complete the detection using the second approach in 45ms, which includes the probe length $L$, which is set to be 5ms, and the network latency needed to transmit the commands, to analyze the received probe packets and to collect the results. In a second experiment, we launch 20 hosts. Again, we are able to accurately find the router with the most number of attached hosts (which has 3 hosts), and we are able to complete the detection using the second approach in 87ms.

## 2.2 Gaining access to enough hosts

In order to saturate an uplink, we must have access to enough hosts connected to the router. How many hosts is enough depends on the uplink capacity, which is not readily known from the topology information. We assume that we can determine the uplink's capacity using a capacity estimation tool, such as Pathload [4], Nettimer[17] or Bprobe [2]. Alternatively, we can keep on increasing the number of hosts in a subnet, and send maximum traffic from all available hosts, until the uplink is saturated.

In this section, we show that it is both possible and economical to launch a large number of VMs in a subnet. We first consider an untargeted attack, i.e., the attacker is interested in gaining access to enough hosts in any subnet.

We first discovered and verified this form of DOS attack in a cloud vendor in Apr. 2009. Through trials of launching many Virtual Machines (VM) in the cloud provider, we learned the VM assignment algorithm used at the time, which is responsible for determining which physical host (thus which subnet) to launch a new VM on. By exploiting the VM assignment algorithm, we were able to launch enough hosts in a subnet in a single trial. Further, in most cloud vendors, we only need to gain access to a small number of hosts in a subnet to launch an attack. This is because the uplink bandwidth is typically the same as the host's interface speed (e.g., 1Gbps). Because of the small number of hosts needed, lauching an attack in those cloud vendors is both fast and economical.

Since the VM assignment algorithm may change over time and since it is different across cloud vendors, we focus on using a brute-force approach in this section. We run the following experiment in a cloud vendor to see how soon we can have enough hosts in a subnet. We launch 10 VMs at a time, then run the probing technique to determine how many are in a subnet. We then launch an additional 10 VMs and probe again. Fig. 5 shows the results when we launch up to 200 VMs. With 20 VMs, there are already 3 2-host clusters, where the two hosts are in the same subnet. At 200 VMs, there are 31 2-host clusters. Larger clusters are harder to form. At 200 VMs, there are 10 3-host clusters, 8 4-host clusters and 2 5-host clusters. Even though harder, our results show that it is still economical to launch a cluster of VMs inside a single subnet.

We should note that our experiment is different than that in [28], where they tried to see how soon one can launch a single VM co-located on the same physical machine as a target VM. Instead of on the same physical hardware, we are

interested in how quickly we can launch a sufficient number of VMs in the same subnet.

In addition to an untargeted attack, we also carry out an experiment to see how fast one can launch a targeted attack. A targeted attack is used when the adversary is not interested in bringing down just any part of the cloud, but rather interested in bringing down a specific application hosted in a specific subnet. The adversary would find out which router the target application is sitting behind through its IP address, then launch an attack for that specific router's subnet.

To simulate a targeted attack, we randomly choose a subnet in the cloud provider's network, then launch 10 VMs at a time to see how fast we can form a cluster in that subnet. We form a 2-host cluster at 60 VMs, a 3-host cluster at 160 VMs, a 4-host cluster at 210 VMs, and a 5-host cluster at 320 VMs. Even though it takes more VMs to launch a targeted attack, it is still quite fast and economical to do.

## 2.3 Carrying out the attack

Once an attacker has gained access to enough hosts, all he needs to do is to send a large amount of traffic through the upstream router to saturate the router's uplink. The easiest way is to send a large amount of UDP traffic. Using UDP packets is important because it will starve the other TCP sessions, who back off during congestion. To make sure the attack traffic goes though the uplink, the attacker needs to send traffic to hosts in other subnets.

Totally saturating the uplinks is not effective against highly adaptive applications. For example, some applications have active standbys which actively communicate with the main application through a heart-beat mechanism. During an attack, if no heart-beat message is able to get through, the standby may think the main application has died. If the standby is in a different subnet than the one under attack, it could trigger the fail-over mechanism, rendering the attack ineffective.

For these adaptive applications, an attacker can avoid sending traffic at full speed. Instead, they can use a bandwidth estimation technique, such as the one we will describe in Sec. 3, to estimate the remaining bandwidth, and send only enough traffic to saturate most of the bandwidth, leaving some for the heartbeat mechanism. Since the heartbeat mechanism requires very little bandwidth, it can go through. However, the application users will still experience a much degraded level of service.

# 3. DOS ATTACK/BANDWIDTH STARVATION DETECTION AND AVOIDANCE

There are several ways a cloud provider could prevent this type of DOS attack, but none of them is attractive. First, a cloud provider could provision more bandwidth up to the full bisectional bandwidth. Although this will guarantee that no DOS attack is possible, this solution is cost prohibitive, especially given the current network architecture. Further, the solution may not even be feasible for large cloud data centers, since the root router must support the full bisectional bandwidth. Second, a cloud provider could cap the bandwidth consumption per server to its proportional share. Since the current data centers are grossly under-provisioned, a proportional share is only a small fraction of the capacity, which significantly limits a server's throughput. Third, a cloud provider could charge for the bandwidth consumption, thus, deterring the adversary economically. Unfortunately, a cloud vendor could not charge a high price for bandwidth, since it would make the cloud business model unattractive. But if a cloud vendor only charges a small price, it is not enough a deterrence for a determined attacker. Fourth, the cloud vendor could attempt to detect high bandwidth usage and shutdown offending applications if needed. Unfortunately, it is difficult to determine whether it is a legitimate bandwidth-hungry application or an attacker. This is especially true since an attacker could get multiple cloud accounts, disguising as multiple legitimate cloud users.

In this section, we present an approach to detect and avoid the DOS attack described in the last section, which leverages the dynamic provisioning capabilities offered by a cloud data center. This solution is a user-side solution, which does not require any cloud service provider cooperation. Thus, even in a data center that does not have any counter-measures (e.g., either due to cost or performance reasons described above), our proposed solution can help a cloud user avoid this form of DOS attack.

Traditional DOS or DDOS (Distributed Denial Of Service) attacks [31] and their counter-measures are well known. There are sophisticated techniques[14] to counter even the most elaborate DDOS attacks where the bot simulates a human behavior. Once a DOS attack is detected, traceback mechanisms [29, 33] can be used to trace and possibly stop the attack at the source. However, all those techniques assume that the attacker is sending packets directly to the application, so that the application can learn that an attack is underway. But, in the new form of DOS attack we described, the application is not even aware that a DOS attack is underway; thus, new techniques to detect and avoid are needed.

Fig. 6 shows our solution architecture. There is a monitoring agent which resides either in a different subnet or outside the cloud infrastructure, e.g., in a corporate data center. The monitoring agent and the application constantly probe each other to see the available bandwidth in both directions (we describe how we probe in Sec. 4). When the application notices that the available bandwidth degrades below a certain threshold, it starts sending a large number of UDP packets to the monitoring agent to ask for help. Even in a total bandwidth starvation, the UDP packets would compete with other attack packets and a portion of them can still get through. When the monitoring agent either detects a bandwidth deterioration or receives a help packet, it initiates application migration. It launches a new active standby in a different subnet behind a different router if there is not an active standby already, and it then measures the bi-directional bandwidth between the monitoring agent and the active standby to make sure there is enough bandwidth. If the bandwidth is not sufficient, it keeps on launching new instances until it can find a free subnet.

When an attacker attacks several subnets at the same time, it may not be possible to find a free subnet in a few trials. The monitoring agent gives up after a set threshold (currently set to 3 trials), it then starts to launch new servers in a different cloud provider. When it finds a good subnet in a cloud provider to host the active standby, it converts the active standby to be the main application.

To prevent further attacks, we can optionally start application hopping – moving the application from one subnet to

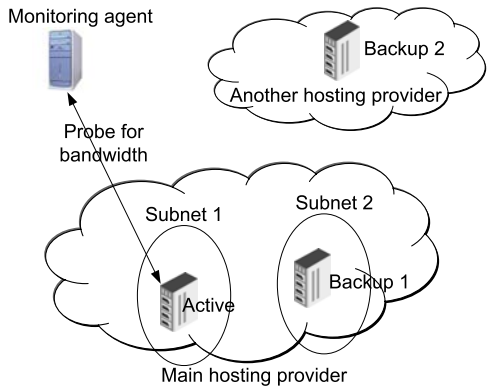


**Figure 6: Bandwidth estimation and migration when under attack.**

another every few minutes before the adversary could launch enough servers in the new subnet to attack again.

Converting an active standby to be the main application requires reprogramming the DNS entry. Due to DNS caching, users of the application may perceive that the application is not available for a short period of time (typically less than a few hours). Even though it may take hours, it is still better than the traditional manual approach to DOS prevention. For example, a DDOS attack launched from Korean paralyzed many US government sites for days in 2009 [16].

The reaction time could dramatically improve with the newer dynamic capabilities offered by a cloud. For example, Amazon Web Service – a prominent cloud provider – offers a service called "ElasticIP", which allows the user to re-assign an IP address to a different VM possibly in a different subnet. Compared to DNS programming, ElasticIP is much faster. In our experiments, it takes roughly 1 to 5 seconds to reassign an IP address.

We have implemented the DOS attack avoidance mechanism. To evaluate its effectiveness, we consider a stateless web application. We use a default Apache web server installation on top of a Ubuntu Linux distribution. The web server and the OS are captured in a machine image, so that when we launch a VM from the machine image, the Apache web server would start automatically. When we migrate the web application, we assume no state information, including the current sessions in progress, needs to migrate. A different application may require a longer time to migrate state information than what we report here, if it is at all possible to migrate under bandwidth starvation.

We set up an experiment where we attack a single subnet where the web application is hosted. We run the experiment 10 times to measure the overall time it takes from attack launching to the application successfully migrating to a new subnet. We first measure the case where there is an active standby VM in a different subnet. The total time to migrate varies from 2.3 seconds to 7.1 seconds with an average of 5.2 seconds, where the majority of time is spent waiting for the reprogrammed ElasticIP to take effect. We then measure the case where there is no active standby. The total migration time ranges from 38.2 seconds to 59.5 seconds with an average of 47.3 seconds. The majority of time is spent launching the standby VM and then booting up the Linux OS. In all cases, we are able to migrate within a minute, which is much less than the days typically required to mitigate a tradition DOS attack.

The proposed solution assumes that not all cloud providers are under attack at the same time. This is reasonable because there are a large number of cloud vendors and each cloud vendor has many data centers; thus, a systematic attack is unlikely. If a cloud provider could put in countermeasures to protect against a data-center-wide attack, our solution would work even if only a single cloud provider is used.

A cloud data-center-wide solution is easy to put in. In order to bring down a whole cloud data center, an attacker must attack the uplink of the root router in the data center (e.g., Link A of router R5 in Fig. 1). In order to attack the root router, an attacker must send a large amount of traffic to hosts outside of the cloud. Since the bandwidth between the cloud and an outside end host is likely small (few end-to-end Internet paths are able to support 1Gbps bandwidth), it would be easier for the cloud provider to identify the attacker's traffic, as its sending rate inside the cloud is substantially higher than the rate sustained outside the cloud. Even if an attacker can secure a high-speed end host and a high-speed link from that host to the cloud, sending traffic at a high rate becomes costly (traffic to outside of cloud is typically not free), making it difficult economically to carry out the attack. Alternatively, an attacker could send traffic (if he is willing to pay for the bandwidth cost) to high-speed hosts that he does not control (e.g., google.com), the high traffic rate would raise alarm at the end host, making it easy to detect the attacker. Once the attacker is identified, the cloud provider could easily shut down the attacker. In comparison, if an attacker attacks a first-tier router (e.g., R1 in Fig. 1) while sending traffic to another host in a different subnet, the traffic could be indistinguishable from legitimate traffic generated by a bandwidth hungry application; thus, it is much harder for a cloud provider to detect and avoid.

Our proposed solution is an application-side solution, i.e., it requires no cooperation from the cloud provider. It compliments any cloud provider solutions. The cloud provider solution focuses on detecting and shutting down easy-to-detect data-center-wide attacks. In contrast, our application-side solution focuses on detecting degradation in quality of service due to either a smaller-scale attack or an increase in legitimate traffic in the same subnet, and proactively relocating the application for a better quality of service.

# 4. BANDWIDTH ESTIMATION

Existing DOS detection methods may use a combination of signals available to the host, such as CPU utilization and memory usage, to detect a DOS is underway. However, in the new form of DOS attack, those signals available to the host are not affected, as the attack is on the underlying network. To detect this new form of DOS attack, we must first know that the available bandwidth is getting depleted; thus, we need a mechanism to accurately estimate how much bandwidth is left. Note that a simple Ping mechanism to see if an application responds is not sufficient because an attacker may not drain the bandwidth completely, leaving small room for Ping or other heart-beat mechanisms.

In this section, we describe a novel technique to accurately detect the available bandwidth in a high speed network.

## 4.1 Prior work

There has been a large body of existing work on estimating network bandwidth (see [24] for a good survey). Some work, such as Pathrate [4], Nettimer[17] and Bprobe [2], focus on measuring the link capacity, while others, such as Pathload [12], TOPP [19], PTR/IGI [11], Packet pair [15], Delphi [26], Pathchirp [27], Spruce [32], Yaz[30], focus on measuring the available bandwidth. We focus on estimating the available bandwidth because it directly corresponds to the quality of service the application would receive.

The available bandwidth is the unused capacity on a link. It is defined by averaging over a time window $T$ as

$$A_i(t) = \frac{1}{T}\int_t^{T+t}(C_i - \lambda_i(t))dt \quad (1)$$

where $A_i(t)$ is the available bandwidth on link $i$ at time $t$, $C_i$ is the link capacity, and $\lambda_i(t)$ is the amount of traffic on link $i$ at time $t$. The available bandwidth along a path is the minimum available bandwidth of all links making up that path.

The existing available bandwidth estimation tools can be divided into two categories based on the underlying approaches to estimation.

- **The probe gap model (PGM)**. PGM tools send a pair of probe packets with a pre-determined time gap of $\Delta_i$, and they arrive at the receiver with a time gap of $\Delta_o$. If we assume there is a single bottleneck and if we assume the queue is not empty between the two probe packets, then $\Delta_o$ is the time taken by the bottleneck to transmit the second probe packet and the cross traffic that arrived during $\Delta_i$. Therefore, the time to transmit the cross traffic is $\Delta = \Delta_o - t_p$, where $t_p$ is the time taken to transmit the second probe packet. The rate of the cross traffic is then $\frac{\Delta_o - t_p}{\Delta_i} \times C$, where $C$ is the bottleneck link capacity. The available bandwidth can be derived as follows:

$$A = C \times (1 - \frac{\Delta_o - t_p}{\Delta_i}) \quad (2)$$

  The PGM model is shown in Fig. 7. Spruce[32], IGI[11], Pathneck[9][10], and Delphi [26] are examples of using the PGM model. The PGM assumes a single bottleneck which is both the narrow link (link with the smallest capacity along a path) and the tight link (link with the minimum available bandwidth along a path). This assumption is necessary for the model analysis, but the tools work well in practice even when the assumption does not hold [11].

- **The probe rate model (PRM)**. The PRM model is based on the theory that if you send traffic at a rate higher than the available bandwidth, packets would be delayed; thus, the received packet rate would be lower. However, if you send traffic at a lower rate, the received packet rate should be no different from the sending rate. One can send a sequence of packets at different rates to search for the turning point, from where the received rate is lower than the send rate. The turning point is the available bandwidth. Pathload [12], Pathchirp [27], PTR[11], and TOPP[19] use the probe rate model.

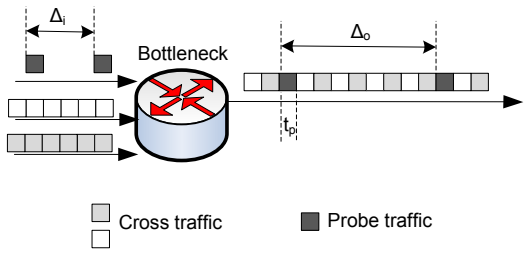


**Figure 7: The probe gap model (PGM).**

These existing bandwidth estimation tools work well when the link capacity is lower. However, in a cloud data center, the link capacity is typically 1Gbps or more. At such a high speed, a slight error in timing measurement could dramatically affect the end result. At 1Gbps, it takes only 12us to send a maximum size 1500B Ethernet packet; thus, there is not much room for errors. As described in [13], a number of system level issues could affect the timing measurement, making it hard to collect accurate timing information. There are two key problems on timing measurement at high speed. First, it is difficult to pace sending packets accurately at high speed on a general purpose CPU. Second, most systems support interrupt coalescence, where the network interface card waits until several packets are received before raising the interrupt to the system. Interrupt coalescence is necessary to reduce the CPU load, but it introduces significant timing error that makes it difficult to estimate available bandwidth accurately. For a full discussion on the timing issues, we refer interested readers to [18][13].

To cope with the timing inaccuracy at high speed, recent proposals for bandwidth estimation, such as PBProbe [3] and ICIM-abw [18], send a burst of packets in place of a single probe packet. The idea is to enlarge the probe packet beyond the 1500B packet limit. Unfortunately, a burst of packets do not behave the same as a giant packet. For example, the burst of packets may be spread out during transmission and the burst of packets may not be received together at the receiver even if they arrive back-to-back.

## 4.2 A new bandwidth estimation tool

In this paper, we propose a new technique called Loaded Spread Pair estimator (LSP). It is an extension of the Spruce [32] tool, which improves upon Spruce to tolerate timing errors. For ease of discussion, we first describe how Spruce works.

Spruce is a tool based on the PGM model, and it computes the available bandwidth based on equation 2. It sends two 1500B packets with a spacing $\Delta_i$ the same as the time it takes to transmit a 1500B packet on the bottleneck link. The spacing is chosen to make sure that the queue is not empty between the departure of the two probe packets, a requirement necessary for deriving equation 2. At the receiver, Spruce measures the received packet spacing $\Delta_o$ and then use equation 2 to compute the available bandwidth.

To average out timing measurements over time, Spruce performs a sequence of measurements, where the time between two measurements is set to be an exponentially distributed function, whose time constant $\tau$ is much larger than the send gap $\Delta_i$. This essentially results in a Poisson sam-

pling process, which ensures that we observe the average cross traffic rate. The time constant $\tau$ could be tuned to adjust the intra-measurement spacing, so that we can adjust the load the measurement tool puts in the network so that it is not intrusive for other applications.

The key problem with Spruce is that the send gap $\Delta_i$ is too small at high speed, yet it is necessary to ensure that the queue at the bottleneck link is not empty between the departure of the two probe packets. To allow a larger send gap, in LSP, we send a burst of $L$ 1500B load packets first, then we send a pair of small 64B probe packets spaced $\Delta_i$ apart. Compared to Spruce, $\Delta_i$ is set to be much bigger. As a result, $\Delta_o$ tends to be bigger also. Because $\Delta_i$ and $\Delta_o$ are much larger, LSP can tolerant small timing errors better than Spruce can. The $L$ load packets are designed to keep the queue loaded so that when the second probe packet arrives, the first probe packet is still in the queue, thus satisfying the requirement for equation 2.

The time difference from sending the first load packet to sending the second probe packet is $T = L \times 1500 \times 8/C + \Delta_i$. We have to ensure that there are enough arriving packets during this period to make sure the queue is not empty; thus, the total arriving traffic during this period $L \times 1500 \times 8 + \lambda \times T$ has to be larger than what the bottleneck link can transmit $C \times T$, where $\lambda$ is the cross traffic arrival rate. Solving the equation, we have

$$\Delta_i < L \times 1500 \times 8(\frac{1}{C-\lambda} - \frac{1}{C}) \quad (3)$$

Note that we ignored the 64B probe packet in the derivation above as an approximation.

Given a level of cross traffic $\lambda$, equation 3 shows how many load packets are needed for a given send gap $\Delta_i$. For example, when $\lambda = C/2$ (half the capacity), $\Delta_i$ should be less than the time to send the $L$ load packets. In our experiments, we have found that $\Delta_i = 100us$ and $L = 8$ work well. Even though, in theory, if $\lambda$ falls below $C/2$, the measurement may not accurately detect the bandwidth, we have found that LSP works well even when $\lambda$ is small.

LSP does not generate significant measurement traffic to overwhelm the network. With $L = 8$, LSP generates roughly 4 times the traffic Spruce would generate, assuming other parameters are equal (e.g., $\tau$). Using the same $\tau$ as Spruce, LSP would generate at most 960Kb/s of measurement traffic, which is much smaller than the link capacity in a cloud data center (e.g., 1Gbps). This is also smaller than other PRM based tools, e.g., Pathload[12] which generates 2.5MB to 10MB of probe traffic per measurement. Although Pathchirp [27], another PRM based tool, may have a comparable traffic load, because it shrinks multiple packet trains in Pathload into one, its accuracy is even worst than Pathload due to the timing errors at high speed.

## 4.3 Implementation details

LSP is implemented as two user-level sender and receiver programs on Linux. When it is time to send one burst of measurement packets, the sender program goes into a tight polling loop, not releasing the processor voluntarily until all packets in the burst have been transmitted. Using the current default parameter ($L = 8$, $\Delta_i = 100us$, $C = 1Gbps$), the sender program would be in the tight loop for roughly 200us, small enough to neither cause problems for other running applications nor tax the CPU heavily.

If the sender's network interface speed is high than $C$ – the bottleneck link capacity, the sender program would pace the load packets at a transmission rate of $C$. In all our experiments (both inside our data center and in a cloud), all links, as well as the network interface cards, are at 1Gbps. Thus, the sender sends the load packets as soon as possible.

In order to make sure the first probe packet is only sent after all load packets have been transmitted, we set the send buffer to be much smaller than 1500B using the SO_SNDBUF socket option before sending the first probe packet. This ensures that the send buffer is clear of the load packets when the first probe packet is sent. Note that we cannot simply time the sending of the first probe packet from the time we sent the first load packet, because we would not know exactly when the first load packet is sent if there are already packets in the send buffer from other applications. We record the time when the first probe packet is sent, then wait for $\Delta_i$ longer before sending the second probe packet.

On the receiving side, we set the SO_TIMESTAMP socket option so that the receiving kernel would timestamp each received packet. The receiver calculates $\Delta_o$ from the timestamp, then use equation 2 to estimate the bandwidth. The receiver averages over a sliding window of 100 individual samples to derive the bandwidth estimation.

## 4.4 Experimental evaluation

To evaluate the proposed bandwidth estimation technique, we conducted extensive experiments, both inside a controlled data center and inside a cloud provider. We compare LSP with Pathload[12], a tool based on the PRM model, and with Spruce [32], a tool based on the PGM model. We choose Pathload and Spruce not only because they cover both models, but also because they both perform well in lower speed networks.

We first setup an evaluation environment in our own data center as shown in Fig. 8. We setup two Cisco routers, one Catalyst 3750G and the other Catalyst 4510, and connect them on their GE interfaces. There are four servers connected to the two routers, all are HP xw4400 workstations with Intel Core2 Duo 2.4 GHz processors and 4GB of memory. The HP workstations are fast enough that they can send packets as fast as the network interface can handle; thus, no extra spacing would be created when pacing load and probe packets. Two HP workstations are used as the source and destination for network probing respectively, i.e., they run the bandwidth estimation tools such as LSP and Spruce. Two other HP workstations are used as the source and sink for the traffic generators.

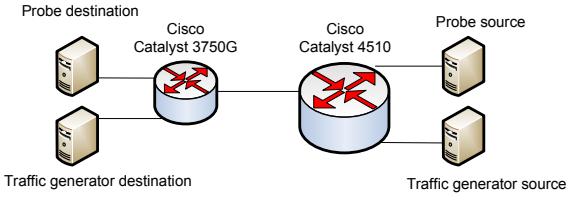


**Figure 8: Experiment setup in our internal data center.**

We use two traffic generators to generate cross traffic. One is Netperf [21]. To send traffic at a specific rate, we enable the `interval` feature on Netperf and we send a burst of

| cross traffic | 250Mbps | 500Mbps | 750Mbps |
|---|---|---|---|
| low | 785.45 | 396.46 | 344.78 |
| high | 1070.31 | 846.83 | 634.17 |

**Table 1: Average Pathload high and low estimates over all measurements for each cross traffic rate.**

packets every millisecond – the smallest granularity in Netperf. The size of the burst controls the average rate of traffic generated. We also use a Poisson traffic generator [23], but we do not see differences in our measurement results. For brevity, we only report the results generated by using Netperf as the traffic generator.

Using Netperf, we generate increasingly higher traffic, first 250Mbps, then 500Mbps, then 750Mbps, and we then run the bandwidth estimation tools repeatedly to collect the results that they are reporting.

Fig. 9 shows the results for Pathload. Both the high and low estimates given by Pathload vary wildly even though the cross traffic does not vary in each segment. In addition, the gaps between the high and low estimates are generally very high, giving no clear indication what the real available bandwidth is. Due to the wildly varying estimates, it is hard to see the overall trend in Fig. 9. So we take an average of both high and low estimates in each segment and show the result in Table 1. Even after averaging, the gap between high and low ranges from 200Mbps to more than 400Mbps. For the 750Mbps and 250Mbps segments, the average low is actually higher than the actual available bandwidth. We believe Pathload's inaccuracy could be attributed to timing errors at high speed.

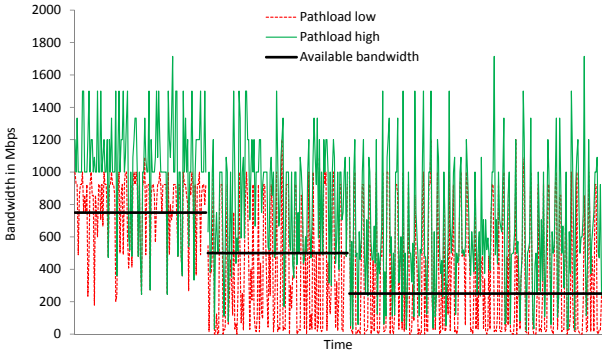


**Figure 9: Pathload performance in our internal data center.**

Fig. 10 shows the results for Spruce. Even though the bandwidth estimate fluctuates over time even in the same segment, its range of fluctuation is much smaller than the gap reported by Pathload. Unfortunately, Spruce appears to consistently over-estimate the available bandwidth. On average, it over-estimates by about 110Mbps for the 750Mbps segment, by 180Mbps for the 500Mbps segment, and by 160Mbps for the 250Mbps segment.

The results for LSP are shown in Fig. 11. Even though its estimates also fluctuate, their averages are much closer to the real available bandwidth. Its magnitude of fluctuation

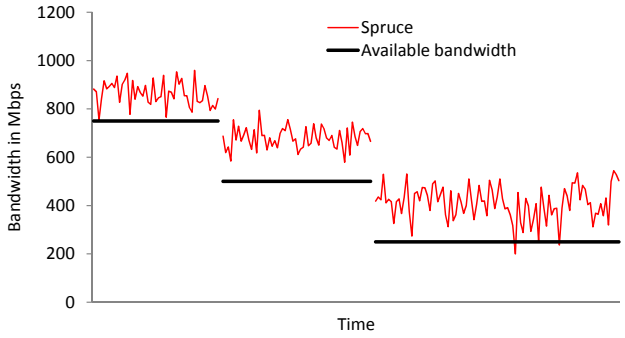


**Figure 10: Spruce performance in our internal data center.**

is similar to that of Spruce's, and it is less than 100Mbps from the real available bandwidth in all cases.

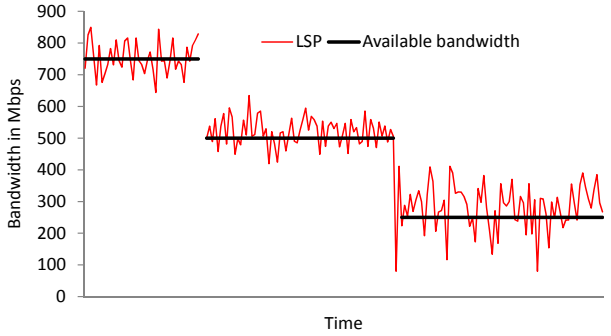


**Figure 11: LSP performance in our internal data center.**

We also evaluate LSP and compare it against Pathload and Spruce in a cloud data center. We launch two VMs in one subnet, and two other VMs in other subnets. One of the two VMs in the first subnet is used as the source for running the measurement tool, and the other is used to generate cross traffic. Being a public infrastructure, the cloud data center does not expose any network statistics. For example, all MIB access to the routers is disabled. Since we do not know the actual available bandwidth, we adopt the D-Test evaluation methodology used in [32], which is based on relative errors.

The D-Test has two phases. In the first phase, the tool estimates the available bandwidth. Let us denote this estimate by $M_1$. In the second phase, we inject a cross traffic at rate of $0.5 \times M_1$, and perform a bandwidth estimation at the same time. Let us denote the second measurement result to be $M_2$. Assuming other background traffic does not change much between the two phases (the two phases are run close together to minimize impact), the tool should report $0.5 \times M_1$. We define *Relative Error* to measure how much off the second measurement is from the ideal result:

$$RelativeError = \frac{M_2 - 0.5 \times M_1}{0.5 \times M_1} \quad (4)$$

We plot the Relative Error for Pathload, Spruce and LSP in Fig. 12 when we run the D-Test in the cloud data center

with them. Fig. 12 shows the cumulative distribution function (CDF), which should be a step function at 0 ideally. For Pathload, we take an average of the high and low estimate and use the average to compute the Relative Error.

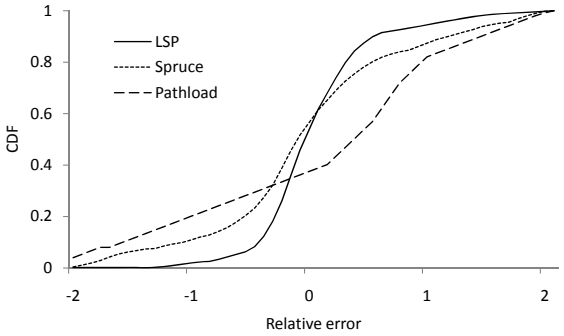


**Figure 12: Relative error for Pathload, Spruce and LSP in a cloud data center.**

Although none of the tools behaves ideally, LSP is closest to a step function. Spruce performs fairly well too. As we have seen in our internal data center, Spruce tracks the bandwidth change fairly well except that it consistently overestimates, so it is no surprise that it performs well in the D-Test. Pathload is the furthest away from being a step function. As we have seen in our internal data center, Pathload sometimes produces wildly varying estimates; thus, it is no surprise that the D-test result would not track the ideal value well.

# 5. CLOUD PERFORMANCE MONITORING

Even though we have presented the avoidance mechanism in the context of a DOS attack, in reality, it is difficult, if not impossible, to distinguish an intentional DOS attack from one or more bandwidth hungry applications. Our proposed avoidance mechanism would be useful even if no DOS attack is underway, so that the application could adapt to the changing network condition.

Using our LSP tool, we conducted a 24-hour measurement of a cloud provider. This cloud provider has a 1Gbps uplink for each subnet. We launch two VMs in two different subnets in the cloud, and we use LSP to measure the available bandwidth between the two VMs. The result is shown in Fig. 13. For the most part, the bandwidth is plentiful at around 700Mbps, which means that other applications in the subnet are using roughly 300Mbps of bandwidth. However, for an hour of so, the available bandwidth decreased dramatically, sometimes dipping below 200Mbps. Although not totally starved, the limited bandwidth could have a direct impact on application performance. Depending on the application, it may be desirable to migrate to a different subnet using our avoidance mechanism to improve network performance.

# 6. CONCLUSION

The large-scale, public nature of a cloud data center brings additional challenges when designing the underlying network infrastructure. Today, these networks are typically grossly under-provisioning, even below the recommended under-provisioning ratio. By exploiting the under-provisioning,

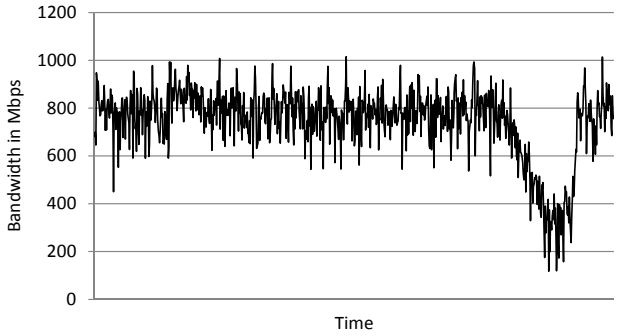


**Figure 13: Available bandwidth probing in a cloud data center for a 24 hour period.**

we show that an adversary can bring down a subnet in the cloud data center with minimal cost. Even though the data center managers can put in measures to counter large-scale attacks, smaller-scale attack is still possible since an attack differs from a normal bandwidth-hungry application only by intent. We propose a dynamic migration architecture, leveraging the dynamic provisioning capability of a cloud, to detect and avoid similar kind of DOS attacks. We also propose a novel available bandwidth estimation tool that works accurately and reliably in high-speed networks. Our work points out that we need a better data center network architecture to support cloud data centers in the future.

# 7. REFERENCES


[1] Al-Fares, M., Loukissas, A., and Vahdat, A. A scalable, commodity data center network architecture. In *Proc. SIGCOMM* (2008).

[2] Carter, R. L., and Crovella, M. E. Measuring bottleneck link speed in packet-switched networks. Tech. rep., Performance Evaluation, 1996.

[3] Chen, L.-J., Sun, T., Wang, B.-C., Sanadidi, M., and Gerla, M. Pbprobe: A capacity estimation tool for high speed networks. *Computer Communications 31*, 17 (2008), 3883 – 3893.

[4] Dovrolis, C., Ramanathan, P., and Moore, D. What do packet dispersion techniques measure? In *In Proceedings of IEEE INFOCOM* (2001), pp. 905–914.

[5] Greenberg, A., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. Vl2: A scalable and flexible data center network. In *Proc. SIGCOMM* (2009).

[6] Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Tian, C., Zhang, Y., and Lu, S. Bcube: A high performance, server-centric network architecture for modular data centers. In *Proc. SIGCOMM* (2009).

[7] Guo, C., Wu, H., Tan, K., Shiy, L., Zhang, Y., and Luz, S. Dcell: A scalable and fault-tolerant network structure for data centers. In *Proc. SIGCOMM* (2008).

[8] Hopps, C. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992 (Informational), Nov. 2000.

[9] Hu, N., Li, L. E., Mao, Z. M., Steenkiste, P., and Wang, J. Locating internet bottlenecks: algorithms, measurements, and implications. In *SIGCOMM '04: Proceedings of the 2004 conference on*

*Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2004), ACM, pp. 41–54.

[10] Hu, N., Li, L. E., Mao, Z. M., Steenkiste, P., and Wang, J. A measurement study of internet bottlenecks. In *Proc. IEEE Infocom* (2005).

[11] Hu, N., Member, S., Steenkiste, P., and Member, S. Evaluation and characterization of available bandwidth probing techniques. *IEEE Journal on Selected Areas in Communications 21* (2003), 879–894.

[12] Jain, M., and Dovrolis, C. End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput. *IEEE/ACM Trans. Netw. 11*, 4 (2003), 537–549.

[13] Jin, G., and Tierney, B. L. System capability effects on algorithms for network bandwidth measurement. In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2003), ACM, pp. 27–38.

[14] Kandula, S., Katabi, D., Jacob, M., and Berger, A. Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds. In *Proc. NSDI* (2005).

[15] Keshav, S. A control-theoretic approach to flow control. In *ACM SIGCOMM* (1991), pp. 3–15.

[16] U.s. eyes n. korea for âĂŸmassiveâĂŹ cyber attacks. http://www.msnbc.msn.com/id/31789294.

[17] Lai, K., and Baker, M. Nettimer: a tool for measuring bottleneck link, bandwidth. In *USITS'01: Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems* (Berkeley, CA, USA, 2001), USENIX Association, pp. 11–11.

[18] Man, C. L. T., Hasegawa, G., and Murata, M. Inline bandwidth measurement techniques for gigabit networks. *Int. J. Internet Protoc. Technol. 3*, 2 (2008), 81–94.

[19] Melander, B., Bjorkman, M., and Gunninberg, P. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Proc. GlobeCOM* (2000).

[20] Mysore, R. N., Pamboris, A., Farrington, N., Huang, N., , Miri, P., Radhakrishnan, S., Subramanya, V., and Vahdat, A. Portland: A scalable fault-tolerant layer 2 data center network fabric. In *Proc. SIGCOMM* (2009).

[21] Netperf. http://www.netperf.org/netperf/.

[22] Cisco data center infrastructure 2.5 design guide. http://www.cisco.com/univercd/cc/td/doc/solution/dcidg21.pdf.

[23] Poisson traffic generator. http://www.spin.rice.edu/Software/poisson_gen/.

[24] Prasad, R. S., Murray, M., Dovrolis, C., and Claffy, K. Bandwidth estimation: Metrics, measurement techniques, and tools. *IEEE Network 17* (2003), 27–35.

[25] Rackspace ipo will fund new data centers. http://www.datacenterknowledge.com/archives/2008/04/29/rackspace-ipo-will-fund-new-data-centers.

[26] Ribeiro, V., Coates, M., Riedi, R., Sarvotham, S., Hendricks, B., and Baraniuk, R. Multifractal cross-traffic estimation. In *Proc. ITC Specialist Seminar on IP Trac Measurement, Modeling, and Management* (2000), pp. 15–1.

[27] Ribeiro, V. J., Riedi, R. H., Baraniuk, R. G., Navratil, J., and Cottrell, L. Pathchirp: Efficient available bandwidth estimation for network paths. In *Passive and Active Measurement Workshop* (La Jolla, CA, Apr. 2003).

[28] Ristenpart, T., Tromer, E., Shacham, H., and Savage, S. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proc. 16th ACM conference on Computer and communications security table of contents* (2009).

[29] Savage, S., Wetherall, D., Karlin, A., and Anderson, T. Practical network support for ip traceback. In *Proc. SIGCOMM* (2000).

[30] Sommers, J., Barford, P., and Willinger, W. A proposed framework for calibration of available bandwidth estimation tools. In *ISCC '06: Proceedings of the 11th IEEE Symposium on Computers and Communications* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 709–718.

[31] Staniford, S., Paxson, V., and Weaver., N. How to own the internet in your spare time. In *Proc. USENIX Security* (2002).

[32] Strauss, J., Katabi, D., and Kaashoek, F. A measurement study of available bandwidth estimation tools. In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2003), ACM, pp. 39–44.

[33] Yaar, A., Perrig, A., and Song, D. Fit: Fast internet traceback. In *Proc. IEEE Infocom* (March 2005).