CloudPolice: Taking Access Control out of the Network

Lucian Popa
U.C. Berkeley / ICSI

Minlan Yu
Princeton Univ.

Steven Y. Ko
Princeton Univ.

Sylvia Ratnasamy
Intel Labs Berkeley

Ion Stoica
U.C. Berkeley

ABSTRACT

Cloud computing environments impose new challenges on access control techniques due to multi-tenancy, the growing scale and dynamicity of hosts within the cloud infrastructure, and the increasing diversity of cloud network architectures. The majority of existing access control techniques were originally designed for enterprise environments that do not share these challenges and, as such, are poorly suited for cloud environments. In this paper, we argue that it is both sufficient and advantageous to implement access control *only* within the hypervisors at the end-hosts. We thus propose CloudPolice, a system that implements a hypervisor-based access control mechanism. We argue that, not only can CloudPolice support more sophisticated access control policies, it can do so in a manner that is simpler, more scalable and more robust than existing network-based techniques.

1. INTRODUCTION

Cloud computing brings the "pay as you go" model to data centers. In particular, the *Infrastructure as a Service* (IaaS) model allows clients to dynamically scale up/down to as many machines as needed inside the cloud.

A major hurdle to the widespread adoption of this model is security, as customers often want to export sensitive data and computation into the cloud. Threats arise not only from privacy leaks at the cloud operating company (outsourcing of data center management is in fact common) but also due to the multi-tenant nature of clouds. For this reason, *network-level access control* policies are a critical component for preserving security when migrating to the cloud computing model. For example, tenants would want their traffic to be, by default, isolated from all other tenants.

Today, access control in cloud environments is typically provided using techniques such as VLANs and firewalls. These techniques, however, were originally designed for enterprise environments and as such are ill suited to meet the challenges unique to cloud environments. Specifically, we argue that existing techniques are challenged by four key characteristics of emerging cloud environments: multi-tenancy, diversity in cloud network architectures, the large scale and the high dynamism of cloud infrastructure. We expand on each of these challenges in what follows.

Multi-tenancy introduces new requirements to access control as intra-cloud communication (*i.e.,* provider-tenant and tenant-tenant) is becoming more popular. For example, Amazon provides their tenants with services such as SimpleDB and Simple Queue Service (SQS); there are also tenants that provide services to other tenants, *e.g.,* mapreduce++ and desktop services [2]. The intra-cloud communication is likely to require new types of access control policies such as fair-sharing between tenants, and rate-limiting tenants as we will discuss later in this paper (§2). VLANs and firewalls do not offer the flexibility to implement these new types of policies.

Network-diversity is another new challenge for access control. The architecture of data centers has evolved significantly from that of the traditional enterprises and is currently in flux, with many new architectures being proposed [8, 15–17]. These new architectures typically employ multiple paths and require specific routing algorithms and address assignments. Therefore, they severely limit the applicability of current mechanisms such as VLANs and firewalls (as we discuss in §4).

Furthermore, today's clouds house tens of thousands of physical machines, and even more virtual machines that are constantly added and removed (AWS [3] reports over 100K VMs started per day). Current access control mechanisms were not designed to handle such scale and churn. For example, firewalls have problems scaling to large numbers of entries and coordinating access control across multiple firewalls is complex (as described in §4), while VLANs do not support dynamic configuration, are limited in scalability and complex to setup and configure [25, 28]. More generally, our observation is that as clouds scale to large numbers of users (AWS reports over 10K users), they will face many of the problems traditionally associated with the public Internet, including DoS attacks between cloud tenants. Such attacks are known to be very difficult to tackle [9, 10, 26, 27] but are not typically the concern of (internal) enterprise access control mechanisms.

Therefore, we believe that these challenges call for a new access control mechanism that provides three properties: (1)

Hotnets '10, October 20–21, 2010, Monterey, CA, USA.

flexibility in providing support for policies in multi-tenant environments such as tenant isolation, fair-sharing, and rate-limiting policies, (2) *network-independence* in decoupling access control from the network topology, routing and addressing, and (3) *scalability* in handling hundreds of thousands of machines and users.

In this paper, we argue that it is both sufficient and advantageous for access control to be implemented only at the end-hosts, within hypervisors. We propose CloudPolice, a new access control mechanism implemented in hypervisors that provides the above properties. Since hypervisors have full software programmability, CloudPolice can provide a broad class of access control polices needed for multi-tenancy. In addition, by embedding access control into the hypervisors, our solution is independent of the network architecture and we are avoiding needlessly tying the development of access control to that of specific network equipment and protocols.

For scalability, CloudPolice proposes a distributed solution, where hypervisors communicate with each other to install access control state. More specifically, CloudPolice uses a runtime approach inspired by network capabilities [26, 27] and push-back filters [9, 19]. Similarly to how destination end-hosts push blocking and rate-limiting filters into the network in these proposals, in CloudPolice hypervisors of destination VMs push blocking and rate-limiting filters to the hypervisors of the source VMs according to the access control policies of destinations. The reason CloudPolice is easy to deploy compared to the work on Internet capabilities is that in a cloud environment the data-center operator controls *both* end-points (source and destination) through the hypervisor, and therefore avoids complex in-network mechanisms to rate-limit/drop packets.

Next we define a set of network access policies desirable in clouds and then present the design of CloudPolice.

2. CLOUD ACCESS CONTROL POLICIES

In this section, we examine a few examples of access control (AC) policies that we believe are important for multi-tenant clouds; our focus is on intra-cloud traffic. Several of these policies bring new challenges to cloud policy support and are not supported by existing AC mechanisms or cloud provider APIs. From these examples we then derive a policy model and use it to guide the design of our proposal.

Note that the AC policies should have the ability to be defined at a finer granularity than per tenant, *e.g.*, for intra-tenant policies. In this paper, we refer to the security principals used in the AC policies as *groups*; this means that a common policy can be defined for all the VMs belonging to the same group and also that policies can be based on (source/destination) groups. A group can be viewed as similar to an AWS security group [3].

2.1 Example Types of Cloud AC Policies

Tenant Isolation: The simplest and most common type of AC policy is to block all traffic from other tenants (in particular groups) and this is the default policy in the current cloud environments. Isolation prevents hosts from being compromised and blocks DoS attacks if correctly implemented, *i.e.*, if the traffic from the other tenants is blocked at the source. Note that DoS attacks in the cloud environment can be easier to mount than in the Internet because attackers may not need to compromise hosts to create botnets, but can also simply pay for the attack hosts. Traffic isolation is traditionally implemented by VLANs, which, however, are not a good fit for the cloud environment (§4).

Inter-tenant Communication: We expect that the shared environment of cloud computing will enable users to offer each other services more easily than with traditional business models, due to the close coupling between the users' machines (*i.e.*, with small latency and large bandwidth). For example, real time advertising [4, 5] is a fast growing ad-paradigm that requires low latency between advertising providers and providers of web content (which are advertising consumers). Cloud computing offers the perfect environment for such a collaboration between tenants. This example requires the ability to communicate between the ad provider and the ad consumer (for ad bidding, ad retrieval) and to isolate the traffic from the other consumers to avoid DoS attacks.[1] Even this simple communication pattern is not well supported by the traditional enterprise mechanisms such as firewalls and VLANs in the cloud environment (§4).

Fair-Sharing among Tenants: Since multiple tenants may access the services offered by one tenant or by the cloud provider, the entity offering the service may want to implement bandwidth fair-sharing among the groups accessing the service. For example, tenants that have more machines or higher available bandwidth (*e.g.*, are better positioned in the network topology) should not be able to get better service nor impact the services available to other tenants more than their fair share. This is not a feature supported by traditional AC mechanisms, but we believe that it should be supported in cloud computing. Scenarios that show the importance of fair-sharing are storage and database services, *e.g.*, Amazon's SimpleDB and Simple Queue Service (SQS).

Rate Limiting Tenants: As a mechanism, rate-limiting is required by AC to implement the previously mentioned fair-sharing policies. But we argue that in clouds rate-limiting is also important as a policy. For example, in a cloud that charges for bandwidth usage, one tenant A may want to rate-limit on tenant B when accessing A's services. In this case, attackers can financially damage their victims by increasing the bandwidth usage of each VM being attacked; the "pay-as-you-go" pricing model will automatically charge the victim. Moreover, tenants and cloud providers may implement elastic services that automatically add more VMs if the demand increases (*e.g.*, AWS auto scaling). Thus, if there is a DoS attack, more VMs will be added automatically; the charge for the added VMs will fall to the victim.

[1] A mechanism for isolation that also protects against DoS attacks can be very important for services such as bidding and betting, where tenants can indirectly influence each other's bids / bets by creating large drop packet rates for those services.

Allowing Locally Initiated Connections: A common envisioned usage of cloud computing is for virtual desktops [2]. The machines hosting the virtual desktops could use various services also located inside the cloud (to take advantage of the low latencies and large bandwidths), such as SMTP, HR database, EPMAP, a service provided by Facebook, *etc*. Different users with different security credentials can log in the virtual desktop hosts. In these circumstances, it is hard to know in advance what cloud services will be accessed by the virtual desktops. Therefore, it is desired to allow incoming traffic from all of the other groups (of the same tenant or not), but only in response to connections initiated by the virtual desktops. This behavior is typically implemented by stateful firewalls and is not available in current cloud provider APIs.

2.2 Policy Model

From the previous examples, we abstract a general AC policy model to be supported by cloud providers.

We use the following definition for an AC policy: an ordered list of *rules* of the form: `if` *`Condition`* `then` *`Action`*. The *`Condition`* is represented by a logical expression containing one or several predicates connected through logical operators. Each predicate can be defined based on: (1) the group of the sender/receiver, (2) the packet header fields (*e.g.*, the five tuple), (3) the current time and (4) state recording the history of past traffic. The predicate is an arithmetic expression using comparison operators ($=, !=, >, <$) and constants, *e.g.*, $port = 80$. The *`Action`* can be: (1) allow, (2) block and (3) rate limit traffic.[2]

We refer to conditions that are not based on recorded state about past traffic as *stateless*, and the remaining as *stateful*. The form of the recorded state could be arbitrary; from our examples, we propose a basic set of four types of state: (1) Incoming Flows: the number, arrival rate and duration of flows incoming from a given source host/group, (2) Incoming Bytes: the number and rate of received bytes from a given source host/group, (3) Outgoing Flows: the locally initiated flows, (4) Rejected flows: the number of locally rejected TCP connections typically indicating port scanning.

Actions can be applied at the granularity of a flow, machine or group, *e.g.*, limit all flows belonging to a group to a total maximum rate. Blocking/Allowing traffic at a granularity larger than a flow can be a useful hint for the AC policy engine to reduce the necessary state, *i.e.*, not keep per flow blocking state, but rather block entire hosts. Rate-limiting can be specified using an absolute rate value or a relative weight. Due to space constraints, we do not elaborate on conflict resolution between rules; we note, however, that conditions with per-flow granularity should typically only trigger per-flow actions (rather than per-VM or per-group).

[2]The described policies do not offer the ability to impose middleboxes as allowed by, for *e.g.*, the policy framework described in [11]. Due to space constraints and since these policies may be less relevant to clouds, we do not discuss them here (see also §5).

3. CloudPolice

There are several design options for implementing the policy model and policies discussed in §2. In CloudPolice, we choose a hypervisor-based approach because hypervisors are (1) *trusted*, (2) *network-independent*, (3) *close to VMs* (and thus can block unwanted traffic before reaching the network), and (4) have full *software programmability*. Next, we discuss the rationale behind our design and then present details of our specific mechanism and analyze its security.

3.1 Design Space

Although the AC policies are defined by the destination, the policies should be enforced close to the *source* rather than at the destination. This prevents unauthorized traffic from abusing the network (*e.g.*, causing congestion, mounting DoS attacks, *etc.*).

A naïve solution is to install all policies and the entire mapping between active VMs and groups in all hypervisors. In this way, the source hypervisor can directly apply the policy of the destination to all the flows sent by its hosted VMs. Unfortunately, this solution scales poorly due to the high churn rate expected for the active VMs. For example, AWS reports about 100K new VMs started per day; in a 100K server infrastructure [3, 6, 14], this translates into more than 100K update messages sent per second on average (peak update rates would likely be much higher).

Another extreme solution is to distribute no policies to hypervisors, but use a centralized repository for policies and group membership. Hypervisors then consult this repository for each new flow and possibly cache the AC policies. However, the centralized resolution service is likely to represent a tempting target for DoS attacks. Moreover, the centralized service has to sustain very high availability and low response times. For example, assuming an average of 10 new flows per second per server [24], and a 100K server cloud, the centralized service would need to process 1M flows per second on average (again, with higher peak rates expected).[3]

For these reasons, we propose a *distributed* solution. In CloudPolice, hypervisors need only know the policies of their hosted VMs and not the policies of any other group in the cloud, nor the group membership. In order to learn which flows should be allowed, blocked or rate limited, CloudPolice uses a *runtime* approach in which hypervisors communicate with one another using a secure channel as we describe next. In designing this technique, we have drawn inspiration from previous approaches on preventing DoS attacks in the Internet such as push-back filters [9, 10] and network capabilities [26, 27].

3.2 CloudPolice Overview

Fig. 1 shows a high level description of the operation of CloudPolice. When a new flow is initiated by a VM, the source hypervisor sends a *control packet* specifying the security group to which the source VM belongs to; this packet

[3]Note also that caching may be ineffective since the traffic is expected to be randomly distributed [20, 24] and since policies and VM locations change in time.

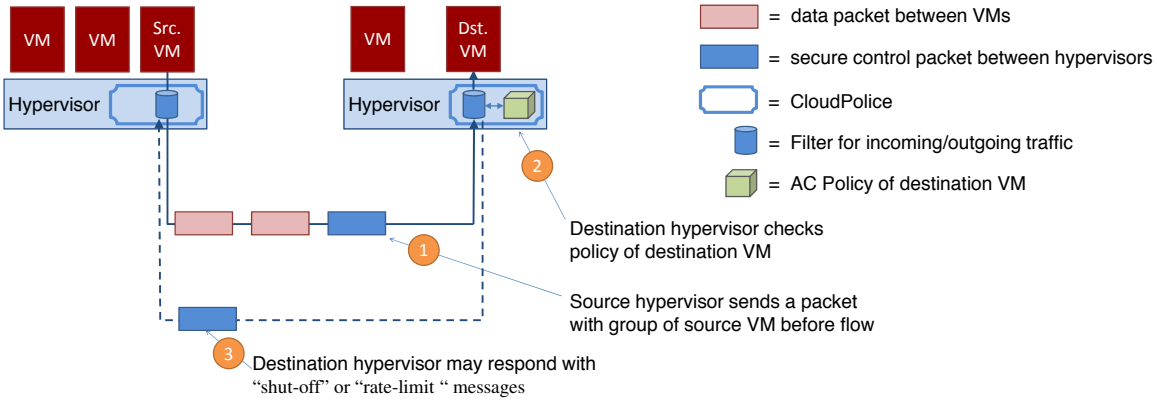

Figure 1: CloudPolice Overview

is sent before the the packets belonging to the flow (step 1). When the destination hypervisor receives such a control packet, it checks the policy for the group of the destination VM (step 2). If the policy allows the traffic, the destination hypervisor creates state for this flow; subsequent packets will be forwarded up to the destination VM by using this entry. If the traffic is not allowed or should be rate limited, the destination hypervisor will send a control packet back to the source hypervisor to block or rate-limit the flow or the VM (step 3). By default, VMs are blocked if the policy contains no rule for that traffic.

3.3 Detailed Design

Soft State: CloudPolice maintains soft state (*i.e.*, removed after expiration) to enforce the policy actions (block, remove and rate-limit). After the expiration of the soft state, the entire process for setting up the state is restarted. Soft state makes it easier to support VM migration and handle packet losses. In our current design, revocation is also handled through the expiration of the soft state. However, explicit state invalidation on policy updates could be implemented, by using control packets between hypervisors in a similar fashion.

Control Packets: There are three types of control packets sent between hypervisors: (a) sent by a source hypervisor (step 1 in Fig. 1), (b) sent by a destination hypervisor to block or rate limit the traffic (step 3 in Fig. 1) and (c) sent by a destination hypervisor to query the source hypervisor, as we will describe later. For case (a), the packet contains the header of the first data packet of that flow, which is used for the destination hypervisor to check its policy. For case (b), the packet specifies if the action should be applied per flow or per VM and the value of the rate limit; to block packets the rate limit is set to zero. Control packets are distinguished from the rest of the data packets by using a special transport protocol number in the IP header (*e.g.*, the protocol number 254, reserved for testing).

Lost/Reordered Control Packets: First, assume the packet sent by the source hypervisor containing the group of the source VM (type (a) above) is lost. If a destination hypervisor receives a flow for which it has no entry, it sets up a querying state for the flow and sends a "querying" control message to the source hypervisor (type (c) aforementioned). At the receipt of the querying message, the source hypervisor resends the type (a) control packet. There is a timer associated with the querying state and a new request is made to the source hypervisor when it expires, *i.e.*, in the case the querying control packet is lost. Second, assume a type (b) control packet, sent by the destination hypervisor, is lost and the destination receives unwanted traffic. In this case, the destination hypervisor sets up a short term state to block the incoming traffic, waiting for the shut-off message (in Fig. 1) to arrive at the source hypervisor. If packets are still received after a short timeout, the destination hypervisor sends back another shut-off packet. In case of rate limiting, the destination monitors the incoming rate and if the limit is not respected, it sends back a new control packet to the source hypervisor.

Policy Updates: We envision two models for distributing and updating policies to hypervisors. In the first model, the cloud provider uploads the group policy to the hypervisor at the VM startup and updates it at all the group members when the policy changes. Since policy changes should be infrequent, we do not expect this service to be a burden for the cloud provider. In a second model, VMs can directly communicate their policies to hypervisors. This model does not require a policy management service from the cloud provider but requires and additional API in both the hypervisor and the VMs.

Global Policies: Some policies may require hypervisors to have knowledge about all the VMs belonging to a group. For example, a policy might specify that the aggregate traffic from all the VMs of group A to be rate limited when accessing the VMs of group B. Such a policy requires communication between the hypervisors hosting B' VMs since the decision should be taken using an aggregate state; this requires hypervisors to know all the members of the group for their hosted VMs. Due to space constraints we do not discuss such policies in this paper, but we note that techniques such as in [22] can be applied.

3.4 Security Analysis

We now analyze possible attacks originated in-cloud by either malicious tenants or compromised VMs. We consider three classes of attacks: (1) bypass the CloudPolice policies and reach a destination VM with unauthorized traffic, (2) mount a DoS attack using unauthorized traffic, and (3) mount DoS attacks by using authorized traffic. Next we discuss how CloudPolice addresses each of them.

Bypass CloudPolice Policies: There are two potential cases in which a VM could receive undesired packets that bypass its policies. The first case is obviously when the hosting hypervisor is compromised. However, when a hypervisor is compromised, access control is not the main concern. For example, through compromised hypervisors attackers might directly corrupt and spoof on private data of other tenants on the same machine. Therefore, we do not consider compromised hypervisors in our threat model. The second case is when a hypervisor receives fake information about the sender. This could occur even without compromised hypervisors, *e.g.,* if VMs could inject spurious control packets into the network. To prevent this case, CloudPolice requires hypervisors and ingress routers to drop control packets incoming from VMs or external traffic; thus, only hypervisors can send control packets.[4]

DoS with unauthorized traffic: Attackers can attempt to DoS a victim V by sending it unauthorized traffic. To enable unauthorized traffic to reach V, two types of attacks can be attempted. First, an attacker VM X can try to prevent control packets from reaching its hypervisor H_X. In this way, H_X would not know that X's traffic should be blocked. To mount this attack, X's group can flood H_X (possibly with authorized traffic), and in this way induce losses of control packets. A simple fix for this attack is to prioritize control packets in switches (most switches today support QoS). But note that even in the absence of traffic prioritization, with a loss rate of 50%, only two control messages are required to block one VM. In an second attack, X tries to exhaust the filters available at hypervisors (either H_X or V's hypervisor H_V). In the unlikely event for this to occur (since we expect CloudPolice to store a small amount of memory and memory to be abundant), CloudPolice can aggregate the per-flow state into per-VM state and block the VMs that have a significant fraction of their flows blocked.

DoS with authorized traffic: AC policies ensure that only authorized traffic will compete for the bandwidth. However, AC does not protect against floods of authorized traffic between colluders that have access to a shared link with the victim (a well known limitation of network capabilities [27]). Unfortunately, due to the virtualized environment, this situation is much more common in clouds than in the Internet. For example, an attacker can attempt to DoS a VM V by sending a lot of authorized traffic to VM X located on the same physical machine with V.

[4]For this reason, CloudPolice does not require packet encryption or other security protocols to protect control packets, and thus can generate control packets with low overhead.

Preventing DoS attacks in the above scenario requires *performance isolation* (PI), *i.e.,* fair bandwidth sharing between VMs, in addition to AC. Note that performance isolation without AC is similarly unable to prevent DoS attacks. Due to space constraints, we do not discuss PI extensively in this paper. However, we point out that CloudPolice can also be used to implement PI and can prevent DoS attacks, while the other PI proposals that we are aware of do not prevent DoS attacks, as we briefly discuss next.

CloudPolice can provide PI by rate limiting VMs in case of congestion with the same mechanism used for AC. For example, if the hypervisor runs N VMs, the VMs sending traffic to each one of these N VMs are together rate limited to $1/N$ (with equal shares or a more sophisticated distribution algorithm). In other words, the bandwidth is evenly shared between the *destination VMs*.[5] In this case, a legitimate VM cannot be DoSed with authorized traffic sent to another VM located on the same physical machine.

On the other hand, existing approaches to implement PI do not block DoS attacks because the sharing is based on the *source VMs* rather than the destination ones. One PI solution is to statically share the bandwidth, *i.e.,* if there are N VMs on each machine, each can send up to $1/N$ of the available bandwidth. This approach cannot prevent DoS attacks since the attacker can use an arbitrary number of VMs to send traffic. In another example, (dynamic) fair sharing per source VM has been recently proposed to be implemented in clouds [23]. This approach is more effective in mitigating DoS attacks compared to static bandwidth sharing, but still suffers from the problem that the attacker can use more source VMs than the legitimate traffic, and thus reduce the bandwidth of the destination VMs proportionally.

3.5 Feasibility

In this section we discuss the feasibility of implementing CloudPolice in hypervisors. We are working on implementing a prototype CloudPolice using Open VSwitch [21].

The main concern with implementing AC inside the hypervisor is the ability of the hypervisor to match the line speed performance requirements. In particular, there are two types of concerns: (1) the ability to maintain and act on per-flow state (the fast path) and (2) the ability to install new state with low enough latencies (the slow path). Next, we discuss each of these in turn.

First, note that Open VSwitch has been shown to scale at the full speed of a linux bridge [21]. Moreover, recent software routers have shown the ability to forward packets at around 10Gbps with a single CPU core [12, 18], even when using routing tables reaching 300,000 entries [12]. Thus, we believe that we can achieve a reasonable baseline forwarding performance with software. However, we recognize that CloudPolice has extra processing performed per flow, and we leave this evaluation for future work.

Second, creating state for each new incoming flow needs to be done fast, before the packets of that flow are dropped

[5]This approach does not guarantee the fair sharing of a congested link, but guarantees fair sharing of the bandwidth at the destination.

from the input buffers. Since the number of hosted VMs is typically small, we expect the policy lookup to be very small and the state setup latency to be dominated by the complexity of the policy. For stateless policies we do not expect any significant latency since these are simple packet matching rules. Even for the several types of stateful policies that we have proposed in §2, we expect low latencies since they mainly involve counters per hosted VM.

4. RELATED WORK

A traditional mechanism to implement AC is through the use of VLANs. However, VLANs have several limitations. First, since VLANs couple access control and switching, they cannot be applied to new network topologies such as [8, 16,17], or to topologies that use L3 routing instead of switching. In addition, VLANs have much overhead both in spanning tree creation/maintenance and in switching between VLANs (which typically requires L3 routing) [13, 25]. VLANs are also limited by the number of hosts in one VLAN and the number of VLANs in a network [28]. Furthermore, VLANs do not offer the flexible policies proposed in this paper.

Firewalls can also be used to block unwanted traffic at the source, *e.g.*, by placing them at the first-hop switches. However, this approach presents a major maintenance overhead, since every time a destination changes its policy, all the firewalls at all possible sources need to be updated. Moreover, in order to support group-based policies, firewalls either need to create an entry for each VM in the group, or group the VMs by the same IP prefixes and create an entry for each prefix. Neither of the solutions is desirable because the former faces a scalability limit, and the latter makes VM address management unnecessarily complex.

Centralized controllers such as OpenFlow and Ethane [7, 11] can be used to provide AC. However, these approaches are network-dependent, *i.e.*, they require changes to the switching hardware. Open VSwitch [21] can achieve network-independence, but it still requires a centralized controller. Thus, Open VSwitch inherits all the drawbacks of centralized approaches – the centralized controller could be a scaling bottleneck, and a potential attraction for DoS attacks from the tenants. Moreover, the OpenFlow API is designed for switches, but much richer policies can (and should) be implemented in hypervisors. VL2 [15] also discusses a mechanism to make address assignment independent of the underlying topology, but still makes use of a centralized service for AC. Thus, it suffers from the same drawbacks listed above for Open VSwitch.

AWS [3] offers a limited set of AC policies [1] such as isolation and on/off access between groups. We are not aware of public information describing AWS's implementation.

5. DISCUSSION AND FUTURE WORK

In this paper we have discussed the design of CloudPolice, a distributed access control mechanism implemented in hypervisors. CloudPolice is designed to meet the need of access control in the era of cloud computing by providing flexibility for supporting policies in multi-tenant environments, network-independence that decouples access control from the network, and scalability to handle hundreds of thousands servers and users. Our work could be a first step towards finding a common access control API and mechanism that can be used across multiple cloud providers.

There are three directions we are pursuing. First, we are working on implementing a CloudPolice prototype and on evaluating its performance. We focus on reducing the cost that this solution would impose in terms of the resources used at servers. Second, we intend to implement and evaluate a performance isolation framework based on CloudPolice. Finally, we want to investigate a class of *dynamic* policies that are controlled at runtime by VMs based on application-level semantics. For example, source VMs could provide access capabilities at runtime (*e.g.*, obtained through external websites) allowing them to access certain groups.

6. REFERENCES

[1] Amazon security white paper. `http://awsmedia.s3.amazonaws.com/pdf/AWS_Security_Whitepaper.pdf`.
[2] Amazon virtual desktop services. `http://desktop-client-for-amazon-s3.qarchive.org`.
[3] Amazon web services. `http://aws.amazon.com`.
[4] Appnexus real-time ad platform. `http://www.appnexus.com`.
[5] Instant Ads Set the Pace on the Web. *The New York Times*. `http://www.nytimes.com/2010/03/12/business/media/12adco.html?emc=eta1`.
[6] Microsoft Azure. `http://www.microsoft.com/windowsazure`.
[7] The OpenFlow Switch Consortium: www.openflowswitch.org.
[8] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*. ACM, 2008.
[9] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol (AIP). In *ACM SIGCOMM*, 2008.
[10] K. Argyraki and D. R. Cheriton. Active Internet traffic filtering: Real-time response to Denial-of-Service attacks. In *USENIX Annual Tech. Conf.*, 2005.
[11] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In *ACM SIGCOMM*, 2007.
[12] M. Dobrescu, N. Egi, K. Argyraki, B.-g. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *ACM SOSP*, 2009.
[13] P. Garimella, Y.-W. E. Sung, N. Zhang, and S. Rao. Characterizing VLAN usage in an operational network. *Workshop on Internet Network Management*, 2007.
[14] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The Cost of a Cloud: Research Problems in Data Center Networks. *Comput. Commun. Rev.*, 2009.
[15] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. *ACM SIGCOMM*, August 17 - 21 2009.
[16] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. *ACM SIGCOMM*, 2009.
[17] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. Dcell: A Scalable and Fault-tolerant Network Structure for Data Centers. In *SIGCOMM*, 2008.
[18] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: a GPU-Accelerated Software Router. In *ACM SIGCOMM*, 2010.
[19] J. Ioannidis and S. M. Bellovin. Implementing Pushback: Router-Based Defense Against DDoS Attacks. In *NDDS*, 2002.
[20] S. Kandula, J. Padhye, and P. Bahl. Flyways To De-Congest Data Center Networks. In *HotNets*, 2009.
[21] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker. Extending Networking into the Virtualization Layer. In *HotNets*, 2009.
[22] B. Raghavan and A. C. Snoeren. A System for Authenticated Policy-Compliant Routing. In *ACM SIGCOMM*, 2004.
[23] A. Shieh, S. Kandula, A. Greenberg, and C. Kim. Seawall: Performance Isolation for Cloud Datacenter Networks. *HotCloud*, 2010.
[24] Srikanth K and Sudipta Sengupta and Albert Greenberg and Parveen Patel and Ronnie Chaiken. The Nature of Datacenter Traffic: Measurements & Analysis. In *Internet Measurement Conference*. ACM, November 2009.
[25] Y.-W. E. Sung, S. Rao, G. Xie, and D. Maltz. Towards Systematic Design of Enterprise Networks. In *ACM CoNEXT*, 2008.
[26] A. Yaar, A. Perrig, and D. Song. SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks. In *IEEE Symp. on Security and Priv.*, 2004.
[27] X. Yang, D. J. Wetherall, and T. Anderson. A DoS-limiting Network Architecture. In *ACM SIGCOMM*, 2005.
[28] M. Yu, X. Sun, N. Feamster, S. Rao, and J. Rexford. Virtual LAN Usage and Challenges in Campus Networks. *Princeton University Technical Report 2010* `http://www.cs.princeton.edu/~jrex/papers/vlan10.pdf`.