CIDS: Adapting Legacy Intrusion Detection Systems to the Cloud with Hybrid Sampling

Qingtang Xia*,Tianjia Chen†,Wei Xu‡
Institute for Interdisciplinary Information Sciences
Tsinghua University
Email: *xqt13@mails.tsinghua.edu.cn, †ctj2015@mail.tsinghua.edu.cn, ‡weixu@tsinghua.edu.cn

***Abstract*—Many attacks originate from inside, and security problems within cloud-computing platforms are becoming more and more severe. Although many Intrusion Detection System (IDS) help monitor and protect the inbound and outbound traffic of data centers, it is still challenging to deploy IDS inside a cloud-computing platform due to extremely high bandwidth within, and the lack of a single ingress point to deploy the IDS. This paper presents two ideas allowing traditional IDS to be adopted to the cloud environment: software-defined-networking (SDN) based packet collection and a hybrid sampling algorithm to significantly reduce workload on the IDS.**

We integrate our data collector in the Open vSwitch of every physical server, making packets capturing highly efficient. Our hybrid sampling algorithm combines both flow statistics and IDS feedback to intelligently chose which packets to sample. The sampling rate is determined by the current workload in the cloud, and thus minimizing the effects to normal workload.

We evaluate our prototype CIDS on a 125-server production OpenStack cloud using real world attack traces, and demonstrate the effectiveness of our approach.

I. Introduction

With the wide adoption of cloud computing and so-called big data, more and more data are processed, stored and transmitted in the cloud. Many reports point out that security and privacy issues are among the biggest challenges facing cloud platforms [12] [4]. Intrusion detection and prevention becomes one of the most important tasks in cloud computing [19], after many attacks on cloud-based applications, impacting hundreds of millions of users in the past few years [13] [11].

Intrusion Detection Systems (IDS) are widely used middle boxes that try to detect intrusions using a combination of data processing techniques on raw packets, such as packet dump, packet decoding, session reassembling, signature matching and machine learning [17]. It has played a key role in traditional area of network security monitoring. IDS has been proven successful as a strong mechanism to protect traditional network systems. Cloud platforms even help deploying the IDS boxes as virtual machine appliances (one form of the network function virtualization [2]). However, many challenges remain when we deploy traditional IDS in the cloud platforms [28].

It is important to realize that many attacks originate from inside the cloud itself, rather from the outside Internet through the network gateway, as many researchers have pointed out [24] [9]. Insider attacks are an important issue for public clouds. Different from traditional attacks, insider attacks in the cloud bring the following challenges to IDS systems in the cloud.

(1) Efficiency and scalability. Most IDS depends on deep packet inspection. As the network traffic within the cloud can be tremendous, with the high-bandwidth data center network (e.g. 10Gbps to server and 100Gbps at core). Comparing to just inspecting the ingress/egress traffic at the gateway, it is prohibitively expensive in terms of the amount of computational and communication resources required to process all packets. In fact, during our experiments, we see high packet loss rate in IDS, even with moderate utilization and many IDS resources.

(2) Virtualized, dynamic changing and multi-tenant network environment [3]. To accommodate multiple tenant, the cloud network often uses tunneling for network virtualization, which hides the real packet header, making it more expensive for IDS to decode packets. Also, the virtual machines (VM), virtual networks, cluster nodes and tenants are dynamically changing, and thus the operators have to consistently reconfigure the IDS to capture meaningful traffic in the cloud.

We design CIDS, a framework for adapting off-the-shelf IDS into the cloud environment, to detect attacks from inside the cloud environment. The key idea of CIDS is to reduce and balance the data collection and computation workload dynamically, according to the resource utilizations in the cloud. The goal of CIDS is to increase the probability that the IDS captures the attack traffic, while keep both the IDS and the data center network from being overloaded.

CIDS consists of a set of *mechanisms* to perform dynamically configurable packet capturing, sampling, filtering, as well as a hybrid sampling *policy* engine to automatically determine the right sampling rate according to the resource utilization. We make no assumptions of the implementation of the IDS itself, and thus support any IDS that runs inside a virtual machine.

We use a combination of both local and global packet processing mechanisms. On each *physical* server, we deploy a host-based data collection module as a virtual switches (Open vSwitch [26] in our setup) plugin to perform both packet filtering and local statistics collection. We also employ a distributed stream processing engine, Storm [1] to pre-process the packets and computes global statistics. The host modules eliminate unnecessary packets as early as possible to reduce communication overhead. The distributed Storm engine keeps global statistics, in order to provide better insights on the

sampling policy decision.

The local data collection module captures packets, samples them according to a central sampling policy and forward them to the correct IDS VMs for processing. The data collection module works before GRE or other tunneling protocol encapsulating IP packets and thus it captures clear-text packets that can be decoded and processed directly by the succeeding modules without additional de-encapsulation operations.

For global packet statistics, we use Apache Storm [1]. Specifically, we compute the flows counting and feature entropy [16] at a global level, and then select the most abnormal top-n flows under large-scale anomaly situation, making sure that they receive the highest priority to be processed.

To decide which packets to sample, we design a hybrid sampling policy that takes into account the network workload (to decide the max amount of packets we can process), the flow size / length, global properties such as feature entropy, as well as allowing the IDS to provide feedback to *mark* suspicious flows. All the sampling decisions are made through a centralized controller. Our hybrid sampling algorithm has two desirable properties: 1) we can dynamically adjust the sampling rate according to the resource utilization; and 2) we intelligently select the most suspicious flows for IDS.

The key policy decision in CIDS is which packets to sample. Naive random sampling methods, either packet-based or flow-based, lead to information loss, and thus are not usable in signature-based IDS. As we will show in our evaluation, random sampling decreases detection accuracy significantly.

We evaluated CIDS on a production OpenStack [10] cloud platform with 125 physical nodes running thousands of virtual machines, using the open source Snort [27] as IDS. Replaying two different sets of public attack traces, we show that CIDS almost doubles the intrusion detection rate than traditional sampling-based methods, automatically adapts to the cloud workload, and significantly reduces the computation overhead and network communication overhead.

Our core contributions of this paper include:

(1) We design a hybrid sampling scheme. In this design, using a combination of local and global sampling decisions, we filter out many packets and yet maintain detection accuracy, making unmodified traditional IDS usable in a cloud environment.

(2) We provide an SDN-based packet collection and monitoring mechanism that allows efficient packet collection from inside a cloud.

(3) We evaluated our system using real world attack traces in a production cloud environment with over a hundred servers, and demonstrate the efficiency and applicability of the system.

The remaining of this paper is organized as follows: We give the overview of CIDS in Section III. In Section IV, we discuss our hybrid sampling algorithm. Section V shows the design details of our CIDS framework. Experimental results of our proposed framework are given in Section VI. And finally we conclude and discuss about future works in VII.

II. Related Work

Researchers have proposed various approaches to detect intrusions for cloud-computing platforms. Baraka et al. [3] p-resented an approach that leverages IDS to protect the compute nodes in cloud. They deployed Snort at the frontend, backend, or in VMs to detect attacks originated from internal and/or external. With their approach, DDoS attacks can be detected highly efficiently. [21] proposed Snort based on signature intrusion detection in open source cloud, Eucalyptus [25]. For this approach, Snort is deployed in each physical machine which host users' VMs to identify attacks initiated from external networks. A number of instances or VMs for running IDS are used to secure cloud. However, this solution requires each physical node to reserve resource for the IDS, causing lots of complexity in task scheduling, as not every single physical node can accommodate the resource requirements. Modi et al. [24] proposed a framework which combined Network Intrusion Detection System (NIDS) with Decision Tree (DT) Classifier in the cloud infrastructure. They deployed NIDS at the frontend and the backend of cloud to detect known intrusions based on configured rules. Uncertain packets were forwarded to DT classifier to identify which class (intrusion or normal) they belonged to. The output of DT were sent back to NIDS to generate rules for accurate detection.

Other researchers focus on sampling-based anomaly detection [15], [20]. But issues how sampling affects intrusion detection accuracy are still open. In [15], they proposed approaches that leverage sampled data to detect port-scan behavior. In [20], they compare and analyze four classic sampling algorithms under distinct sampling rate and different detection parameters empirically and they get the conclusion that flow sampling is better than packet sampling. The authors of [16] proposed an approach to detect anomaly from backbone networks based on analysis of the entropies for principal packet features. Their experiment showed that anomalies naturally fall into distinct and meaningful clusters according to feature distribution. In [6], they concluded that entropy-based anomaly detection had high tolerance degree despite of the information loss for sampling. Worm outbreak events could also be exposed at low sampling rate using entropy-based approaches.

III. CIDS Overview

Instead of designing a all-new IDS, the goal of CIDS is to provide a framework so that the cloud operators can adapt existing IDS software or hardware boxes to detect intrusions from inside the cloud. System-wise, we combine software-defined networking techniques, scalable data stream processing and hybrid sampling algorithms to provide a practical and efficient system design. We assume that the existing IDS system runs in virtual machines, which is a black box for CIDS. The core ideas of CIDS are the following:

Configurable and scalable packet monitoring and filtering framework. As the traffic is huge within a cloud, we need to eliminate unnecessary traffic as early as possible. Thus, we

decide to filter out traffic at the source. Most cloud platforms already have built-in software switches at the hosts to handle multiple virtual machines, and we leverage these software switches to perform packet filtering. In CIDS, we integrate our data collection module with the Open vSwitch (OVS) on each physical server. We dynamically choose which (virtual) port to monitor and set up rules to include / exclude specific packet headers, and then forward them to the correct IDS VM for processing. The filtering behavior is controlled by a centralized controller.

Collecting data at the software switches also solves the tunneling problem discussed in Section I, as we capture the raw packet headers before the encapsulation, so we can process them later without complex decoding, saving significant bandwidth and computation overhead.

Local and global packet statistics. To make detailed sampling decisions, we need to maintain packet and flow statistics, such as rate and feature entropy at both *local* level and *global* level. At the local level, the host-based module computes statistics, and we use Apache Storm [1] to perform flow-counting and calculation of feature entropy at a global level. All the statistics are sent to the centralized sampling controller to determine the sampling rate, as well as the priority.

Hybrid sampling algorithm to capture more potential attacks. Sampling is necessary considering the amount of traffic in the cloud. However, the IDS uses a stateful algorithm for detection, which means randomly dropping packets or flows can severely affect the detection accuracy. CIDS provides a way to select the most suspicious traffic to feed into the IDS.

There are two classes of attacks prevalent in the cloud, based on the amount of attack traffic, small scale and large scale [23] [15] [16] [20]. Small scale attacks include exploiting specific vulnerabilities to get root privilege, attempting to crash victim machines or steal user data. They sometimes only involve a handful of packets, and happen quite irregularity. Large scale attacks, such as denial-of-service (DoS), on the other hand, involves tremendous intrusion traffic and distributed attack agents. Both kinds of attacks can originate from inside of the cloud.

Our hybrid sampling algorithm first make an estimate of the network resource utilization at each host, and decide a maximum sampling rate. Based on this max rate (workload aware), it then decide which flows / packets to sample, based on local metrics such as flow length, and global metrics such as feature entropies. Moreover, it allows the IDS system to provide feedback on which (partially observed) flows are suspicious, and all subsequent packets from the suspicious flow are sampled.

System architecture. Figure 1 shows the architecture of CIDS. There are three major components in CIDS: highly efficient local data collection module, global packet statistics module, and a centralized controller.

Data collection works with Open vSwitch (OVS) of each physical host. It collects raw data from OVS and executes

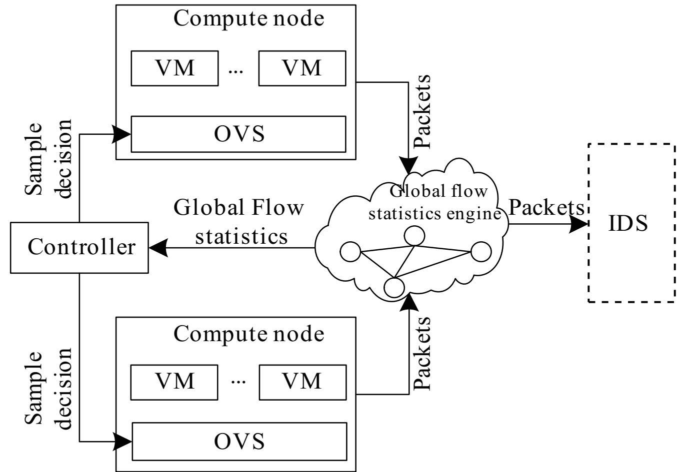

Fig. 1: CIDS architecture.

a series of actions including packets decoding, flow size estimation, flow filtering and flow sampling.

The global packet statistics runs in a set of dynamically allocated virtual machines, and it is implemented with Apache Storm [1], a distributed data stream processing framework. It computes global flow statistics so that we can detect large scale attacks.

The centralized controller coordinates local and global actions. The controller collects flow statistics and resource utilization from local and global components, computes the global sampling strategy, and then send control command to all agents to perform the actual packet filtering actions.

Although we can use CIDS with many off-the-shelf IDS, throughout this paper, we use Snort [27], a popular open source system, as our IDS. We use a custom program to dynamically add / delete Snort VM instances according to the workload, in order to to emulate the modern virtualized IDS setup.

IV. CIDS Sampling Strategies

Sampling strategy is the core of CIDS. There are two levels of sampling consideration in CIDS. First, we need to compute a maximum allowed sampling rate, p_M, for each host, based on the resource utilization. To achieve the sampling rate, naively we can randomly drop packets. However, as we will show in our evaluation, random sampling reduces detection accuracy significantly. Thus, our goal is to intelligently utilize the allowed sampling rate to capture the most likely attacks. In CIDS, we make the sampling decisions based on a combination of three types of statistics, local, global and feedback from IDS cluster. Although we keep global statistics, most of the sampling mechanism actually happens the host, and thus eliminate the data collection overhead early in the process.

A. Determine the Maximum Allowed Sampling Rate

For intrusion detection, we want to capture as much traffic as possible. However, we need to avoid overloading the IDS cluster and avoid competing with production workload for shared resources in the cluster, most importantly, the data

center network capacity. In a virtualized cloud environment, the network bottleneck is usually at the software switch (Open vSwitch), and thus we determine the max allowed sampling rate on each physical server, p_M, using the following equation

$$p_M = U_{IDS} \times U_{network}, \tag{1}$$

where U_{IDS} is the utilization of the IDS cluster. It is a global parameter that is periodically broadcast to each host from the centralized controller. $U_{network}$ is the network utilization observed at the server.

B. Local Sampling Strategy

In the cloud, a few long flows occupy the majority of network traffic, while a large number of small flows generate a small amount of total traffic [29] [15]. As researchers have pointed out, intrusion traffic does not consists of much of the total network traffic (usually no more than 0.05%) [31], and super long flows are highly unlikely to be attacks [8]. This is because attackers usually use small flows to scan the network to collect information about victims, and use moderate flows to upload malware [14]. In fact, researches show that the size of more than 95% abnormal flows is less than 10 packets [31].

With these observations, we want to put more priority on capturing small flows at a given maximum sampling rate. As we do not know the flow size, each host computes the sampling rate for a particular flow f, p_f, as

$$p_f = p_M \times \frac{1}{L_{cur}} \tag{2}$$

where p_M is the maximum sampling rate for the host, calculated using Eq.1, L_{cur} is the current flow size. Note that the simple equation reduces the sampling probability for f quickly as more packets arrive.

There is one exception to the sampling rule. If the IDS detect a flow as suspicious from the first few packets, the IDS system can *mark* the flow and inform the host. Then all subsequent packets from the flow is sampled. Capturing marked flows has the highest priority, and we further reduce p_M for other flows to accommodate the resource need.

C. Global Sampling Strategy

The sampling decisions discussed above are based on local information only. However, there are many large scale attacks, such as distributed denial-of-service (DDoS) attack, use many flows. Detecting these types of attacks requires global flow statistics. However, it is beyond the processing capacity of IDS to track all these flows. Thus, we need a simple yet efficient global sampling strategy to detect abnormal flow patterns in a global sense. We adopt the feature entropy method proposed in [16]. The advantage of the method is its simplicity: it only tracks a few metrics per flow, and it is easy to implement as a distributed stream processing algorithm.

The key idea of feature entropy is to detect sudden changes of value distributions of different *features* across all the flows, such as source / destination addresses / ports. For example, during a DDoS attack, there might be a sudden increase in the number of distinct source IP addresses, and a majority of destination IP addresses will be concentrated to a few values. As another example, with a worm propagating in the cloud, we can see an increase in the number of distinct destinations IP addresses, but the destination ports will be more concentrated to a single value.

Formally, we use the *entropy* of a feature f_x to capture the randomness of its value distribution. We define a feature f_x as a set of N distinct possible values as

$$f_x = \{(x_i, n_i), i = 1, 2, \ldots, N\}, \tag{3}$$

where each value x_i appears n_i times in the feature set. We can calculate the feature entropy using the following equation.

$$H(f_x) = -\sum_{i=1}^{N} \frac{n_i}{|S|} \log_2 \frac{n_i}{|S|}, \ \ where\ |S| = \sum_{i=1}^{N} n_i. \tag{4}$$

It is easy to see that $H(f_x)$ has the range of $(0, log_2 N)$. On one extreme, if f_x only has a single value, $H(f_x) = 0$. On the other extreme, if all values of feature f_x are distinct, $H(f_x) = log_2 N$ [16].

We compute the global entropy across all flows for each feature periodically. From the example above, we can see that during a DDoS, $H(source_ip)$ increases while $H(dest_ip)$ decreases, while during a worm propagation, $H(dest_ip)$ increases and $H(dest_port)$ decreases.

Of course the feature entropy changes all the time. We capture amount of change using expectation deviation, $\xi(H)$, of an entropy H, and we have

$$\xi(H) = \frac{X - E(H)}{\delta(H)}, \tag{5}$$

where $E(H)$ and $\delta(H)$ are the expectation and standard deviation of H respectively. Intuitively, the magnitude of $\xi(H)$ reflects the amount of change of the entropy value.

We need to note that entropies of different features have different variations. Thus we train a threshold for each feature, $\hat{\xi}(H)$ to capture the normal variation, based on historical traffic traces (during a period with no known large attacks).

We periodically compute $\xi(H)$ for each feature globally across the entire cloud using the Storm cluster (to be described in the next section). If we found $\xi(H) > \hat{\xi}(H)$, we would suspect there is a large-scale attack, and the controller will instruct all hosts to start sampling the top flows contributed to the entropy changes. Again, each host will reduce the sampling rate for general traffic to accommodate the resource requirement.

V. System Design and Implementation

CIDS consists of the following modules. Per physical server data collection modules, the global flow statistics engine, a centralized controller, and a adapter to connect to different off-the-shelf IDSes. Figure 2 shows the implementation structure of CIDS.

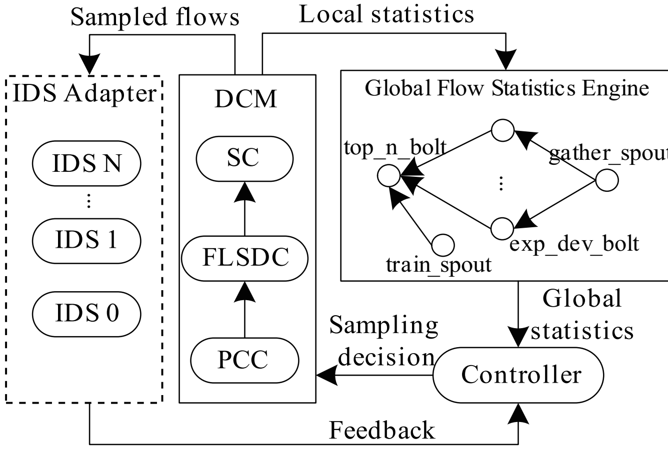

Fig. 2: Implementation of CIDS.

A. Data Collection Module (DCM)

The DCM modules are installed on each physical server. A DCM consists of three components: packets capturing component (PCC), flow filtering and local sampling decision component (FLSDC) and sampling component (SC). For each fixed length time slot, DCM collects a set of packets and computes their statistics. We leverage software-defined network features in the OVS to perform packet sampling. Specifically, we add a virtual interface to OVS and set it to mirror all other interfaces of VMs according to the OpenFlow protocol [22]. PCC listening on this virtual interface to capture packets, and thus we can obtain all VMs communications. PCC works on the virtual interface of OVS for each physical server. All captured packets are passed to the local flow filtering and local sampling decision component (FLSDC) for further processing.

FLSDC accepts input flows from PCC and determine whether the flows belong to (user defined) normal services running in cloud, such as the common storage accesses and data transfers in Storm. If so, we abandon these packets, and if not, we report the local 5-tuple flow statistics to the global flow statistics engine. All filtered traffic are sent to the sampling component (SC). At the same time, FLSDC calculates local sampling rate for each flow using flow length and information from the central controller.

SC decides whether to drop a packet at the local server. It makes the decision according to both the decision from FLSDC component, as well as feedback from the central controller (discussed later in this section). If SC does not drop a packet, the packet is sent to the global flow statistics engine for further processing.

B. Global Flow Statistics Engine

To identify attacks at a global level, we implement a global flow statistics engine. In order to maintain scalablity and reliability of the processing, we decide to implement the engine using Apache Storm [1], a popular open source data stream processing framework. In the Storm framework, there are two kinds of processing units, the *spout* representing the producer of data streams, and the *bolts* representing the consumer to compute on the data streams. These processing units form a *topology* that execute the entire stream processing pipeline. We leverage the Storm framework to handle common tasks like failover and load balancing over multiple compute (virtual) machines.

We implement four tasks on top of the Storm framework. The *gather_spout* spout task receives local flow statistics from DCM, and transforms them into message streams processed by succeeding tasks. The *exp_dev_bolt* bolt tasks receive messages from *gather_spout* every time slot. At the end of each time slot, it computes all the feature entropies and their expectation deviation, using the following steps:

- Receive messages from *gather_spout* and calculate the sets for every feature (e.g. *srcip, srcport, destip, destport*) according to Eq.3 for the time slot.
- According to Eq.4, calculate feature entropies of current time slot.
- According to Eq.5, calculate the expectation deviation until current time slot.

A *top_n_bolt* bolt task compares the expectation deviation outputted from *exp_dev_bolt* with threshold value from the output of *training_spout* to ascertain whether large-scale anomaly happens. If yes, then calculate top-n flows according to global sampling strategy. A *training_spout* spout task takes normal traffic as the input dataset for *training_spout*, and calculate the feature entropies, mathematical expectation and standard variance under normal conditions. At the end, we get the critical threshold of expectation deviation under general/common conditions.

C. Controller

A centralized controller collects data from all the local data collection units, as well as the global statistics engine. The data include both the resource utilization and flow-level statistics. Based on these statistics, the controller decides whether there is potentially a global-scale attacks (e.g. DDoS or port scan) going on, and if so, it computes the most suspicious N flows, and informs all the DCMs on each server, so that all packets from these suspicious flows are collected. Otherwise, the controller will let the local DCMs to make their own local decisions.

D. IDS Adapter

The goal of CIDS is to adapt unmodified off-the-shelf IDS to the cloud. To this end, we only add an (optional) adapter module to the IDS. IDS can use this adapter to provide *feedback* to the controller. The feedback is simple: IDS can only tell the controller which flows it wants to monitor closely. These flows are then considered *marked* so the system will make all the efforts to keep these flows sampled and delivered to the IDS. This feedback mechanism is a key to improve sampling accuracy.

In addition, in a cloud environments, it is likely that an attack stream can be captured by several different DCMs, causing duplicate alarms. In order to suppress these alrams, we

TABLE I: Attack traces selected from KDD'99 and CAIDA'14

Category	No.	Name	Dataset	Attack packets	Enlarge
U2R	1	loadmodule	KDD'99	10*23	10
R2L	2	imap	KDD'99	10*84	10
R2L	3	named	KDD'99	10*47	10
R2L	4	sshtrojan	KDD'99	10*60	10
DOS	5	teardrop	KDD'99	512*7*254*1	512
DOS	6	mailbomb	KDD'99	512*1*1*667	512
DOS	7	land	KDD'99	512*9*254*2	512
DOS	8	synflood	CAIDA'14	1,440,562	1
Prob	9	nmap	KDD'99	256*38*254*3	256
Prob	10	ipsweep	KDD'99	256*9*254*1	256
worm	11	Code redII	CAIDA'14	5,609,294	1

also provide an optional alarm filtering and logging module that records and supresses alarms in a user configurable way.

Though we can adapt to different IDSes, in this paper, we use Snort [27] for experiment, because it is open source yet quite effective.

VI. Evaluation Results

A. Experiment Setup

We evaluate our CIDS in an in-house OpenStack production cluster. The platform has 125 2U servers, and each server has 12 CPU cores, 128GB DRAM, 10TB disks and 10Gbps Ethernet. We deploy data collection module on each of the 125 physical servers. We implement the IDS adapter using a Snort cluster of 31 virtual machines (VMs). We compute the global flow statistics using a Storm cluster with 16 VMs. Each VM has 2 virtual CPUs, 4GB RAM and 100GB disk.

Dataset. We adopt two widely used dataset for IDS evaluation, KDD'99 [18] and CAIDA UCSD anonymized 2014 Internet Traces [7] to evaluate CIDS. KDD'99 includes five weeks of data and CAIDA'14 anonymized passive traffic traces on high-speed Internet backbone links.

We replay traces in both datasets using the Scapy [5] tool in our cloud platform. To match the high network capacity of the cloud, We enlarge the normal and attack traffic in the trace by modifying packets header of source IP address, destination IP address, source port, or destination port. Thus, we have traffic with mixed attack and normal flows originating from different VMs in the cloud. Table I summarizes the attacks contained in these datasets, and our enlargement factor for each types of attacks.

Evaluation Metrics. In the evaluation, we compare CIDS with the following three naive scheme:

- **Full Traffic.** Send all traffic to the IDS, and let the IDS to drop extra traffic if overloaded.
- **Random Packets.** Provide a uniformly random sample of *packets* at the source.
- **Random Flows.** Provide a uniformly random sample of *flows* at the source.

We focus on comparing the most important metric in IDS evaluation, namely, the *intrusion detection rate (IDR)*. We also compare the *Detection Latency* of different schemes. Finally,

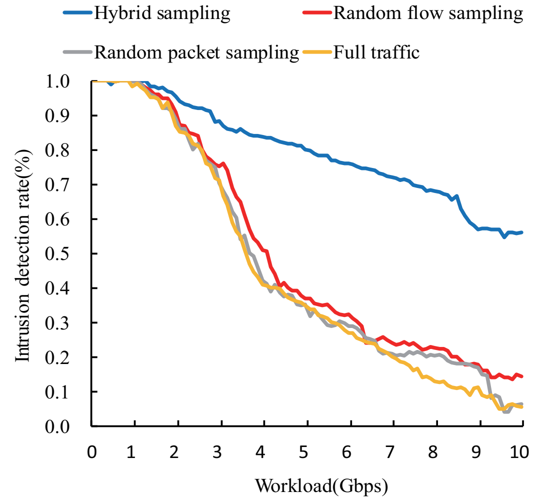

Fig. 4: Intrusion detection rate with increased workload.

TABLE III: Average detection latency (seconds)

Attacks	DOS	U2R	R2L	Probe	Worm
Hybrid	3.45	0.23	0.89	3.03	3.76
Random flow	9.10	1.12	2.90	12.94	10.38
Random packet	13.31	1.47	3.16	13.22	16.97
Full traffic	15.95	1.52	3.48	13.91	17.0

we show that CIDS helps significantly reduce the resource utilization of the IDS nodes.

B. Amount of valid attack packets captured.

Using the 11 types of attacks above, we first evaluate how many valid attack packets we can capture using each sampling scheme, and Table II shows the results. We can see that with CIDS we can capture the most valid packets.

It is especially true for the seven large-scale attacks, such as DDoS and probing, as CIDS captures almost twice as many valid attack packets than other schemes. For example, $86,319$ *ipsweep* attack packets are captured by CIDS while only $43,891$ and $20,483$ are captured by random flow / packet sampling, respectively.

For these small-scale and short-lasting attacks, such as *named*, *sshtrojan*, and *imap*, all three other schemes miss almost all attack packets, while CIDS captures a large portion of them, under the same system workload.

C. Intrusion detection rate (IDR).

Being able to capture more attack packets directly leads to a significant improvement on IDR. Figure 3 shows the IDR comparison over different schemes.

For common attacks, such as *loadmodule* (230 attack packets lasting about 3 minutes), CIDS achieves an IDR of 37%, higher than that of 10% for random flow and 0% for random packet sampling. This is because of the preference to smaller flows, as well as the IDS feedback used by CIDS.

CIDS's preference on short-flows causes many initial attack packets to be captured with high probability, and this is why

TABLE II: The number of valid attack packets captured using different sampling schemes.

Attacks	No.1	No.2	No.3	No.4	No.5	No.6	No.7	No.8	No.9	No.10	No.11
Total packets	231	840	474	600	910, 336	341, 504	2, 340, 864	1, 440, 562	1, 853, 184	146, 304	5, 609, 294
Hybrid	70	320	193	200	491, 581	167, 337	1, 193, 841	619, 442	1, 037, 783	86, 319	3, 477, 762
Random packet	0	130	32	102	118, 344	40, 980	234, 086	129, 651	222, 382	20, 483	617, 022
Random flow	32	220	85	71	236, 687	64, 886	421, 356	273, 707	333, 573	43, 891	1, 290, 138

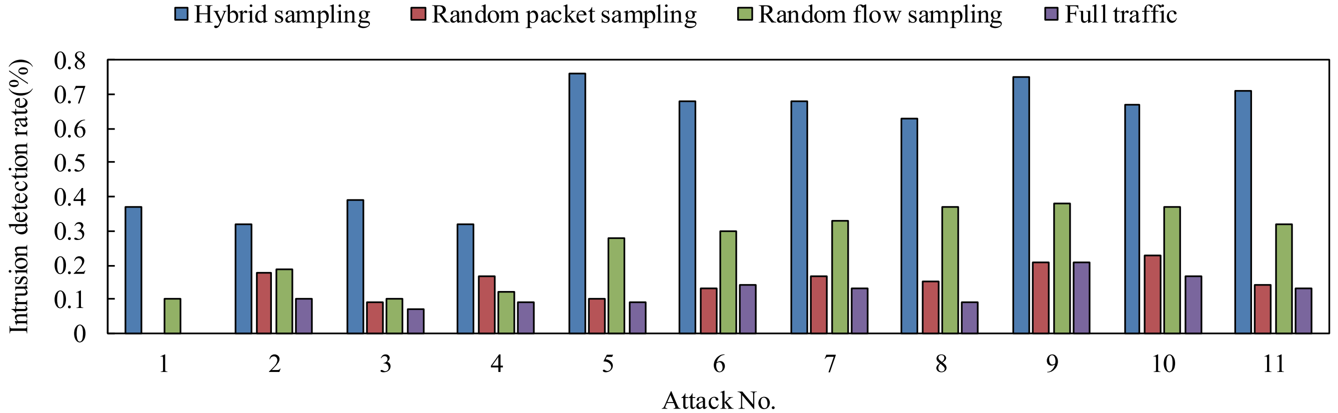

Fig. 3: IDR comparison for 11 attacks.

CIDS achieves higher IDR for attack types 1-4 than all other schemes.

D. Stability of IDR under different load.

It is preferable that the IDR degrades gracefully when the system load increases. Here we show that CIDS achieves this desirable property. We replay the *synflood* attack traffic, using a varying amount of normal *iperf* [30] traffic as background traffic to adjust the workload in the network.

Figure 4 shows the change of IDR under different network load. We can see that under low load (smaller than 1Gbps), all schemes achieves similar IDR. However, IDR of other all other three naive sampling schemes degrade dramatically when load increases. On the contrary, CIDS automatically selects flows that are more likely to be abnormal and thus provides a more graceful degradation on IDR.

E. Detection latency.

It is also important to be able to detect attacks quickly. We show that although CIDS adds a bit of detection latency, the overhead is quite acceptable.

CIDS gets small detection latency of user-to-root and remote-to-local attacks, with 0.23 and 0.89 seconds respectively. Detection latency of large-scale anomalies, such as *DDoS*, *probe*, and *worm* is about 3 seconds, also much lower than the random sampling schemes. Large-scale anomalies take longer time to detect in CIDS because CIDS need several time slots to calculate feature entropies. CIDS presents better detection latency because it is less likely to miss attack packets, letting the IDS to capture and accumulate these attack packets quickly. Table III shows the detailed data.

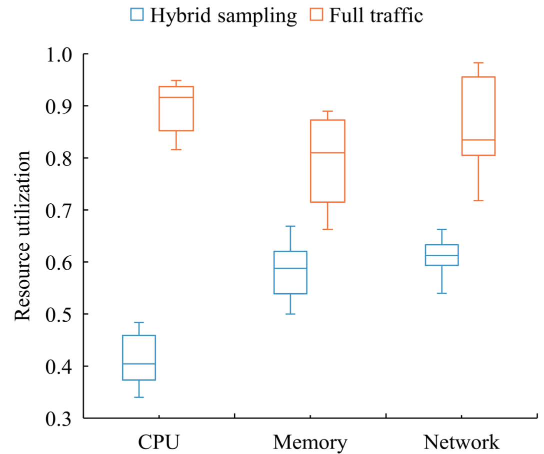

Fig. 5: IDS-VMs resource utilization comparison.

F. Resource utilization on IDS.

As IDS systems are scares and expensive resources in the system (especially when the cloud adopts commercial IDSes that requires a per-CPU license), we want to save as much IDS resources as possible. We show that CIDS can help achieve this goal.

We perform this evaluation using no attacks. Thus all traffic is normal from iperf, and we measure the resource utilization on the IDS to process these traffic. Figure 5 shows that CIDS can save much resource on the IDS system, keeping them from being overwhelmed.

VII. Conclusion and Future Work

Attacks within cloud are hard to detect using traditional IDS. However, the tremendous bandwidth in the data center network renders most of existing IDS systems too expensive to apply directly. Random sampling partially helps solving the problem, but uniform random packet or even flow sampling significantly decreases the detection accuracy in a traditional IDS. Instead of redesigning an IDS, we propose CIDS, a novel architecture that integrates flow filters built on software defined networking technology, IDS-aware sampling mechanisms and policy, to allow adapting traditional IDS system in cloud, maintaining their detection ability while keeping the overhead low. In the experiments on a sizable cluster running real world traces, we demonstrate the effectiveness of CIDS.

As future work, we will focus on the mechanisms of small scale intrusions to improve IDR for such type of attacks. Furthermore, a type of long term intrusion (e.g. advanced persistent threat, or APT) is a dangerous category of threats to cloud computing platform. It is hard to detect such kind of attacks mainly because it is impossible to keep track of all flows for an extended period of time, due to the resources limitation. We consider detecting APT in the cloud an important future direction and we are designing a approximation algorithm to keep (partial) states, in order to obtain enough states for APT detection.

References

[1] Quinton Anderson. *Storm real-time processing cookbook*. Packt Publishing Ltd, 2013.

[2] Fatemeh Azmandian, Micha Moffie, Malak Alshawabkeh, Jennifer Dy, Javed Aslam, and David Kaeli. Virtual machine monitor-based lightweight intrusion detection. *ACM SIGOPS Operating Systems Review*, 45(2):38–53, 2011.

[3] Hifaa Bait Baraka and Huaglory Tianfield. Intrusion detection system for cloud environment. In *Proceedings of the 7th International Conference on Security of Information and Networks*, page 399. ACM, 2014.

[4] Stephen Biggs and Stilianos Vidalis. Cloud computing: The impact on digital forensic investigations. In *Internet Technology and Secured Transactions, 2009. ICITST 2009. International Conference for*, pages 1–6. IEEE, 2009.

[5] Philippe Biondi. Scapy. *see http://www. secdev. org/projects/scapy*, 2011.

[6] Daniela Brauckhoff, Bernhard Tellenbach, Arno Wagner, Martin May, and Anukool Lakhina. Impact of packet sampling on anomaly detection metrics. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 159–164. ACM, 2006.

[7] UCSD CAIDA. Anonymized internet traces 2014.

[8] Scott Campbell and Jason Lee. Intrusion detection at 100g. In *State of the Practice Reports*, page 14. ACM, 2011.

[9] Massimo Ficco, Luca Tasquier, and Rocco Aversa. Intrusion detection in cloud computing. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on*, pages 276–283. IEEE, 2013.

[10] Tom Fifield, Diane Fleming, Anne Gentle, Lorin Hochstein, Jonathan Proulx, Everett Toews, and Joe Topjian. *OpenStack Operations Guide*. " O'Reilly Media, Inc.", 2014.

[11] Joseph Galante, Olga Kharif, and Pavel Alpeyev. Sony network breach shows amazon clouds appeal for hackers. *Bloomberg News*, 16, 2011.

[12] Frank Gens. It cloud services user survey, pt. 2: Top benefits & challenges. *blog, http://blogs. idc. com/ie*, 2008.

[13] Doug Gross. million compromised in evernote hack. *CNN Tech*, 2013.

[14] David Jaeger, Martin Ussath, Feng Cheng, and Christoph Meinel. Multi-step attack pattern detection on normalized event logs. In *Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on*, pages 390–398. IEEE, 2015.

[15] Jaeyeon Jung, Vern Paxson, Arthur W Berger, and Hari Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 211–225. IEEE, 2004.

[16] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM Computer Communication Review*, pages 217–228. ACM, 2005.

[17] Wenke Lee and Salvatore J Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM transactions on Information and system security (TiSSEC)*, 3(4):227–261, 2000.

[18] Richard Lippmann, Joshua W Haines, David J Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. *Computer networks*, 34(4):579–595, 2000.

[19] M Lockneed. Awareness, trust and security to shape government cloud adoption. *LM Cyber Security Alliance and Market Connection White Paper*, 2010.

[20] Jianning Mai, Chen-Nee Chuah, Ashwin Sridharan, Tao Ye, and Hui Zang. Is sampled data sufficient for anomaly detection? In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 165–176. ACM, 2006.

[21] Claudio Mazzariello, Roberto Bifulco, and Roberto Canonico. Integrating a network ids into an open source cloud computing environment. In *Information Assurance and Security (IAS), 2010 Sixth International Conference on*, pages 265–270. IEEE, 2010.

[22] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[23] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. A survey of intrusion detection techniques in cloud. *Journal of Network and Computer Applications*, 36(1):42–57, 2013.

[24] Chirag Modi, Dhiren Patel, Bhavesh Borisanya, Avi Patel, and Muttukrishnan Rajarajan. A novel framework for intrusion detection in cloud. In *Proceedings of the Fifth International Conference on Security of Information and Networks*, pages 67–74. ACM, 2012.

[25] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus open-source cloud-computing system. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, pages 124–131. IEEE, 2009.

[26] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The design and implementation of Open vSwitch. In *12th USENIX symposium on networked systems design and implementation (NSDI 15)*, pages 117–130, 2015.

[27] Martin Roesch and Chris Green. Snort users manual. *Snort Release*, 1(3), 2003.

[28] Sebastian Roschke, Feng Cheng, and Christoph Meinel. Intrusion detection in the cloud. In *Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference on*, pages 729–734. IEEE, 2009.

[29] William Stallings. High speed networks. *TCP/IP and ATM Design Principles, Upper SaddleRiver, NJ, Prenti c eHall, In c*, 1998.

[30] Ajay Tirumala, Qin Feng, Jon Dugan, Jim Ferguson, and Kevin Gibbs. iperf: TCP/UDP bandwidth measurement tool. 2005.

[31] Arno Wagner and Bernhard Plattner. Entropy based worm and anomaly detection in fast IP networks. In *14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05)*, pages 172–177. IEEE, 2005.