Attacking Data Center Networks from the Inside

Anurag Khandelwal[*]
UC Berkeley

Navendu Jain
Microsoft Research

Seny Kamara
Microsoft Research

ABSTRACT

The ubiquitous growth in the popularity of public cloud computing platforms as seen today entails an inherent risk: the shared nature of data center networks (DCNs) renders co-hosted tenants susceptible to attacks from *within* the network. In this paper, we discuss the security mechanisms offered by popular cloud service providers at present, and explore the extent to which existing data center networks might be vulnerable to internal denial-of-service attacks. We describe two categories of attacks: the first comprising those that are easy to mount, but are ill-disguised and conform to traditional attack patterns (*overt* attacks), and the second comprising hard-to-detect *covert* attacks that involve a greater complexity of deployment, but also show greater impacts. Finally, we discuss attack-mitigation schemes that employ common network mechanisms, such as TCP pacing and Network Virtualization, and also present a new source-based filtering scheme to regulate network communications.

1. INTRODUCTION

Cloud computing offers a novel way to deploy applications and services. In this paradigm, a cloud service provider makes its infrastructure and computational resources—compute, storage and network—available as a service to users. Providers can increase profits by multiplexing their resources to multiple tenants and tenants can deploy their applications and services in a scalable and inexpensive manner.

While the popularity of public clouds like Amazon EC2 [1], Microsoft Azure [3] and Google Compute Engine [2] indicates the promise of this model, the shared nature of public cloud infrastructures leaves them vulnerable to malicious tenants. In fact, potential attacks by malicious co-tenants remains one of the main concerns and barriers to the adoption of public clouds by enterprises and governments [22].

In virtualized data centers, the hypervisor fairly divides resources like CPU, memory and disk amongst co-hosted virtual machines (VM), but the network still remains open to contention. The lack of isolation in data center networks (DCN) provides malicious tenants an opportunity to gain an unfair share of bandwidth or to degrade the quality of service of other tenants (e.g., competitors).

[*]Work done while at Microsoft Reserch.

DCNs are be particularly susceptible to network attacks due to (at least) three conditions:

- **Bandwidth over-subscription and shallow buffers**: DCNs are typically setup as multi-rooted trees with servers at the leaves and core routers at the roots. The links are (often) highly oversubscribed; ranging from 1:2 at the Top of Rack (ToR) switch, to 1:240 from servers to the core [11]. Such a setup allows an attacker to congest oversubscribed links with high utilization to maximize impact. Furthermore, network switches have shallow buffers (typically 4-16 MB) which makes them susceptible to traffic bursts that quickly exhaust the switch memory or maximum permitted buffer for an interface, risking packet losses.

- **TCP dominance and diverse traffic workloads**: Typically, 99.5% of DCN traffic is TCP [5]. While the original 1974 paper proposed packet-switching to share resources [9] and later efforts incorporated congestion control to provide fairness, these techniques were primarily focused on low-bandwidth, high-latency setup with high statistical multiplexing such as the Internet. In contrast, a DCN is comprised of paths with high bandwidth-delay product where even a slight variation in RTTs can cause TCP to back-off, significantly degrading throughput. A related point is that data centers run diverse workloads from ones that require small predictable latency with others requiring large sustained throughput. Since these flows compete for limited buffer space at switches, they can hurt each other's performance. For example, bandwidth-intensive flows of backup services can fast build up queues increasing delay in latency-sensitive foreground services.

- **Shared network services and multiple vantage points**: Data centers host a broad range of network services including DHCP, ARP, and DNS to bootstrap services and enable communication between instances. By targeting and disabling these services, a malicious tenant can effectively attack a large number of cotenant services in the data center. While DCNs deploy defense mechanisms to prevent IP spoofing and gratuitous ARP, they are still vulnerable to several classes of attacks that can be masqueraded as legitimate protocol messages such as ARP storms and bandwidth

.

flooding. Note that while cloud providers do impose per-VM budgets for outgoing bandwidth, an attacker can still deploy multiple VMs to mount distributed flooding attacks so as to meet per-VM constraints.

This paper makes a first attempt to examine a broad range of attacks on data center networks. Our results show that there are a number of ways for tenants to attack services hosted in a public cloud. Some of the attacks we consider are well-known (e.g., SYN, ARP and DHCP floods) but some, as far as we know, are new and make use of subtle problems that occur when TCP is implemented in data center networks. The attacks we consider range from covert attacks, which are virtually undetectable, to more overt ones that can be detected but are very powerful. We also consider attacks that are targeted at a specific victim service and attacks that are more indiscriminate and aimed at the network.

For all the attacks we consider, we study the tradeoffs between cost of deployment, level of impact, and risk of detection. We also discuss means to mitigate the impact of these attacks based on our experimental observations.

In summary, we make the following contributions:

1. we implement and empirically validate a broad range of attacks targeted at data center networks,

2. we identify the network resources that facilitate these attacks (e.g., switch buffers),

3. we observe and show how TCP-based performance problems (e.g., Incast [18] and Outcast [20]) can be leveraged by malicious tenants to mount *highly covert* attacks against other tenants.

4. we demonstrate the feasibility of low-rate DoS in a DCN requiring modest overhead but that can cause significant impact.

5. we show the feasibility of well-known bandwidth flooding attacks while still meeting per-VM rate caps enforced by cloud providers.

6. finally, we outline techniques to defend against these attacks using common network mechanisms and discuss engineering challenges in implementing them.

2. BACKGROUND

DCN Topologies. DCNs that host public clouds typically have a multi-rooted tree topology as shown in Figure 1. The links are high-bandwidth, ranging from 10 to 40Gbps and are often highly oversubscribed. At the lowest level (that of the top of rack (ToR) switches) the oversubscription rate is usually 1:2 but at higher levels (e.g., between servers and core routers) the oversubscription rate can be almost 1:240 [11]. Moreover, the switches in the lower layers of the network are generally inexpensive and have shallow buffers that range from 4-16 MB.

DCN Workloads. The workloads associated with DCNs usually include application traffic such as web search, advertisements, content streaming and data analytics. Such applications have a diverse set of communication patterns which include short-lived latency-sensitive flows and long-lived throughput-sensitive flows. Broadly speaking, we can categorize them into three categories: bursty query traffic or mice flows (2-20KB), delay-sensitive short messages (100-1MB) and long or elephant flows (1-100MB) [11]. Typically, 99.5% of DCN traffic is TCP [5].

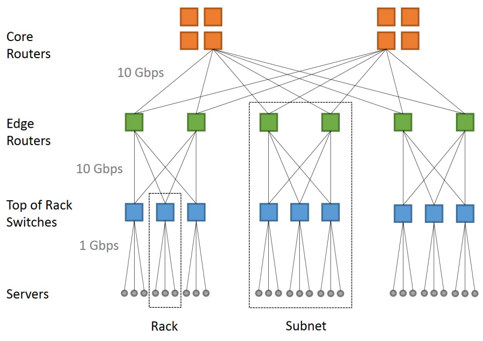


Figure 1: Multi-rooted tree topology

DCN security. Public clouds such as EC2 and Azure implement security mechanisms at multiple levels, including at the host OS, the guest OS, and at the firewall.

EC2 and Azure claim to provide protection against several traditional network security attacks but recommend that tenants implement their own protection as well. Standard DDoS mitigation techniques such as SYN cookies and connection limiting are employed within EC2 [6]. Azure claims to be able to detect incipient internal DDoS attacks and remove the responsible VMs or accounts [16]. In EC2, external DDoS attacks are mitigated by providing internal bandwidth that well exceeds the provider-supplied Internet bandwidth [6] and Azure has load-balancing mechanisms that mitigate, in part, DDoS attacks from inside and outside the DCN [16].

IP spoofing is not possible within EC2 or Azure. In EC2 the firewalls ensure the authenticity of all network traffic, i.e., an instance may only send traffic with source IP/MAC that is assigned to it [6]. In Azure, filters at the hypervisor VM switch block any broadcast and multicast traffic with the exception of DHCP leases and ARP requests [16]. These filters, along with a restricted IP/MAC to VM mapping that is ensured by ToR switches, block any source-spoofed packets. This imposes a strict check on the identity of the source of the traffic.

Per-VM bandwidth limits are explicitly imposed in Azure - the amount of bandwidth allocated is proportional to the instance size, with traffic caps of about 5Mbps for the XSmall (smallest) compute instance, to 800Mbps for XLarge (largest) instances [16]. These bandwidth caps, while imposed primarily to ensure a proportionality between the VM cost and the corresponding network share, also provide a means to prevent unchecked saturation of bandwidth by a single or a few VMs. Bandwidth is not limited to set rates in EC2; instead different instances sizes are assigned different I/O priorities. If no other instance is utilizing bandwidth on a particular host, a VM can potentially get as much bandwidth as the host is capable of providing [6].

3. ATTACK SETUP

Adversarial model. We assume the cloud service provider is trusted and any potential flaws in the infrastructure is un-

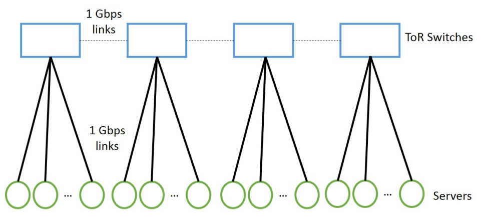


Figure 2: Network topology for experimental setup

intentional. The attacks that we consider are all mounted from *user space*. We also assume the root OS and the hypervisor are secure: the adversary requires no elevated privileges to deploy and execute the attacks. We assume the adversary intends to either (a) gain an unfair share of the network resources (e.g., bandwidth), or (b) deny service to a targeted network application. The first scenario includes tenants that want to obtain a greater share of the network than what they paid for. The second scenario includes tenants that deliberately deny service to other tenants' applications; for instance, a competing service that floods the network so as to degrade the quality of their target's application would fall under this category.

Overview of attacks. We consider four types of attacks which we classify according to two dimensions. The first is whether the attack is targeted or indiscriminate, and the second is whether it is covert or overt (i.e., hard or not to detect). While targeted attacks are usually harder to deploy, they also have greater impact and a lower risk of detection. Indiscriminate attacks, on the other hand, can be easier to deploy but easier to detect.

Experimental setup. All our experiments were carried out on a high performance computing cluster with 152 servers distributed under 4 racks. The cluster employs Cisco SGE2010's as ToR switches, with each server-to-switch link having a bandwidth of 1Gbps. The switches are all connected in a daisy-chain topology with switch-to-switch link bandwidth of 1Gbps, as shown in Figure 2. For all experiments, we restrict our network traffic to a single rack, the inter-rack isolation ensuring no bottleneck at the switch-to-switch links.

In all our experiments, CPU and memory are never bottlenecked (unless explicitly shown). Each servers is equipped with two Quad Core AMD Opteron processors with clocking speed of 2.10 GHz, 32GB RAM and 1Gbps NICs. The nodes run Windows Server 2008 R2 (SP2), and retain all of their native network parameters. The default TCP $\mathrm{RTO}_{\mathrm{min}}$ is 300ms for each node.

3.1 Cloud Inference

All our attacks rely on a setup phase, in which the attacker gathers information about the cloud including its network topology, its VM placement strategy and some network parameters. As we describe below, previous work has already shown how this information can be obtained in public clouds; in our experiments we assume the attacker has already gathered this information.

Topology inference. Raiciu et al. [21] demonstrate the steps involved in topology inference by inferring the topology of an EC2 data center network. The inference relies on traceroute, iperf and ICMP ping and could be carried out in any DCN that permits the use of these tools.

Recall that in EC2, VMs run on top of the Xen Hypervisor [8] with each physical machine hosting several VMs along with a special Dom0 VM. Dom0 mediates the access of all other VMs on that machine to the network and reports itself as an IP hop in traceroutes.

The attack proceeds by launching many instances and by running ping, traceroutes and iperfs between pairs of instances. The results of these pairwise experiments will differ depending on whether the two VMs are co-located on the same machine, are within the same rack, are in the same subnet or in different subnets. Based on these observations, the attacker can infer the DCN topology. For more details we refer the reader to [21].

Cloud cartography and co-location. Cloud cartography enables an attacker to establish a mapping between instance creation parameters and location within the cloud. With such a mapping, the attacker can estimate the instance creation parameters for a target VM and use them to increase the probability of co-residence with an adversarial VM. In [22], Ristenpart et al. explore how to perform cloud cartography in EC2 and even using their brute-force techniques, it is shown that co-residence can be achieved with probability .4.

We note that our attacks require a comparatively weaker form of co-residence. In particular, our malicious VMs only need to be placed on the same rack or subnet, as opposed to the same physical machine as in the case of [22]. In theory, this form of rack- or subnet-level co-residence can be carried out with a higher probability and can be confirmed with inexpensive checks. For instance, to check if a malicious VM is under the same rack as a target VM, a simple traceroute can be carried out; if they are indeed under the same rack, the traceroute will be of the form VM_1-$Dom0_1$-$Dom0_2$-VM_2. Similar checks can be carried out to confirm subnet-level co-residence.

Network parameters. We also require estimates of a number of network parameters. Most of these can easily estimated in any public cloud. For instance, TCP RTTs can be estimated by methods described by Jiang et al. [14]. For the Traffic Cap imposed on a VM, the available bandwidth reported by iperf can be used as an estimate; Jain et al. [13] also propose mechanisms for end-to-end available bandwidth estimation. Simple network tools such as ICMP ping are used to detect increased latency in the network. The TCP $\mathrm{RTO}_{\mathrm{min}}$ can be estimated by using a variation of the Low Rate Denial of Service (LDoS) attack [15] which we describe in Section 4.3.

3.2 Comparison of Attacks

We provide a summarizing comparison of all the attacks explored, evaluating them on the basis of their cost to launch, impact on the victim, challenges faced due to security measures, and their detectability. The comparison is depicted in Table 1.

The comparison elucidates the various tradeoffs involved in mounting these attacks. The flood-based attacks, which fall under the overt class of attacks, are easy to deploy, but are just as easy to detect due to their inconspicuous traffic patterns. The cost of mounting these attacks, which is proportional to the amount of resources (e.g. number of adversary VMs) required to achieve a minimum thresh-

Attack Methodology	*Cost*	*Impacts*	*Detection*	*Challenges in launching the attack*
Bandwidth Flood	High	Saturation of access link to victim VM, High occupancy at switch buffer	Easy	Filtering of Data packets at source
SYN Flood	Low	Denial of service for connection attempts to victim VM	Easy	Filtering and throttling of SYN packets at source
ARP Flood	High	Saturation of links within subnet, High occupancy at switch buffer	Easy	Throttling of ARP packets at source to very low rates
DHCP Flood	High	Saturation of links within subnet, High occupancy at switch buffer	Easy	Throttling of ARP packets at source to very low rates
Low Rate DoS	Low	Low throughput for long TCP flows	Difficult	Reduced impact and greater detection at low values of RTO_{min}
TCP Incast	Low	Low throughput for long TCP flows, High latency for short TCP flows	Difficult	Placing VMs in target rack
TCP Outcast	High	Low throughput for long TCP flows, High occupancy at switch buffer	Difficult	Placing VMs outside target rack

Table 1: Comparison of Attack Strategies

old impact on the victim's network performance parameters (e.g. throughput, latency, etc.), is generally high. However, these attacks form an important class of attacks, since they demonstrate the feasibility of launching distributed attacks to circumnavigate per-VM traffic caps that are typically imposed in DCNs.

The covert attacks (Incast, Outcast, Low Rate DoS) are relatively difficult to deploy due to the co-location constraints associated with them. At the same time, these attacks are comparatively much tougher to detect, and their degenerative impacts are much more pronounced. These are generally low cost attacks, since a few adversarial VMs are sufficient to achieve the desired performance degradation. The covert attacks add a new dimension to mounting attacks within DCNs, since they specifically exploit the properties of the data center networks to attain their effectiveness.

A study of these attacks also reveals a number of challenges associated with deploying them effectively; aggressive throttling of specific flood traffic (e.g., SYN, ARP and DHCP packets) at their source could significantly decrease their impact, consequently increasing the cost of deployment. For the covert attacks, co-location adds to the complexity of launching the attacks, since their effectiveness relies on strategic placement within the network. Modification of the parameters on which the attacks rely can also nullify their impact to a certain extent; for instance, reducing the TCP RTO_{min} affects the impacts of the Low Rate DoS and Incast attacks.

4. COVERT ATTACKS

A majority of traffic in public clouds is TCP traffic, albeit in varying mixtures of short-lived latency-sensitive flows and long-lived throughput-oriented flows. TCP, however, was designed to acheve long-term throughput fairness in the Internet and its application in DCNs suffer from several shortcomings. Two of the most important ones are the TCP Incast [18] [24] and the TCP Outcast problem [20].

We observe that these shortcomings can be effectively used to perform *highly* covert targeted and indiscriminate DDoS attacks in DCNs. As we show experimentally, these attacks can be very powerful; throttling the throughputs of targeted VMs down to as much as 10% of their expected values.

4.1 The Incast Attack

The TCP Incast problem was first observed by Nagle et al. [18] and arises due to all-to-one communication patterns in high bandwidth, low latency networks that employ TCP such as DCNs and commodity clusters. Typically, the Incast problem is observed in *barrier-synchronized* request workloads, where a client requests data from multiple servers and can only proceed once it has received responses from all the servers. Examples include distributed storage, MapReduce and web-search workloads. In such settings, the responses from the servers get synchronized at the input ports of the connecting switch (usually at the ToR level) and fill up its shared memory pool. This results in packet losses for one or more of these responses and, under severe losses, TCP experiences a timeout which lasts at least $\mathrm{RTO_{min}}$ amount of time. The timeout, along with the barrier synchronized nature of the communication, results in a drastic drop in application throughput.

Deployment. Our first attack is based on the observation that the incast problem can be exploited to attack co-hosted tenants. In fact, if an attacker can strategically place adversarial VMs throughout the DCN in such a way that it can re-create the incast communication pattern, then it can significantly affect the target's traffic. The attack is particularly effective if the target traffic is latency-sensitive. Figure 3 describes the setup needed by the attacker. First, a number of malicious VMs have to be co-located on the same rack as the target VM. As discussed in Section 3.1, this can be done using previous techniques. The adversarial VMs then periodically emit short, synchronized, bursty flows to the target VM. This causes high buffer occupancy at the shallow-buffered ToR switch and induces packet drops for any other flow destined to the target. The impact is especially drastic for latency-sensitive short flows, since the TCP retransmission timeouts triggered by the packet drops cause them to default their flow deadlines.

Experimental results. We experimentally validated the impact of this attack by simulating the communication pattern observed in cluster filesystems in our own setup. In our experiment, a victim client issues a request for a block of data of size B that is striped across N servers. Each sender responds with B/N bytes of data. Only after having

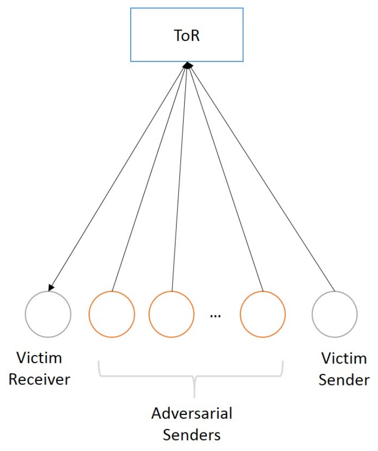


Figure 3: Schematic for Incast based attack

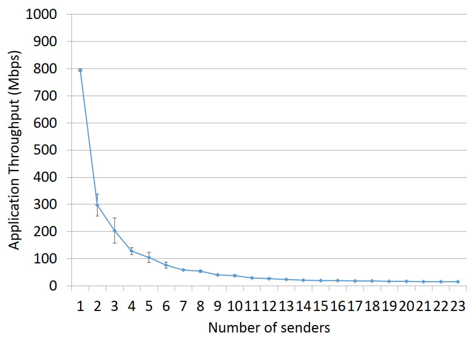


Figure 4: Incast: Drop in application throughput with increase in number of synchronized flows

received responses from every server does the client issue requests for the subsequent data block. Some or most of these flows can be visualized as adversarial flows, where the adversary attempts to add to the number of synchronized flows at the switch buffers. This adds to the likelihood of buffer overflows, increasing the number of TCP timeouts and, therefore, the impact of TCP's congestion control on the victim flow(s).

Each experiment is run for 200 block transfers of block-size $B = 1MB$ to observe steady-state performance, calculating the application throughput at the client over the entire duration of the transfer. All flows are TCP flows with $\mathrm{RTO}_{\min} = 300ms$.

Since the attack traffic resembles the traffic pattern of the victim, i.e, both the patterns conform to common barrier synchronized workloads, it becomes increasingly difficult to distinguish the attack traffic from the legitimate flows required by the victim. In fact, a traffic pattern analysis alone would fail to distinguish the attack pattern from a typical Map-Reduce workload – validating the classification of this attack as covert.

The results of the experiment are shown in Figure 4. We confirm the sharp drop in application throughput on increasing the number of barrier synchronized flows. The drop is extremely rapid at the begining, but slows down with the further addition of flood-senders; at $N = 6$ senders, the application throughput drops to as much as 10% of the original

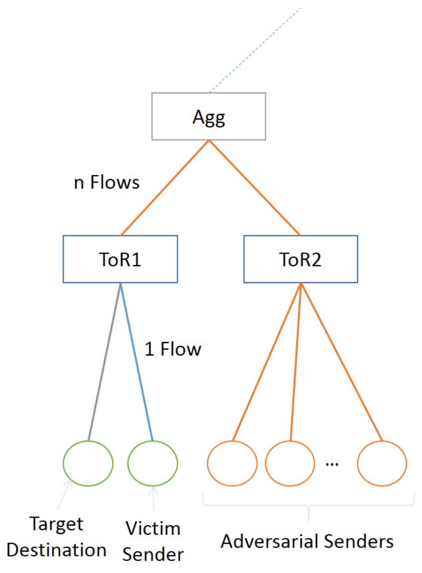


Figure 5: Schematic for Outcast based attack

($\approx 800Mbps$), but does not fall significantly beyond that. This implies that a potential adversary could cause desired degradation of throughput even with a small number of adversarial flows, imposing a minimal cost on attack deployment, while causing a sustained impact on victim throughput.

4.2 The Outcast Attack

The TCP Outcast problem is another problem that arises when TCP is used in low-latency DCNs. More specifically, it occurs when commodity switches that employ a droptail queuing policy at the output buffers are used. Outcast was first observed by Prakash et al. [20], and works as follows. Whenever a burst of packets arriving at any two incoming ports of a switch are destined to a common output port, the tail-drop queuing mechanism causes a batch of consecutive packet drops at the input ports. This phenomenon is referred to as a *port-blackout*. Since the tail-drop scheme is unbiased, the port-blackout is also unbiased and affects one of the two ports uniformly at random. However, when a small set of flows and a large set of flows—each at different input ports—compete for the same output port, the small set receives a much lower throughput and is said to be *outcast* from the large set. This occurs because port-blackout causes packet drops for each set at random, but the number of packet drops is distributed across a larger number of flows in the case of the larger set. Thus the flows belonging to the smaller set are more likely to lose an entire tail of their congestion window during the blackout, which results in TCP timeouts. This causes severe damage to the throughput for the smaller set of flows, whereas the larger set remains relatively unaffected. This bias in the throughput distribution is referred as TCP Outcast.

Deployment. As in the previous Section, we observe that the outcast problem can be exploited to attack co-hosted tenants. By strategically deploying malicious VMs throughout the DCN and re-creating the outcast communication pattern, an attacker can greatly affect the throughput of the target's traffic by artificially inducing port blackouts at particular switches. Figure 5 shows one such placement strategy where N malicious flows under ToR2 compete with a single victim flow under ToR2 for a common output port at ToR1 that leads to the target VM. We use this placement strategy to carry out our attack in our experimental testbed.

The impact of outcast attacks is diminished if the cloud enforces low traffic caps, which is the case in Azure (see Section2). Traffic caps can be circumvented, however, by using a *distributed* outcast attack, where the malicious flows are distributed over multiple VMs so as to stay under the cap.

Experimental results. The setup to verify the impacts of the outcast attack was modeled on the placement strategy described in Figure 5. The N sources, along with the victim sender, issue uncapped bulk TCP flows to the target destination. As discussed, the placement model ensures that the set of adversarial flows compete with the victim flow at different switch input ports, for the same switch output port leading to the target destination.

Note that the outcast attack remains covert, since it mimics the workload observed in common all-to-one communications in DCNs; since the attack employs TCP flows, it does not conform to the aggressive flood based DDoS attack-patterns that generally flout network protocols. The Outcast attack achieves its impact in spite of the adversarial flows being subject to TCP congestion control mechanism, which reduces its likelihood of being identified as malignant traffic.

The results of our experiments are shown in Figure 6 which measures the average throughput at the victim as a function of the number of adversarial flows. We see that the drop in throughput for the victim flow increases significantly with the increase in the number of competing adversarial flows; the throughput falls to as much as $\approx 10Mbps$, which is just 10% of it fair share ($\approx 100Mbps$), at $N = 9$ adversarial flows. As with the incast attack, this shows that the adversary is capable of causing severe degradation of the victim's network performance at a relatively meager cost of deployment, with the outcast attack.

We also describe the impacts of distributed outcast attacks in DCNs with different per-VM bandwidth limits. Figure 7 compares the impact of a traffic cap of R on the victim throughput drop for 6 malicious VMs. Thus, at low traffic caps, the adversary must face a trade-off between the magnitude of damage caused to the victim flow (quantified as the throughput drop) and the cost incurred (quantified as the number of VMs used to issue the adversarial flows).

4.3 Low-Rate Denial of Service Attack

TCP's congestion control mechanism, albeit robust to a range of network conditions, implicitly relies on end-system co-operation. This gives rise to a well known vulnerability, that has known to have been exploited by malicious high-rate unresponsive flows. However, these flows are easy to detect and remedial measures are relatively simple. On the other hand, low-rate attacks, which are just as harmful as the high-rate counterparts, pose much more of a threat since they are difficult for routers and counter-DoS measures to detect [15].

Deployment In an LDoS attack, an attack TCP stream is generated and made to flow through a buffer B that is shared with the target flow. The attack flow consists of short

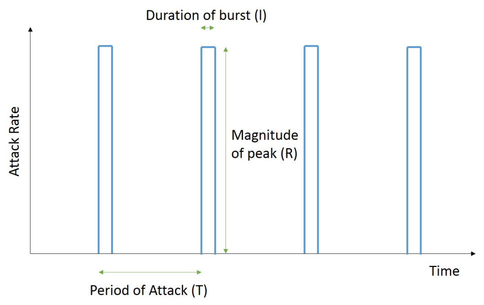


Figure 8: The attack stream with a periodic square-wave pat

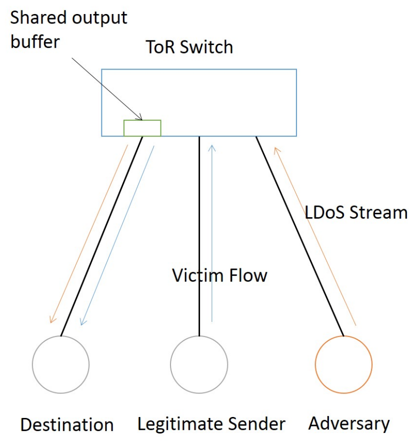


Figure 9: Schematic for low-rate DoS attack

bursts of packets that repeat after a chosen slow-timescale frequency. The stream has a periodic square-wave pattern composed of bursts of duration ℓ at a rate R that repeat every T seconds (see Figure 8). The parameters ℓ, R, and T are changed to achieve maximum impact. In particular, the burst rate R is kept sufficiently high to cause buffer overflows at B (which results in packet drops for the target stream) but low enough to avoid detection. Similarly, the burst duration ℓ is kept larger than the RTT of the target stream in order to ensure that all packets within the TCP congestion window for that RTT are dropped. Standard results from queuing theory show that if the buffer is cleared out at c Mbps, then to ensure a buffer overflow at B one must have

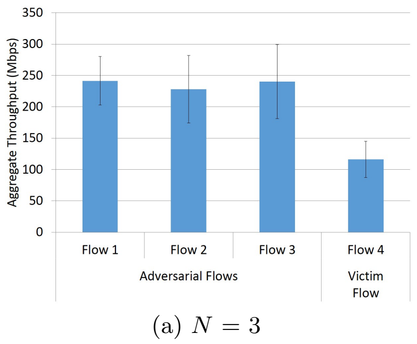
$$(R + R_{\text{TCP}} - c) \cdot \ell < |B|$$

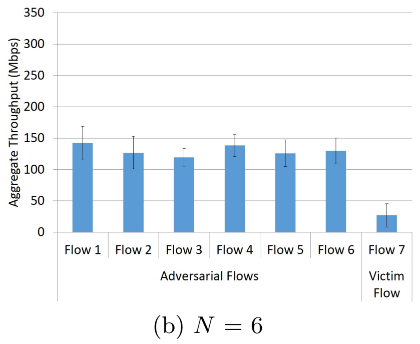
where R_{TCP} is the rate of the target stream.

Every time the target stream's packets are dropped, the victim waits RTO amount of time to retransmit another packet. Ignoring packet losses due to slow start in the target flow (i.e., the victim can send data at full rate for ($t -$ RTO) time in each period of the attack stream) the victim's throughput is expected to reduce to at least

$$\frac{T - \text{RTO}}{T} \times R_{\text{TCP}}.$$

Experimental results. We study the impact of LDoS attacks in DCNs. Our experimental setup is depicted schematically in Figure 9. The adversary issues periodic UDP bursts to the destination with burst duration $\ell = 20$ms and time period T varied from 20ms to 1000ms in 20ms intervals. The burst rate R is uncapped, and is allowed to saturate the available bandwidth. At the same time, the victim is-


(a) $N = 3$


(b) $N = 6$

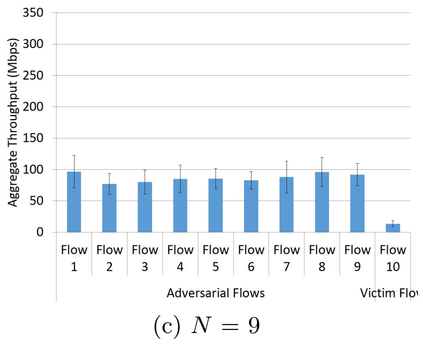
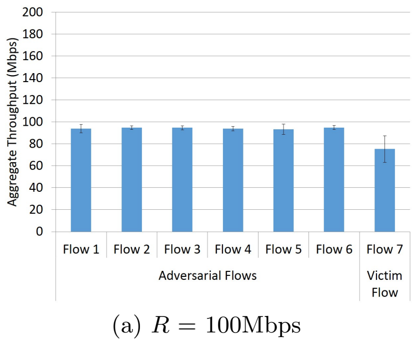
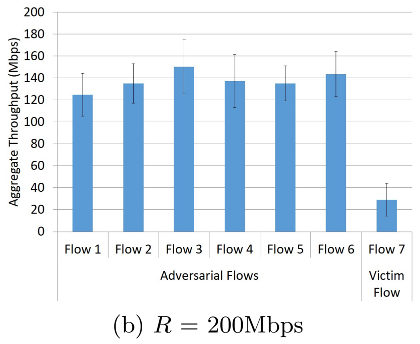

(c) $N = 9$

Figure 6: Drop in Victim flow throughput at $N = 3$, 6 and 9.


(a) R = 100Mbps


(b) R = 200Mbps

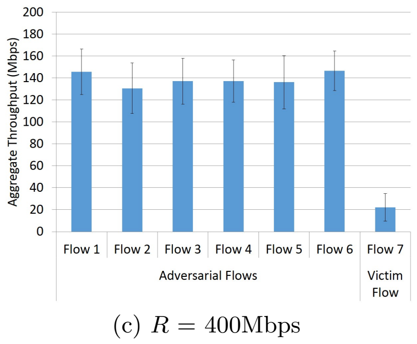

(c) R = 400Mbps

Figure 7: Drop in Victim Flow throughput at different traffic caps: R = 100, 200 and 400Mbps

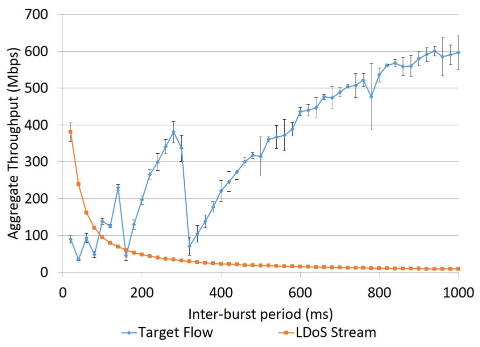


Figure 10: Variation of aggregate throughput of victim flow and attack stream with inter-burst period (T) (no traffic caps)

sues a TCP flow to the same destination. The victim flow and the LDoS stream share an output buffer at the switch port leading to the destination. The TCP RTO$_{\text{min}}$ is kept at 300ms, and the victim flow RTT is around 1ms.

The results of our experiment are shown in Figure 10. It is evident from the results that with the increase in the inter-burst duration (i.e., the time period of the attack stream) the *aggregate* throughput of the attack stream decreases, making it increasingly harder for it to be detected. Also, the maximum impact of the attack stream occurs at roughly around T = 300ms, 150ms, 100ms, etc. (i.e., when the inter-burst duration is a factor of the TCP RTO$_{\text{min}}$). This is explained as follows: every time the victim experiences packet drops, it waits for RTO$_{\text{min}}$ (or a multiple of RTO$_{\text{min}}$, as per the exponential back-off mechanism) amount of time before it retransmits packets. However, if the inter-burst duration is roughly equal to (or a factor of) the TCP RTO$_{\text{min}}$, then the attack bursts coincide with the victim's attempts to retransmit packets, throttling its throughput to a minimum.

Inferring RTO$_{\text{min}}$. We note that this analysis also provides a way to determine the best values for the parameter T for given ℓ and R: since the aggregate throughput for the attack stream keeps decreasing with increase in T, the value of T at which the last minimum for the victim throughput occurs ($T = T_{\text{ideal}}$) is ideal. As discussed before, T_{ideal} is roughly equal to the TCP RTO$_{\text{min}}$ and offers maximum victim throughput degradation at minimum attack stream detectability. This method is also useful to estimate the TCP RTO$_{\text{min}}$ ($\approx T_{\text{ideal}}$), an important network parameter as discussed in Section 3.1.

Distributed LDoS. In order to study the impact of traffic caps, we performed the above experiment with varying values of R. Both the attack rate R as well as the victim flow rate R_{TCP} were capped at different values, and the results obtained are depicted in Figure 11. The primary observation is that with lower traffic-caps, the throughput degradation is lower (almost non-existent for a traffic cap of 200Mbps). However, this provides us with another insight into increasing the impact of the traffic-capped attack: the use of multiple synchronized attack streams to carry out a *distributed* low-rate attack.

On using multiple attack streams at a traffic cap of 200Mbps (Figure 12), we see that the impact increases with the num-

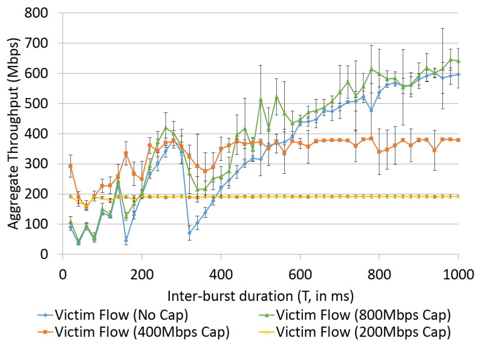


Figure 11: Effect of traffic caps on Victim flow throug

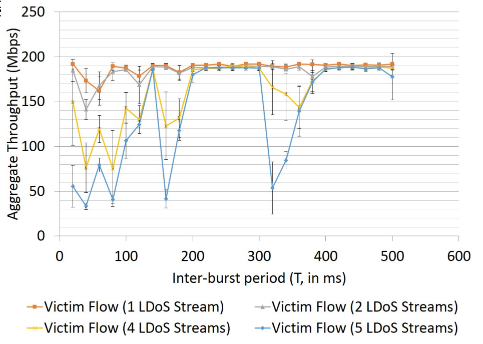


Figure 12: Effect of multiple attack streams on victim flow throughput at a traffic cap of 200Mbps

ber of synchronized attack streams employed. The victim throughput is reduced from 200Mbps to 50Mbps with 5 attack streams at $T = T_{\text{ideal}}$.

We also study the impact of burst-duration ℓ on the throughput degradation. We carried out the experiment as described above with burst rate R fixed at different values (R = 200Mbps in Figure 13 (a), R = 400Mbps in Figure 13 (b), R = 800Mbps in Figure 13 (c)), and burst-duration ℓ = 5ms, 10ms, 20ms. The victim sends data at full rate, i.e., attempts to saturate the available bandwidth. We see that at lower attack-rates (R = 200Mbps), increasing the burst-duration increases the victim throughput degradation significantly; however, at higher attack rates (R = 800Mbps) the throughput degradation remains the same at all ℓ values. Thus, the study of the trade-off between the burst-rate R and the burst-duration ℓ provides the adversary a means to minimize both the burst-duration and the burst-rate and still achieve maximum victim throughput degradation.

5. OVERT ATTACKS

Flood based attacks have been widely explored in the Internet scenario, where a targeted service is flooded with a large volume of traffic to overwhelm the network and deny service to legitimate users. These attacks are typically carried out by means of botnets, whereby a large number of compromised hosts are used to send the flood traffic to the victim service.

Deployment We explore the applicability of flood based attacks (which we categorize as *overt* attacks) to the Data Center Network, where a malicious tenant attempts to flood a target service or VM in the cloud from *within the DCN*. The basic schematic followed by the flood based attacks is described in Figure 14. A number of malicious VMs (Flood-Senders) are used to send a barrage of packets to a victim

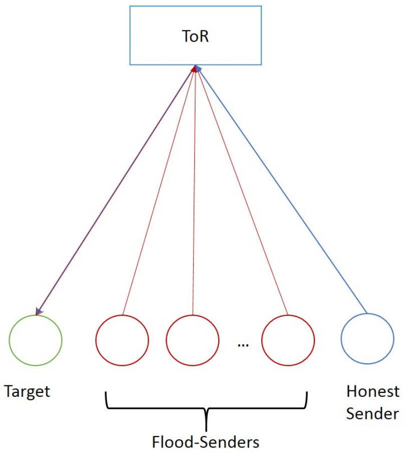


Figure 14: Schematic for flood-based attacks

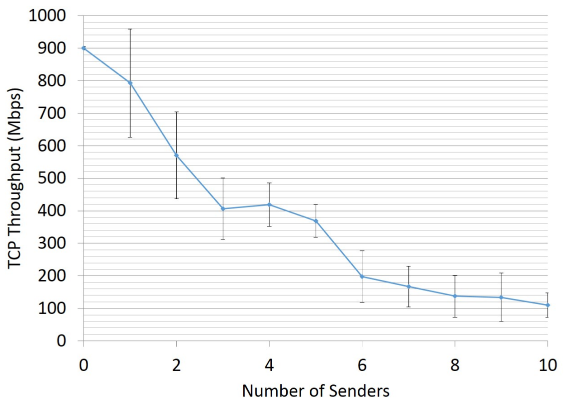


Figure 15: Victim throughput decreases significantly with increase in the number of senders

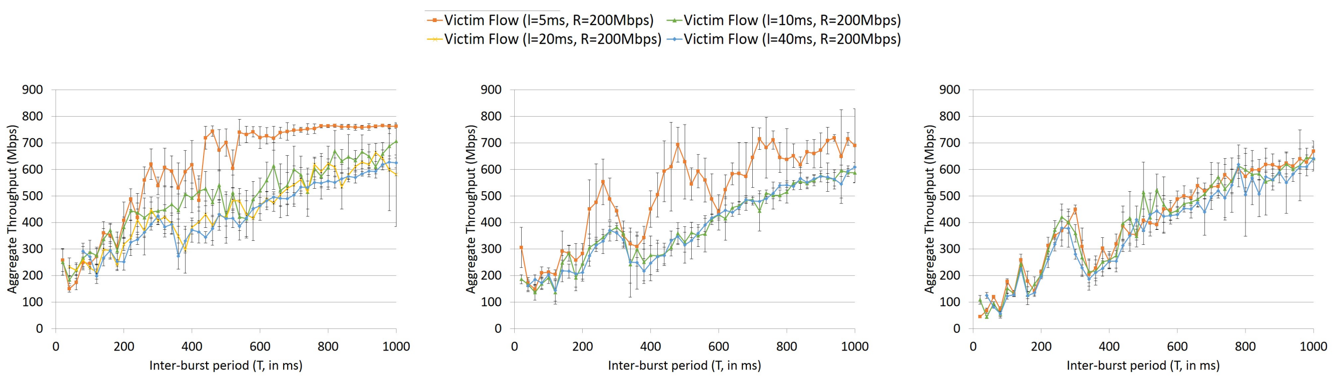
VM (Target). Meanwhile, any VM sending legitimate traffic (Honest-Sender) would experience denial of service.

5.1 Bandwidth Flood

The bandwidth flood relies on flooding the access link to the target VM with a flood of data packets. As a distributed attack, several malicious VMs generate a flood of traffic with the same target destination. Even with a traffic cap imposed on each VM, the adversary utilizes the distributed nature of the attack to not only saturate the victim access link saturated but also keep high buffer occupancy at the switch port leading to victim.

Experimental results. Our experimental setup to study the impacts of bandwidth flood comprised N malicious senders that send a barrage of non-spoofed TCP packets to a target host. Each sender issues the flood stream at a packet rate of R = 1000 packets/sec, with each packet having a payload of 1460 Bytes ($\approx$ 12 Mbps). The Honest-Sender attempts to establish a bulk TCP flow to the same target VM, and the denial of service experienced is quantified as the drop in TCP throughput experienced in the presence of the flood.

Figure 15 shows that drop in TCP throughput experienced is not proportional to the flood traffic; for instance, at 3 Flood-Senders, TCP throughput drops from 900Mbps



(a) $R = 200$Mbps (b) $R = 400$Mbps (c) $R = 800$Mbps

Figure 13: Effect of burst duration ℓ and burst rate R on victim throughput

(at no flood traffic) to 400Mbps, where as the cumulated flood traffic is only about 36Mbps. This is explained by the network-buffer overflows at the switch port leading to the target: the high volume of incoming traffic at the switch port results in packet drops for the both the well behaved TCP flow and the flood traffic; however, the TCP congestion control mechanism causes the victim flow throughput to fall by a considerable amount, while the malicious flood traffic continues at its fixed rate.

5.2 SYN Flood

The TCP SYN flood works to exploit TCP's three-way handshake mechanism to establish a connection between the server and a client. An adversary sends out a flood of TCP SYN packets to the server, and each of these packets is handled as a new connection request. This results in a large number of open connections and the server responds to each of the SYN packets with SYN-ACK packets. The half open connections saturate the number of available connections that the server is able to make, and the server is unable to respond to legitimate connection requests. In modern Operating Systems, protection against SYN flood include measures like dropping connections randomly when the number of half-open connections exceeds a certain threshold; this would still prevent a legitimate client from connecting to the server successfully during a SYN flood attack.

Experimental results. Our setup to study the impacts of the SYN flood attack consisted of N senders that send a flood of non-spoofed TCP SYN packets to the server at a packet rate of R packets/sec. We measure the number of connections in the SYN received state (i.e., the number of half open connections) at the server during the flood. Figure 16a shows the time-variation of the number of half-open connections at $R = 1000$ packets/sec, Figure 16b shows the variation at $R = 5000$ packets/sec and Figure 16c at $R = 10000$ packets/sec. The number of half open connections provides an estimate of the amount of resources used up at the server.

We see that the number of half-open connections at the server saturate with two or more senders flooding it with SYN packets. This is most likely a protection scheme provided by the operating system to prevent over-allocation of resources to the half-open connections; however, this also shows that the server is likely to drop incoming connection requests with higher probability under those flood conditions.

We also study the impact of the flood on a legitimate

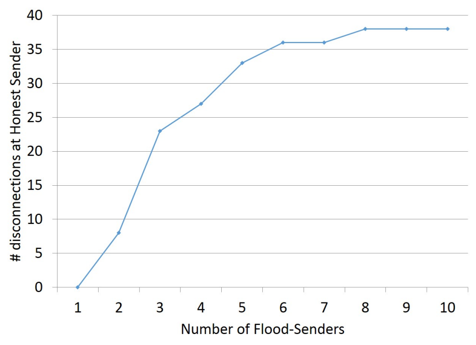


Figure 17: Number of failed connections saturates beyond 8 Flood-Senders

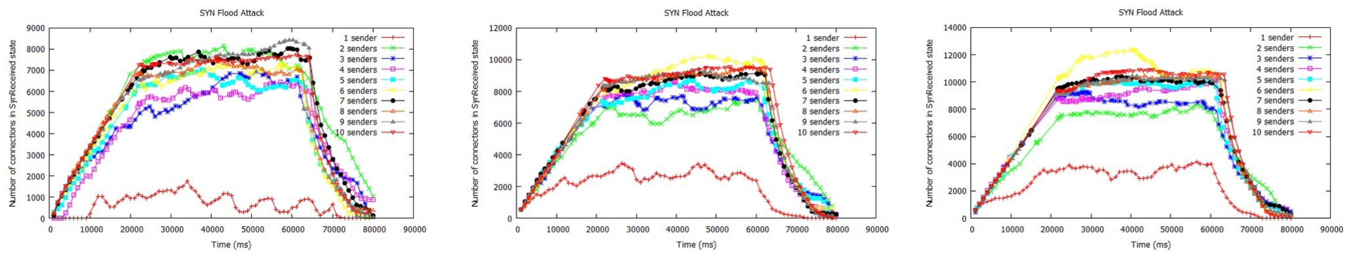
client attempting to connect to the server. We setup the SYN flood as before, but also include an honest client which continuously attempts to connect to the server; as soon as the the client establishes a connection successfully, it immediately disconnects and re-attempts connecting to the server. However, if the client is unable to connect, it waits for the connection timeout before it re-attempts connecting. We measure the number of disconnections experienced by the legitimate client in the duration of the flood (800s) at a flood rate $R = 1000$ packets/sec.

Figure 17 shows that the number of failed connections increases with increase in the number of Flood-Senders, and saturates at over 8 Flood-Senders, when the client is unable to establish any connection successfully. This provides a means to deny service to cloud-based services with a relatively small number of malicious VMs.

5.3 ARP Flood

The Address Resolution Protocol is ubiquitously employed to map network addresses (IP) to physical addresses (MAC). By means of this protocol, a sender that needs to determine the MAC address of a known destination IP node broadcasts an ARP request to every host in the network. Only the destination node responds to the request through an ARP reply in unicast mode. Under usual operation, only the intended destination responds to the ARP request. The protocol has proved to work well under normal circumstances - however, it was not designed to cope with malicious hosts.

One possible way to exploit the protocol is through ARP cache poisoning - where an adversary sends out spoofed ARP



(a) $R = 1000$ packets/sec (b) $R = 2000$ packets/sec (c) $R = 3000$ packets/sec

Figure 16: Time-variation of the number of half-open connections, at different flood-rates R

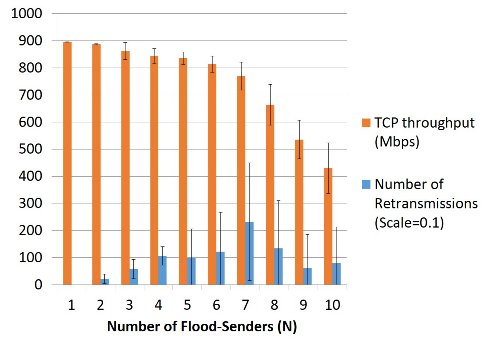


Figure 18: Impact of ARP flood on TCP throughput at flood rate $R = 10000$ packets/sec

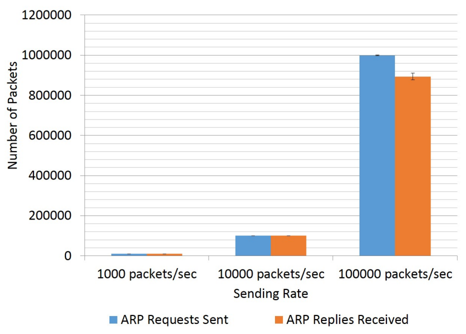


Figure 19: Comparison of ARP Requests sent vs. ARP Replies received at different R

packets to associate the its own MAC address to a target IP address, so that it can intercept all of the target's incoming traffic. However, this attack is infeasible on a Data Center network, since the switches and routers are configured to intercept and filter source-spoofed traffic.

We look at another ARP based attack that exploits the broadcast nature of ARP packets to cause 'broadcast storms' within the network. The adversary issues a flood of (non-spoofed) ARP packets requesting resolution of a particular IP. On reaching the switches, these packets are broadcasted to all the hosts in the subnet. This results in a 'storm' of ARP packets in the network, causing bandwidth saturation, resource utilization at the hosts and high buffer occupancy at the switches.

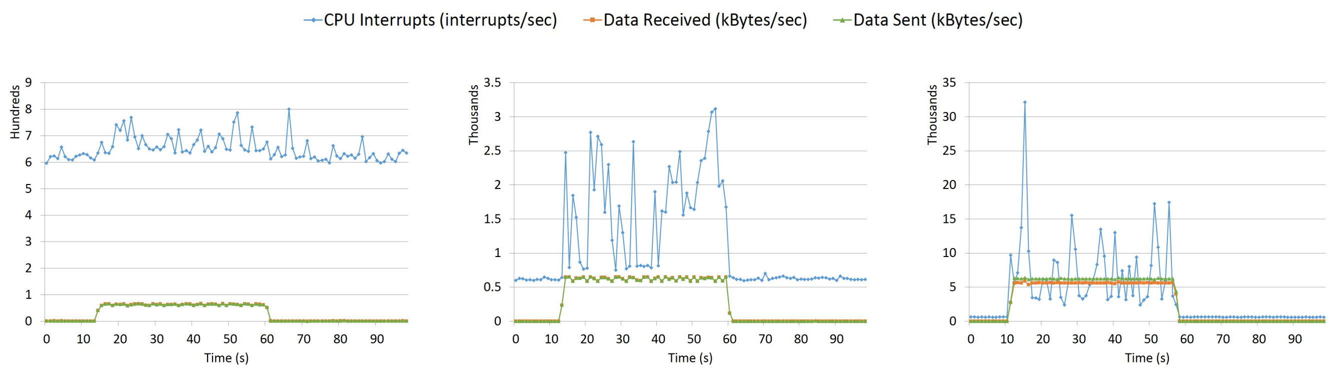
The ARP flood falls into the category of indiscriminate attacks due to their widespread impact. Although the malicious ARP requests are issued with a single specified IP to resolve, their broadcast nature causes them to affect all hosts within a particular subnet (which we verify shortly in the experimental results). However, the attack's indiscriminate nature is what adds to its overtness.

Experimental results. The experimental setup to study its impact comprised N malicious senders sending out a flood of ARP packets at a packet rate R, requesting the MAC address of a particular target IP. Meanwhile, the legitimate sender establishes a bulk TCP flow to the same target; as before, the impact of the flood is quantified as the drop in the TCP flow throughput during the flood.

Figure 18 shows the decrease in throughput of the TCP flow, along with the number of TCP retransmissions experienced at the legitimate sender, on increasing the number of malicious senders at $R = 10000$ packets/second. Each ARP request packet has a total size of 60 bytes, bringing the flood traffic volume to 4.5 Mbps per malicious sender. As with the bandwidth flood, the throughput drop for the victim flow is not proportional to the traffic, due to packet drops at the switch port leading to the target.

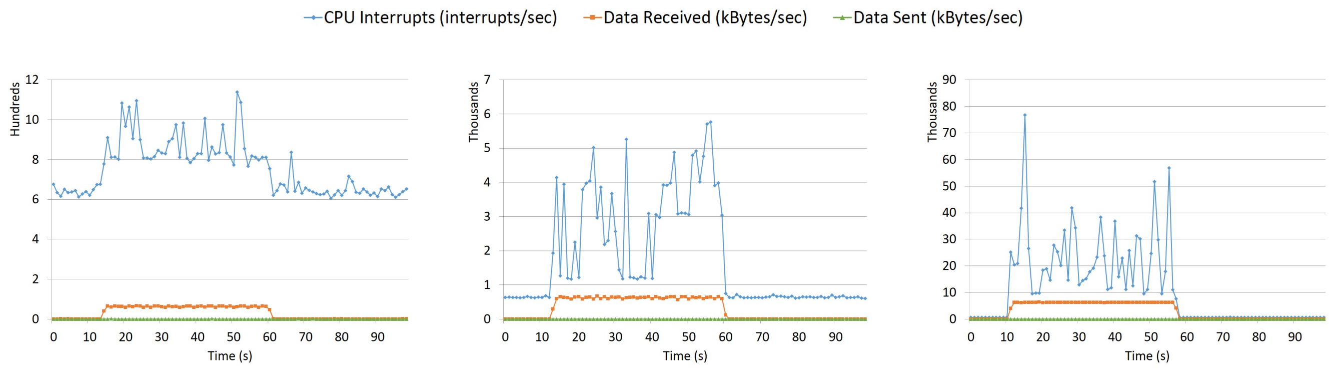
A comparison of the number of ARP replies received to the number of ARP requests transmitted yields interesting observations. Figure 19 shows the ARP packets sent and received at different sending rates R over a duration of 10s. We observe that the number of replies per request sent out decreases at extremely high sending rates, indicating a bottleneck of resources at the victim. We verify this by measuring CPU interrupts at the victim at the different flood rates (R) as shown in Figure 20. On increasing the flood-rate R at a fixed number of malicious senders ($N = 1$), the interrupts/sec experienced by the victim increases significantly, limiting the victim's ability to respond to each of the ARP requests at extremely high flood rates ($R = 100000$ packets/sec).

We also look at the spikes in CPU interrupts at hosts that are not the target of the malicious ARP requests, but receive them nonetheless due to the broadcast nature of these packets. Figure 21 shows the incoming and outgoing data traffic contrasted with the CPU interrupts that occur per second at one of these hosts. Thus, it is evident that even the non-target hosts experience significant bottlenecking of its CPU resources, as much of it is utilized in handling the interrupts as a result of the incoming ARP requests.



(a) R = 1000 packets/sec (b) R = 10000 packets/sec (c) R = 100000 packets/sec

Figure 20: CPU interrupts/sec at target host contrasted with incoming/outgoing data traffic due to ARP flood, at different flood-rates R



(a) R = 1000 packets/sec (b) R = 10000 packets/sec (c) R = 100000 packets/sec

Figure 21: CPU interrupts/sec at non-target host contrasted with incoming/outgoing data traffic due to ARP flood, at different flood-rates R

5.4 DHCP Flood

The Dynamic Host Configuration Protocol is another broadcast based network protocol widely used to configure network devices to enable communication over an IP network. DHCP has several modes of operation, but we focus on how DHCP is used for allocation of network addresses to hosts. Initially, a client broadcasts a DHCPDISCOVER packet containing its MAC address, on the physical subnet to discover available DHCP servers. The DHCP servers then respond with DHCPOFFER packets (either broadcast or unicast), each offering an available IP address to the client. The client may only respond to any one of these offers (if there are multiple offers) with a DHCPREQUEST packet, which is again broadcast, since it is meant for all DHCP servers. The chosen server then responds with either a DHCPACK (along with reservation of the requested IP address for the client) or DHCPNACK packet depending on whether or not the IP address is still available. If the response is a DHCPACK, the client tests out its new IP by sending out an ARP request for it: if no other client responds, the allocation is complete; otherwise, the client responds to the DHCP server with a DHCPDECLINE, declining the IP lease and restarts the whole process.

Again, one of the common attacks on DHCP involves a malicious client sending out multiple DHCP requests with spoofed MAC addresses, to exhaust the available IP addresses at the DHCP server, denying the allocation of IPs to legitimate clients. As we have already seen, Data Center networks are resilient against source spoofing.

We focus on another attack strategy which utilizes DHCP to create a broadcast storm, quite akin to the ARP based attack. We only rely on the first two steps of DHCP operation: the adversary sends out a flood of DHCPDISCOVER packets, which are broadcasted across the subnet, and the DHCP server(s) associated with that subnet respond with DHCPOFFER packets, which may be broadcast or unicast. Again, as with the ARP storm, the DHCP storm also falls into the category of indiscriminate attacks; the attack does not target a particular VM, but the subnet as a whole.

Experimental results. We use a setup similar to the previous flood based attacks to study its impacts: N malicious hosts broadcast a flood of non-spoofed DHCPDISCOVER packets at a packet rate R. Figure 22 shows the drop in throughput for a bulk TCP flow between two hosts in the same subnet, at different values of R and N. Increasing the number of senders and the packet rate causes bandwidth saturation and high buffer occupancy at the switch ports, resulting in a larger throughput drop.

We compare the number of DHCPDISCOVER packets

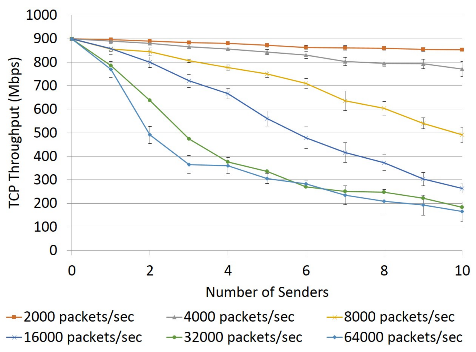


Figure 22: Impact of DHCP flood on TCP throughput

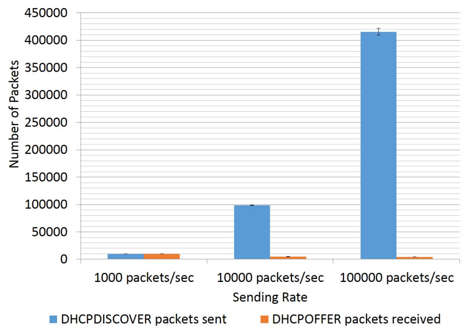


Figure 23: Comparison of DHCPDISCOVER packets sent vs. DHCPOFFER packets received at different R

sent out with the number of DHCPOFFER packets received by the client at different sending rates R. The results are depicted in Figure 23; we observe that the number of DHCPOFFER packets received in reply is much lower than than the number of requests sent out, as the DHCP server throttles the rate at which it responds to the requests with DHCPOFFER packets.

6. MITIGATING ATTACKS

A majority of the attacks discussed have significant impacts with moderate to low costs of deployment, exploiting shortcomings in the protocols that regulate network communication in the data center network. In this section, we outline possible ways for the mitigation of these attacks -Ű each has its pros and cons and pose several engineering challenges in its implementation; our aim in their discussion is to motivate the research community to work in this important area.

6.1 Network Virtualization

Network virtualization aims to improve costs and manageability of the network akin to server virtualization. One approach is to leverage proposals such as Open-Flow [17] that maintain rules for routing state in a logically centralized controller which can be queried at the time of connection-setup. The rules can be cached locally to minimize overheads. This approach can help prevent network based attacks, for example, by

(a) effectively converting broadcast (ARP/DHCP) requests into single lookups (e.g., via a directory service),

(b) optimizing traffic flow management by monitoring the types of flows (throughput-oriented or latency-sensitive) carried by each device/link and re-routing them on performance degradation, and

(c) managing ACLs at the data center level (in contrast to the current hypervisor-level at each server) to enable network isolation between services.

6.2 TCP Pacing

TCP pacing has been proposed to handle bursty flows that reduce network efficiency. The key idea is to evenly space transmitted data over an RTT instead of sending a single burst. This technique has been analyzed to provide better fairness and throughput; it stands to reason that TCP Pacing would mitigate the impacts of attacks that rely on DCN-specific unfairness among TCP flows. TCP Pacing has been shown to reduce the impact of the Outcast problem to an extent [20]. However, a key engineering challenge is to implement it in an environment with a high bandwidth-delay product, as in a datacenter. In particular, pacing can in fact worsen the throughput due to synchronized losses and delay of congestion signals. Moreover, it requires parameter tuning as the performance is sensitive to the choice of parameters. Techniques such as intelligent active queue management [4] can be explored to address some of these concerns.

6.3 Filtering Mechanisms

In any attack, the adversary relies on the ability of the attack traffic to reach its target host or victim, unfiltered at key points in its flow-path. It is inherently difficult to classify any data traffic within the network as malicious traffic, sans cases in which the traffic betray certain ill-disposed characteristics (such as source-spoofed packets); this makes the adversary's task significantly simpler. The situation demands a stricter monitoring of communication between the different VMs within the network.

A preliminary conclusion would suggest that such a mechanism should prevent malicious traffic from reaching their target VMs; this ideally requires that only pairs of trusted VMs should be allowed to communicate with each other in the DCN. While this can be ensured by filtering incoming malicious traffic at a host, an even better alternative would require that the malicious traffic be dropped at the source itself. There are challenges associated with both the approaches; we compare and contrast the two with regards to both their pros and cons.

Destination-based filtering. One possible approach is destination-based filtering: packets arriving at a node (that hosts several VMs) are checked to determine whether the intended destination of the packet has approved communication from the packet's specified source. This prevents the target VM from being overwhelmed by packets from untrusted sources, and this filtering can easily be implemented at the hypervisor or the host operating system (or typically as firewall rules for each VM, as in Azure [6] and Amazon EC2 [16]). However, this still allows the malicious packets to saturate the links and fill up the switch buffers along the path leading up to the target - virtually denying any legitimate communication to the target VM or service.

Source-based filtering. To overcome the shortcomings of the destination-based scheme, we propose a source-based filtering mechanism that detects and filters malicious traffic at source. As discussed, in an ideal situation, only groups of mutually trusted VMs should be able to communicate with each other. In our scheme, any network traffic that violates this is identified and filtered at source. We define *communication groups* as groups of VMs that are allowed to communicate with each other within the DCN. By default these groups are allocated according to the ownership of the VMs - all VMs belonging to the same owner (i.e., a single tenant) would form a closed communication group. These groups can be extended to include other trusted owners or tenants, upon verification by both parties.

The membership of VMs is authenticated by a Directory System, akin to that employed by VL2 [11] to maintain a mapping between application-specific addresses (names) to location-specific addresses (locators). Our Directory System maps the VM IP addresses to the communication group that they belong to; before issuing a packet, the host checks its local cache of directory entries to verify if the destination of the packet belongs to the same communication group as its source. If the directory entry is not present in the local cache, a query is issued to the Directory System, requesting the same. If the destination is verified to be in the source VM's communication group, the packets are transmitted, and dropped otherwise.

The primary advantage of this mechanism is that it prevents potentially malignant traffic from ever leaving its source, thereby minimizing instances of intentional link saturation and switch-buffer congestion. However, potential drawbacks include performance overheads and added latency associated with the directory lookups.

7. RELATED WORK

In this section we briefly present some of the research literature related to vulnerabilities in DCNs and mechanisms to ensure fairness in sharing the network.

Vulnerabilities in DCNs. Ristenpart et al. [22] describe inherent risks in public cloud platforms by exploring the feasibility of co-location - the ability to place VMs within the network that are resident on the same physical machine as a target VM. With extensive analysis, it is possible to map instance parameters to its assigned IP; this mapping is exploited to determine a likely host where the target VM may reside, and several instances are instantiated with similar parameters until co-location is achieved. Raiciu et al. [21] show the feasibility of deducing the network topology information in DCNs, despite the service provider's intentions to keep it confidential. The authors utilize a set of relatively simple network tools, including iperf, ICMP ping, and traceroute, to infer the network topology for an Amazon EC2 data center.

TCP has been shown to suffer from a number of issues in data center networks. Nagle et. al [18] introduced the TCP Incast Problem - the loss in throughput experienced by application employing barrier synchronized workloads in storage networks. Subsequent works [24], [10], [25] further explore the Incast problem, and Vasudevan et al. [24] suggests several mitigation mechanisms including high-resolution kernel timers and randomized timeouts.

Prakash et al. [20] discuss the TCP Outcast Problem, which arises when a small set of flows at one input port of a shallow-buffered switch competes with a large set of flows at another input port, for the same output port; in such situations, the smaller set of flows experiences significant throughput drop. The same work also describes several solutions to the problem, namely, RED, SFQ, TCP pacing and Equal Path Routing.

Implementing fairness in Network Sharing. Significant effort has been invested in developing a fair mechanism to share network resources in DCNs in recent works. SecondNet [12] and Oktopus [7] are mechanisms that provide static allocation of network resources per-tenant; while SecondNet employs rate-controllers at the hypervisor to ensure per-flow rate-caps, Oktopus utilizes virtual networks connecting the VMs to provide the tenant with a fixed share of network resources.

Sheih et al. [23] describe *(*Seawall) as a performance isolation scheme that allocates bandwidth to customers on a per-source basis; it assigns equal weights to all the sources communicating over a link and distributes the bandwidth accordingly.

Popa et al. [19] identify the primary requirements of bandwidth allocation in DCNs – Network Proportionality, Minimum Guarantee, and High Utilization – and showed that the difficulty in realizing fair allocation mechanisms lie in the tradeoffs associated with these requirements. The authors also describe different allocation mechanisms to the navigate these tradeoffs.

Alizadeh et al. [5] introduce DCTCP as a solution to performance impairments faced by a varied mix of workloads in data center networks. DCTCP modifies TCP to leverage Explicit Congestion Notification (ECN) for provide feedback to end hosts; this ensures low switch buffer utilization, alleviating impacts of TCP Incast and queue buildups that are characteristic of mixed workloads.

8. CONCLUSION

In this work, we show how the public cloud platform, due to its shared network infrastructure, is inherently vulnerable to attacks from a malicious tenant. The experiments on the various attacks and their results verify the feasibility of launching such attacks in a cloud environment, and allow us to compare them on the basis of their cost of deployment and the resulting detrimental impacts.

Based on the observations, we outline how common network schemes (TCP Pacing, Network Virtualization) can be employed to mitigate the impacts of such attacks. We also propose a new mechanism to regulate network communications in the data center network - a source-based filtering mechanism that scrutinizes network packets for malignant traits at their origin, and restricts network based communication to pre-defined communication groups. This inhibits the adversary's ability to direct malicious traffic to their targeted victims, adding a new degree of complexity to mounting attacks within DCNs.

9. REFERENCES

[1] Amazon ec2. http://aws.amazon.com/ec2/.
[2] Google compute engine. http://cloud.google.com/compute/.
[3] Windows azure. http://www.windowsazure.com/.
[4] AGGARWAL, A., SAVAGE, S., AND ANDERSON, T. E. Understanding the performance of tcp pacing. In *INFOCOM* (2000).

[5] Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. Data center tcp (dctcp). In *SIGCOMM* (2010).

[6] Amazon. Amazon web services: Overview of security processes.

[7] Ballani, H., Costa, P., Karagiannis, T., and Rowstron, A. Towards predictable datacenter networks. *SIGCOMM Comput. Commun. Rev.* (2011).

[8] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization. In *SOSP* (2003).

[9] Cerf, V. G., and Khan, R. E. A protocol for packet network intercommunication. *IEEE TRANSACTIONS ON COMMUNICATIONS* (1974).

[10] Chen, Y., Griffith, R., Liu, J., Katz, R. H., and Joseph, A. D. Understanding tcp incast throughput collapse in datacenter networks. In *WREN* (2009).

[11] Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. Vl2: a scalable and flexible data center network. *Commun. ACM* (2011).

[12] Guo, C., Lu, G., Wang, H. J., Yang, S., Kong, C., Sun, P., Wu, W., and Zhang, Y. Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *Co-NEXT* (2010).

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

[14] Jiang, H., and Dovrolis, C. Passive estimation of tcp round-trip times. *SIGCOMM Comput. Commun. Rev.* (2002).

[15] Kuzmanovic, A., and Knightly, E. W. Low-rate tcp-targeted denial of service attacks and counter strategies. *IEEE/ACM Trans. Netw.* (2006).

[16] Marshall, A., Howard, M., and Harden, B. Security best practices for developing windows azure applications.

[17] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.* (2008).

[18] Nagle, D., Serenyi, D., and Matthews, A. The panasas activescale storage cluster: Delivering scalable high bandwidth storage. In *SC* (2004).

[19] Popa, L., Kumar, G., Chowdhury, M., Krishnamurthy, A., Ratnasamy, S., and Stoica, I. Faircloud: sharing the network in cloud computing. *SIGCOMM Comput. Commun. Rev.* (2012).

[20] Prakash, P., Dixit, A., Hu, Y. C., and Kompella, R. The tcp outcast problem: exposing unfairness in data center networks. In *NSDI* (2012).

[21] Raiciu, C., Ionescu, M., and Niculescu, D. Opening up black box networks with cloudtalk. In *HotCloud* (2012).

[22] 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 *CCS* (2009).

[23] Shieh, A., Kandula, S., Greenberg, A., Kim, C., and Saha, B. Sharing the data center network. In *NSDI* (2011).

[24] Vasudevan, V., Phanishayee, A., Shah, H., Krevat, E., Andersen, D. G., Ganger, G. R., Gibson, G. A., and Mueller, B. Safe and effective fine-grained tcp retransmissions for datacenter communication. *SIGCOMM Comput. Commun. Rev.* (2009).

[25] Zhang, J., Ren, F., and Lin, C. Modeling and understanding tcp incast in data center networks. In *INFOCOM* (2011).