# Stealth Migration: Hiding Virtual Machines on the Network

S. Achleitner, T. La Porta and P. McDaniel
Pennsylvania State University
Computer Science and Engineering
{sachleitner,tlp,mcdaniel}@cse.psu.edu

S. V. Krishnamurthy
University of California, Riverside
Computer Science and Engineering
krish@cs.ucr.edu

A. Poylisher, C. Serban
Applied Communication Sciences
{apoylisher,cserban}@appcomsci.com

***Abstract*—Live virtual machine (VM) migration is commonly used for enabling dynamic resource or fault management, or for load balancing in datacenters or cloud platforms. A service hosted by a VM may also be migrated to prevent its visibility to an external adversary who may seek to disrupt its operation by launching a DDoS attack against it. We first show that current systems cannot adequately hide a VM migration from an external adversary. The key reason for this is that a migration typically manifests a traffic pattern with distinguishable statistical properties. We introduce two new attacks that can allow an adversary to effectively track a migration in progress, by leveraging observations of these properties. As our primary contribution, we design and implement a stealth migration framework that causes migration traffic to be indistinguishable from regular Internet traffic, with a negligible latency overhead of approximately 0.37 seconds, on average.**

## I. Introduction

Today, use of virtualized computing is more the norm than the exception. For purposes such as dynamic load balancing, or for fault management, virtual machines or even entire virtual operating systems are moved between servers within a datacenter or across distributed datacenter establishments.

In this paper, we ask the question: "Can a VM migration be identified by an external adversary if it transmitted between datacenter establishments over the Internet?". In a security-agnostic setting, an adversary can easily do so by examining the IP addresses and port numbers associated with migration flows, and in some cases, may even be able to examine the payloads in the flow.

There have been proposals for protecting VM migrations to some extent. Some of these for example, are based upon encrypting the payloads in such flows, or the establishment of virtual LANs [11], [12]. However, these are not fool proof; for example, an attacker can still potentially examine IP addresses and port numbers, even if the payloads themselves are encrypted. In order to hide the end-points of a migration, anonymous or onion routing techniques such as Tor [18] can be used. Unfortunately, as we show in this work, while this could work for general types of traffic, it may not suffice for VM migrations which, exhibit specific timing patterns; by observing these timing patterns (which manifest themselves even if the migration flow is mixed with other Internet traffic) an adversary can effectively track the flow and overcome anonymous routing.

Next, we ask: "How can we prevent an attacker from analyzing the traffic to infer VM migration patterns?". In other words, how can we design a stealth migration system to make VM migration traffic indistinguishable from generic Internet traffic flows? Unfortunately, one cannot blindly apply pre-existing traffic camouflaging techniques that have been previously proposed ([30], [14], [13]). This can adversely affect the service in terms of its performance (drastically increased downtimes) or create other signatures that can easily be identified by an adversary. Thus, as the first challenge, we need to implement traffic shaping techniques that are effective (make the migration traffic virtually indistinguishable from other traffic) and yet, keep the performance degradation experienced by the migration process to a minimum. Second, the stealth migration system has to dynamically, and fairly rapidly adapt to changes in variations in both migration traffic and normal traffic. Thus, it needs to be *adaptive* while still ensuring that the performance of the migration is unaffected.

As our main contribution, we design and implement a stealth migration framework that effectively addresses the above challenges. Our framework incorporates a novel adaptive transmission rate variation scheme along with highly dynamic and flexible traffic chaffing, to achieve extremely high accuracy in terms of making the migration traffic indistinguishable from normal traffic, but with minimum performance penalties. In more detail, the following is a summary of our contributions:

- ***Showcasing the vulnerabilities of VM migrations:*** We show that live VM migration can be effectively profiled via the statistical features of its flow.
- ***Design and implementation of a stealth migration framework:*** We design and build a stealth migration framework that intelligently combines transmission rate variation and chaffing to hide migration traffic from outsider adversaries that monitor flows.
- ***Experimental evaluations:*** We demonstrate that our framework virtually renders VM flow migration traffic indistinguishable from regular Internet traffic while only increasing the VM downtime by a miniscule 0.37 seconds on average.

## II. Background and Definitions

### A. Live VM migration

Most modern hypervisors (also called Virtual Machine Monitors or VMM) can support live VM migration. A key goal during migration is to minimize the *downtime* of the service provided by the VM. The *total migration time* refers to the duration between when a migration is initiated until the time when the original VM may be discarded. Different mechanisms can be used for live VM migration. A *pre-copy* approach is

978-1-5090-5336-0/17/$31.00 ©2017 IEEE

used by the hypervisors we use in our experiments; further details can be found in [17].

### B. Motivation: Distinguishable migration characteristics

While we defer a detailed discussion of our experimental testbeds and set up to later sections, we present a result which in essence, motivates our work. We inject a mixture of real Internet traffic traces in our test network and perform a live migration. We then present the observed traffic profile.

Specifically, in Figure 1 the distinguishable traffic characteristics of the VM migration (rate limited to 4MB/s) are shown. Note that these results are without our stealth migration system in place; we will refer to these migrations as *native migrations* in the rest of the paper. In more detail, the migration injects a constant periodic burst of packets onto the network. We analyzed approximately 80,000 network packet flows from a source *IP:port* to a destination *IP:port* with realistic Internet traffic (without migration flows), and compared it to the case with migration flows. We found that distinguishing features that reveal a migration from a regular flow include (i) variability in throughput and (ii) inter-packet transmission times. These features can be easily used in a maximum-entropy based detection algorithm and to perform pattern-matching, to detect VM migration flows. In Section IV, we show that one can identify over 90% of the VM migration flows (in our test set) with a false-positive rate of about 2%.

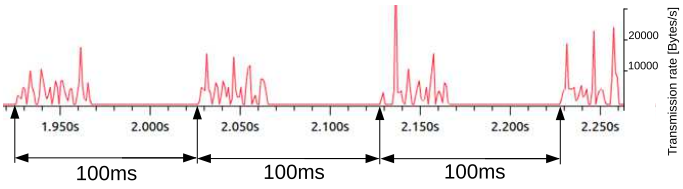


Fig. 1. Throughput characteristics of a live VM migration

### C. Discussion of state-of-the-art defense techniques

Recent published algorithms that enable traffic analysis attacks identify a packet stream based on its statistical characteristics. These traffic characteristics depend on features such as the unique distribution of inter-packet times or packet sizes. The proposed defense techniques against traffic analysis attacks introduced in recent publications, such as [30], [14] or [13], are often based on injecting dummy packets into a data stream or padding packets so all packets have the same size. These methods are effective in changing the statistical features exploited in a traffic analysis attack, such as in website-fingerprinting, but have the drawback of causing significant overheads. Applying these techniques on network traffic which includes VM migration flows, causes a significant latency overhead (300% on average with the system proposed in [13]). It is important to minimize the latency overhead since the process of live VM migration is very sensitive to increased latency, which can have a significant impact on the VM downtime as observed during our experiments and shown in [17]. The introduced delay of a defense system applied on live VM migrations can even cause a state where the migration process is not able to finish, leaving the sender and receiver in an idle state occupying resources on source and destination host. In comparison to existing approaches, our system only causes a latency overhead of 10-20%, which results in an additional VM downtime of 0.37 seconds on average. This makes our system a lightweight defense approach.

In addition, altering the size of packets in a VM migration flow is not necessary to hide it. According to the most recent Global Internet Phenomena Report [4], over 50% of today's Internet traffic consists of packets from video streaming platforms. Traffic from video-streaming applications, such as Youtube or Netflix, mainly consist of full-size TCP packets, [29], [10], as does VM-Migration traffic. However, as we discuss later and show in Figure 1, the timing characteristics of VM flows are unique, even compared to streaming and FTP flows, so attackers can distinguish VM flows based on these characteristics. This motivates our proposed attack- and defense-strategies which are focused on traffic analysis based on timing features.

## III. Experimental setup

To analyze virtual machine migration traffic on the network and perform experiments we used two different testbeds: (i) a small hardware testbed of Linux hosts connected with 100Mbit/s routers and (ii) a NS3 based network emulator for VM migration. By using two independent test-environments, we were able to confirm that the distinct traffic pattern of VM migration flows is caused by the actual migration process. To confirm that the traffic pattern of a VM migration is preserved even on Tor networks, we also performed multiple VM migrations over the real-world Tor network between two hosts located in two US states as discussed in Section IV-D.

### A. Virtualization platform

We use *libvirt* [6] as a virtualization management platform. It supports multiple hypervisors, such as KVM/QEMU, XEN, VMWare, LXC, OpenVZ and VirtualBox. In this paper we present data observed during the use of KVM/QEMU which is one of the most used open-source hypervisors. We omit the data for other hypervisors due to space issues. The distinct traffic pattern of VM migrations shown in Figure 1 is caused by the transmission speed control mechanism in the *libvirt* platform. Therefore the traffic pattern of VM migrations are not specific to certain hypervisors, but are manifested during VM migrations which are executed using the *libvirt* platform. We also confirmed in our experiments that the current state of a VM has no impact on the manifested migration traffic pattern on the network. Several cloud computing platforms including Eucalyptus, Nimbus or OpenStack [6] are based on libvirt. Multiple options for VM migration are offered by *libvirt*. We tested different combinations of these options during our experiments as listed in Table I.

TABLE I. Virtual Machine migration types and features

| | |
|---|---|
| **Migrated OSs** | Ubuntu 14.04 32/64, Alpine Linux 3.2 32/64 |
| **Apps on VM** | XAMPP, MySQL Server, Lighttpd, OpenOffice |
| **Hypervisor** | KVM/QEMU |
| **VM sizes** | 1GB, 150MB, 85MB |
| **Migration speeds** | 2, 4, 5, 6, 8, 10, 12MB/s |
| **Transmission types** | encrypted libvirt tunnel, TLS, unencrypted |
| **Authentication** | certificate, public key, username/password |
| **Migration types** | Live, Offline |

### B. Hardware testbed

To analyze virtual machine migration on actual network hardware we used a small setup of Linux hosts connected with 100Mbit/s routers to a local subnet. All hosts were running Ubuntu 14.04 with 64bits, and use the program *Virtual Machine Manager*, a user interface for *libvirt*, to trigger VM migrations.

*1) Background traffic:* For introducing background traffic into our hardware testbed we replay network traffic trace files from the Japanese WIDE project. The WIDE project operates a nationwide test-network in Japan focusing on research and development of global-scale networks specifically for data-centers and cloud computing systems. The WIDE project is maintaining a repository of traffic traces [8] from a 150Mbps upstream link to an Internet Service Provider. We used several of these traces from 2012, 2014 and 2015 as background traffic in our local test-network.

### C. Emulation testbed

In addition to analyzing virtual machine migration traffic on our hardware testbed, we also used a NS3 based network emulation testbed, to run experiments. Our test-setup consists of two server farms migrating virtual machines over the network and a man-in-the-middle adversary performing traffic analysis attacks.

*1) Background traffic:* Background traffic in the emulation testbed is generated based on historical Internet user activity patterns. This user model generates transitions between activities using a Markov transition matrix to predict the next activity a user will perform on the Internet. Such user activities include website browsing, blog interaction, e-mail, video streaming and downloading files via FTP. To generate this user activity model, we use a dataset as discussed in [19]. The resulting background traffic consists of realistic short- and longterm HTTP, TCP and FTP connections.

### D. VM types

We migrate different types of virtual machines over the network in our experiments. Table I provides a summary of the different operating systems and settings we used in our migration experiments. The settings like migration speed or transmission type can be tuned using the *libvirt* virtualization platform. For evaluating the attack channels and defense mechanism as we introduce in Section IV and V, different combinations of the migration types and settings presented in Table I were used.

## IV. Detection and Attack Channels

In this Section we introduce two highly effective, traffic analysis attacks to detect VM migrations on the network.

### A. Threat model

We consider an adversary using Internet route hijacking to place itself as a man-in-the-middle on routes over the Internet to intercept inter-data center VM migration traffic. It has been observed in recent reports [5] that such attacks are emerging and represent real security threats. There are multiple ways an attacker can place itself on the transmission channel. One such way is to hijack a route [2], [5]. A targeted attack on the migration process can be executed by an attacker as a counter-maneuver when VM migration is used as a proactive defense mechanism as discussed in [26]. In our experiments we discovered that a VM migration process is very sensitive to disruptions on the network. As we show in Figure 2, a simple malicious packet dropping attack, as discussed in [31], on the migration process is able to increase the service downtime by a factor of 3-6, or even prevent the process from finishing if more than 10% of packets are dropped.

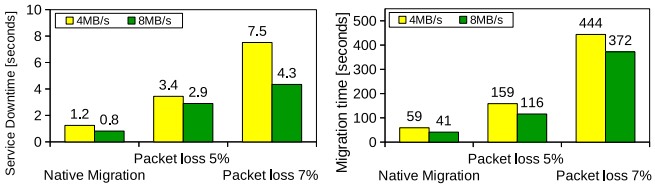


Fig. 2. Packet dropping attack on intercepted VM migration flow

The pre-condition for launching such a targeted attack is that an adversary is able to identify the exact VM migration flow on the network. In our experiments we assume that state-of-the-art security measures like encryption, onion routing and/or the use of secure tunnels are applied on the migration stream as discussed in the literature [18], [11], [12], [28]. Even if all these mechanisms are in place, an attacker can still perform a traffic analysis attack to detect a VM migration flow based on its traffic characteristics, and observe critical information such as the migration size, duration, transmission speed and the endpoint VMMs involved, as discussed in [20], [23] and launch a targeted attack on the migration flow. For detecting a VM migration flow and distinguishing it from realistic background traffic, we introduce two different traffic analysis attacks. First we consider an attacker who is able to actively monitor the network traffic and apply a pattern matching algorithm to identify a migration by its distinct traffic pattern. As a second attack we consider an adversary who is able to constantly monitor the network traffic. This attacker has a historical collection of migration flows which are used for training a classifier; the attacker then executes a maximum-entropy based detection algorithm to identify an ongoing VM migration, based on its traffic features.

### B. Pattern matching detection

We consider an adversary who can actively monitor the network traffic. Our experiments reveal that the very specific traffic patterns exhibited by migration flows are caused by the transmission speed control mechanism of the virtualization platform. The observed traffic pattern in different test environments are shown in Figure 1. For VM migration, these traffic patterns depend on features like byte counts per unit of time or inter-packet transmission intervals, and can be observed at different locations in the network, and are usually independent of the packet contents or encryption as discussed in [23]. The traffic pattern we observed in our experiments is characterized by bursts of packets which are sent out every 100ms. The length of the packet burst depends on the specified migration rate. Figure 1 shows the pattern caused by a VM migration with a defined transmission speed of 4 MB/s. We observed the same pattern in our hardware testbed (Section III-B), in the emulation testbed (Section III-C) as well as when migrating a VM over the real-world Tor network (Section IV-D). Being aware of this pattern, an adversary is able to apply a pattern matching algorithm to flows on the network and determine the presence of a VM migration.

*1) Pattern matching detection algorithm:* As shown in Figure 1, packet bursts transmitted at very constant time intervals are characteristic traffic patterns of VM migrations. Counting the number of burst intervals within multiple packet windows and calculating the variation coefficient of the distance between packet bursts, allows the detection of a migration flow. First, Algorithm 1 determines the start time of a packet burst and calculates the time duration between the previous and current

**Algorithm 1** PatternAttack($PacketFlow$,$P_{win}$,$\delta$,$PC_t$)

```
1: previousBurstTime = 0; BurstList = {}; patternCounter = 0
2: while PacketFlow do
3:     for all trafficFlow < IP : Port > in PacketFlow do
4:         if start of packet burst in PacketFlow then
5:             BurstList.add(1)
6:         else
7:             BurstList.add(0)
8:         end if
9:         if BurstList.length > P_win then
10:            Comment: Determine distance between spikes
11:            VariationList = {}
12:            BurstCounter = 0
13:            for all b in BurstList do
14:                BurstCounter++
15:                if b == 1 then
16:                    VariationList.add(BurstCounter)
17:                    BurstCounter = 0
18:                end if
19:            end for
20:            varCoef = StDev(VariationList)/Avg(VariationList)
21:            if varCoef < δ then
22:                patternCounter++
23:            end if
24:            BurstList = {}
25:        end if
26:        if patternCounter > PC_t then
27:            return Migration at connection IP : port detected
28:        end if
29:    end for
30: end while
```

packet burst. If the start of a packet burst is detected in a packet flow from source $< IP : port >$ to destination $< IP : port >$, a '1' is added to the list of bursts for that flow; else a '0' is added (lines 4-8). This results in a list of 0s and 1s; the distance between 1s reflects the distance between packet bursts. This is done for all packets within a specified packet window. In lines 9-19, all observed packets between two burst are counted and stored in a list. We then calculate the variation coefficient of the observed packets within the specified window in line 20. The variation coefficient $c$ is defined as the standard deviation divided by the mean:

$$c = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}}{\bar{x}} \quad (1)$$

Since the distance between two packet bursts of a migration flow has very little variance, a variation coefficient lower than a defined threshold $\delta$ (lines 21-23), of the seen packets between bursts, is an indication of a VM migration flow. If this pattern was seen more than $PC_t$ times (lines 26-28) Algorithm 1 returns that a migration flow is detected.

*2) Detection effectiveness:* Figure 3 shows the detection results with our pattern matching algorithm in terms of identifying VM migration packet flows on the network and distinguishing them from non-migration background traffic. We show the effectiveness of our pattern matching detection algorithm by comparing the number of packet flows of detected migrations, the total number of migrations, the false-positives (on the left y-axis) and the number of non-migration packet flows (on the right y-axis). The values on the bars in Figure 3 show a comparison of the number of packet flows in our experiments to evaluate the detectability of VM migration based on its traffic pattern.

TABLE II. COMPARISON OF VARIATION COEFFICIENTS OF TRAFFIC FLOWS FOR DIFFERENT PACKET WINDOWS

| Type | 100 | 200 | 300 | 500 | 1000 | 2000 | 3000 | Load | Speed |
|---|---|---|---|---|---|---|---|---|---|
| TCP | 0.862 | 0.890 | 0.865 | 0.932 | 1.205 | 1.104 | 1.104 | | |
| TCP | 0.393 | 0.407 | 0.494 | 0.466 | 0.715 | 0.715 | 0.674 | | |
| TCP | 0.615 | 0.709 | 0.599 | 0.707 | 0.366 | 0.503 | 0.336 | | |
| Nat. Mig. | 0.003 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | H | 8MB/s |
| Nat. Mig. | 0.146 | 0.119 | 0.114 | 0.113 | 0.107 | 0.097 | 0.098 | M | 8MB/s |
| Nat. Mig. | 0.060 | 0.039 | 0.030 | 0.021 | 0.017 | 0.012 | 0.008 | S | 4MB/s |

Our pattern matching detection algorithm is able to identify about 80% of VM migration flows on the network, while maintaining a false positive rate of approximately 1-2%. As shown in Figure 3, the detection rate based on pattern matching depends significantly on the algorithm parameters, especially on the chosen packet window. The observed results suggest that the use of a packet window of 1000, achieves the best results. For the shown results, we used values of $\delta = 0.05$ and $PC_t = 3$ to adjust the sensitivity of the algorithm. We wish to point out here that with the above pattern-matching algorithm, no historical training data is required.

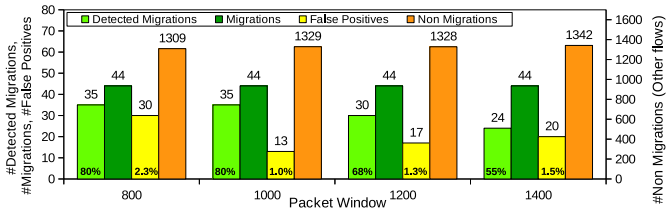


Fig. 3. Detection results with the pattern matching algorithm

### C. Maximum entropy based detection

For our second migration detection attack we also consider an attacker positioned on the communication channel. Once on the migration path, the attacker uses an anomaly detector to identify a VM migration. Anomaly detectors are trained to identify specific types of traffic anomalies by inspecting traffic flows as discussed in [24], [25], [22]. Traffic anomalies are specified by features that reflect a distinct traffic pattern which distinguishes them from the rest of the traffic. A network anomaly detector can be trained on distinct features of traffic patterns to flag a packet flow when it matches the predefined model.

*1) Maximum entropy based detection algorithm:* For this detection attack, we use the principle of an anomaly detector for a malicious purpose, i.e. as a method to identify a VM migration flow on the network. Our anomaly detection based attack algorithm uses maximum entropy estimation to identify a VM migration flow. We use the *MAXENT Library* which is part of the *Apache OpenNLP* [1] package to create a detection model for VM migration flows. The most important part for creating a detection model is to identify a distinct feature of a network flow. We determined that native migration traffic flows (Nat. Mig.) have a very low variability in terms of the number of transfered bytes within a specified window of packets, compared to common TCP packet flows as we show in Table II. We use the recorded network traffic from our test network for training and evaluating our migration detection model. To measure the variability of a flow we calculate the variation coefficient of the throughput within a specified window of packets. In Table II we show the variation coefficient of the

throughput for different packet window sizes, as determined by our maximum-entropy detection algorithm viz. 2.

**Algorithm 2** MaximumEntropyAttack($P_{win}$)

```
1: HashMap ByteWindow < Window, List >
2: HashMap Flows < IP : port, ByteWindow >
3: while PacketStream do
4:    Comment: Determine transmitted bytes within window
5:    for all Flow IP : port in PacketStream do
6:       for all Window w in P_win do
7:          List bytes = ByteWindow.get(w)
8:          bytes.add(transmitted bytes in w of IP : port)
9:          ByteWindow.put(w, bytes)
10:      end for
11:      Flows.put(IP : port, ByteWindow)
12:   end for
13:   Comment: Calculate detection features
14:   for all IP : port in Flows do
15:      HashMap bw = Flows.get(IP : port)
16:      Features = {}
17:      for all Window w in bw do
18:         List bytes = bw.get(w)
19:         varCoef = StDev(bytes)/Avg(bytes)
20:         Features.add(varCoef)
21:      end for
22:      Comment: Determine prediction with pre-trained model based on
         calculated features
23:      if Predict(Features)[Migration] ≥ 0.5 then
24:         return Migration at connection IP : port detected
25:      end if
26:   end for
27: end while
```

We compare common Internet traffic flows with native VM migration flows. The column labeled *Load* in Table II shows the background traffic load. We use the categories H(heavy = $> 80$ % traffic), M(medium = $40 - 50\%$ traffic) and S(small = $< 20\%$ traffic) to describe the traffic load in the network while the migration was performed. The last column shows the transmission speed of the VM migration flow. As shown in the table, common Internet traffic flows, which are using the TCP protocol, have much higher variation coefficients of their transmission rates compared to native VM migration flows. We compare VM migrations to TCP flows, since the TCP protocol is used by the Linux KVM VMM when a migration is performed. We assume an attacker is performing some sort of basic traffic flow filtering for migration detection, based on known facts about VM migration (such as only considering TCP streams and discarding flows that are too short to be a migration). For each flow (IP:port), our algorithm determines the number of transmitted bytes within consecutive windows of packets passing through the detection system. For our experiments, we use different window sizes between 100 and 3000 packets. As shown in Algorithm 2, lines 5-12, for a flow defined by *IP:Port*, we determine the number of transmitted bytes within a window of packets and store it in a HashMap. For each flow, we calculate the variation coefficient of transmitted bytes within different packet windows, as shown in lines 14-21. This procedure results in a set of features for each flow.

To specify a feature set in our detection model we use a triplet of three different packet window sizes. In this paper we present results for the following sets of packet windows: $P_{win}$ = {{*100,200,300*}, {*100,500,1000*}, {*500,1000,2000*}, {*1000,2000,3000*}}. The resulting feature triple for each flow is passed to our pre-trained detection model to determine a migration prediction for a particular flow (line 23). In our experiments we use a threshold of 50% for the prediction result, to decide if a flow matches the features of a VM migration or not. If our maximum entropy based detection model determines a value greater than 50% that the corresponding packet flow matches the migration features, we mark the flow as a VM migration.

*2) Detection effectiveness:* Our introduced maximum-entropy detection attack is able to identify approximately 90% of VM migration packet flows that are integrated with non-migration packet flows. We note that our detection algorithms only identify migration flows by their traffic characteristics and do not analyze packet content, port numbers, host addresses, etc. Traffic flows like large file transfers can have similar features of a low throughput variability, causing false-positives for an attacker. In our detection attack experiments we experienced a false-positive rate of about 1-2%. Figure 4 shows the effectiveness of our maximum-entropy based detection attack. The values on the bars in Figure 4 show a comparison of the number of packet flows in our experiments to evaluate the detectability of a VM migration based on its traffic pattern. For the results in Figure 4, we use 80% of traffic data randomly chosen for training and the remaining 20% for testing the detection system. Our maximum-entropy based VM migration detection algorithm is able to detect about 90% of the VM migrations with a false positive rate of about 1-2%. These results show that an attacker targeting VM migrations is able to identify 90% of migration flows only based on their traffic characteristics. In this case, using encryption, or anonymous routing techniques to protect the VM migration would not prevent an attacker from detecting it and identifying VMM endpoints which can be used to launch further attacks as we discuss in Section IV-A.

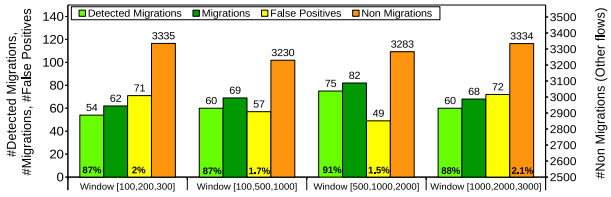


Fig. 4. Detection with a maximum entropy based detection system

## D. Detecting VM migration in Tor networks

Previous work has shown that low-latency anonymous communication networks, such as Tor, are vulnerable to traffic analysis attacks [15], [27]. Recently the authors of [15] show that the endpoints of anonymous traffic flows can be identified with an accuracy of 81% in a real-world Tor network. We show in our experiments that traffic analysis attacks are able to detect VM migration flows with an accuracy of 90%, even if anonymization techniques, such as the Tor network, are used. To demonstrate this, we setup an experiment to perform a VM migration over the Tor network between two hosts located in different US states. Both hosts were running a recent Tor version (*0.2.4.27*) on a Linux OS. We used the command *torify* to send the VM migration traffic over the Tor network to its destination host. As in our previous experiments, *libvirt* was used as the virtualization platform. Our migrations through Tor were performed over three hops, the default number in the real-world Tor network, to anonymize source and destination

of the VM traffic. We record the migration traffic at the migration destination after it exits the Tor network. Analyzing the recorded migration traffic, we confirm that the distinct pattern of a VM migration, is preserved while progressing through the Tor network and results in the same traffic pattern as shown in Figure 1. Observing the distinct pattern of VM migrations at different locations in the Tor network, an adversary can track the packet flow and can determine the source and destination of the VM migration.

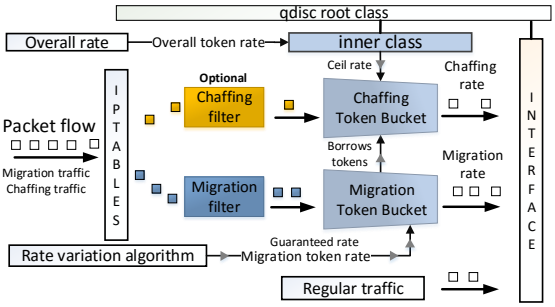


Fig. 5. Dynamic Hierarchy Token Bucket implementation

## V. Stealth Migration System

To defend against traffic analysis attacks on VM migrations over unprotected networks, we seek to make the migration packet flow indistinguishable from other traffic. Towards this, we design a novel stealth migration framework which applies different camouflaging techniques on the migration flow to achieve a similar traffic profile comparable to common background traffic, while limiting the impact on the migration process performance. The core component of our stealth system is a *Dynamic Hierarchy Token Bucket (DHTB)* which adjusts the transmission rate at high frequency and is able to balance the load intelligently across the migration traffic, and chaffing traffic that is injected when there is very low background traffic load. The transmission rate of the VM migration flow is dynamically adjusted by a *Rate Variation Algorithm* within the DHTB. The *Chaffing Generator* creates an adjustable parallel stream of dummy packets in a flexible and dynamic manner. The composition of these techniques generate a traffic flow with a randomized pattern, while avoiding high increases in latency which would negatively affect the migration flow.

### A. Hierarchy Token Bucket (HTB)

A Hierarchy Token Bucket is in essence a traffic shaper; an inherent scheduling algorithm is used to transform a packet flow into a defined form. Further details about the theory of hierarchy token buckets can be found in [9].

*1) Dynamic Hierarchy Token Bucket (DHTB):* In our stealth system we extend the concept of a HTB with a dynamic rate variation algorithm and refer to it as Dynamic Hierarchy Token Bucket as shown in Figure 5. In our implementation, the token arrival rate for migration traffic is dynamically adjusted by our rate variation algorithm, viz. 3, with a Linux system command. The different classes of our DHTB can be seen as multiple Token Buckets that are used in parallel and are able to borrow tokens from each other. Our DHTB also includes two filters and iptables rules, to distinguish the migration and chaffing traffic based on their specified port numbers and route it to their appropriate class. Regular non-migration traffic transmitted from the migration host, is sent into the network without any manipulation. Our rate variation algorithm dynamically adjusts the token arrival rate of the migration token bucket and helps shape the traffic into a randomized pattern.

### B. Dynamic rate variation

The purpose of dynamic rate variation is to adjust the token arrival rate of the DHTB to vary the transmission rate of migration packets and make the flow indistinguishable from common network traffic. We measure the traffic variability in a packet flow by computing the variation coefficient of the flow throughput for different packet window sizes. A comparable variation coefficient, similar to common Internet traffic, should be achieved, while limiting the impact on the VM migration process. As defined in Equation 1, the variation coefficient is calculated of a series of $n$ numbers, where $x_i$ represents a transmission rate previously used in our DHTB to send migration packets over the network.

In our defense system, we chose an approach to dynamically determine the migration packet transmission rate by generating a high variability within the given transmission bounds while introducing randomness into the rate selection process as shown in Algorithm 3. This makes the pattern of a migration stream unpredictable while creating a traffic variability similar to common Internet traffic.

To execute our rate variation algorithm, viz. 3, the lower and upper transmission rate bounds ($minR$,$maxR$), the rate update interval ($int$), the number of previous stored rates ($win$) and the start rate ($r$) have to be supplied. The upper and lower variation coefficient $uC$ and $lC$ are used to specify bounds on the preferred variation coefficient. To determine the next transmission rate for the migration flow, the algorithm compares the current and previous variation coefficients and decides how the current transmission rate should be adjusted (lines 9-15). Depending on the difference between previous and current variation coefficients, defined as $d$, the algorithm determines if the current rate $r$ should be increased or decreased by a specific factor. We make the resulting rates of our algorithm unpredictable by using randomized values within the predefined bounds. The value $c$ on line 17 triggers a random change in the direction, and ranges of the random values $([x_1, y_1], [x_2, y_2], [x_3, y_3], [x_4, y_4])$ in lines 21-35 control the adjustment of the rate change factor.

The variability of the generated transmission rates is controlled by these ranges and should be adjusted according to the specific application scenario of the stealth system. For example, in a network were the majority of the traffic has a higher variability, the numbers should be calibrated to make the migration traffic indistinguishable from the common background flows. The ranges $([1.0, 1.4], [1.4, 1.9], [0.8, 1.0], [0.1, 0.8])$ are used for the results presented in this paper, which were calibrated during our experiments. This calibration aims to achieve a flow variability between $lC = 0.5$ and $uC = 1$ that matches the variation coefficient of TCP streams we found, in reported prior analysis of realistic Internet traffic (Table II).

For the presented results, we change the transmission rate of our DHTB every $10ms$ and keep track of the previous 50 transmission rates. In Figure 6 we show the transmission flow of a stealth VM migration. The introduced rate variation algorithm generates a randomized traffic pattern that has a similar variation coefficient as a common TCP flow and can

therefore not be identified by an attacker analyzing the network traffic for flows with VM migration characteristics.

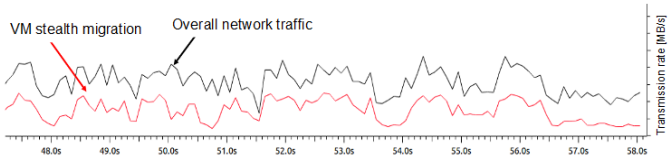


Fig. 6. Transmission variation of a stealth VM migration

### C. Chaffing

We refer to *chaffing* as the technique of injecting a parallel stream of dummy packets into the network, to camouflage changes in the overall network load indicating the start- and end-points of a VM migration. We use this defense technique as an optional defense method which is primarily applied in a network with a low traffic load to hide sudden changes in the network traffic which could reveal the presence of VM migration flows. For constructing a dummy packet generator, we used existing tools, those in [7] and [3]. In addition to creating dummy packets at the migration source, a dummy virtual server should be used on a remote machine to generate appropriate response packets when dummy TCP packets are introduced. We also change the address and port numbers of chaffing packets at random time intervals to make it appear that normal connections are established to different hosts.

## VI. Evaluation of Indistinguishability

The potency of traffic analysis attacks in identifying a specific packet flow by its statistical characteristics depends on the traffic pattern of the flow. This traffic pattern of a VM migration can be specified in its entirety by the distribution of its inter-packet arrival times as discussed earlier in Sections II-B and II-C. To evaluate the indistinguishability of our stealth traffic, we analyze the distribution of inter-packet arrival times of VM migration flows. We define time categories and count the number of packets which are associated with a specific inter-packet time, in each category.

A category of inter-packet times is defined by a lower time bound $t_l$ and an upper time bound $t_u$ where $t_u$ = $t_l$ + 1 microsecond. For our analysis we considered time categories starting from $t_l$ = 0 microseconds up to the highest seen inter-packet time within a flow. To evaluate the differences in traffic patterns of migration flows and non-migration flows, we compared the distribution of inter-packet times of the 10 most common time categories of traffic flows. To categorically demonstrate the indistinguishability of stealth migration traffic from common background traffic, we show that the distributions of inter-packet times of stealth- and background traffic are statistically indistinguishable. Statistical indistinguishability is achieved when the statistical distance between two distributions is negligible as defined in [21]. We define $\chi = \{X_n\}_{n\in N}$ and $\psi = \{Y_n\}_{n\in N}$ as sequences of probability distributions. We say that $\chi$ and $\psi$ are statistical indistinguishable if $\Delta(X_n, Y_n) = negligible(n)$.

To show that our stealth system shapes the migration traffic into a statistically indistinguishable form, we perform a *Chi-Square* test over the average inter-packet time distribution of native migration, stealth migration and common TCP and FTP flows. We define a significance level of $\alpha = 0.05$ to decide if the null-hypothesis of a test result should be rejected or accepted. If the result of the *Chi-Square* test returns a probability $> \alpha$, we accept the null hypothesis which shows that there is no statistically significant difference between two traffic flows.

**Algorithm 3** RateVariation($minR$,$maxR$,$int$,$win$,$r$,$d$,$c$,$lC$,$uC$)

```
rates = {}; prevVarCoef = 0.0; varCoef = 0.0; direction = true; changeRate = 1.0
while Stealth enabled do
    prevVarCoef = varCoef
    calculate stdDev = StandardDeviation(rates)
    calculate avg = Average(rates)
    calculate varCoef = stdDev/avg
    Comment: Change direction if the difference to the previous variation is greater than d
    if (prevVarCoef - varCoef) > d then
        direction = !direction
    end if
    Comment: Change direction if variation didn't change
    if (prevVarCoef equals varCoef) then
        direction = !direction
    end if
    Comment: Randomly change direction with a chance of c%
    if (RandomNumber(0, 100) ≤ c) then
        direction = !direction
    end if
    if direction then
        if varCoef < uC then
            changeRate = RandomNumber([x1, y1])
        end if
        if varCoef < lC then
            changeRate = RandomNumber([x2, y2])
        end if
    else
        if varCoef < uC then
            changeRate = RandomNumber([x3, y3])
        end if
        if varCoef < lC then
            changeRate = RandomNumber([x4, y4])
        end if
    end if
    r = r * changeRate
    if r < minR then
        r = minR
    end if
    if r > maxR then
        r = maxR
    end if
    Set DHTB migration class rate to r
    if rates.size() > win then
        remove oldest entry from rates
    end if
    rates.add(r)
    wait(int)
end while
```

TABLE III. Chi-Square test results to show indistinguishability of stealth migrations

| Compared flows | Result | If $> \alpha$ |
|---|---|---|
| Native migrations <> TCP flows | 0.019 | Distinguishable |
| Native migrations <> FTP flows | 0.038 | Distinguishable |
| Stealth migrations <> TCP flows | 0.999 | Indistinguishable |
| Stealth migrations <> FTP flows | 0.999 | Indistinguishable |

In Table III we show the *Chi-Square* test results of comparing the distributions of the average observed inter-packet times of migration and background traffic flows. The resulting values of comparing the traffic pattern of native VM migration flows to common TCP and FTP traffic flows observed in our test-environments are less than the defined significance level $\alpha$ and therefore distinguishable. The test result of stealth migration flows show a considerably higher value than the

defined significance level and thus these flows are statistically indistinguishable from background traffic.

## VII. Evaluation of defense techniques

To demonstrate the effectiveness of our defense techniques, we applied the introduced traffic analysis algorithms on VM migration flows, after they pass through our stealth migration system.

TABLE IV. Comparison of variation coefficients of traffic flows for different packet windows

| Type | 100 | 200 | 300 | 500 | 1000 | 2000 | 3000 | Load |
|---|---|---|---|---|---|---|---|---|
| Nat. Mig. | 0.074 | 0.057 | 0.047 | 0.037 | 0.025 | 0.008 | 0.007 | M |
| TCP | 0.752 | 0.842 | 0.859 | 0.870 | 0.870 | 0.945 | 0.686 | |
| TCP | 0.746 | 0.913 | 0.965 | 1.044 | 0.955 | 0.726 | 0.453 | |
| Ste. Mig. | 0.634 | 0.657 | 0.653 | 0.655 | 0.639 | 0.600 | 0.572 | S |
| Ste. Mig. | 1.074 | 1.049 | 0.923 | 0.826 | 0.680 | 0.453 | 0.312 | M |
| Ste. Mig. | 0.718 | 0.744 | 0.751 | 0.754 | 0.743 | 0.712 | 0.687 | H |

### A. Evaluation of dynamic rate variation

By dynamically adjusting the transmission rate for migration traffic (using the rate variation algorithm described in Section V-B), our stealth migration system effectively modulates the distribution of inter-packet times in a way that it cannot be distinguished from non-migration packet flows. In Table IV we compare the variation coefficients for different types of flows via experimentation on our hardware testbed. We find that the variation coefficients for stealth migration flows (Ste. Mig.) are very similar to that of non-migration TCP flows. The last column in Table IV characterizes the background traffic load (*Load*) in the network during the migration process. We use the categories H(heavy = $\sim 80\%$ traffic), M(medium = $40-50\%$ traffic) and S(small = $< 20\%$ traffic). Our stealth system performs well *even with* different coexisting network loads; the observed variation coefficients are always comparable to common TCP flows.

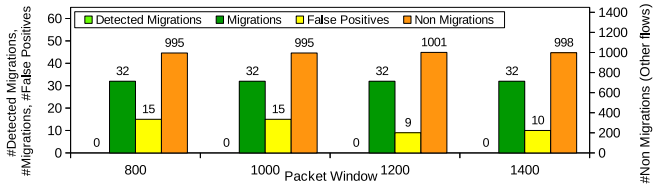


Fig. 7. Detection of stealth VM migrations with our pattern matching attack

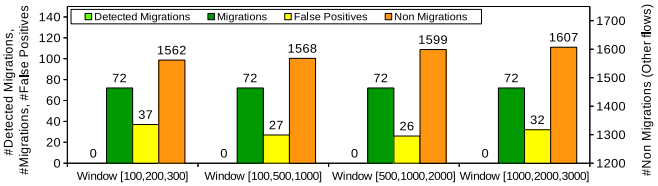


Fig. 8. Detection of stealth VM migrations with our maximum entropy attack

In Figure 7 we show the results from our experiments, when the pattern matching algorithm is applied on stealthy VM migration flows. The pattern matching attack, was not able to identify any stealthy VM migrations, since the traffic pattern is randomized and cannot be exploited. In Figure 8 we present detection results of the maximum-entropy based detection algorithm when our stealth system is applied on VM migration flows. Again, the attack cannot distinguish the migration traffic from non-migration traffic, since the stealth system effectively camouflages the traffic pattern of the migration. Our results show that we are able to reduce the detectability with different traffic analysis attack channels of VM migrations, to 0% and therefore significantly reduce the attack surface.

### B. Stealth migration performance

In this section we evaluate the performance of our stealth migration system with regards to total migration time and VM downtime; we compare the results with that of native migrations without any stealth techniques. In these experiments, we migrated a 32bit Alpine Linux operating system with 80MB RAM in our emulation testbed.

*VM downtime:* The downtime of a virtual machine during a live migration process refers to the time for which the service provided by a virtual machine is completely unavailable. This downtime is usually significantly lower than the overall migration time. We measure the service downtime of a migration process by sending frequent ping messages (every 10ms) from a remote host to the migrated VM. The service downtime is then calculated as the number of lost ping messages, multiplied by the ping interval. The impact on the VM service downtime of our stealth migration system is minor. We observed an increase of 0.44 seconds (from 1.24s to 1.68s) on average when the virtualization platform is migrating a VM with 4MB/s and 0.31s on average (from 0.81s to 1.12s) with a rate of 8MB/s. On average, our stealth migration system delays the VM service downtime by an additional 0.37 seconds. A fractional increase like this is not observable by users of cloud computing services, hosting websites or video streaming platforms (to name a few examples). The reason for downtime variability of VMs during live migration is discussed in [17].

*Total Migration time:* The total migration time of a virtual machine refers to the time frame between when a migration is triggered at the original VM host until the process is finished and the virtual machine is running on the destination host. Users of services provided by virtual machines are usually not affected by the overall migration time. To compute the migration time, we collected timing data of multiple migrations when transmission rates of 4MB/s and 8MB/s are used. The overall migration time, when our stealth system is applied to the migration flow, is only increased by 1.8s (from 59s to 60.8s) on average with a rate of 4MB/s and 5.5s (from 41s to 46.5s) on average with a migration rate of 8MB/s. Comparing the latency overhead ratios of our system to recently proposed traffic analysis defense systems, such as CS-BuFLO [13], [14], shows a significant difference as we present in Table V.

TABLE V. Comparison of latency overhead ratios

| Latency overhead ratio | | | | |
|---|---|---|---|---|
| **CS-BuFLO** | | | **Stealth Migration** | |
| Lower | Upper | | Lower | Upper |
| 2.7 | 3.4 | | 1.1 | 1.3 |

As shown, current state-of-the-art traffic analysis defense systems, such as CS-BuFLO have a latency overhead ratio of over 3.0 on average. In comparison, our system only has an average latency overhead ratio of 1.2, making our system a lightweight defense approach.

## VIII. Related work

In [17] different design options for migrating running services of an operating system from a source to a destination host in cluster environments and data centers are introduced. Different attacks against live VM migrations in the context of the control plane, data plane and migration module are explored in [28]. By using techniques like ARP spoofing, DNS poisoning or route hijacking [2], an attacker is able to logically

position themself in the migration transit path. Even if proper encryption and identity management is used, it is still possible to gain valuable information such as the endpoint VMMs involved in the migration process. We show in this paper that by using our stealth system an attacker is not able to identify a migration stream, which is the foundation for a successful attack. The authors of [12] address the problem of different legislation and untrustworthy destinations when VMs, containing user data, are migrated across national borders. In [23] the weaknesses of censorship circumvention systems which mimic specific network traffic flows are discussed. Mimicking typical network protocols to bypass censorship systems of specific network traffic is very complex and error-prone and does not address changing of typical traffic patterns. In particular, it is stated in [23] that: *Many protocols produce characteristic patterns of packet sizes, counts, inter-packet intervals, and flow rates. These patterns are often stable across the network, observable even when packet contents are encrypted, and can be exploited for traffic analysis*.

The process of inter-datacenter VM migration is discussed in [16]. The authors analyze the traffic features observed during VM migrations between geographical regions over the Internet.

## IX. Conclusions

In this paper, we demonstrate that explicit patterns that manifest themselves during a VM migration, allows an adversary to detect and profile the migration. To prevent attackers from launching targeted attacks against a VM migration flow, we introduce a stealth migration system that prevents the detection of a migration flow by making it indistinguishable from normal Internet traffic. We perform extensive experimental evaluations to showcase the benefits of our system in hiding migration flows and showing that it is lightweight.

## Acknowledgment

The effort described in this article was sponsored by the U.S. Army Research Laboratory Cyber Security Collaborative Research Alliance under Cooperative Agreement W911NF-13-2-0045. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation hereon.

## References

[1] Apache opennlp maximum entropy. http://opennlp.apache.org/.

[2] Bgp hijacking. https://www.blackhat.com/docs/us-15/materials/us-15-Gavrichenkov-Breaking-HTTPS-With-BGP-Hijacking-wp.pdf.

[3] Bittwist packet generator. http://bittwist.sourceforge.net/.

[4] Global inet report. https://www.sandvine.com.

[5] Hierarchy token bucket theory. http://research.dyn.com/2013/11/mitm-internet-hijacking/.

[6] libvirt virtualization management platform. http://www.libvirt.org/.

[7] Mausezahn packet generator. http://netsniff-ng.org/.

[8] Mawi working group traffic archive. http://mawi.wide.ad.jp/mawi/.

[9] The new threat: Targeted internet traffic misdirection. http://luxik.cdi.cz/~devik/qos/htb/manual/theory.htm.

[10] V. K. Adhikari, S. Jain, and Z.-L. Zhang. Youtube traffic dynamics and its interplay with a tier-1 isp: an isp perspective. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010.

[11] M. Aiash, G. Mapp, and O. Gemikonakli. Secure live virtual machines migration: issues and solutions. In *Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on*. IEEE, 2014.

[12] S. Biedermann, M. Zittel, and S. Katzenbeisser. Improving security of virtual machines during live migrations. In *Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on*.

[13] X. Cai, R. Nithyanand, and R. Johnson. Cs-buflo: A congestion sensitive website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. ACM, 2014.

[14] X. Cai, R. Nithyanand, and R. Johnson. New approaches to website fingerprinting defenses. *arXiv preprint arXiv:1401.6022*, 2014.

[15] S. Chakravarty, M. V. Barbera, G. Portokalidis, M. Polychronakis, and A. D. Keromytis. On the effectiveness of traffic analysis against anonymity networks using flow records. In *Passive and Active Measurement*. Springer, 2014.

[16] Y. Chen, S. Jain, V. K. Adhikari, Z.-L. Zhang, and K. Xu. A first look at inter-data center traffic characteristics via yahoo! datasets. In *INFOCOM, 2011 Proceedings IEEE*.

[17] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005.

[18] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[19] W. DuMouchel and M. Schonlau. A comparison of test statistics for computer intrusion detection based on principal components regression of transition probabilities. *COMPUTING SCIENCE AND STATISTICS*, 1998.

[20] X. Fu, B. Graham, R. Bettati, and W. Zhao. On effectiveness of link padding for statistical traffic analysis attacks. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*. IEEE.

[21] O. Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2004.

[22] Y. Gu, A. McCallum, and D. Towsley. Detecting anomalies in network traffic using maximum entropy estimation. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. USENIX Association, 2005.

[23] A. Houmansadr, C. Brubaker, and V. Shmatikov. The parrot is dead: Observing unobservable network communications. In *Security and Privacy (SP), 2013 IEEE Symposium on*.

[24] A. Lakhina, M. Crovella, and C. Diot. Characterization of network-wide anomalies in traffic flows. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, 2004.

[25] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM Computer Communication Review*, 2004.

[26] S.-J. Moon, V. Sekar, and M. K. Reiter. Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[27] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of tor. In *Security and Privacy, 2005 IEEE Symposium on*.

[28] J. Oberheide, E. Cooke, and F. Jahanian. Empirical exploitation of live virtual machine migration. In *Proc. of BlackHat DC convention*. Citeseer, 2008.

[29] A. Rao, A. Legout, Y.-s. Lim, D. Towsley, C. Barakat, and W. Dabbous. Network characteristics of video streaming traffic. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*. ACM, 2011.

[30] W. Tao, C. Xiang, N. Rishab, R. Johnson, and I. Goldberg. Effective attacks and provable defenses for website fingerprinting. In *23rd USENIX Security Symposium (USENIX Security 2014)*.

[31] X. Zhang, T.-L. Wu, Z. Fu, and T.-L. Wu. Malicious packet dropping: how it might impact the tcp performance and how we can detect it. In *Network Protocols, 2000. Proceedings. 2000 International Conference on*. IEEE.