Active Profiling of Physical Devices at Internet Scale

Xuan Feng*¶, Qiang Li †, Qi Han‡, Hongsong Zhu*, Yan Liu §, Jie Cui ‖, Limin Sun*

* Institute of Information Engineering, Chinese Academy of Sciences, China

†School of Computer and Information Technology, Beijing Jiaotong University, China

§ College of Software and Microelectronics, Peking University, China

‡Department of EECS, Colorado School of Mines, Golden, CO USA

¶ University of Chinese Academy of Sciences, China

‖China General Technology Research Institute, China

***Abstract*—Nowadays, more and more physical devices embed computing and networking capabilities and are visible on the Internet. These devices include webcams, net-printers, and industrial control equipments, etc. Collecting information about these devices is crucial to preserve cyber-security and facilitate security auditing for system administrators. In this paper, we propose a scalable framework for physical device profiling. It leverages banner grabbing to identify device types and running services, and uses clock skew to determine a device ID. Our framework scales well. We implement a prototype system and use it to profile Webcams and industrial control device. The results show that our system can effectively profile and identify Webcams in real time. We deploy it on the cloud server and use it to detect 4 billion IP addresses to profile 1.2 million Webcams and more than 60 thousand industrial control devices in 20 hours.**

I. Introduction

We are on the brink of a new era in the development of Internet of Things (IoT) [1], where more and more physical devices are visible on the Internet. There are many types of these physical devices with an IP address, such as routers, net-printers, webcams, switches, bridges and industrial control devices. These devices are fundamental to daily life and work place, including homes, factories, environments, grids and cities, enabling numerous people-centric smart applications. Perhaps one of the most overlooked “things”, available on every physical devices, is the profile of a physical device. A device profile contains its ID, type, services, and IP address, necessary to things in the cyber-space. This paper focuses on remotely extracting information of physical devices at Internet scale.

A device’s profile is rich in the sense that the information can be used to identify a physical device in the cyber space, bringing about many benefits. One advantage is to help system administrators with security auditing [2], discover and patch servers to preserve device security in the cyber-space. It also helps forensic and track individual devices on the Internet [3]. For instance, we can provide an evidence to show that whether a given device was involved in a recorded event or not. The last but not least is that network measurement statistics can be improved by detailed information of devices [4], [5]. This is because network measurement is largely influenced by dynamic host configuration protocol (DHCP) as hosts dynamically join and leave the network. Network statistics only capture a certain snapshot of network status. Extracting device information including its type, ID, and IP address can provide identification out of anonymized traces to network measurements.

In this paper, we propose a scalable physical device profiling framework on the Internet. It is the first general-purpose framework designed to address several challenges arising from profiling devices: (i) scaling well to large scale (4 billion IP addresses), (ii) associating IDs with physical devices rather than simply labeling with an IP address assigned by DHCP, (iii) extracting physical device information such as device type, running services, and its IP. We leverage the existing approaches on network reconnaissance and clock skew (e.g., [6] [7]) and realize a system running in the cloud to accomplish this task. We first scan a large number of Internet space to discover live hosts, then adopt banner grabbing from application layer protocols to extract physical devices. If a physical device is identified, the clock skew separates device IDs with microscopic precision deviations of device hardware. A key design goal of our work is the scalability.

To verify our framework for profiling devices, we implemented a prototype system to extract Webcams information. We conducted experiments to test accuracy and precision for different stages of the framework. We test five application layers to extract Webcam type information and clock skew for Webcam device ID. Our results show that physical device type can be extracted from banner grabbing and devices of the same type can be separated by clock skew. We deployed the system on Amazon EC2 [8] to study how it performs at Internet scale. We scanned the entire Internet space and discovered 1.2 million Webcams and 60 thousand industrial control device and extracted their profiles in less than 20 hours.

Specifically, the contributions of this paper are as follows.

- We have proposed a scalable physical device profiling framework. It is the first work to extract detailed device information(ID, type, running services) at a large scale.
- We have implemented a prototype system to extract

† Qiang Li is corresponding author.

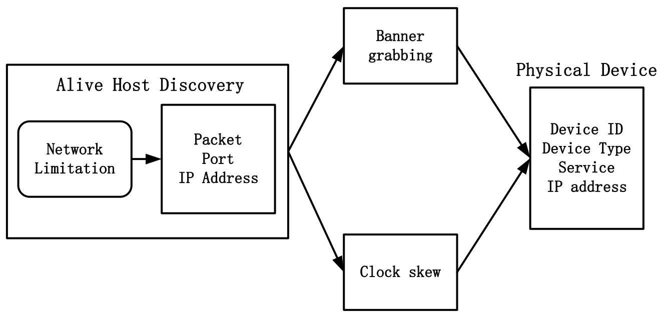


Fig. 1: The framework for extracting physical device information.

Webcams. Our experimental results show that Webcam information can be effectively collected in real time.

- We have deployed physical device profiling system on Amazon EC2. We scanned the entire Internet space to discover physical devices including 1.2 million Webcams and 60 thousand industrial control devices and extracted their profiles in less than 20 hours.

The rest of the paper is structured as follows. We describe the related work in Section V. We then propose our framework and design details in Section II. In Section III, the real world experiments are conducted and empirical results demonstrate the performance of our approach. Finally, we make final remarks in Section IV and Section VI.

II. A Scalable Framework for Devices Information Extraction

A. Live host discovery

In this section, we propose a scalable framework for physical device information extraction. We first present the details on how to get device ID, type, running services, and its IP address. We then discuss a heuristic algorithm for collect physical device information in a large-scale.

There are many physical devices visible on the Internet, such as Webcam, net-printer and industrial control devices. These devices differ from each other in many attributes and characteristics. In order to identify them we propose a device profile extraction framework as shown in Fig. 1. It consists four parts: live host discovery, banner grabbing, clock skew, and physical device information extraction. The input to the live host discovery module is a long list of IP addresses and the output is a list of IP addresses whose host is active or turned on. Its main function is filtering out unqualified IP address and keep qualified ones for the next module. The banner grabbing module uses the application layer interaction between the detector (our system) and candidates to extract physical device information. The clock skew module associates IDs with devices using clock skews of these devices. Physical device information is stored as the device's profile. As shown in Table I, we record information for each device as a record in a table. A device info tuple consists of $(ID, Type, Service, IP, Time)$. Device IP address might change over time and services may be turned off or on, but its device ID and type remain stable. We use a timestamp to label the snapshot of the device at that time. A key design goal of our framework is to be able to remotely extract device information in a scalable manner by only sending active packets to an IP address.

TABLE I: Physical device information.

Device ID	*Device Type*	*Services*	*IP address*	*Timestamp*
(α_1, β_1)	Webcam	HTTP, RTSP	201.22. $*$.$*$	2015-6-3
(α_2, β_2)	Net-printer	HTTP	201.22. $*$.$*$	2015-6-3

Discovering live hosts on the Internet is the first step by filtering out unqualified hosts from a huge number of IP addresses before extracting information of physical devices. Determining whether an IP address is associated with a live host is straightforward. We send a packet to an address, if we receive a response, then it is an live host, otherwise no host is present. While the discovery methodology seems simple, three design challenges arise in practice.

Network condition limitations. Due to network limitations, we can not send packets to IP addresses at an unlimited speed. Packets may be dropped during live host discovery. For instance, a busy router in the middle of a network path may drop most probing packets. There are two network limitations, local network bandwidth and the router limitations. For the first limitation, keeping the transmission rate under local network bandwidth can avoid packet loss. For the second one, a randomized algorithm can be used so that probing packets are sent to a random IP address each time rather than in sequential order. This can avoid congested paths at a certain time. Previous work claims that [9] an Internet-wide host detection within 45 minutes cannot be achieved in practice.

Port Selection. Port selection is important to identify whether a host is alive or down. There are 65536 ports to be tested. Detecting all ports is unnecessary, impractical, and time consuming. We adopt two improvements for port selections. First, we rank ports and only open popular ones with a high probability [10]. Second, we observe that application layer services usually run in particular ports, such as HTTP on 80, RTSP 554.

Packet Selection. Probing packets are sent to an IP address with a port and a response is used for extracting physical devices. During live host discovery, we send packets to determine the existence of a host. To speed up the process, we send a packet in the transport layer without storing every state for building TCP connections. During banner grabbing, we need to send application layer packets to extract information of physical devices. It requires building a complete TCP connection before sending an application layer request. During clock skew calculation, we send a sequence of TCP packets with timestamps to a device.

B. Banner grabbing

Banner grabbing is an enumeration technique used to glean information about a computer system on a network and the services running on its open ports. A simple banner grabber

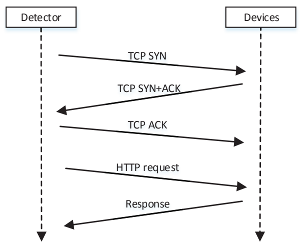
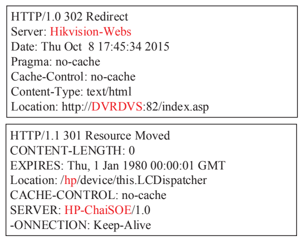


Fig. 2: The interactions involved in getting a HTTP protocol header for banner grabbing.



```
HTTP/1.0 302 Redirect
Server: Hikvision-Webs
Date: Thu Oct  8 17:45:34 2015
Pragma: no-cache
Cache-Control: no-cache
Content-Type: text/html
Location: http://DVRDVS:82/index.asp
```

```
HTTP/1.1 301 Resource Moved
CONTENT-LENGTH: 0
EXPIRES: Thu, 1 Jan 1980 00:00:01 GMT
Location: /hp/device/this.LCDispatcher
CACHE-CONTROL: no-cache
SERVER: HP-ChaiSOE/1.0
-ONNECTION: Keep-Alive
```

Fig. 3: The banner grabbing for application protocol header.

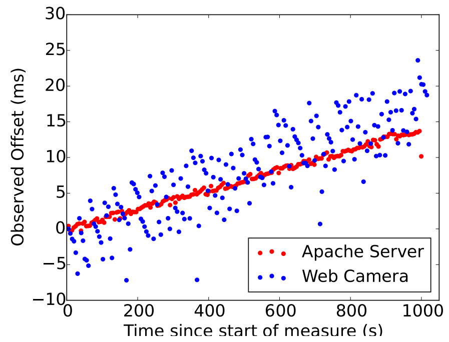


Fig. 4: The clock skews of two devices.

connects to an open TCP port and prints out anything sent by the listening service within several seconds. After getting the live host from a large number IP address, we use banner grabbing to extract two types of information: services running on the device and device type.

The first type of information, the running services, is simply collected from application layer protocols. For instance, If we send a HTTP request to a live host and receive a response, the host is indeed running the HTTP service. The method is simple, but the question is what application protocol to use. There are many application protocols and using every one is impossible. Table I shows two instances of physical devices running HTTP and RTSP services. We rank the popularity of application layer protocols to identify whether this host is running the service or not, according to existing rank list [10]. We describe application protocol selection combined with their ports in the next section II-D.

The second type of information, the device type, can be extracted from response headers of application layer protocols. Table I shows two types of physical devices: Webcams and Net-printers. To extract device type, the detector first builds a three-way handshake based on TCP connections with the device, then sends a request of application layer protocol to a device. Fig. 2 shows the interaction between the detector and the device in order to get an HTTP response. After the detector receives the response, it extracts device type information from the option field of the response packet. Fig. 3 shows two HTTP responses from two live hosts. We found the “Hikvision-Webs” and “ HP-ChaiSOE” labels from the “server” field (in red). “Hikvision-Webs” is a typical Webcam device as the company name is “Hikvision”, and “ HP-ChaiSOE” is a typical Net-printer as the company name is “Hewlett Packard”. We store these strings into a pre-defined file and extract the information from the response header. If a matching is found via regular expression, we consider this live host as this device type.

C. Clock skew calculation

There are several ways to label a device. IP address usually represent a device, but it is easily affected by DHCP with the dynamic joining and leaving of devices. MAC address of a network card is unique, however, it is only available in the local network. High level information such as cookie and user login can describe a device, but it is only suitable for a few instances and can not be used for a large number of devices. In our work, we use clock skew to label a device.

Every device has its own system clock. To synchronize its clock, network time protocol [11] provides a method to calibrate a device’s system clock. During a short period of time, every device might still have slight clock skew. We identify physical devices based on disparate clock skews. The clock skew of a device can be calculated as follows. The detector’s system clock is recorded as S, the device’s system clock is recorded as D. We use S_i and D_i as the interval time between ith timestamp and the initial timestamp, then

$$S_i = T_i^S - T_1^S$$

$$D_i = T_i^D - T_1^D$$

where T is the timestamp of system clock. We get the detector’s system clock by accessing system function and get the device’ clock by extracting the optional field timestamp in TCP packets. We send a sequence of TCP packets to a device, and receive their responses with the timestamps. The set $(S_i, D_i), i \in \{1, ..., N\}$ represents the clock skews between the detector and the device. Fig. 4 shows clock skews of a Webcam and an Apache server. To intuitively represent clock skews, we use S_i as the value of X-axis and the difference $(D_i - S_i)$ as the value of Y-axis. We see that the clock skew of Webcam is different from the clock skew of Apache server.

To approximately represent these sequential values $(S_i, D_i), i \in \{1, ..., N\}$, we adopt the polynomial regression to fit S_i to D_i, as follows:

$$D_i' = b_0 + b_1 * S_i + b_2 * S_i^2 + ... + b_m * S_i^m$$

where the cost function is

$$\frac{1}{2 * N} \Sigma_{i=1}^{N} (D_i' - D_i)^2$$

We adopt gradient descent [12] to minimize the cost function. As shown in Table I, we use two parameters (α, β) to describe the clock skews of Webcam and Net-printer.

However, directly using clock skews to label devices is not practical. When the number of devices is large, there

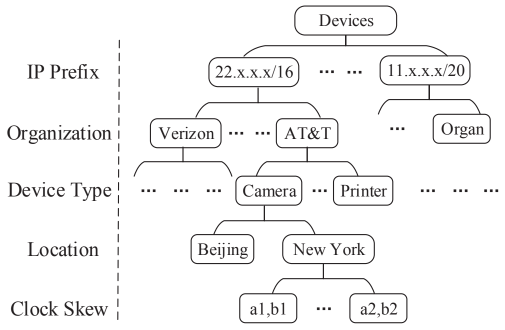


Fig. 5: A hierarchical structure for device labeling.

```
1:  **Input**: a list of IP address $list$ $\{1,...,N\}$;
2:  **Output**: physical device records $Tab$ as shown in Table I;
3:  randomly arrange $list = random(list)$;
4:  rank application protocols $p_1,...,p_k$
5:  for (each IP in $list$) do Alive detection
6:      if (get a response) then live host is stored in $list'$
7:      end if
8:  end for
9:  for (each live host in $list'$) do
10:     for (try $p_i$) do extract the banner from header;
11:         if (type is checked) then record type, IP, service $i$;
12:         test and record its clock skew
13:         end if
14:     end for
15: end for
```

Fig. 6: A heuristic Algorithm for extracting physical devices

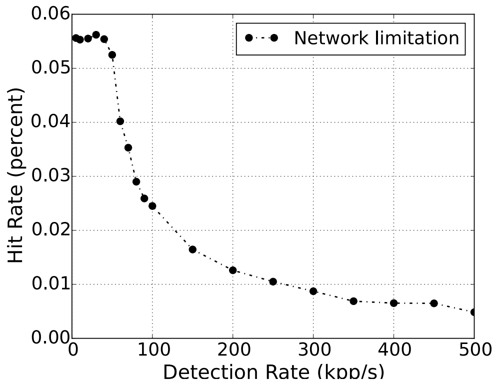


Fig. 7: Network condition affects the hit rate of live host discovery.

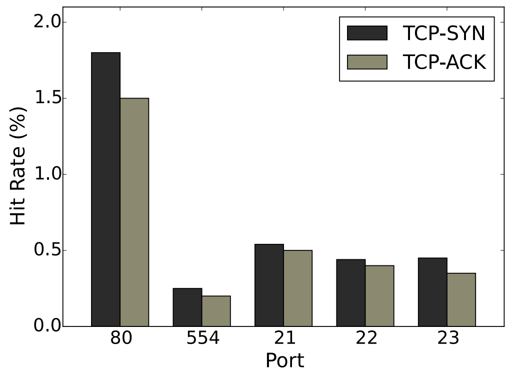


Fig. 8: Hit rate of packets and ports.

might be some devices whose clock skews overlap with others. To address this issue, we propose a hierarchical structure to calibrate clock skews. The intuition is that we can extract lots of information from IP address [13], [14], such as network prefix, organization, location, and device type, from banner grabbing. These attributes are stable but at the coarse-grained level. We use these attributes to cluster devices into different groups. Devices that have the same attributes are clustered into the same groups. As shown in Fig. 5, the hierarchical structure is organized by five attributes. In the lowest level, we use the clock skew to label devices. To normalize clock skews, we add offsets to the parameters to represent different groups. For instance, if device one (α_1, β_1) and device two (α_2, β_2) are in different groups, we add the offset to their parameters as $(\alpha_i + d_i^{\alpha}, \beta_i + d_i^{\beta})$.

D. A Heuristic Algorithm

We propose a heuristic algorithm for extracting information of physical devices via techniques mentioned in the sections, as shown in Fig. 6. Line 3, in the initial stage, we randomly re-arrange IP list to form new list as uniform permutation [15]. We also rank the application protocols to extract physical device type. From line 5, we begin to discover live hosts from the IP lists. This process filters out unqualified IP addresses. Because we send stateless packets to IP addresses, we can discover hosts as soon as possible under the constraint of local network bandwidth. At the next stage (line 9), we check each live host to determine wether it is a physical device or not. We test every application protocol, if there is any string matching with pre-stored device type files, we store its type, services and IP. This process determines the existence of a physical device.

Then we record their clock skews as shown line 12. This clock skew recording is time consuming. We can do it either online or offline. The information (Fig. 5) about network prefix, organization, location, and device type can be added offline. At last the algorithm outputs the table as the information of physical devices in the form of $(ID, Type, Service, IP, Time)$.

III. Performance Evaluation

To verify our framework, we use webcam as an instance of physical devices. A webcam is a type of digital video camera commonly deployed for surveillance. It can send and receive data via a computer network. They are visible in public Internet and remotely accessible. We first implement a prototype system to extract information of Webcam on the Internet. We then test banner grabbing for extracting service and type of the Webcam. We test and verify the effectiveness

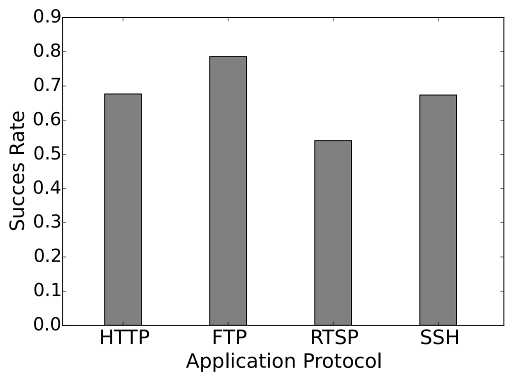


Fig. 9: Success rate of extracting application layer protocols

TABLE II: Time needed for banner grabbing.

	Telnet	*HTTP protocol*	*RTSP*	*SSH*
TCP connection	0.09258s	0.01279s	0.19660s	0.08698s
Header	0.11587s	0.01819s	0.20203s	0.11551s

of using clock skews to label Webcam devices. Finally, we extract Webcam device information at Internet scale (nearly 4 billion IP addresses).

A. Implementation

We have implemented the heuristic algorithm (Fig. 6) to extract information of Webcams on the Internet. For live host discovery, we send a TCP packet with "SYN" field option. If a host responds with a TCP packet with "SYN-ACK" field option, we set it as a live host, otherwise inactive. We use the network scanner tool Zmap [9] randomizize the IP address list. We rank the ports (port 80, 554, 81, 85,21 and 22) based on popularity for webcam. We also rank application layer protocol for banner grabbing as HTTP, RTSP, FTP and SSH. These protocols have a high probability that their response header contains a camera type information. After identifying a Webcam device, we send a sequence of packets to this IP address with the timestamp optional field. We label a Webcam device using two parameters (α, β) and the hierarchical structure uses the network prefix to cluster different device groups. We deployed our heuristic algorithm on Cloud computing server, Amazon EC2 [8]. It is Ubuntu 14.04.2 LTS (GNU/Linux 3.13.0-48-generic x86_64) with 2 vCPU and 8 GB of memory, 450Mbps bandwidth. After probing the entire IP space, we fetch the date set from the server and extract device profile in the form of $(ID, Type, Service, IP, Time)$.

B. Performance of live host discovery

Live host discovery filters out unqualified IP addresses and selects candidates for the next stage. We have conducted two experiments for live host discovery, one is the impact of network limitation on discovery rate and the other is hit rate for packet and port selection. Fig. 7 shows how the network bandwidth affects the live host discovery rate. The unit of X-axis is packets per second. The Y-axis is the hit rate and equal to $N_{candidates}/N_{total}$, where N_{total} is the total number of IP addresses and $N_{candidates}$ is the number of responding hosts. When the detection rate is 50,000 packets per second, a stable hit rate is achieved for live host discovery. When the detection rate increases, the time needed to finish the discovery slightly decreases smaller but the hit rate drops very fast. For live host discovery, we need a high and stable hit rate. Although Zmap [9] claims that it can finish the entire IP space detection under 45 minutes, the time is only a theoretical bound and it only includes host discovery. A practical issue is that network congestion reduces the hit rate. We decided to use the most stable detection rate while keeping the hit rate high.

We next study the hit rate for packet and port selection during live host discovery. Fig. 8 shows the hit rate for different ports and packets. Two type of packets, TCP packets with "SYN" field and TCP packets with "ACK" field, are used to discover live hosts. We can see that TCP packets with "SYN" field have a slightly better performance. In our prototype system, we adopt the TCP packets with the "SYN" field. The results also show different hit rates under different ports. Port 80 have the highest hit rate, nearly 1.9%. It suggests that ranking ports based on their popularity is important. We also tested ICMP packets to discover live hosts. As ICMP does not require a port, its hit rate is the highest, close to 9%. However, we do not use ICMP because application layer protocols usually use particular ports (e.g., HTTP uses Port 80) but ICMP is not associated with a particular port.

C. Performance of banner grabbing

Banner grabbing extracts services and determine whether a live host is a Webcam or not. We have evaluated the success rate and latency involved in extracting information out of a response of application layer protocols.

Figure 9 shows the success rate of extraction. Success rate is equal to $N_{application}/N_{candidates}$, where $N_{candidates}$ is the number of live hosts and $N_{application}$ is the number of hosts returning the response of application layer protocol. We can extract information from the response header. We see that different application protocols have different success rates. Once a response is returned, we identify this host is running this service and record it. We then extract strings from the response header to match the Webcam device type. Due to the lack of ground truth for remote hosts on the Internet, we only perform matching in controlled environments. If we extract the string to match with Webcam type that is pre-stored in a file, the precision is 100%. However, we can not provide the false negative rate for matching.

Table II is the latency in getting a response within application layer protocols. This latency includes the time needed for the three-way handshake TCP connection requires. Combining Fig. 9 and Table II, we can determine the running services and Webcam device type.

D. Performance of clock skew

Fig. 10 shows the time-varying clock skews of two Webcam devices. When the time is short (less than 1000s), the device's

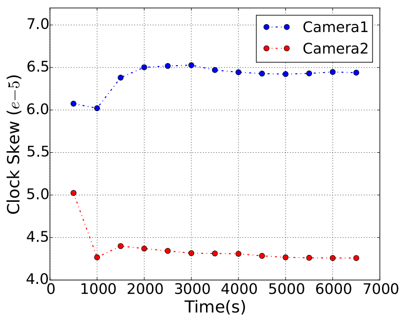


Fig. 10: Clock skews of two cameras over time.

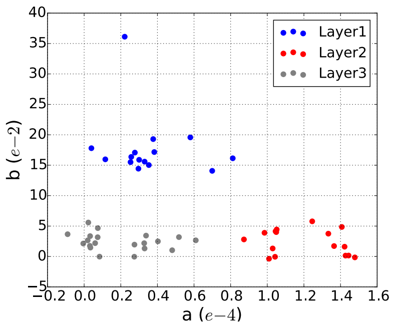


Fig. 11: Clock skews of 50 devices

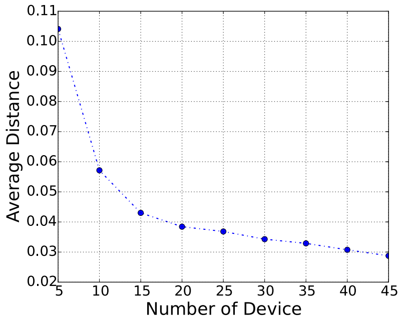


Fig. 12: Average distance of devices

clock skew has a high fluctuation. Due to the inherent noise between the detector and the device. Over time, the device's clock skew stabilizes. Worth to point out is that noticeable latency is needed to get an available clock skew. We also see that two Webcam devices have different clock skews that can be used to differentiate them.

Furthermore, we investigate whether the large number of devices affects the diversity of clock skews. Fig. 11 shows clock skews of 50 Webcam devices. The camera is represented as (α, β), where they are fitting parameters between clock skews. X-axis degree is the α magnitude and Y-axis is the β magnitude. Each point is a Webcam device. We use the network prefix (shown in Fig. 5) as the attribute to group these devices. We see that clock skews of these devices are different enough to be used as labels.

We use the Euclid distance to represent the difference between Webcam devices. As shown in Table I, the two devices' distance is calculated as $d_{12} = \sqrt{(\alpha_1 - \alpha_2)^2 + (\beta_1 - \beta_2)^2}$. The distance shows the difference between two devices. We use the average distance to measure the overall diversity of devices' clock skews. Fig. 12 shows the average distance changes with the number of Webcam devices. When the number is small, devices' clock skews are distinct. When the number becomes larger, the distance decreases. Here, we do not use other attributes to group devices, but we add the offset to clock skews. Our hierarchical structure for device labeling would alleviate the situation.

E. Real world experiment

To further verify our approach, we deployed the system on the cloud server, Amazon EC2. We conducted the experiments to record Webcam device information on September 20, 2015 and the details are shown in Table V. We have exhaustively searched the entire IP address space (close to 3.7 billion addresses). We excluded both reserved/unallocated IP space from IANA [16]. We also excluded IPs that send some emails to complain about our scanning activities during our collection experiments. Altogether, we have about 610 million IP address in our blacklist to be excluded. In order to profile webcam, we have tested five normal application layer protocol usually running on webcam , including HTTP, RTSP, FTP ,Telnet, and SSH. We have found 1.2 million Webcams from 3.7 billion IP addresses in 20 hours. Note that in this experiment, we did not record Webcam device IDs using clock skews. The reason is that getting a clock skew from a device takes more than 2,000s. Therefore, labeling 1.2 million devices will require several years.

TABLE III: Six protocols ICS device counts in Internet-wide range

Protocol	IP Space	Count
Modbus	3.7 billion	21131
EtherNet/IP (ENIP)	3.7 billion	3704
Siemens S7	3.7 billion	3602
Tridium Niagara Fox	3.7 billion	21038
BACnet	3.7 billion	13781
Red Lion	3.7 billion	1604

There are some kinds of physical device apart from webcam and the most crucial device is industrial control device. These devices use supervisory control and data acquisition (SCADA) to control remote equipments with coded signals, and these equipments are typically computer-based systems with access to the Internet used to monitor and control industrial processes. There are dozens of protocols running on SCADA-enable devices, including Siemens S7, Modbus, IEC 60870-5-104, DNP3, EtherNet/IP, BACnet, Tridium Niagara Fox, Crimson V3, OMRON FINS, PCWorx, ProConOs and MELSEC-Q.

Like wecam, we did not record industrial device IDs using clock skews for time-consuming and security concern. And the detailed reason is that industrial control device have a less dynamic IP address and we can extract more useful information to profile it. For example, we extract device type from Modbus protocol. Table IV shows ICS device type and vendors with Modbus. A problem is we cant extract many device type and vendor. Only a little proportion of ICS devices have their field information. Many ICS devices just deny to give their detail information. Unknown is most commonly seen among ICS devices. One security problem is these ICS device with exposing themselves information may have security issues.

Due to the characteristic of industrial control device, we have implemented six protocols running industrial control system devices and extract some detailed informationm to profile it as following illustrating:

- Modbus: it has since become a de facto standard communication protocol and a commonly available means of connecting ICS devices. We can get their "Vendor Name", "Network Module", "CPU Module", "Firmware", "Memory Card" and "Project Information". We can infer their usage functions, e.g. water station gateway. All of these give us the opportunity to profile it.
- EtherNet/IP (ENIP): it is similar as Modbus. It is an industrial Ethernet network combining standard Ethernet technologies with the media-independent Common Industrial Protocol. We can get their "Vendor", "Product Name", "Serial Number" "Device Type", "Product Code", "Revision", Device IP. We also can infer their function based on these information. Much information is also vacant. The detail information of ENIP is different from Modbus.
- Siemens S7: differing from Modbus and ENIP, it is a particular ICS protocol for automation systems similar as Simatic corporation. We can get their "Module", "System Name", "Version", "Plant Identification". It is not necessary to infer their usage functions because S7 usually is used as automation systems. Furthermore, there are more probing packets need to extract the information. In Modbus and ENIP, we just need one packet enough to extract the information.
- Tridium Niagara Fox (Fox): it is used for building automation solutions rather than standard protocol Modbus or ENIP. Its function focuses on the building automation control. Much information (e.g. "Version", "Application")can be got by our approach, where several ones are vacant.
- BACnet: it is most commonly used in industrial automation control systems, such as for water processing plants, manufacturing facilities and utilities. We also can get their information and infer their functions.
- Red Lion: it acts similarly as BACnet.

As shown in Table VI, we also conducted some experiments to record industrial devices from August, 2015 to November, 2015. We have collected above six ICS protocols in the whole IPv4 space. Table V shows the total ICS devices count in the 3.7 billion address space we discovered. Their total count is nearly 64860.

In brief, we are trying to discover industrial devices at internet scale and to profile it.

IV. Discussion

Our work can be improved in three different areas.

Firewall and VPN. For remote access, physical devices should be access-controlled by a firewall or a VPN. However, we have collected information devices running special protocols at Internet scale some tomes and found tens of thousands of devices being exposed to the public Internet. No matter what security protections these protocols have, these Internet-connected devices are at risk of security attacks. In fact, some industrial control protocol lacks of security protections. To approximate a "bird's eye view" of these devices at Internet scale, we can not estimate the total number and distribution of them. This is because that many devices are in the local network behind firewall or VPN. Although we can recognize NAT or firewall and divert their interference over industrial control systems. However, the problem is our NAT detection relies on-shelf software or third tools. They take a long time to find NAT or firewall and it is impractical to straight use

TABLE IV: Types and vendors of devices running Modbus in Internet-wide range

Device type	Conut		Device vendors	Count
light controller	32		Computer OY	90
network ananlyer	6		Entes	9
power controller	13		Lantronix	6
power monitor	1		Nivus	436
programmable logic controller	861		Schneider Elevtric	1866
scada controller	212		Solar Log	620
solar panel	620		Telemecanique	557
water flow controller	436		Veris Industries	1
unknown	17546		unknown	17546

TABLE V: Internet-scale physical devices discovery

IP Space billion	Application Protocol	Ports	Webcam Devices (million)	Time (hours)
3.7	HTTP, RTSP FTP, Telnet, SSH	80, 81,85, 21 554 , 22, 23	1.2	20

TABLE VI: Physical devices discovery at Internet scale

Begin Time	IP Space	Application Protocol	Ports
2015-08-31	3.7 billion	Modbus,Siemens S7	502,102
2015-09-02	3.7 billion	Modbus,Siemens S7	502,102
2015-09-05	3.7 billion	Modbus,Siemens S7	502,102
2015-10-23	3.7 billion	Modbus,Siemens S7 EtherNet/IP,BACnet, Tridium Niagara Fox Crimson Red Lion	502,102, 44818,47808, 1911,789
2015-10-25	3.7 billion	Modbus,Siemens S7 Tridium Niagara Fox	502,102,1911
2015-10-28	3.7 billion	Modbus	502
2015-10-29	3.7 billion	Modbus	502
2015-10-30	3.7 billion	Tridium Niagara Fox	1911
2015-11-2	3.7 billion	BACnet, Crimson Red Lion	47808, 789
2015-11-30	3.7 billion	Modbus	502
2015-12-02	3.7 billion	BACnet, Modbus, EtherNet/IP, Tridium Niagara Fox	502,102, 44818,47808, 1911,789

it at internet scale. Passive listening [6] on the outgoing data traffic might be a good way to estimate the number of these devices, such as a listener at an Internet service provider.

Banner Grabbing. We get device type by probing application-layer information and banner grabbing. But this kind of banner can modify by the owner, we could improve its precision and recall via multiple layers combination probing. However, more probing means the more disturbing remote physical device. These systems are sensitive to these probing and usually block us. Therefore, it become hard to improve our performance at the current situation.

Reduce latency. Our framework can use in the small scale and get the good performance, but it need too long time to get the clock skew at internet scale. In our experiment, getting a clock skew from a remote physical device takes more than 2,000s and get good result. Therefore, labeling 1.2 million devices will require several years use one machine. Although we can use multithreading or muti-host to accelerate this process, the best way is to find an new algorithm to reduce the latency in getting the clock skew of a device.

V. Related Work

There is a growing interest in both academia and industry for remote discovery of devices on the Internet. It is a primary step of remotely locating vulnerable systems. The classic on-shelf tool Nmap [10] discover lots of device information by sending complicated packets to every IP address. It is time consuming and only suitable for a small-scale space. In contrast, our work discovers as much device information as possible in large-scale network space. Large-scale device discovery efforts have been focusing on how to speed up the measurement in the large space [2] [9], [17]–[29]. Over time it does get progressively faster - from 4 months down to 30 days, one day, and more recently 45 minutes. Heidemann et.al [2] explored the visible Internet to characterize the edge hosts and evaluate their usages via active scanning in 30 days. Leonard et.al [17] implemented IRLscanner as an Internet-wide detection mechanism. It uses multiple detectors to speed up the measurement and can complete in 24 hours. Durumeric et.al [9], [20] proposed Zmap for Internet-wide host discovery that completes in 45 minutes. This time duration is a theoretical upper bound and cannot be achieved in practice due to limitations in network conditions. Zmap only sends one packet to every IP address but our approach aims to extract as much device information as possible. Xuan et.al [30] propose a two-stage discovery mechanism to identify industrial control device. It take a first step in this direction by investigating physical devices at internet scale. One fundamental difference between our work and previous work is our goal: whereas all previous work focuses on speeding up network exploration, our work aims to extract physical device information without sacrificing speed.

Another research aspect related to our work is fingerprinting devices. Well-known operating system fingerprinting tools include Nmap [10], xprobe2 [31] and p0f [32]. They use transport layer packets to remotely fingerprint the information of operating systems. In contrast, we focus on device profiles, including device ID, type and services. Hong et.al [33] searched the entire Internet and then classified IP addresses as popular or unpopular. Xie et.al [3] de-anonymized the Internet to minimize the influence of DHCP effect. It used hotmail login information to identify device IDs. Our work is different since we adopt the active scan to collect TCP timestamp and calculate the clock skew to separate devices rather than using application layer information. A network card's MAC address of device is supposed to be unique and can serve as device ID. However, it can not be extracted when there are more than two hops way, which is common on the Internet. Our work uses packet timestamp to calculate the clock skew, avoiding the influence of multi-hop effect. Previous work [34] used clock skew to label devices. However, it used active scan to collect packet timestamp and passive listen to collect packet timestamp. Stable timestamp can effectively minimize the noise of network path. Our work adopts active scan to collect packet timestamps and combine that with other attributes using a hierarchical structure to separate diverse devices. Moreover, we extract as much physical device information as possible while still keeping the time short.

VI. Conclusion

In this paper, we have proposed a scalable framework for extracting device information in the Internet space. We have used live host discovery to find out active hosts and used banner grabbing to extract device type and its services. We used clock skews to label a physical device's ID. The results show that Webcam profiles can be effectively extracted and identified in real time. We have implement a prototype system and run it on Amazon EC2 to extract information of Webcam devices and physical control devices from the entire IPV4 space. We deployed it on the cloud server and used it to detect 4 billion IP addresses to profile 1.2 million Webcam devices and 60 thousand physical control devices in 20 hours. In the future, we will further improve two aspects. One is banner grabbing. We will try to reduce its false negative rate. The other is clock skews. We will find a way to reduce the latency in getting the clock skew of a device.

VII. Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grant No. U1536107), National Defense Science and Technology Innovation Fund, CAS (No. CXJJ-16-M118) and "Strategic Priority Research Program" of the Chinese Academy of Sciences,Grant No. XDA06040101.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] J. Heidemann, Y. Pradkin, R. Govindan, C. Papadopoulos, G. Bartlett, and J. Bannister, "Census and survey of the visible internet," in *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*. ACM, 2008, pp. 169–182.

[3] Y. Xie, F. Yu, and M. Abadi, "De-anonymizing the internet using unreliable ids," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4. ACM, 2009, pp. 75–86.

[4] Y. Xie, F. Yu, K. Achan, E. Gillum, M. Goldszmidt, and T. Wobber, "How dynamic are ip addresses?" in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 301–312.

[5] M. Khadilkar, N. Feamster, M. Sanders, and R. Clark, "Usage-based dhcp lease time optimization," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. ACM, 2007, pp. 71–76.

[6] E. Bou-Harb, M. Debbabi, and C. Assi, "Cyber scanning: a comprehensive survey," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 3, pp. 1496–1519, 2013.

[7] M. Allman, V. Paxson, and J. Terrell, "A brief history of scanning," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. ACM, 2007, pp. 77–82.

[8] Amazon elastic compute cloud (amazon ec2). [Online]. Available: https://aws.amazon.com/ec2/

[9] Z. Durumeric, E. Wustrow, and J. A. Halderman, "Zmap: Fast internet-wide scanning and its security applications." in *Usenix Security*, 2013, pp. 605–620.

[10] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, 2009.

[11] D. Mills, "Network time protocol (version 3) specification, implementation and analysis," 1992.

[12] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[13] M. J. Freedman, M. Vutukuru, N. Feamster, and H. Balakrishnan, "Geographic locality of ip prefixes," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. USENIX Association, 2005, pp. 13–13.

[14] Maxmind geoip2. [Online]. Available: https://www.maxmind.com/en/geoip2-services-and-databases

[15] S. Staniford, V. Paxson, N. Weaver *et al.*, "How to own the internet in your spare time." in *USENIX Security Symposium*, 2002, pp. 149–167.

[16] Assigned numbers authority (iana). [Online]. Available: http://www.iana.org/

[17] D. Leonard and D. Loguinov, "Demystifying service discovery: implementing an internet-wide scanner," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 109–122.

[18] Q. Li, Q. Han, and L. Sun, "Collaborative recognition of queuing behavior on mobile phones," *Mobile Computing, IEEE Transactions on*, vol. 15, no. 1, pp. 60–73, 2016.

[19] Q. Li, Q. Han, X. Cheng, and L. Sun, "Queuesense: Collaborative recognition of queuing on mobile phones," in *Sensing, Communication, and Networking (SECON), 2014 Eleventh Annual IEEE International Conference on*. IEEE, 2014, pp. 230–238.

[20] D. Adrian, Z. Durumeric, G. Singh, and J. A. Halderman, "Zippier zmap: internet-wide scanning at 10 gbps," in *Proceedings of the 8th USENIX Workshop on Offensive Technologies*, 2014.

[21] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A search engine backed by internet-wide scanning," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 542–553.

[22] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the https certificate ecosystem," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC '13. New York, NY, USA: ACM, 2013, pp. 291–304. [Online]. Available: http://doi.acm.org/10.1145/2504730.2504755

[23] Q. Li, Q. Han, and L. Sun, "Context-aware handoff on smartphones," in *Mobile Ad-Hoc and Sensor Systems (MASS), 2013 IEEE 10th International Conference on*. IEEE, 2013, pp. 470–478.

[24] D. Leonard, Z. Yao, X. Wang, and D. Loguinov, "Stochastic analysis of horizontal ip scanning," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 2077–2085.

[25] A. Dainotti, A. King, k. Claffy, F. Papale, and A. Pescapè, "Analysis of a "/0" stealth scan from a botnet," in *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, ser. IMC '12. New York, NY, USA: ACM, 2012, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/2398776.2398778

[26] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, and V. Paxson, "The matter of heartbleed," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, ser. IMC '14. New York, NY, USA: ACM, 2014, pp. 475–488. [Online]. Available: http://doi.acm.org/10.1145/2663716.2663755

[27] X. Cai and J. Heidemann, "Understanding block-level address usage in the visible internet," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 99–110. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851196

[28] X. Cai and J. S. Heidemann, "Understanding block-level address usage in the visible internet." in *SIGCOMM*, S. Kalyanaraman, V. N. Padmanabhan, K. K. Ramakrishnan, R. Shorey, and G. M. Voelker, Eds. ACM, 2010, pp. 99–110. [Online]. Available: http://dblp.uni-trier.de/db/conf/sigcomm/sigcomm2010.html#CaiH10

[29] J. Czyz, M. Allman, J. Zhang, S. Iekel-Johnson, E. Osterweil, and M. Bailey, "Measuring ipv6 adoption," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 87–98, Aug. 2014. [Online]. Available: http://doi.acm.org/10.1145/2740070.2626295

[30] X. Feng, Q. Li, Q. Han, H. Zhu, Y. Liu, and L. Sun, "Identification of visible industrial control devices at internet scale," in *IEEE International Conference on Communications (ICC)*, 2016.

[31] O. Arkin, F. Yarochkin, and M. Kydyraliev, "The present and future of xprobe2," *The Next Generation of Active Operation System Fingerprinting*, vol. 27, no. 2, 2003.

[32] M. Zalewski, "p0f: Passive os fingerprinting tool," 2006.

[33] C.-Y. Hong, F. Yu, and Y. Xie, "Populated ip addresses: classification and applications," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 329–340.

[34] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," *Dependable and Secure Computing, IEEE Transactions on*, vol. 2, no. 2, pp. 93–108, 2005.