Invisible Probe: Timing Attacks with PCIe Congestion Side-channel

Mingtian Tan*, Junpeng Wan*, Zhe Zhou†
School of Computer Science
Fudan University
{18210240176,19210240003,zhouzhe}@fudan.edu.cn

Zhou Li
University of California, Irvine
zhou.li@uci.edu

***Abstract*—PCIe (Peripheral Component Interconnect express) protocol is the de facto protocol to bridge CPU and peripheral devices like GPU, NIC, and SSD drive. There is an increasing demand to install more peripheral devices on a single machine, but the PCIe interfaces offered by Intel CPUs are fixed. To resolve such contention, PCIe switch, PCH (Platform Controller Hub), or virtualization cards are installed on the machine to allow multiple devices to share a PCIe interface. Congestion happens when the collective PCIe traffic from the devices overwhelm the PCIe link capacity, and transmission delay is then introduced.**

In this work, we found the PCIe delay not only harms device performance but also leaks sensitive information about a user who uses the machine. In particular, as user's activities might trigger data movement over PCIe (e.g., between CPU and GPU), by measuring PCIe congestion, an adversary accessing another device can infer the victim's secret indirectly. Therefore, the delay resulted from I/O congestion can be exploited as a side-channel. We demonstrate the threat from PCIe congestion through 2 attack scenarios and 4 victim settings. Specifically, an attacker can learn the workload of a GPU in a remote server by probing a RDMA NIC that shares the same PCIe switch and measuring the delays. Based on the measurement, the attacker is able to know the keystroke timings of the victim, what webpage is rendered on the GPU, and what machine-learning model is running on the GPU. Besides, when the victim is using a low-speed device, e.g., an Ethernet NIC, an attacker controlling an NVMe SSD can launch a similar attack when they share a PCH or virtualization card. The evaluation result shows our attack can achieve high accuracy (e.g., 96.31% accuracy in inferring webpage visited by a victim).

I. Introduction

Devices peripheral to CPU are innovated at a fast pace. For instance, the speed of NIC (Network Interface Card) has been increased from 10 Mbps to 10 Gbps in the recent decade [1]. The throughput of hard drives is improved 100x due to new storage techniques like NVMe (Non-Volatile Memory Express) [2]. Followed by the upgrade of the peripheral devices, the data volume exchanged between them and CPUs is "exploding". The traditional PCI (Peripheral Component Interconnect) protocol cannot meet such demand, hence PCIe (PCI express) was proposed and has become the *de facto* protocol for I/O between CPU and peripheral devices [3].

I/O switch and congestion. Though PCIe supports high throughput I/O, *e.g.*, 16 GB/s for a PCIe 3.0 link [4], the support from CPU seems to fall behind. For example, only three 16-lane PCIe interfaces are provided by the Intel high-end CPUs [5], but a computer can integrate a large number of peripheral devices. For instance, 8 GPUs, 4 NVMe SSDs, and 1 x16 high-speed NIC are equipped by Tyan Thunder HX FT77DB7109 [6], a mainstream server for deep learning. Apparently, a gap exists between CPU's limited PCIe support and the strong demand for high-speed peripheral devices. As an intermediate solution, PCIe switch and PCH (Platform Controller Hub) are invented to expand CPU's PCIe support, which we term *I/O switch* in this paper. With I/O switch, two or more I/O devices can share a PCIe link and send data to the same PCIe interface of CPU. Besides link share, a GPU can directly talk to NICs and SSDs without the involvement of CPU, with the help of PCIe switches, which is termed RTX IO by NVIDIA [7].

However, I/O switch also introduces a new problem: I/O congestion. When one device fills a PCIe link (*e.g.*, Nvidia GTX 1080Ti GPU can produce 480 GB/s data [8]), the PCIe packets generated by the other devices sharing the same I/O switch will be delayed due to point-to-point credit-based flow control of PCIe [9]. While congestion has been well studied in the network protocols like TCP [10], [11] and the security issues like denial-of-service (DoS) attacks are well known, no prior works have looked into the I/O congestion brought by PCIe on a single machine, not to mention its security implications. In this work, we make the *first* attempt to study these issues.

PCIe congestion side-channel. Through exploratory analysis, we found the degree of PCIe congestion might reflect the operational status of a connected device. When a device transfers data through a congested PCIe link, the interval between request and response will be increased. Therefore, if an attacker has access to a device that shares a PCIe link with another device used by a victim, the attacker can keep probing the link and use the delay to infer the status of the victim device. Furthermore, as the device I/O patterns can be determined by the user's activities (*e.g.*, typing and browsing), the attacker can recover the sensitive user information from the probed delays potentially.

Though the high-level idea is simple, measuring the delay at high resolution and recovering the user's information at high

*The first two authors are equally contributed. † Zhe Zhou is the corresponding author.

accuracy is non-trivial. The attack seems impossible initially using traditional probing methods (*e.g.*, keep reading `procfs` of Linux [12]) and time measurement (*e.g.*, `clock()`). But the new functionalities provided by the software and hardware stacks of peripheral devices enable our attack, as we observed. Here we name a few: 1) Kernel-bypass removes the random delays from CPU scheduling and interrupts, conserving the attacker's measurement to the communication latency. 2) RDMA (Remote Direct Memory Access) NICs widely installed on data centers and public clouds allow the attacker to transmit data at ultra-low latency, leading to a very high sampling rate of a PCIe link. 3) The hardware clock provided by RDMA NIC enables high-precision measurement.

Our attack and evaluation. Based on the above observations, we propose a new side-channel attack that exploits PCIe I/O congestion, termed INVISIPROBE. We demonstrate INVISIPROBE in two scenarios.

Firstly, when the victim uses a high-speed device like GPU on a server, an attacker can use an RDMA NIC on her machine to probe the RDMA NIC on the server and infer the victim's secret. We demonstrate three attack settings, including 1) inferring keystroke events and words typed by the victim; 2) inferring the webpages visited by the victim; 3) inferring the machine-learning model trained by the victim. We found the delay (for keystroke inference) or delay sequence (for webpage and model inference) patterns are distinguishable, due to the unique data movement patterns between CPU and GPU. To achieve accurate inference on the probed delays, we develop an inference model based on *Attention-Based Bidirectional LSTM (AttBLSTM) [13]*, which is capable of handling long sequences with variable length.

In addition to the high-speed device, we found the low-speed device used by a victim is not immune to INVISIPROBE. When a victim is using an Ethernet NIC, *e.g.*, 1Gbps NIC, an attacker can probe the server's NVMe SSDs at high frequency to introduce PCIe congestion, and measure the delays at the same time. We demonstrate that the collected delays can help the attacker infer which website is visited, due to that loading different websites results in different file downloading patterns.

We evaluate the four proposed attacks in a lab environment consisting of one server and one PC. To highlight the evaluation results, 98.97% keystroke events can be detected, and 96.31% visited webpages can be correctly classified, in the first attack scenario. We also evaluate the second attack scenario in a public cloud, Alibaba cloud, and found high accuracy (91.02% for webpage inference) can be achieved. Our results indicate the threat is practical when PCIe congestion is exploited by an attacker. As I/O switches become prevalent in data centers and the recent trend encourages PCIe switches to be shared by I/O devices across machines, *i.e.*, through PCIe fabric, we believe more attention should be paid by the security community. We propose a few directions for threat mitigation and are discussing them with cloud providers like Alibaba cloud.

Contributions. We summarize our contributions below.

- We identify a new side-channel attack exploiting the unique features of PCIe congestion.
- We develop two concrete attack strategies: using RDMA NIC to attack GPU and using NVMe SSD to attack Ethernet NIC.
- We evaluate the two strategies under 3 victim scenarios: keystroke typing, webpage browsing and training machine-learning model. The result shows our method, INVISIPROBE, is effective.
- We will release the code and experimental data after the discussion with the stake-holders to help other researchers investigate the related issues.

II. BACKGROUND

A. PCIe

PCIe (Peripheral Component Interconnect Express) is the *de facto* protocol for the peripheral devices to transmit data to a processor [3], thanks to its prominent advantages like higher maximum bus throughput. There are 4 ways to let a device communicate through PCIe. 1) Installing the device on the PCIe slot of the motherboard, *e.g.*, discrete GPU; 2) Installing the device on another type of slot which is compatible with PCIe protocol, *e.g.*, installing NVMe (Non-Volatile Memory Express) SSDs on M.2 slots; 3) Soldering a device on the motherboard and using PCIe protocol, *e.g.*, onboard NIC (Network Interface Card); 4) Leveraging a controller to convert a different protocol used by a device to PCIe protocol, *e.g.*, SATA hard disks.

The communication between PCIe devices is carried out by *interconnect* (or *link*). A link consists of one or more *lanes*, which carries a full-duplex byte stream, producing 985 MB/s bandwidth per lane per direction for the version supported by the mainstream Intel CPUs [14]. The number of lanes per link can be 1, 2, 4, 8, or 16. Thus, the device's PCIe speed can be adjusted based on the number of occupied lanes.

Different from PCI protocol in which a shared parallel bus architecture is used and the bus clock is constrained by the slowest peripheral device, PCIe protocol is *packet-based*, which enables more efficient bus usage. Similar to the network stack, PCIe has three layers and they are transaction layer, data link layer, and physical layer. The data are encapsulated into packets by transaction layer and routed based on memory address, I/O address and device ID [15]. The data link layer handles integrity check, flow control, and re-transmissions, ensuring the data integrity and reliability for the upper layers. To this end, *point-to-point credit-based flow control* [9] is employed by PCIe. On a PCIe link, the receiver notifies the sender the size of the receiving buffer reserved for it ahead, and the data transmitted by the sender is always under the buffer limit.

B. PCIe Topology

PCIe allows devices to connect to the processor in a tree-like topology [15][1], with the help of *PCIe switch* and *PCH*

[1]In the future, PCIe fabric may be used in a data center, changing the topology to a network-like topology.

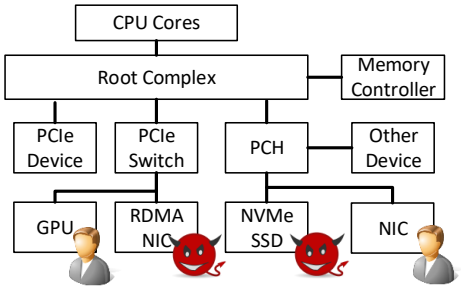


Fig. 1: PCIe topology and adversary model.

(Platform Controller Hub). In this work, we term them jointly as *I/O switches* as they have similar functionalities in routing PCIe packets. An I/O switch has upstream and downstream ports connecting to different devices, in order to separate the data flow. Figure 1 illustrates one common PCIe topology. Specifically, the PCIe lanes interfaced by CPU and the memory controller are connected to a component named *root complex* which is further connected to the peripheral devices. The connection between the root complex and peripheral devices can be divided into three classes.

Firstly, a small number of high-speed interfaces of root complex are directly connected to high-speed devices. For mainstream Intel CPUs like "Coffee Lake" CPUs [16], interface to 16-lane PCIe link is provided, which can be used by a discrete GPU.

Secondly, relatively slow devices, like hard disks, sound cards, and NICs, are connected to PCH before the root complex, which is the standard component on desktop machines and also servers. *Direct Media Interface (DMI)* is used as the channel between CPU and PCH, which has nearly the same design as *4-lane PCIe link*. A PCH can connect to many devices, each of which possesses at least one PCIe lane. However, the resulted total I/O throughput can be much larger than what DMI can provide. Thus, DMI allows the connected devices to share the bandwidth of the four PCIe lanes.

Thirdly, the PCIe switch is used to expand CPU's capabilities in connecting more PCIe devices, especially on high-performance servers. While the CPU integrated by a server usually provides high-speed interfaces for direct connection to PCIe devices, it might be insufficient to meet the demand of adding PCIe devices. For example. Tyan Thunder HX FT77DB7109 [6], a mainstream server for deep learning, equips 8 GPUs, 4 NVMe SSDs, and 1 x16 high-speed NIC. As only three 16-lane PCIe interfaces are provided by the CPU ("Cascade Lake-SP" [5]), the server manufacturer converts one 16-lane PCIe interface into two or more interfaces by using a PCIe switch.

C. RDMA

To reduce the network latency and CPU consumption on the server, RDMA (Remote Direct Memory Access) was proposed to allow a data sender to directly access the memory region of a receiver, without waiting for the receiver's CPU. Most RDMA functionalities are implemented by RDMA NICs, supporting one-sided or two-sided RDMA operations. In the one-sided setting, the RDMA application at the server registers one or multiple memory regions with a RDMA NIC. The virtual address of the memory region as well as a key corresponding to the memory region will be sent to the client. When the client is going to read the memory in the server, she constructs and sends a special RDMA read packet that contains the address of the memory to be read as well as the key. The server's RDMA NIC directly reads the requested memory for the client once receiving the packet, without interrupting the CPU. The value of the requested memory will be encapsulated in the response packet that will be delivered back to the client. In the two-sided setting, the CPUs of the sender and the receiver are both involved.

A few RDMA implementations have been developed under the standard RDMA protocol, namely InfiniBand [17], RoCE [18] and iWARP [19]. We tested our attack under InfiniBand as it is the native physical layer protocol of RDMA.

Currently, RDMA has seen strong adoption in data centers and cloud, resulting in high-performance applications related to key-value storage [20], graph processing [21], machine-learning [22] and *etc*. We consider RDMA as one attack vector to enable our attack, INVISIPROBE.

III. ATTACK OVERVIEW

Using I/O switches to serve multiple PCIe devices and share the PCIe bandwidth has become the standard solution to address the constraints of PCIe interfaces. However, it also opens up possibilities for attacks. In this section, we firstly describe the threat model. Then, we elaborate on the issue of PCIe congestion and how it can be exploited as a side-channel. Finally, we overview the attack procedure.

A. Threat Model

As shown in Figure 1, we assume that among a pair of I/O devices integrated by a target machine, one interacts with an attacker and another is used by a victim. Both devices share the same I/O switch which connects to the upstream CPU. We assume the attacker cannot access the victim's data or code directly, but plans to use INVISIPROBE to infer the victim's secrets.

We showcase two scenarios. In the first scenario, the attack happens in a cloud environment or a data center where a server allows a remote machine to directly access its memory through its RDMA NIC (*e.g.*, by running an RDMA key-value storage). The attacker can control a VM in the same cloud or a machine in the same data center to interact with the RDMA NIC of the server. In the meantime, a *high-speed device*, in particular a discrete GPU, is used by a victim. GPU and NIC share the same PCIe switch, which is the setting appeared in the slides of NVIDIA [7] when NVIDIA is launching the 30 series GPUs. Using RDMA to steal a victim's secret was researched recently [23], [24] and our attack setting is similar,

i.e., assuming attacker can interact with an RDMA NIC in a server, but the goal of the attacks differ. In Section VIII, we compare to these works in detail. We assume the server providing GPU acceleration allows multiple users/tenants to use the machine simultaneously. This assumption has been justified by previous GPU side-channel attacks [25], [26]. Moreover, we assume the victim can use the GPU exclusively, which is an even weaker condition comparing to previous works [25], [26] assuming GPU is shared between the victim and the attacker.

Under this scenario, we studied three types of attacks. 1) The server has a VM leased to a victim who types sensitive text over remote desktop to the VM; 2) The server has a VM leased to a victim who opens chrome to view webpages through remote desktop; 3) A victim trains a deep-learning model on the server. The attack goals are to infer: 1) which word is typed by the victim; 2) which webpage is browsed by the victim; 3) which model is trained by the victim.

In the second scenario, we assume the attacker has access to an NVMe SSD on the server. The attacker can be local (*e.g.*, executing a program on the server) or remote (*e.g.*, accessing the SSD through NVMe fabric [27]). In the meantime, a victim uses a *low-speed device*, in particular a standard NIC, on the server. As described in Section II, PCH is used to serve the traffic from SSD and NIC together. Under this scenario, we investigate an attack that is the same as the second attack under the prior scenario: inferring what pages are visited by the victim.

B. PCIe Congestion

As PCIe connections resemble network connections, congestion could happen when the data volume to be transmitted surpass the capacity of PCIe links or I/O switches. Specifically, when the sender learns that the requested receiving buffer exceeds the capacity of the receiver, it holds the packet to be sent till the receiver's buffer is freed, in a process called *back-pressure* [15]. The packet transmission will be delayed, and the delay grows when the data pressure is increased on the PCIe link, as shown by previous studies [3], [28].

PCIe congestion can be caused by both high-speed and low-speed devices. For the first case, the data volume generated by a device alone, like GPU, can cause congestion. For instance, Nvidia GTX 1080Ti GPU can occupy 480 GB/s PCIe bandwidth [8], but the 16-lane PCIe link offers no more than 16 GB/s bandwidth. For the second case, when multiple devices transmit packets to a PCH simultaneously, congestion can happen. For instance, a PCH only occupies four PCIe lanes from CPU. However, an NVMe SSD alone has a four-lane link to PCH, so it is able to fill all PCH lanes. When another device shares the same PCH, congestion will happen. To notice, though PCIe congestion happens frequently, prior works mostly focused on its impact to the performance of PCIe fabric [29], [30]. The issue on the single machine and how it can be exploited for attacks have not been studied.

Through experiments, we found the overhead caused by PCIe congestion is not negligible, and reflects what happens on the devices sharing the same I/O switch, to some degree. An attacker can exploit this finding to measure the I/O latency on the devices of the target machine (*e.g.*, RDMA NIC and NVMe SSD) and analyze the portion related to PCIe congestion. However, the measurement of I/O operations is known to be fluctuating, due to interrupts [31]. Yet, we found that when the kernel is bypassed, the measurement becomes stable. For the first attack scenario, RDMA directly bypasses the kernel [32]. For the second attack scenario, when a *kernel-bypass driver*, like SPDK [33] or `io_uring`, is installed for the NVMe SSD, the same effect can be observed. In fact, kernel-bypass driver is widely installed on servers of data centers [34]. In Appendix A, we explain how kernel-bypass driver helps the I/O measurement in detail.

C. Attack Procedure

Based on our insights into PCIe congestion, we design INVISIPROBE, a new attack exploiting PCIe congestion as side-channels. In essence, when congestion is caused by the high-speed victim device, the attacker can measure the variation of the I/O latency by sharing PCIe switch to infer the secret states of the victim. When the victim device is low-speed, unable to introduce congestion alone, the attacker can tunnel in a high volume of data to "saturate" the shared PCH switch and measure the I/O latency at the same time. This subsection overviews the workflow of INVISIPROBE and Section IV and V elaborate how INVISIPROBE are implemented against PCIe switch and PCH respectively.

1. Device pairing for congestion. The attacker first needs to identify a pair of I/O devices that share an I/O switch. While the attacker does not directly know the PCIe topology of the target machine, the chances of finding a device sharing a PCIe link with the victim device are high, especially at servers. In Section VI-A we give an example of the PCIe topology.

2. Delay measurement. The attacker probes the I/O latency of her device using the timing API (*e.g.*, `RDTSCP` instruction [35]) to infer the status of the victim device. The probe must satisfy the following requirements:

- Attacker's device can complete each probe request within a short interval, achieving a high sampling rate on a PCIe link.
- The latency of the probe should be stable, so its variance should be small when the same data volume is encountered by a PCIe link.
- The congestion level should be proportional to the probe delay.

In Section IV and V, we describe how the probes on NIC and NVMe SSD are constructed under the above requirements.

3. Inferring the secret. After the delays of the probes are collected, the attacker will analyze them to infer the victim's activities. Based on our exploratory analysis, the mapping between them is not straightforward, therefore we leverage supervised learning techniques to obtain such mapping and use it to classify user's activities.

Challenges. While the attack procedure is simple at a high level, there are some challenges we need to address in attack implementation. Firstly, the choices of the probe are paramount (*e.g.*, different I/O APIs, their parameters, and device access patterns), but most of them cannot fulfill the three proprieties in the delay measurement. Secondly, high-resolution tools are required to measure the I/O latency so the "weak signals" from users' sensitive activities can be captured. The regular measurement tools provided by OS cannot provide enough accuracy for latency measurement. Thirdly, the attacker cannot directly measure the victim's device, and the probes she issues might interfere with her observations. To solve those challenges, the probe and the inference methods have to be carefully designed and we elaborate them in the two attack scenarios.

IV. Attacking GPU with RDMA NIC

In this scenario, we assume a high-speed device, in particular a discrete GPU, is used by a victim. The adversary has access to an RDMA NIC on the same machine. For the attacker to obtain reliable measurement through the RDMA probe, the processing time of the RDMA request at the server should be small. Fortunately, this requirement can be fulfilled readily. For example, 1.3 million ops/sec can be achieved by an RDMA key-value store on Infiniband [20].

One advantage of the RDMA probe is that it cannot be interfered by other network traffic, like LAN traffic. A server usually uses an RDMA NIC to handle RDMA traffic and an Ethernet NIC to handle LAN traffic, so they are isolated at the physical layer. During our experiment, we also use separate NICs. Though RDMA NIC can be configured to handle IP packets, *e.g.*, through IPoIB [36], this feature is disabled by us (the default setting). Therefore, no IP traffic would go through the RDMA NIC.

A. Design of Probe

RDMA supports three types of connections, Reliable Connected (RC), Unreliable Connected (UC), and Unreliable Datagram (UD) [37]. Among them, RC resembles TCP protocol, and RDMA read can be done only on this connection type, so we assume the attacker probes the I/O delay with RC. Algorithm 1 overviews the whole probing method. We write around 400 lines of C code for this RDMA probe.

When RC is set up, a send queue will be created on the attacker side to send the read requests. Each read request accesses 4 bytes only. When the number of bytes is large, PCIe switch may increase the buffer size to ease congestion. When the number of bytes is less than 4, we found the throughput and delay remain the same. To keep sampling, the attacker needs to monitor the send queue and make sure it is non-empty by continuously adding requests. However, this strategy might make multiple requests arrive at the I/O switch at the same time, resulting in inaccurate measurement. We address this issue by making the requests *"mutually exclusive"*: we set a fence (`IBV_SEND_FENCE`) after each request, holding the next request to wait for the reply of the prior one.

Algorithm 1: Collection of Delay Sequence with RDMA

```
Result: DelaySeq
Alloc SendQueue(withFence), CompletionQueue;
SendQueue.enqueue(a batch of RDMA read
  operations);
lastTimestamp = device.getTimeStamp();
while notEnough(DelaySeq) do
    wr = CompletionQueue.pop();
    DelaySeq.append(wr.timestamp - lastTimestamp);
    lastTimestamp = wr.timestamp;
    if SendQueue.isAlmostEmpty() then
        SendQueue.enqueue(a batch of RDMA read
          operations);
    end
end
```

We set the `IBV_EXP_CQ_TIMESTAMP` flag to the sending queue in order to collect the *hardware timestamps* of the replies, which are stored into the completion queue. The collected timestamp is much more precise than the timestamp of CPU (*e.g.*, `RDTSCP`). The interval between consecutive hardware timestamps is close to the delay of the request, because 1) kernel and CPU are bypassed; 2) no congestion should happen at the sender and receiver NICs due to mutually exclusive requests; 3) the Infiniband network introduces ultra low and stable latency (*e.g.*130 nanosecond from one switch port to another [38]).

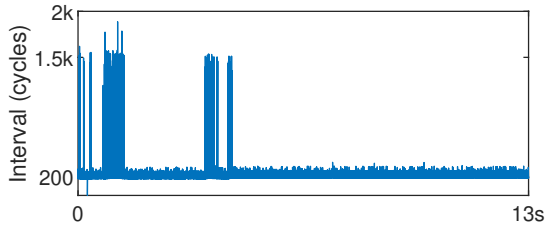


Fig. 2: Delay variance based on the PCIe congestion status.

We found the designed probe can achieve a very high sampling rate and high sensitivity to congestion. On our experiment platform, the sampling rate can be as high as 0.77 million points per second (the interval between probes is around 1.3 microseconds or 206 RDMA NIC cycles) when the upstream PCIe link is not shared with other devices. When the PCIe link is busy, as we let the CPU pass a large array (30,000×30,000 integers) to GPU and read it back, the interval is increased to 9.6 microseconds or 1,520 RDMA NIC cycles. As shown in Figure 2, *6 times* difference can be observed. The high entropy embedded within the interval variance enables fine-grained profiling of the victim device.

A recent work, NetCat [23], carried out an RDMA Prime+Probe attack against LLC (last level cache) of a server to infer the victim's secret. Though our RDMA probe also touches LLC, what is measured is different from the RDMA

Prime+Probe of NetCat [23]. First, Prime+Probe requires an eviction set filled with carefully selected memory addresses [2], but our probe reads fixed 4 bytes only. Second, Prime+Probe introduces the timing difference of accessing LLC (between hit and miss) at 100 nanoseconds, but the difference is far larger in our attack (8.3 microseconds).

B. Attack 1: User-input Inference

We observe that even a keystroke a victim enters, which introduces fairly small amount of data, is distinguishable under INVISIPROBE. When a keystroke is entered to a GUI input, a character will be rendered by GPU. In addition, the UI element around it, *e.g.*, textbox, will be refreshed. Before the rendering, CPU passes the character and UI elements from memory to GPU via PCIe link and issues the rendering commands to GPU. Though the data volume is small, CPU will try to use all PCIe lanes to transmit the data, still introducing noticeable congestion when INVISIPROBE is launched at the same time, leading to a surge of probe delays, as shown in the top of Figure 3. In contrast, when the displayed content is not updated, the CPU-GPU PCIe link will stay "quiet". Therefore, INVISIPROBE can infer the keystroke dynamics. To notice, though the display is refreshed at a constant rate, *e.g.*, 60Hz, the frequency of data transmission on PCIe and GPU rendering are not constant.

This attack assumes the rendering is facilitated with the GPU on the remote server. This feature has been supported by remote desktop services of Windows [40] and Linux [41], which can be turned on by changing the system configurations on the server. Mainstream cloud providers like Amazon EC2 also offered this support in their VMs [42]. Game cloud providers like Steam Cloud Play started to render games for users in remote server, which aroused over 94 million users [43]. When a user is using a resource-constrained device, *e.g.*, a thin client in an enterprise network, for tasks requiring GPU acceleration, like browsing using WebGL, gaming, and data visualization [44], remote GPU acceleration is recommended [45]. We expect this setting will be encountered more often along with the increased adoption of remote desktop [46].

In Section VI-B, we evaluate whether INVISIPROBE can learn when a keystroke occurs and how likely a word typed by a victim can be inferred. Below we describe the steps.

Inferring keystroke events. By modeling the surge of probe delays, the occurrences of keystrokes could be detected. The main problem we need to solve is how to distinguish keystrokes with other UI updates (*e.g.*, screen transition) from the delay surge. After analyzing the delay sequences resulted from a set of keystrokes, we found that any keystroke will increase the delay of 7 consecutive probes, while other UI activities rarely exhibit the same pattern (*e.g.*, UI transition impacts far more probes). According to this observation, we propose the following method to identify a keystroke:

1) We compute the intervals between all probe requests and save them into a delay sequence.
2) We set a lower-bound TH_L and an upper-bound TH_H to select the sampling points. These points are likely to be related to keystrokes and we call them suspected points. As shown in the bottom of Figure 3, unrelated sampling points can be filtered out reliably.
3) We use a sliding window of Win readings to scan the intervals. If there are over K suspected points, we consider there is a suspected keystroke happening in the time window. The red stars and blue circles in Figure 3 show the suspected keystrokes.

After empirical analysis, we set the values of TH_L, TH_H, Win and K to 1000, 3000, 50,000 and 6. Though for a different machine the delay patterns might be varied (*e.g.*, 5 might be sufficient for K), the adversary can update the parameters according to the target machine.

Removing caret. Though the above method produces good recall of keystroke events, we found the blinking caret in the textbox is detected as well, as it has a similar impact on our probe intervals. On the other hand, the blinking caret can be filtered out due to their unique UI patterns: 1) the blink has a constant interval; 2) the caret blinks only when the user is not typing in most applications[3].

Specifically, we compute the intervals between the suspected keystrokes and remove the ones with the constant intervals (*e.g.*, 598 milliseconds or 1193 milliseconds for Google Chrome, and 403 milliseconds or 793 milliseconds for gedit). The chances of removing real keystrokes are very small, as keystroke usually takes a much shorter interval. The blue circles in Figure 3 show the carets that are successfully detected.

Recovering word. Though the learnt keystroke events do not tell which keys are typed, according to prior studies, their intervals reveal what words are typed [12], [47], [48], based on a language model. Take English words as an example. In essence, the attacker can first collect the keystroke timings about English words from a dictionary words, and then use Hidden Markov Model (HMM) [49] to build the relationship between the intervals and the hidden states (*i.e.*, pair of characters). The transition probabilities between certain states such as ('i', 'o') are adjusted based on their probabilities of occurring together. The "SPACE" character between words are detected based on its longer inter-keystroke interval [12]. Then, the attacker applies Viterbi algorithm [49] to obtain the top n most likely character sequences given an interval sequence. The candidate sequences that fail spelling checks are removed before the result is presented.

C. Attack 2: Webpage Inference

Using GPU to accelerate webpage rendering has become a standard technique for browsers, like Google Chrome [50], Mozilla Firefox and Internet Explorer. Take Google Chrome

[2]Maurice et al. [39] showed Prime+Probe is feasible under addressing uncertainty, but it is used to construct covert channel.

[3]We tested 10 applications and only found the caret keeps blinking in Firefox regardless of typing. The 10 applications are described in Appendix C.

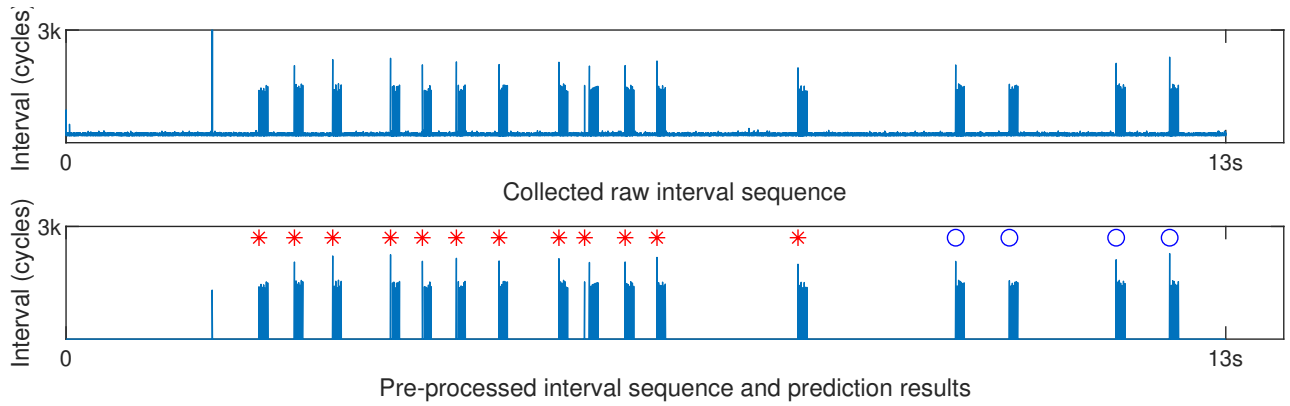
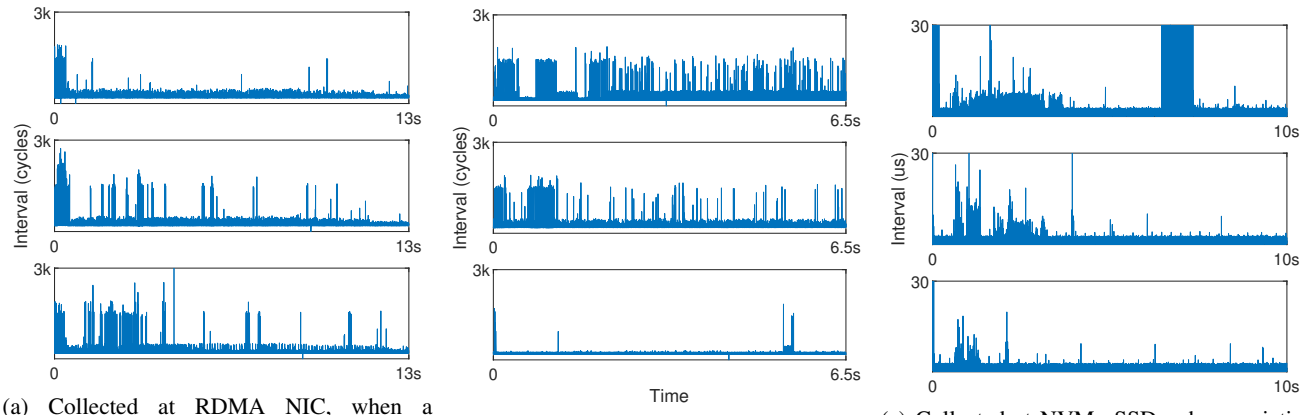


Fig. 3: Recovery of keystroke events. The top figure shows the raw delay sequence. The bottom figure shows the sequence after the pre-processing. The predicted keystrokes are marked with red stars while carets are marked with blue circles.



(a) Collected at RDMA NIC, when a victim visits three different websites (`www.android.com`, `www.12306.cn`, `www.pptv.com`).

(b) Collected at RDMA NIC, when a victim trains three different ML models (DCGAN, RNN-LSTM, AutoEncoder).

(c) Collected at NVMe SSD, when a victim visits three different websites (`sohu.com`, `tmall.com`, `baidu.com`).

Fig. 4: Interval sequences in the three attacks.

as an example. The renderer process of CPU rasterizes (*i.e.*, converts the geometry description of the image to pixel description) the webpage and pushes the data to the CPU memory shared with GPU. The GPU process copies the data from CPU memory to the GPU memory and invokes OpenGL [50] APIs to draw the bitmaps region-by-region. Lastly, the Chrome compositor stitches the rendered images together to draw the whole webpage using GPU. Initially, a website needs to call WebGL APIs [51] to direct Google Chrome to use GPU for rendering. But recently, GPU acceleration is configured as the default setting in Google Chrome [52]. Since the data related to the webpage has to be moved from CPU to GPU, PCIe congestion can be introduced as well when INVISIPROBE is launched.

We speculate visiting a webpage could yield a unique delay sequence, because it usually triggers downloading many web files (*e.g.*, JavaScript, HTML and image files) from different web origins. The browser executes each file after it is downloaded instead of waiting for all of them [53]. As such, the data transmission between CPU to GPU becomes intermittent, resulting in distinguishable I/O patterns. Figure 4a (a) compares the sequences when visiting 3 different webpages. Based on the above observation, we take the entire delay sequence and classify it.

In Section VI-C, we evaluate INVISIPROBE in a close-world setting: we profile 100 webpages ahead and examine if INVISIPROBE can identify the webpage being visited.

Classifying delay sequence. Loading the same webpage might observe different delay sequences due to network conditions, OS events, and dynamic content (*e.g.*, online advertisement). To achieve robust classification on the ever-changing delay sequences, we leverage *LSTM (Long short-term memory)*, an RNN model family capable of handling complex time-series [54], [55]. LSTM is able to track long-term dependencies in a sequence, which is ideal for our problem setting, as file downloading events can span the whole lifetime of webpage rendering. In particular, we choose *AttBLSTM (Attention-Based Bidirectional LSTM)* model [13] under the LSTM family to classify our delay sequence. AttBLSTM considers both forward and backward dependencies and uses an attention

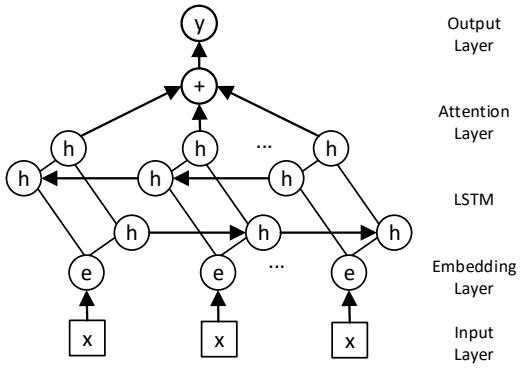


Fig. 5: AttBLSTM Model for classifying webpage visits.

layer to focus on the elements that have a decisive role in the sequence. It outperforms the vanilla LSTM model especially when the number of sequence elements (*e.g.*, downloaded files in our case) is not fixed [56]. Figure 5 shows the structure of our classifier based on AttBLSTM.

While AttBLSTM can directly process the delay sequence, we found the performance is unsatisfying as each sequence has a huge number of data points, ranging from 1 million to 10 million. Therefore, we add a pre-processing layer before the input layer of AttBLSTM. We split the sequence into windows and each window contains Win intervals with no overlapping with other windows. Then we compute the *frequency* of each interval value and aggregate them into three buckets $[0-100]$, $[100-500]$ and $[500-]$. As a result, each window is converted into a vector of three frequency buckets.

After processing the input, the LSTM layer learns the predictive features from the sequence and the attention layer captures the dependencies between the sequence of features and the output. The embedding layer after that produces a vector of 64 numerical values. Finally, the fully-connected layer converts the embedding vector into a classification vector with the size the same as the number of profiled webpages. The soft-max layer assigns probabilities to each class, and the one with the highest probability is returned to the adversary.

Through this pipeline, we found that a webpage can be classified at high accuracy, but it also requires many sequences of a webpage to be collected beforehand for training. If removing the layers after the embedding layer, we can turn the AttBLSTM model into an embedding model and classify a webpage without retraining. For instance, we can generate the embedding (E_1) of one sequence (S_1) of a webpage W with the pre-trained model. For a new sequence S_2 encountered in the testing stage, we can generate its embedding E_2 and compute its distance to E_1 (Cosine distance) and classify it into W if the distance is lower than a threshold. We adjust the pre-processing step by using the average delay as the feature for each time window instead of the three bucketed frequencies. We call the two settings classification mode and embedding mode.

D. Attack 3: Machine-learning Model Inference

Nowadays, GPU is extensively used to train machine-learning models. The structure of a machine-learning model can be considered as "intellectual property" (IP) [57], as many companies have invested a large amount of resources to develop it. In this attack, we consider the adversary is interested in learning the model structure for IP infringement or use the information to improve the adversarial attacks [57]. This attack setting has been studied by a number of works recently (surveyed in Section VIII), but none of them exploited the PCIe congestion side-channel.

More specifically, we consider the structure of a DNN (Deep Neural Network) model has been decided by a developer and it is trained using TensorFlow on the targeted server. When the training is initialized, TensorFlow compiles the model structure to a Tensorflow graph where each node is a TensorFlow operation (or op). For each op in the graph, the graph executor on CPU will transfer a batch of data to GPU memory as initialization. Thus, there is data movement from CPU to GPU through PCIe link [58] and different ops could introduce different data movement patterns, and our goal is to infer the op sequence, which can be leveraged to recover the secret model structure. Though the valid combinations of ops are virtually infinite, only a few ops can be chosen, including Conv (convolution), FC (fully-connected), BN (batch normalization), ReLU, Pool, and *etc*. As shown in a recent work [59], those ops all introduce unique PCIe usage patterns, and their combination may result in a unique delay sequence.

Similar to Attack 2, the attacker probes the shared PCIe link to obtain a delay sequence. Then she uses the same AttBLSTM model described in Section IV-C to classify the whole delay sequence into a model structure that has been profiled. In evaluation (Section VI-D), we profiled 10 models and classifies a sequence to one of them.

V. Attacking NIC with NVMe SSD

In this scenario, we show that even when a low-speed device, in particular an Ethernet NIC, is used by the victim, the sensitive information can be leaked under InvisiProbe. Similar to the prior section, we describe the probe design and a concrete attack under this scenario.

A. Design of Probe

When attacking GPU using NIC, the congestion is mainly caused by the GPU used by the victim, and the I/O delay is measured by the RDMA NIC. In contrast, the attacker in this scenario uses her device (*i.e.*, NVMe SSD) to cause congestion and measure delay together, because the data volume introduced by the victim's NIC alone might be limited. Ethernet NICs that support 10Mbps, 100Mbps, 1Gbps data rates are usually connected to PCH via a single PCIe lane [60] [4]. A PCH can forward 4GB/s data but a 1Gbps NIC only introduces 0.125GB/s data, filling 3.1% bandwidth of PCH at maximum.

[4]Higher-speed NICs, like 10Gbps NIC, are rarely plugged to PCH.

Although NVMe SSDs can generate a large amount of traffic in a small duration to fill the bandwidth gap, the latency of a single request has a large variance, because SSDs run many internal jobs, *e.g.*, garbage collection and anti-wearing [61]. To address the problem, we use pipelined requests instead of mutually exclusive requests with a fence. Specifically, the attacker always fills up the send queue of her NVMe SSD, so the SSD will be busy sending responses to the PCIe link. The attacker does not measure the latency on each request, but measure the interval between consecutive response completions, which eliminates the variance caused by SSD internal jobs.

We experimented with numerous settings of data transmission, and found that reading 4KB (8 LBAs, 512 bytes each LBA) at a fixed position in each request can produce the desired traffic volume and maintain a high and stable sampling rate. As a result, the attacker can issue more than *812K read operations per second (IOPS)* that generates about 3.1 GB/s throughput. On top of this throughput, the Ethernet NIC (x1 lane) can easily saturate the upstream of the PCH (x4 lane). Besides, we investigated three reading orders of NVMe SSD, including random read, sequential read and read from a constant block, and found constant read yields the most stable delays.

To precisely measure the NVMe SSD I/O interval, we use Storage Performance Development Kit (SPDK) [33], a toolkit for low latency kernel bypass I/O. SPDK allocates a pair of send queues and completion queues, which are shared between the user's process and the device. When there is an I/O operation launched by the process to the NVMe SSD, SPDK generates an item in the send queue containing all the parameters about the request, and then notifies the SSD via *door bell*, which writes a memory-mapped register of the SSD. The controller of the SSD will immediately fetch the request from the send queue and process it. After that, the result will be written to a predefined memory location and an item will be saved in the completion queue. Developer can get the completion information by a SPDK procedure called (`spdk_nvme_qpair_process_completions()`). The algorithm of this NVMe SSD probe is shown by Algorithm 2. We write about 200 lines of C code for this NVMe SSD probe.

Algorithm 2: Collection of Intervals with NVMe SSD

```
Result: IntervalSeq
Alloc SendQueue, CompletionQueue;
SendQueue.enqueue(a batch of read operation);
lastTimestamp = RDTSCP();
while notEnough(IntervalSeq) do
    while isEmpty(CompletionQueue) do
        ;
    end
    currentTime = RDTSCP();
    CompletionQueue.dequeue();
    IntervalSeq.append(currentTime - lastTimestamp);
    lastTimestamp = currentTime;
    if SendQueue.isAlmostEmpty() then
        SendQueue.enqueue(a batch of read operation);
    end
end
```

B. Attack 4: Webpage Inference with NVMe SSD

Among the three attacks demonstrated in Section IV, the attack about webpage inference is more relevant in this scenario, as users' activities on the other two do not directly involve NIC. Similar to Attack 2, we collected a delay sequence for each webpage visit, and classify it using the AttBLSTM model. During pre-processing, we again split the sequence into windows, but this time we count the maximum delay in the window as a feature instead of using the three frequency buckets, because the delays do not have the same distribution. Other statistical features, like average delay, lead to a similar result, so we use the maximum delay for simplicity.

VI. Evaluation

In this Section, we evaluate the effectiveness of Attack 1-4. In Section VI-A, we describe the platform used for experiment. Section VI-B, VI-C, VI-D, VI-E report the results for each attack. Section VI-F describes the evaluation on a public cloud.

A. Experiment Platform

We use a regular desktop PC and a server as the experiment environment, which is similar to prior works in studying remote side-channel attacks [23], [24], except that we use one less desktop PC because we assume the victim's application directly runs on the server. The specifications of the machines are shown in Table I.

In Figure 1, we give an abstraction of PCIe topology on a machine. Here we elaborate on the topology (also illustrated in Figure 6) on our server, which is more complex. Though the CPU only has 3 PCIe interfaces, the PCIe switch (PEX 8747 PCIe) and the 8 I/O multiplexers (IT8898 chips) expand the support to 4 PCIe x16 devices at full speed. Two PCIe devices have high chances to share the same switch (*e.g.*, plugged into Slot 3 and Slot 5 to have x16 PCIe link).

The RDMA NIC of the server has a direct Infiniband connection to the PC's RDMA NIC. Though in data centers, there is usually an Infiniband switch between two of its machines to enable RDMA connection, we do not introduce the switch into the experiment platform because it is too expensive (over 240,000 US dollars [62]). On the other hand, the latency with and without the Infiniband switch are similar (*e.g.*, extra 130 nanoseconds [38]).

B. User-input Inference with RDMA NIC

We use gedit and Google Chrome (version 79.0.3945.88) to simulate the applications that receive the victim's input. Two webpages, Google login page (`accounts.google.com/ServiceLogin/signinchooser`) and Amazon login page (`www.amazon.com/ap/signin`), are tested on

TABLE I: Specification of the evaluation platform.

	Server	PC
CPU	Intel Xeon Platinum 8260	Intel Core i3 8100
Motherboard	Supermicro X11SPA-TF	MSI B360
Memory	64G	32G
RDMA NIC	Mellanox MCX556A	Mellanox MCX455A
Ethernet NIC	Intel I-210	Intel I-219-V
GPU	Nvidia 1080 Ti	-
NVMe SSD	-	GP-ASM2NE6100TTTD
OS	Ubuntu 18.04 (4.15.0-112)	Ubuntu 18.04 (5.0.0-31)

Google Chrome. The victim types in the text on the applications on the server through a remote desktop software TigerVNC 1.7.0, and the attacker recovers keystroke events at her PC side. The remote GPU acceleration is enabled by configuring the X display name field of TigerVNC to 0.

Inference of keystroke events. We first evaluate the inference accuracy of the victim's keystroke events under INVISIPROBE. We recruit 1 volunteer to type in an English article, containing 480 characters. We align the timeline of the actual keystrokes and the detected ones, and count the false positives (the keystrokes detected by mistakes, or FP) and false negatives (the missed keystrokes, or FN). Assume the number of true positives (all real keystrokes) is C, false-positive rate (FPR) and false-negative rate (FNR) are defined as $\frac{FP}{C}$ and $\frac{FN}{C}$.

The result shows all events can be identified at the right time, resulting in **0%** FNR and FPR. However, without removing the blinking caret, some of the detected keystrokes are false positives, resulting in 5.88% FPR (30 caret blinkings). After removing the carets based on the heuristics described in Section IV, we found FPR drops to 1.03% (5 carets recognized as keystrokes), but FNR is still 0%. Table II lists the accuracy (*i.e.*, 1-FPR).

TABLE II: Accuracy of recovering keystroke events.

Keystroke & Caret Events	w/o Caret Removal	w/ Caret Removal
100%	94.18%	98.97%

Word recovery. We construct an HMM by following the steps in [12] with a ten-letter alphabet ("etaoinshrd"). There are 100 possible transitions between a pair of letters in the alphabet and we collect 40 keystroke intervals for each pair with a volunteer, and use the intervals to train the HMM. Then we draw the first 43 words from a 1000-words dictionary [63] that are composed of our alphabet as the words to be tested. Appendix D lists the selected words. Each word was typed 3 times by another volunteer and the recovered keystroke timing by the prior stage was sent to the HMM model for the word inference. We found the real words are highly ranked among the 1000 words in the dictionary after sorted by HMM: **66.7%** ranked in the top 10 and **95.3%** among the top 50. Our result is comparable to [12], which tested 39 words from a 2103-words dictionary, and the top-10 accuracy and top-50 accuracy are 40% and 86%.

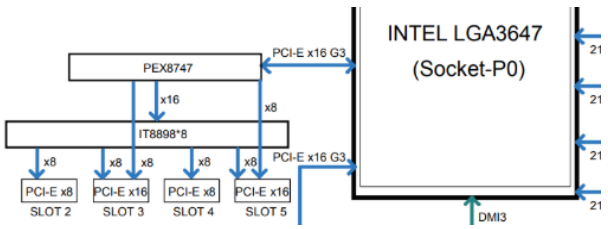


Fig. 6: Part of the block diagram of Supermicro X11-SPA TF.

TABLE III: Paramters of AttBLSTM model.

	Attack		
Paramter	Web-GPU	ML-GPU	Web-NIC
Hidden Size	48	48	64
Attention Size	64	64	64
Embedding Size	64	48	64
Learning Rate	0.022	0.016	0.016
LR Decay	0.992	0.995	0.992
Weight Decay	0.0003	0.0006	0.0004
Dropout	0.16	0.3	0.18

C. Webpage Inference with RDMA NIC

We select the Alexa top 100 website homepages [64][5] as the webpages targeted by the attacker. The scale of our data is comparable to prior works that infer webpage visits through side-channels, *e.g.*, 100 websites for [65], [66]. We use the top 100 websites to demonstrate attack effectiveness. In other words, an attacker targeting a specific victim can adjust the list to include websites that reflect the victim's preferences, political views, *etc*.

Each webpage was visited 150 times using Google Chrome and each visit lasts 8 seconds. In total 15,000 delay sequences were collected. We carried out cross-validation by splitting the data into training and testing by 85% and 15% randomly. We train the AttBLSTM model for 400 iterations with the parameters shown in the second column of Table III.

Classification mode. The evaluation metrics are similar to Section VI-B in that we measure the accuracy of the top-N candidates against the 100 tested webpages. We found the top-1 accuracy can achieve **96.31%**, *i.e.*, 96.31% sequences can be correctly classified with the top choice from our model, while the top-3 accuracy can achieve **99.16%**. We analyzed the mis-classified sequence and found that the main reason is the unstable network condition for some visits.

Embedding mode. We collected 100 new webpages (termed W_{new}) to test this mode. The trained AttBLSTM model is the same as the one used for the classification mode. For each webpage in W_{new}, we first collect 1 delay sequence (e_i) and generated a set of embedding vectors (termed E_{new}). Then, we collect another delay sequence (e_i') for every webpage in W_{new}, and compute its Cosine distance to every vector in E_{new}. If the distance between e_i and e_i' under the same webpage is under the threshold (0.474) while the distances between e_i and all other embedding vectors are larger, we consider it as a true positive. In the end, **90.77%** accuracy can be achieved on W_{new}.

[5]Due to that our machine is located in China, we selected the top 100 Chinese websites, which are not blocked by the Great Firewall.

D. Model Inference with RDMA NIC

We choose ten popular machine-learning models (8 DNN and 2 simple models) and collect their delay sequences for evaluation. Each model is trained for 100 iterations and each iteration uses 64 images from MNIST [67] in a batch. In total 1,000 sequences are collected, and the adversary's goal is to learn which model it corresponds to. Table IV shows the models we use for evaluation. The third column of Table III shows the parameters of AttBLSTM for this task.

TABLE IV: Machine-learning models evaluated under INVISIPROBE.

Model	Description
FC	A simple model with only 1 FC layer
CPF	A model with Conv-Pool-FC layers
CPCPF	A model with Conv-Pool-Conv-Pool-FC layers
DCGAN [68]	Deep Convolutional GAN
RNN-LSTM [69]	RNN model with LSTM layer
AutoEncoder [69]	CNN model with encoder and decoder
RNN-Attention [70]	RNN model with Attention layer
Resnet-18 [71]	A widely used CNN model for image recognition
Inception-v3 [72]	A widely used CNN model for image recognition
Regression	A simple logistic regression model

Similar to the setting of Section VI-C, we split the sequences to 85% training and 15% testing, and evaluate the top-N accuracy. The result shows the top-1 accuracy can achieve **100%** for all models. We speculate such high accuracy is caused by the long execution time for each iteration, which leaves abundant information to be used by INVISIPROBE.

We also attempted to infer the layers composing a DNN model from a delay sequence, however, we are unable to obtain a satisfying result, because segmenting the sequence by layer is very difficult. Appendix B elaborates this attempt.

E. Webpage Inference with NVMe SSD

In this attack, we use the same set of webpages (Alexa top 100) and experiment setting (85% training and 15% testing) as Attack 2 (Section VI-C), except that the attacker collects delay sequence from NVMe SSD. The last column of Table III shows the parameters of AttBLSTM for this task.

Overall, the top-1 accuracy reaches **93.29%**, and top-3 accuracy reaches **98.71%** for the classification mode. The accuracy is comparable to that of Attack 2, though a slight drop is observed. We found the main cause of the errors is also the unstable network conditions.

Classification Model	Top-1 Acc.	Top-3 Acc.	Embedding Acc.
3-layer fully-connected	48.92%	64.44%	67.26%
LSTM	84.25 %	94.00%	83.00%
Bi-directional LSTM	87.38%	96.10%	83.46%
AttBLSTM	93.29%	98.71%	88.17%

TABLE V: Accuracy (Acc.) of Attack 4 with different classification models.

Comparison between inference models. Comparing with a normal neural network model, like a fully-connected model, AttBLSTM adds three features, including long-sequence learning with LSTM, bi-directional sequence processing, and attention mechanism. Here we rerun the experiment of Attack 4 by stripping off each feature, in order to learn how much performance gain is brought by them. Table V shows the comparison. We use the 3-layer fully-connected model as the baseline, and LSTM significantly improves the performance (Top-1 accuracy raised from 48.92% to 84.25%), indicating the recurrent models are much better positioned to handle this task. BLSTM raises the Top-1 accuracy over LSTM by 3.13% and AttBLSTM improves over BLSTM by 5.91%, suggesting these two features are necessary. The trend of Top-3 classification accuracy and embedding accuracy are similar.

Performance gain with kernel-bypass. We assume the kernel-bypass driver (*e.g.*, SPDK) has been installed on the server for the attack leveraging NVMe SSD, as described in Section III-B. The kernel-bypass driver is expected to help the attack obtain a stable measurement of delays, and here we evaluate this claim.

In particular, we rerun Attack 4 but use the standard system calls to collect the delays. Specifically, `sys_read` system call is repeatedly invoked to read a 4K Block from a file at random positions. To force the OS to fetch data from NVMe disk without using buffers, we open the file in `O_DIRECT` mode. We look into the quality of sampled delays, and the result proves our assumption. Table VI compares the average IOPS (I/O operations per second) by the probe when one webpage is tested, and it shows without SPDK, much fewer samples can be collected (57,603 compared to 839,546). The reason is that the context switching between ring 0 and ring 3 wastes many CPU cycles. Besides, more random noise is introduced with `sys_read`: the standard deviation of the collected delay becomes 71 microseconds, while it is only 0.22 microseconds for the SPDK case. Regarding the accuracy of Attack 4, it is dropped to 90.13% and 96.53% for top-1 and top-3, shown in Table VI.

	IOPS	Top-1 Acc.	Top-3 Acc.
SPDK	839,546	93.29%	98.71%
`sys_read`	57,603	90.13%	96.53%
Alibaba Cloud with `io_uring`	12,757	91.02%	97.77%

TABLE VI: IOPS of different probe designs and inference accuracy for Attack 4.

F. Attack on Public Cloud

The prior experiments are done in a lab environment. To evaluate the practical impact of INVISIPROBE, we implement Attack 4 in the Alibaba cloud. We did not implement Attack 1-3 because we cannot find a public cloud matching the attack condition, that two VMs with RDMA connection can be rented. Tsai *et al.* [24] tested their RDMA attack in CloudLab [73], but we found CloudLab uses AMD CPUs, which uses I/O die inside the CPU to forward PCIe traffic. Specifically, AMD systems do not break CPU's PCIe port into multiple ports with a switch. Instead, they use an I/O chip to connect all PCIe ports, cores and memory controllers together,

in which case a PCIe port not only conflict with other ports but also cores and memories, producing much more noises.

One interesting observation we gain from inspecting the public cloud is that *I/O virtualization* is extensively leveraged. In this case, cloud vendors do not use the PCH to serve SSDs and NICs together. I/O devices including SSDs, NICs are all virtualized by hardwares, which are called Nitro Cards by Amazon [74] and MOC Cards by Alibaba [75]. We call them *vcard* collectively. The vcard rather than PCH act as an I/O switch to handle PCIe traffic. Particular for Alibaba, a cloud server has only one MoC, no matter how many devices it virtualizes. Therefore, similar to PCH, vcard becomes the bottleneck for PCIe traffic.

As for the end-to-end attack, we rented an instance (type `ecs.ebmc6.26xlarge`) from Alibaba cloud and launched Attack 4. We run the attacker's code on the instance to access NVMe SSD and the webpages are visited by Google Chrome on the instance. We found directly running the attack code is infeasible, because the virtualized SSD does not support SPDK. Fortunately, the OS kernel of the instance supports `io_uring`, a kernel-bypass framework provided by Linux. Therefore, we substitute the part relying on SPDK with `io_uring` and run the attack code. It turns out top-1 accuracy and top-3 accuracy are dropped to 91.02% and 97.77%, as shown in Table VI. The main reason is that the sampling quality of INVISIPROBE degrades on vcard: only 12,757 IOPS is produced. Yet, high attack effectiveness is demonstrated.

VII. DISCUSSION

A. Potential Mitigation

The evaluation result suggests INVISIPROBE can harm another user's security and privacy sharing a machine. On the other hand, defending against INVISIPROBE is quite challenging. Other remote cache side-channel attacks [23], [24] can be mitigated by isolation (*e.g.*, LLC partitioning [23] and separating protection domains [24]). However, simply isolating traffic between devices might eliminate the benefit brought by PCIe. Below we discuss a few directions.

Blocking high-resolution clock instructions. Assuming the data center operator can decide what instructions are available to the upper-layer applications, *i.e.*, the probe code written by the attacker, restricting the access to those instructions might deter INVISIPROBE. For instance, when using NIC to attack GPU, the hardware clock has to be obtained by the adversary by setting flags in the RDMA read queue. When using NVMe SSD to attack NIC, `RDTSCP` needs to be invoked by the adversary. However, legitimate applications also use those primitives to measure their performance. Even if certain instructions can be banned, *e.g.*, `RDTSCP`, the attacker can switch to other instructions, *e.g.*, using multiple threads and monitoring the interrupts ("Low-Requirement Interrupt Timing Attack" of [47]).

Reducing the delay variance. As the attacker can construct a probe quite sensitive to the delays caused by congestion, another solution is to reduce the delay variance, so the congestion states of the PCIe link can be concealed. One can increase the capacity of PCIe links, but doing so would incur high upgrading cost and there is no guarantee that congestion will never be triggered. Introducing noises, *i.e.*, increasing delays, would confuse the attacker, but the extra overhead could be non-negligible, especially to scenarios that are delay-sensitive, *e.g.*, training deep neural networks.

Detecting the suspicious probe requests. The adversary needs to maintain a non-empty queue of probe requests to keep collecting measurement data for the whole life-cycle of a victim operation. Blocking probe request individually is not a feasible solution, as reading a small chunk of memory by RDMA has legitimate use cases like distributed locks [76]. A more viable solution could be letting a security application on I/O switch to inspect the request sequence, identify the anomalies, and notify the data center operators. In fact, a few recent works have shown that programmable network switches can be leveraged to detect DDoS attacks [77], [78]. However, the current hardware of I/O switch does not support this idea, *i.e.*, not programmable. As revealed by our end-to-end experiment (Section VI-F), I/O switch is virtualized by some public cloud. Hence, an alternative approach is to deploy the defense on the vcard, and we are discussing this approach with cloud providers like Alibaba.

To notice, though processor vendors like Intel provide counters like PCM [79] to monitor PCIe bandwidth, it covers PCIe stop inside the chip only. What happens at the PCIe switch is oblivious to those counters, so the congestion caused by INVISIPROBE cannot be detected.

Better QoS on I/O switch. Though programming I/O switch to detect INVISIPROBE might be infeasible now due to the hardware restrictions, improving the QoS logic of I/O switch to better serve legitimate applications might deter INVISIPROBE. For example, by prioritizing I/O devices of high IOPS (I/O per second), the switch can guarantee relatively stable and lower delay on those devices, which reduces the inference accuracy of INVISIPROBE. PCIe congestion has been studied in PCIe fabric and a few congestion-aware models were proposed [29], [30]. Within the PCIe standard, *virtual channel* [80] aims at a similar goal, by mapping dedicated physical resources like buffers to high-priority transactions, eliminating resource conflicts with the low-priority traffic. We plan to investigate whether they could thwart INVISIPROBE and the follow-up counterattack in the future.

B. PCIe fabric

Resource disaggregation is a general trend for data centers and cloud [81]. In a fully resource-disaggregated rack, I/O devices are all connected to PCIe switches, which constitutes a PCIe fabric, as shown by Figure 7. CPUs also connect to the PCIe fabric as nodes. In this case, the attack surface will be largely broadened: *all I/O devices in the rack will be exposed to attackers for probing*. So far, there lacks comprehensive security analysis of PCIe fabric and we believe this is an urgent topic for the security community.

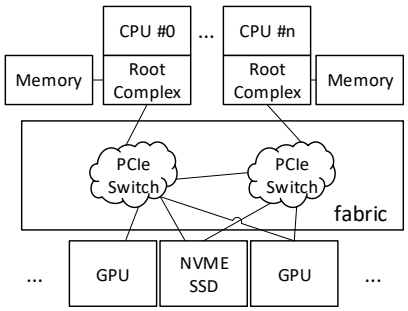


Fig. 7: PCIe fabric.

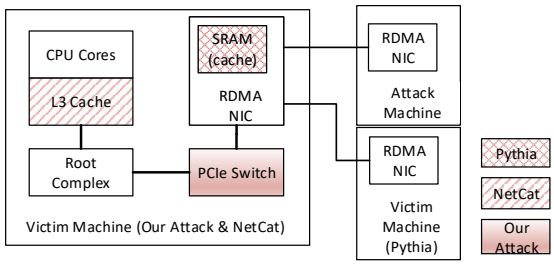


Fig. 8: Comparison between Pythia, NetCat and our attack. Shaded parts show where the attack is focusing on.

C. Limitations

User-input inference. We aim to accurately infer the words typed by a victim in Attack 1, though there is still room for improvement (*e.g.*, top-10 accuracy is 66.7%). Future work could evaluate models other than HMM and use the context to filter the inferred words. Given only the coarse-grained feature, say keystroke timing, is available to the adversary, INVISIPROBE cannot directly learn more sensitive information like password and credit card numbers. Still, we believe the word recovery example has demonstrated that PCIe congestion side-channel should be mitigated.

Webpage inference. For Attack 2 and 4, we classified 100 webpages in both scenarios, which might be considered as a small dataset. The main reason is that data collection is time-consuming: every visit costs 10 seconds and each webpage is visited repeatedly for 150 times. We plan to increase the dataset in the future by running multiple machine instances. On the other hand, as our dataset covers webpages of a variety of categories, we believe the result is representative. We also assume a close-world setting where the victim visits the webpages that have been profiled. As a next step, we will evaluate whether the victim is also vulnerable under the open-world setting.

Model inference. For Attack 3, INVISIPROBE can tell which model a victim is executing but the layers cannot be inferred separately. We acknowledge this cannot fully reveal the model structure. On the other hand, if the attacker can profile a large number of models ahead, and compute the distance between a delay sequence to the profiles, she might find a similar model structure, which still helps her attempt in IP infringement and adversarial attacks. We will investigate this option.

Combinations of attack and victim devices. We tested two device combinations. Potentially, other device combinations might be vulnerable as well, but due to the high cost of testing one combination, we leave the exploration of other pairs as future work. We also plan to investigate principled methodologies to reduce the search space of vulnerable pairs.

Public cloud setting. Since INVISIPROBE is implemented based on the features and ISA of Intel CPUs, the public cloud supporting RDMA, in particular CloudLab, cannot be evaluated against, because it only has AMD CPUs. We leave the attack implementation AMD platforms as a future work.

Single-CPU setting. Though INVISIPROBE is expected to be used as a cross-CPU attack vector, e.g., in a full disaggregated server rack, so far, we only evaluated INVISIPROBE under the single-CPU setting because our motherboard only has one CPU slot. We plan to evaluate INVISIPROBE under the cross-CPU setting with a new motherboard, e.g., Supermicro X11DPH-T.

PCIe topology. In evaluation, we assume the adversary knows the PCIe topology of the targeted machine, and then selects attacker and victim peripheral devices accordingly. When the information is unavailable, the attacker would need to infer which victim device shares the I/O switch. This task is similar as VM co-residency attack [82], in which the adversary leverages side-channels to determine when her VM shares a physical machine with another victim VM. The probing delays observed by the adversary might fulfill this task, and we plan to validate this hypothesis.

VIII. RELATED WORKS

Side-channel attacks in data center. Recently, a few works studied how RDMA can be exploited to break the confidentiality of machines/programs in a data center with cache-based side-channel attacks. In the attack named NetCat [23], Kurth *et al.* showed that an attacker can remotely Prime+Probe the LLC of the victim machine with the help of RDMA NICs and Intel's special cache mechanism named DDIO (Data-Direct I/O). The attacker is able to infer the memory access pattern of the victim machine, resulting in consequences like password leakages. In the attack named Pythia [24], Tsai *et al.* found that Evict+Reload can be launched against metadata stored in the SRAM on RDMA NIC. With this attack, other RDMA nodes' access patterns can be inferred.

Our attacks explore another direction to launch attacks on RDMA NICs. The two literature [23], [24] focused on cache timing on host CPU and NICs respectively. Our attacks focus on timing related to PCIe links between host CPU and NICs. Figure 8 shows the comparison.

Security implications of bandwidth contention. Previous works have shown bandwidth contention can be exploited for side-channel and covert-channel for attacks. Hu *et al.* [83], [84] and Gray *et al.* [85], [86] studied the covert channels based on bus/cache contention between VMs managed by VAX security kernel. Wu *et al.* investigated a similar attack in the contemporary public cloud environment [87]. DRAMA exploits the DRAM row buffers that are shared in multiprocessor systems for cross-CPU attacks [88]. Irazoqui *et al.* also conducted cross-CPU attacks on the CPU interconnect [89]. None of the prior works investigated the issues on the high-speed I/O protocols like PCIe, and we made the first attempt.

Keystroke inference. Keystroke inference was studied based on the network communication patterns [90]. Following that, a number of works studied how information leaked from software and hardware can be exploited for the same purpose. For instance, Zhang *et al.* [12] proposed that the pattern of ESP register value of a thread can be used as fingerprints of keystroke events. Schwarz *et al.* proposed KeyDrown [47] showing 1) timing attack on keyboard interrupt and 2) cache attack on the interrupt handler in the kernel can lead to keystroke inference. Wang *et al.* showed that by launching cache-based side-channel attacks against graphic libraries, the same goal can be achieved [48]. INVISIPROBE reveals a new remote side-channel for keystroke inference.

Website inference. When the adversary is able to eavesdrop the traffic between the victim user and a remote entity, even if the traffic is encrypted, which website might is visited can still be inferred, based on packet sizes and time intervals [91]–[95]. As more and more browsers choose to render webpages in the GPU, which webpage is visited can be recovered through GPU-based side-channel, in memory residue, access patterns and performance counters [25], [65], [96]. Besides, cache side-channel attacks have also been found effective for website inference [66], [97].

Stealing machine-learning model. As the machine-learning model structure can be considered as a secret, recently a number of works studied how it can be inferred with side-channel attacks. The exploited side-channels include power consumption [98]–[100], CPU cache [101]–[103] and GPU resource contention [25], [26]. The one closest to our work snoops PCIe bus [59]. The attacker needs physical access to the edge device to obtain such information. Our adversary can be remote.

IX. CONCLUSION

PCIe congestion resulted from the insufficient forwarding capability of I/O switches introduces I/O delays to a connected device. When exploited by an attacker, who intentionally introduces PCIe congestion, sensitive user activities on a device can be inferred. We identified four attacks in two scenarios (using RDMA NIC to attack GPU and using NVMe SSD to attack NIC), showing sensitive information like the keystroke timings, webpage visits, trained machine-learning models can be inferred at high accuracy. We call the awareness of server manufacturers and the security community, and our study can serve as the motivation to design security-enhanced PCIe implementations.

X. ACKNOWLEDGEMENT

We thank the valuable comments from the anonymous reviewers, who have guided us to significantly improve the paper from its initial version. The authors from Fudan University are supported by NSFC 61802068 and Shanghai Sailing Program 18YF1402200.

REFERENCES

[1] "The three most common ethernet speeds," https://smallbusiness.chron.com/three-common-ethernet-speeds-69375.html, accessed: 2020-02-11.

[2] "Nvme ssds: Everything you need to know about this insanely fast storage," https://www.pcworld.com/article/2899351/everything-you-need-to-know-about-nvme.html, accessed: 2020-02-11.

[3] R. Neugebauer, G. Antichi, J. F. Zazo, Y. Audzevich, S. López-Buedo, and A. W. Moore, "Understanding pcie performance for end host networking," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 327–341.

[4] "Pcie 3.0 x8 vs. x16: Does it impact gpu performance?" https://www.gamersnexus.net/guides/2488-pci-e-3-x8-vs-x16-performance-impact-on-gpus, accessed: 2020-02-11.

[5] "2nd generation intel xeon scalable processors with intel c620 series chipsets (purley refresh)," https://www.intel.com/content/www/us/en/design/products-and-solutions/processors-and-chipsets/cascade-lake/2nd-gen-intel-xeon-scalable-processors.html, accessed: 2020-02-29.

[6] "Tyan thunder hx ft77db7109," https://www.tyan.com/Barebones_FT77DB7109_B7109F77DV14HR-8X-2T-F, accessed: 2020-02-20.

[7] A. Burnes, "Introducing NVIDIA RTX IO," https://www.nvidia.com/en-us/geforce/news/rtx-io-gpu-accelerated-storage-technology/, 2020, [Online; accessed 18-December-2020].

[8] L. Yin, X. Chen, Z. Qin, Z. Zhang, J. Feng, and D. Li, "An experimental perspective for computation-efficient neural networks training," in *Conference on Advanced Computer Architecture*. Springer, 2018, pp. 168–178.

[9] R. Budruk, "Pci express basics," in *PCI-SIG Developers Conference*, 2007.

[10] W. Sun, L. Xu, S. Elbaum, and D. Zhao, "Model-agnostic and efficient exploration of numerical state space of real-world TCP congestion control implementations," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019, pp. 719–734.

[11] S. Jero, M. E. Hoque, D. R. Choffnes, A. Mislove, and C. Nita-Rotaru, "Automated attack discovery in tcp congestion control using a model-guided approach." in *ANRW*, 2018, p. 95.

[12] K. Zhang and X. Wang, "Peeping tom in the neighborhood: Keystroke eavesdropping on multi-user systems," in *USENIX Security Symposium*, vol. 20, 2009, p. 23.

[13] P. Zhou, W. Shi, J. Tian, Z. Qi, B. Li, H. Hao, and B. Xu, "Attention-based bidirectional long short-term memory networks for relation classification," in *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*, 2016, pp. 207–212.

[14] "Intel pcie introduction," https://www.intel.com/content/www/us/en/io/pci-express/pci-express-architecture-general.html, accessed: 2020-02-29.

[15] "Down to the tlp: How pci express devices talk," http://xillybus.com/tutorials/pci-express-tlp-pcie-primer-tutorial-guide-1, accessed: 2020-02-11.

[16] "8th generation intel core processor family and intel xeon processor e-2100m family (coffee lake h)," https://www.intel.com/content/www/us/en/design/products-and-solutions/processors-and-chipsets/coffee-lake-h/overview.html, accessed: 2020-02-29.

[17] T. Shanley, *InfiniBand network architecture*. Addison-Wesley Professional, 2003.

[18] M. Beck and M. Kagan, "Performance evaluation of the rdma over ethernet (roce) standard in enterprise data centers infrastructure," in *Proceedings of the 3rd Workshop on Data Center-Converged and Virtual Ethernet Switching*, 2011, pp. 9–15.

[19] F. D. Neeser, B. Metzler, and P. W. Frey, "Softrdma: Implementing iwarp over tcp kernel sockets," *IBM Journal of Research and Development*, vol. 54, no. 1, pp. 5–1, 2010.

[20] C. Mitchell, Y. Geng, and J. Li, "Using one-sided RDMA reads to build a fast, cpu-efficient key-value store," in *Presented as part of the 2013 USENIX Annual Technical Conference (USENIXATC 13)*, 2013, pp. 103–114.

[21] J. Shi, Y. Yao, R. Chen, H. Chen, and F. Li, "Fast and concurrent RDF queries with rdma-based distributed graph exploration," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 317–332.

[22] J. Xue, Y. Miao, C. Chen, M. Wu, L. Zhang, and L. Zhou, "Fast distributed deep learning over rdma," in *Proceedings of the Fourteenth EuroSys Conference 2019*, 2019, pp. 1–14.

[23] M. Kurth, B. Gras, D. Andriesse, C. Giuffrida, H. Bos, and K. Razavi, "Netcat: Practical cache attacks from the network," in *Proceedings of IEEE Security & Privacy 2020*. IEEE, 2020.

[24] S.-Y. Tsai, M. Payer, and Y. Zhang, "Pythia: remote oracles for the masses," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 693–710.

[25] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, "Rendered insecure: Gpu side channel attacks are practical," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2139–2153.

[26] J. Wei, Y. Zhang, Z. Zhou, Z. Li, and M. A. A. Faruque, "Leaky DNN: stealing deep-learning model secret with GPU context-switching side-channel," in *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2020, Valencia, Spain, June 29 - July 2, 2020*, 2020, pp. 125–137.

[27] "Nvme over fabrics," https://nvmexpress.org/wp-content/uploads/NVMe_Over_Fabrics.pdf, accessed: 2020-03-05.

[28] D. J. Miller, P. M. Watts, and A. W. Moore, "Motivating future interconnects: a differential measurement analysis of pci latency," in *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 2009, pp. 94–103.

[29] M. Martinasso, G. Kwasniewski, S. R. Alam, T. C. Schulthess, and T. Hoefler, "A pcie congestion-aware performance model for densely populated accelerator servers," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 739–749.

[30] M. Martinasso and J.-F. Méhaut, "A contention-aware performance model for hpc-based networks: A case study of the infiniband network," in *European Conference on Parallel Processing*. Springer, 2011, pp. 91–102.

[31] L. Soares and M. Stumm, "Flexsc: Flexible system call scheduling with exception-less system calls." in *Osdi*, vol. 10, 2010, pp. 1–8.

[32] "What is rdma?" https://community.mellanox.com/s/article/what-is-rdma-x, accessed: 2020-08-27.

[33] Z. Yang, J. R. Harris, B. Walker, D. Verkamp, C. Liu, C. Chang, G. Cao, J. Stern, V. Verma, and L. E. Paul, "Spdk: A development kit to build high performance storage applications," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2017, pp. 154–161.

[34] "Userspace networking with dpdk," https://www.linuxjournal.com/content/userspace-networking-dpdk, accessed: 2020-02-29.

[35] "Rdtscp," https://www.felixcloutier.com/x86/rdtscp, accessed: 2020-02-29.

[36] "Ip over infiniband (ipoib)," https://docs.mellanox.com/pages/viewpage.action?pageId=12004991, accessed: 2020-08-27.

[37] "Which queue pair type to use," https://www.rdmamojo.com/2013/06/01/which-queue-pair-type-to-use/, accessed: 2020-02-29.

[38] "Introducing 200g hdr infiniband solutions," https://www.mellanox.com/related-docs/whitepapers/WP_Introducing_200G_HDR_InfiniBand_Solutions.pdf, accessed: 2020-02-25.

[39] C. Maurice, C. Neumann, O. Heen, and A. Francillon, "C5: cross-cores cache covert channel," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2015, pp. 46–64.

[40] "Enabling gpu rendering on windows server 2016 / windows 10 rdp," https://community.esri.com/thread/225251-enabling-gpu-rendering-on-windows-server-2016-windows-10-rdp, accessed: 2020-08-27.

[41] "Creating a virtual gpu-accelerated linux workstation," https://cloud.google.com/solutions/creating-a-virtual-gpu-accelerated-linux-workstation, accessed: 2020-08-27.

[42] "Deploying a 4x4k, gpu-backed linux desktop instance on aws," https://aws.amazon.com/cn/blogs/compute/deploying-4k-gpu-backed-linux-desktop-instance-on-aws/, accessed: 2020-02-20.

[43] Unknown, "Steamworks," https://partner.steamgames.com/, 2020, [Online; accessed 18-December-2020].

[44] ——, "Rethinking Visual Cloud Services for Evolving Media," https://www.intel.ru/content/dam/www/public/us/en/documents/guides/vcd-wp-v6.pdf, 2020, [Online; accessed 18-August-2020].

[45] "Remote desktop services - gpu acceleration," https://docs.microsoft.com/en-us/windows-server/remote/remote-desktop-services/rds-graphics-virtualization, accessed: 2020-08-27.

[46] "Remote desktop software statistics and trends," https://www.trustradius.com/vendor-blog/remote-desktop-buyer-statistics-and-trends, accessed: 2020-08-27.

[47] M. Schwarz, M. Lipp, D. Gruss, S. Weiser, C. Maurice, R. Spreitzer, and S. Mangard, "Keydrown: Eliminating software-based keystroke timing side-channel attacks," 2018.

[48] D. Wang, A. Neupane, Z. Qian, N. B. Abu-Ghazaleh, S. V. Krishnamurthy, E. J. Colbert, and P. Yu, "Unveiling your keystrokes: A cache-based side-channel attack on graphics libraries," in *NDSS*, 2019.

[49] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[50] "Gpu accelerated compositing in chrome," https://www.chromium.org/developers/design-documents/gpu-accelerated-compositing-in-chrome, accessed: 2020-02-20.

[51] "Webgl: 2d and 3d graphics for the web," https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API, accessed: 2020-02-29.

[52] "How to turn hardware acceleration on and off in chrome," https://www.howtogeek.com/412738/how-to-turn-hardware-acceleration-on-and-off-in-chrome/, accessed: 2020-02-11.

[53] B. Pourghassemi, A. Amiri Sani, and A. Chandramowlishwaran, "What-if analysis of page load time in web browsers using causal profiling," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 2, pp. 1–23, 2019.

[54] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proceedings*. Presses universitaires de Louvain, 2015, p. 89.

[55] N. Laptev, J. Yosinski, L. E. Li, and S. Smyl, "Time-series extreme event forecasting with neural networks at uber," in *International Conference on Machine Learning*, vol. 34, 2017, pp. 1–5.

[56] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[57] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman, "Sok: Security and privacy in machine learning," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 399–414.

[58] D. Lustig and M. Martonosi, "Reducing gpu offload latency via fine-grained cpu-gpu synchronization," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 354–365.

[59] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood *et al.*, "Deepsniffer: A dnn model extraction framework based on learning architectural hints," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 385–399.

[60] "Intel ethernet server adapter i210-t1," https://www.intel.com/content/www/us/en/products/network-io/ethernet/gigabit-adapters/server-i210-t1.html, accessed: 2020-02-29.

[61] "Storage system design analysis: Factors affecting nvme ssd performance," https://www.alibabacloud.com/blog/storage-system-design-analysis-factors-affecting-nvme-ssd-performance-2_594376, accessed: 2020-02-29.

[62] "infiniband switches," https://store.mellanox.com/categories/switches/infiniband-switches.html, accessed: 2020-02-29.

[63] Unknown, "English Word lists," https://github.com/mahavivo/english-wordlists/blob/master/COCA_20000.txt, 2020, [Online; accessed 18-August-2020].

[64] "Alexa top sites in china," https://www.alexa.com/topsites/countries/CN/, accessed: 2020-03-05.

[65] S. Lee, Y. Kim, J. Kim, and J. Kim, "Stealing webpages rendered on your browser by exploiting gpu vulnerabilities," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 19–33.
[66] A. Shusterman, L. Kang, Y. Haskal, Y. Meltser, P. Mittal, Y. Oren, and Y. Yarom, "Robust website fingerprinting through the cache occupancy channel," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 639–656.
[67] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[68] https://github.com/carpedm20/DCGAN-tensorflow, accessed: 2020-02-11.
[69] "Rnn-lstm/autoencoder/regression cite," https://github.com/MorvanZhou/Tensorflow-Tutorial/blob/master/tutorial-contents/, accessed: 2020-02-11.
[70] https://github.com/ilivans/tf-rnn-attention, accessed: 2020-02-11.
[71] https://github.com/Natsu6767/ResNet-Tensorflow, accessed: 2020-02-11.
[72] https://github.com/tensorflow/models/tree/master/research/inception, accessed: 2020-02-11.
[73] R. Ricci, E. Eide, and C. Team, "Introducing cloudlab: Scientific infrastructure for advancing cloud architectures and applications," *; login:: the magazine of USENIX & SAGE*, vol. 39, no. 6, pp. 36–38, 2014.
[74] "Aws nitro system," https://aws.amazon.com/ec2/nitro/, accessed: 2020-08-27.
[75] "Ali pingtou is developing a dedicated soc chip for cloud server core moc card," https://www.firstxw.com/view/236611.html, accessed: 2020-08-27.
[76] D. Y. Yoon, M. Chowdhury, and B. Mozafari, "Distributed lock management with rdma: decentralization without starvation," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1571–1586.
[77] Q. Kang, L. Xue, A. Morrison, Y. Tang, A. Chen, and X. Luo, "Programmable in-network security for context-aware byod policies," in *USENIX Security Symposium*, 2020.
[78] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," in *NDSS*, 2020.
[79] "Processor counter monitor (pcm)," https://github.com/opcm/pcm, accessed: 2020-08-26.
[80] I. Granovsky and E. Perlin, "Integrating pci express ip in a soc," https://www.design-reuse.com/articles/15545/integrating-pci-express-ip-in-a-soc.html, accessed: 2020-08-26.
[81] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, "Legoos: A disseminated, distributed OS for hardware resource disaggregation," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 69–87.
[82] A. O. F. Atya, Z. Qian, S. V. Krishnamurthy, T. F. L. Porta, P. D. McDaniel, and L. M. Marvel, "Malicious co-residency on the cloud: Attacks and defense," in *INFOCOM*. IEEE, 2017, pp. 1–9.
[83] W.-M. Hu, "Reducing timing channels with fuzzy time," *Journal of computer security*, vol. 1, no. 3-4, pp. 233–254, 1992.
[84] ——, "Lattice scheduling and covert channels," in *Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE Computer Society, 1992, pp. 52–52.
[85] J. W. Gray, "On introducing noise into the bus-contention channel," in *Proceedings 1993 IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE, 1993, pp. 90–98.
[86] J. W. Gray III, "Countermeasures and tradeoffs for a class of covert timing channels," Tech. Rep., 1994.
[87] Z. Wu, Z. Xu, and H. Wang, "Whispers in the hyper-space: high-bandwidth and reliable covert channel attacks inside the cloud," *IEEE/ACM Transactions on Networking*, vol. 23, no. 2, pp. 603–615, 2014.
[88] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM addressing for cross-cpu attacks," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 565–581.
[89] G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cross processor cache attacks," in *Proceedings of the 11th ACM on Asia conference on computer and communications security*, 2016, pp. 353–364.
[90] D. X. Song, D. A. Wagner, and X. Tian, "Timing analysis of keystrokes and timing attacks on ssh." in *USENIX Security Symposium*, vol. 2001, 2001.
[91] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in *2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 191–206.
[92] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail," in *2012 IEEE symposium on security and privacy*. IEEE, 2012, pp. 332–346.
[93] X. Luo, P. Zhou, E. W. Chan, W. Lee, R. K. Chang, and R. Perdisci, "Httpos: Sealing information leaks with browser-side obfuscation of encrypted flows." in *NDSS*, vol. 11, 2011.
[94] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, "Analyzing android encrypted network traffic to identify user actions," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 1, pp. 114–125, 2015.
[95] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 1187–1203.
[96] Z. Zhou, W. Diao, X. Liu, Z. Li, K. Zhang, and R. Liu, "Vulnerable gpu memory management: towards recovering raw data from gpu," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 2, pp. 57–73, 2017.
[97] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, "The spy in the sandbox: Practical cache attacks in javascript and their implications," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1406–1418.
[98] L. Batina, S. Bhasin, D. Jap, and S. Picek, "CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 515–532. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/batina
[99] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu, "I know what you see: Power side-channel attack on convolutional neural network accelerators," in *Proceedings of the 34th Annual Computer Security Applications Conference*, ser. ACSAC '18. New York, NY, USA: ACM, 2018, pp. 393–406. [Online]. Available: http://doi.acm.org/10.1145/3274694.3274696
[100] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18. New York, NY, USA: ACM, 2018, pp. 4:1–4:6. [Online]. Available: http://doi.acm.org/10.1145/3195970.3196105
[101] V. Duddu, D. Samanta, D. V. Rao, and V. E. Balas, "Stealing neural networks via timing side channels," *CoRR*, vol. abs/1812.11720, 2018. [Online]. Available: http://arxiv.org/abs/1812.11720
[102] M. Yan, C. W. Fletcher, and J. Torrellas, "Cache telepathy: Leveraging shared resource attacks to learn DNN architectures," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 2003–2020.
[103] S. Hong, M. Davinroy, Y. Kaya, S. N. Locke, I. Rackow, K. Kulda, D. Dachman-Soled, and T. Dumitras, "Security analysis of deep neural networks operating in the presence of cache side-channel attacks," *CoRR*, vol. abs/1810.03487, 2018. [Online]. Available: http://arxiv.org/abs/1810.03487

APPENDIX A
I/O LATENCY UNDER KERNEL-BYPASSING

Traditionally, an I/O operation consists of four segments, as shown in Figure 9 (left). Here we use a C library API `fread()`, which reads file content into memory buffer, as the example. When it is executed by the CPU, `A`) a request will be issued from the user space to the kernel space, going through drivers (file system, block, and PCI), till delivered to the hard drive; `B`) the hard drive processes the request; `C`) the results are written to memory; `D`) the code waiting for the result is notified about the request completion. To measure the I/O latency of `fread()`, the developer can execute `clock()` before and after and compute the interval, say `I`, which equals to the sum of `A`, `B`, `C`, and `D`.

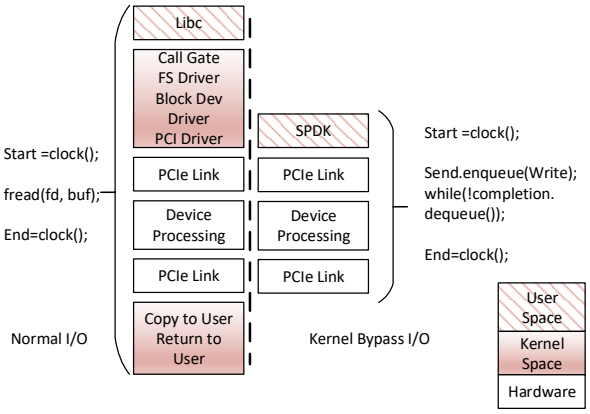


Fig. 9: Normal I/O vs Kernel-bypass I/O.

After PCIe congestion, `I` is supposed to be increased and the adversary tries to use it to infer the status of the victim device. However, such measurement is not always reliable as `A` and `D` involve system calls and interrupt handling, whose latency is pretty random [31]. `B` and `C` are directly related to the operational status of the I/O device, which can reflect the degree of PCIe congestion, but they are overwhelmed by `A` and `D`.

However, the procedure is largely different when the kernel-bypass drivers are used. As shown in Figure 9 (right), the developer uses *send queue* and *completion queue* to decouple OS from the I/O path. The user-space application directly writes a request to a memory region that is mapped to the I/O device, eliminating `A`. On completion, the result is written into a pre-allocated memory region to be polled by the application without interrupting CPU, eliminating `D`. As a result, `I` is composed of `B` and `C`, making measurement on PCIe congestion possible. In fact, the latency `I` can be "amplified" several times. For instance, In the setting that a GPU shares a link with a NIC, the delay of packet receiving on the NIC (`I`) can go up *6 times*, when link congestion is caused by the GPU whose status turns from idle to exchanging data, as we will show in Figure 2 in Section IV.

Appendix B
Attempt of Recovering Layers

According to Wei *et al.* [26], when TensorFlow is leveraged to run a DNN model, executing each layer involves data and code exchange between CPU and GPU, which should introduce intermittent PCIe traffic. We attempted to segment the delay sequence and recognize the layers, but found the PCIe traffic of different layers are interleaved. We execute two models (one with a conv layer and one with relu and pool layers) and their combination (conv-relu-pool). The delay sequence is shown in Figure 10. As we can see, the delay sequence from conv-relu-pool model is not simply the concatenation of conv and relu-pool.

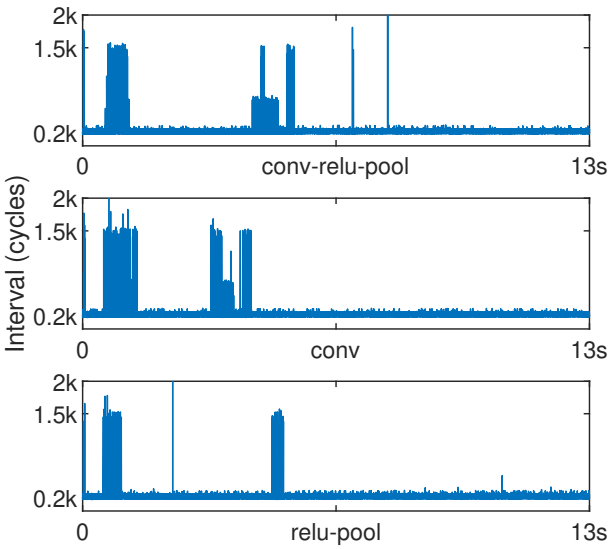


Fig. 10: The delay sequences of the three tested models, which are collected by RDMA NIC.

Appendix C
Applications Investigated for Caret Blinking

We investigated 10 applications in Ubuntu to see how many of them have blinking caret during user typing, but found only Firefox, as shown in by Table VII, has this behavior.

App name	Version	Comment
Google Chrome	79.0.3945.88	
Firefox	80.0.1-1	Caret Blinks
gedit	3.28.1	
PyCharm	2020.2.1	
Sublime Text	3211	
VS Code	1.48.2	
LibreOffice Writer	1:6.0.7	
LibreOffice Calc	1:6.0.7	
IDEA Community	2020.2.1	
Skype	8.64.0.67	

TABLE VII: Applications investigated for caret blinking.

Appendix D
List of Words Tested for Word Recovery

hero, test, need, that, rise, star, aids, hate, diet, road, hers, host, than, tree, sit, dad, eat, dna, sin, net, its, rid, ear, her, art, toe, tie, hit, radar, sense, their, heart, share, taste, harsh, ratio, north, trend, shore, order, noise, trash, stand.