IMap: Toward a Fast, Scalable and Reconfigurable In-Network Scanner With Programmable Switches

Menghao Zhang, *Member, IEEE*, Guanyu Li, Cheng Guo, Han Bao, Mingwei Xu, Hongxin Hu, *Member, IEEE*, and Fenghua Li

***Abstract*— Network scanning has been a standard measurement technique to understand a network's security situations, e.g., revealing security vulnerabilities, monitoring service deployments. However, probing a large-scale scanning space with existing network scanners is both difficult and slow, since they are all implemented on commodity servers and deployed at the network edge. To address this, we introduce IMap, a fast, scalable and reconfigurable in-network scanner based on programmable switches. In designing IMap, we overcome key restrictions posed by computation models and memory resources of programmable switches, and devise numerous techniques and optimizations, including an address-random and rate-adaptive probe packet generation mechanism, and a correct and efficient response packet processing scheme, to turn a switch into a practical runtime-reconfigurable high-speed network scanner. We implement an open-source prototype of IMap, and evaluate it with extensive testbed experiments and real-world deployments in our campus network. Evaluation results show that even with one switch port enabled, IMap can survey all ports of our campus network (i.e., a total of up to 25 billion scanning space) in 8 minutes. This demonstrates a nearly 4 times faster scanning speed and 1.5 times higher scanning accuracy than the state of the art, which shows that IMap has great potentials to be the next-generation terabit network scanner with all switch ports enabled. Besides, our experiments also show that IMap supports the reconfiguration of scanning tasks at runtime, without incurring switch downtime. Leveraging IMap, we also discover several potential security threats in our campus network, and report them to our network administrators responsibly.**

***Index Terms*— Network scanner, programmable switch, single-packet probes.**

Manuscript received 4 February 2023; revised 18 July 2023 and 16 September 2023; accepted 18 October 2023. Date of publication 25 October 2023; date of current version 22 November 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62221003, in part by the China Post-Doctoral Science Foundation under Grant 2022M720202, and in part by the Beijing Post-Doctoral Research Foundation under Grant 2022-ZZ-078. An earlier version of this paper was presented in part at the NSDI 2022. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Chunyi Peng. *(Corresponding authors: Guanyu Li; Mingwei Xu.)*

Menghao Zhang is with the School of Software, Beihang University, Beijing 100191, China (e-mail: zhangmenghao0503@gmail.com).

Guanyu Li, Cheng Guo, Han Bao, and Fenghua Li are with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China (e-mail: dracula.guanyu.li@gmail.com; guoc21@mails.tsinghua.edu.cn; bh22@mails.tsinghua.edu.cn; lifh@cernet.edu.cn).

Mingwei Xu is with the Department of Computer Science and Technology and the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China, and also with the Zhongguancun Laboratary, Beijing 100194, China (e-mail: xumw@tsinghua.edu.cn).

Hongxin Hu is with the Department of Computer Science and Engineering, University at Buffalo, SUNY, Buffalo, NY 14260 USA (e-mail: hongxinh@buffalo.edu).

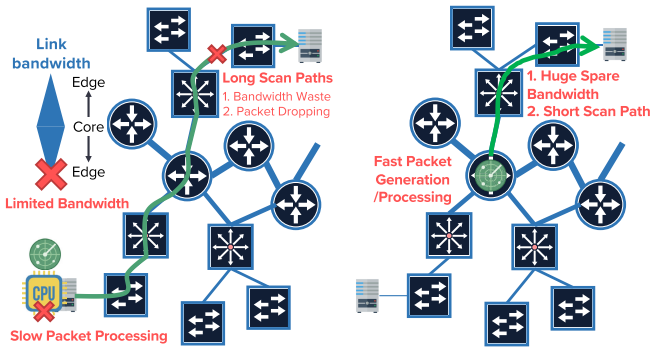
Digital Object Identifier 10.1109/TIFS.2023.3327665

I. Introduction

NETWORK scanning is a typical procedure to discover active hosts, ports and services in a network, which is mainly used by network operators/researchers for security assessment and system maintenance of the network. Enabled by tools such as Nmap [2], ZMap [3] and Masscan [4], network scanning has become a standard measurement technique to understand host behaviors in the target network, even the entire Internet. Recent studies have demonstrated that network scanning can help reveal new security vulnerabilities [5], [6], [7], monitor service deployments [8], [9], [10], [11] and shed light on previously opaque distributed systems [12], which are essential for people to understand the network's security situations.

Today's network scanners, however, cannot keep pace with today's soaring scanning space and provide a timely security snapshot. Recently IPv6 has progressed to the stage of large-scale deployment, and reports show that IPv6 is used by 21.5% of all the websites [13]. Along with the adoption of 5G networks, more and more Internet-of-Things (IoT) devices and mobile devices are connecting online [14]. The increased address space and the numerous online devices mean that the network scanner should be *scalable* to this much larger scanning space easily. Moreover, since these IoT and mobile devices go online and offline frequently, it is necessary for network scanners to conduct a comprehensive scanning quickly. Otherwise, a large number of security snapshots cannot be captured, potentially missing numerous security incidents [15]. This raises the requirement that the network scanner should complete a comprehensive scanning as *fast* as possible.

However, a closer look into today's network scanners shows that they are far from being fast and scalable, due to their implementation targets and deployment locations, as shown in Fig. 1(a). First, in terms of implementation targets, current network scanners are all implemented on commodity servers. As CPUs on servers are not specialized for high-speed packet processing, the scanning speed of these CPU-based network scanners is intrinsically limited. This makes it difficult to cover a large scanning space timely. Second, in terms of the deployment locations, state-of-the-art network scanners are all located at the network edge. Scanning from the edge is usually limited by the upstream bandwidth of the end host, which inevitably constrains the utmost scanning speed for network scanning tasks. Besides, the end-to-end scanning paths indicate more bandwidth waste for edge networks and larger possibilities of dropping probe/response packets.



(a) Existing network scanners are limited by their CPU implementation and host deployment locations.

(b) IMap is benefited from its ASICs implementation and in-network deployment location.

Fig. 1. Existing network scanners v.s. IMap.

In this paper, we propose IMap, a fast and scalable in-network scanner to address the aforementioned issues. The technology enabler for IMap is the emergence of programmable switches [16], which offer remarkable performance, unprecedented programmability, and unique deployment location, as shown in Fig. 1(b). Generally speaking, one single programmable switch could provide a packet processing capability as high as multiple Tbps, which is several orders of magnitude higher than highly-optimized servers. Besides, such switches support stateful packet processing with domain-specific languages (e.g., P4 [17]), which allows programmers to enforce user-defined packet processing logics in the switch pipeline directly. Moreover, switches (especially core switches) provide a unique vantage point for network scanning, which is no longer constrained by the upstream bandwidth of the end host or plagued by the bandwidth waste and probe/response packet drops of the end-to-end scanning paths. These unique characteristics of programmable switches are incredibly valuable for the next-generation fast and scalable network scanners.

Nevertheless, designing IMap is a non-trivial effort. First, as a switch-based network scanner, IMap should first function as a traditional switch correctly, which requires that IMap should support the reconfiguration of scanning tasks at runtime without incurring switch downtime and interrupting routing functionality. Second, as a high-speed network scanner, when sending probe packets, IMap must cover the scanning space completely, and also be aware of network conditions to avoid affecting the normal packet routing functionality. Besides, once response packets arrive, IMap should distinguish normal packets and response packets correctly, and also process the response packets efficiently to avoid saturating the storage server. However, switches only have constrained computational models and limited memory resources, which cannot satisfy these requirements easily.

To meet these requirements, IMap designs a set of techniques and optimizations, i.e., an address-random and rate-adaptive probe packet generation mechanism, and a correct and efficient response packet processing scheme, to turn a switch into a runtime-reconfigurable high-speed network scanner. We implement a prototype of IMap in an Intel Tofino switch [18], and make the source code publicly available [19]. Testbed experiments and real-world deployments show that even with one switch port enabled, IMap can survey all ports of our campus network (i.e., 6 Class B IP address blocks), a total of up to 25 billion scanning space, in 8 minutes, achieving a nearly 4 times faster scanning speed and 1.5 times higher scanning accuracy than state-of-the-art network scanners. Besides, from the perspective of functionality, IMap is able to support the reconfiguration of scanning tasks at runtime, without reloading a new P4 program and incurring switch downtime. IMap also discovers several potential security threats in our campus network. To the best of our knowledge, IMap is the first network scanner that can potentially reach multiple Tbps scanning speed, benefiting from its implementation targets and deployment locations. We hope IMap can serve as the foundation for next-generation terabit network scanners.

In summary, we make the following contributions in this paper:

- We analyze the limitations of current network scanners, and identify the opportunities brought by programmable switches (§II).
- We propose IMap, a fast, scalable and reconfigurable in-network scanner with programmable switches. IMap consists of a probe packet generation module to generate high-speed probe packets with random address and adaptive rate, and a response packet processing module to process response packets correctly and efficiently (§III, §IV).
- We implement an open-source prototype of IMap, and conduct extensive testbed experiments and real-world deployments to show advantages of IMap (§V, §VI).

Finally, we make some discussions in §VII, describe related works in §VIII, and conclude this paper in §IX.

II. Background, Motivation and Observation

A. Backgrounds and Limitations of Current Network Scanners

Network scanning is usually used to discover active hosts, ports and services in a network, which can help network operators/researchers understand the security situation of the target network, even the entire Internet. One typical network scanning paradigm is single-packet probing, including TCP SYN scans, ICMP echo request scans, and application-specific UDP scans, which is also the default scanning strategy for many existing network scanners such as Nmap [2], ZMap [3] and Masscan [4]. Taking TCP SYN scans as an example, a replied TCP SYN-ACK packet indicates the probe port is active, and a replied TCP RST packet means the port is closed. By sending a single probe packet and checking the corresponding response packet, the single-packet probing strategy can quickly locate the activeness of a specific service port, which can easily support follow-up actions like completing a protocol handshake on discovered hosts. Currently, this single-packet scanning strategy has been widely used in many cases, such as revealing new security vulnerabilities [5], [6], [7], monitoring service deployments [8], [9], [10], [11] and illuminating previously opaque distributed systems [12].

With the rapid growth of scanning spaces and security incidents recently, today's network scanners are falling behind

the times, especially in terms of scanning *scalability* and scanning *speed*. First, network scanners should be able to scale to large scanning spaces easily. Recently IPv6 has been in the stage of large-scale adoption, for instance, Google's statistics show that around 36.99% of its users access Google via IPv6 [20]. Since IPv6 has a much larger address space than IPv4, the scanning space increases drastically. Besides, along with the deployment of 5G networks, more and more IoT/mobile devices are connecting online [14]. All these require that network scanners should be able to cover a large scanning space easily. Second, network scanners should be fast enough to provide timely security snapshots. Today's networks become more and more dynamic, and IoT/mobile devices switch between online and offline frequently. Meanwhile, we have also witnessed that security incidents occur more and more frequently, and some of them occur in a very small time scale (e.g., from tens of seconds to several minutes). For example, according to Cybint's monthly newsletter, since COVID-19, the frequency of cybercrimes increases 300%, and hackers attempt to attack vulnerable home networks as people are working from home [15]. As a consequence, network scanners should be able to complete a comprehensive scanning as fast as possible. Otherwise, some security snapshots cannot be captured and important security incidents may be missed.

However, today's network scanners are intrinsically slow, which are far from being fast and scalable to satisfy the aforementioned new requirements. For example, even with Zippier ZMap [21], one of the most powerful network scanners today, the scanning capability only reaches a throughput of 10 Gbps and a rate of 14.2 Mpps [22]. The capability of today's network scanners is limited by two key factors fundamentally. First, in terms of implementation targets, current network scanners are all implemented on commodity servers. Packet processing on commodity servers is intrinsically slow, since CPUs are not specialized for high-speed packet processing. Even with software optimizations like DPDK [23], the throughput cannot reach more than 40 Gbps easily [24], [25], [26]. Second, in terms of deployment locations, today's network scanners are all located at the edge of the network. Scanning from the edge is not only limited by the upstream bandwidth of the end host, but also incurs longer scanning paths and non-negligible bandwidth waste because of end-to-end scanning paths. As a result, even if the scanners are capable of scanning at a higher rate (e.g., 40 Gbps), the scanning results (e.g., hit rate, active/inactive rate) may suffer from low accuracy because of undesirable probe/response packet drops on the end-to-end scanning paths (§VI-B). Not surprisingly, because of these fundamental limitations, since the publication of Zipper ZMap [21], the network scanning tools have not experienced any progress, and researchers have turned to improve the scanning accuracy with the help of various algorithmic techniques [27], [28], [29], [30], [31].

B. Opportunities by Programmable Switches

Programmable switches [16], [17] bring unprecedented opportunities to address the limitations of current network scanners.

1) High Packet Processing Capability: Switching ASICs are specialized for high-speed line-rate packet processing, which can provide several orders of magnitude higher throughput than highly-optimized servers [24]. Specifically, today's latest CPU-based network scanner, Zipper ZMap [21], could only provide a scanning rate of 14.2 Mpps and a scanning throughput of 10 Gbps. In contrast, switching ASICs can easily process a few billion packets per second, which shows great potentials to be a terabit network scanner. Other hardware alternatives, such as FPGA and NPU, cannot match the performance of switching ASICs [24], thus not promising for a high-speed network scanner.

2) Flexibility to Support Diverse Scanning Tasks: The most prominent characteristic of the new-generation switching ASICs is programmability. Such switching ASICs can be programmed with domain-specific languages like P4 [17], and also support stateful packet processing with user-defined logics. Besides, programs can run collaboratively between the data plane switching ASICs and the control plane switch CPUs, enabling advanced and flexible packet processing. Moreover, the packet processing logics at the switching ASICs can be reconfigured at runtime from the control plane, through interactive APIs such as match-action tables and registers. As a result, diverse scanning tasks can be implemented in advance and a specific scanning task can be configured at runtime, which would potentially be the foundation of next-generation high-speed network scanners.

3) Vantage Points to Conduct Network Scanning: Existing network scanners are all located at network edges and implemented in end hosts, where the utmost scanning rate is usually constrained by the bandwidth of the end hosts. Worse yet, scanning from the end host requires an end-to-end scanning path, which inevitably results in the waste of bandwidth resources and the degradation of scanning accuracy. In contrast, switches provide a unique vantage point for network scanning tasks. Core switches usually have huge spare bandwidths (i.e., more than 50% spare bandwidth [32]), which shows substantial potentials for network scanners to tap. Moreover, scanning from a core switch is no longer plagued by the bandwidth waste or the scanning accuracy degradation resulted from the end-to-end scanning path. This scanning vantage point is particularly valuable for high-speed network scanners.

III. IMap Overview

A. Deployment Scenario

Our scenario focuses on a network-centric deployment model, where the administrators of an ISP or a cloud network deploy IMap to understand their own network's security situations. IMap could also be used for Internet-wide scanning, but this should avoid causing any ethical concerns, as pointed out in ZMap [3]. Ideally, IMap should be built on a core switch, which provides both routing services and scanning functionality simultaneously. In other words, the IMap switch should first preserve the functionality of packet switching, and then behave as a high-speed network scanner when there is spare bandwidth (e.g., reports show that bandwidth occupation ratio for core switches is usually less than 50% [32]). Note that the deployment of programmable switches is not a new requirement; several ISPs/cloud networks have already replaced their legacy switches with programmable switches in

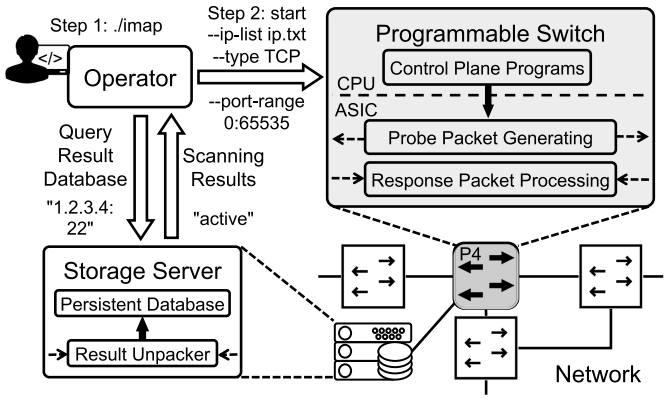


Fig. 2. The workflow of IMap.

their networks, which we believe is an irresistible trend in the foreseeable future [33], [34], [35]. Besides, an in-core-network scanner also raises the bar for attackers to take advantage of this powerful network scanner, as it is difficult for normal attackers to obtain such a deployment location. Currently IMap supports three types of single-packet probes, including TCP SYN scans, ICMP echo request scans and application-specific UDP scans, all of which can be specified and changed by operators at runtime.

B. Workflow and Design Requirements

IMap is designed to be a high-speed, easy-to-use network scanner based on core switches, so the usage of IMap is a bit different from traditional network scanners at the network edges, such as ZMap [3] and Masscan [4]. As shown in Figure 2, the workflow of IMap mainly includes two steps. First, operators should start the IMap switch (e.g., via command `./imap`) and make it serve as a traditional switch, which forwards the normal traffic accordingly. Second, at runtime, operators should specify a scanning task, including the scanning address space, the scanning type, and the scanning port range, via switch Command-Line Interface (CLI) (e.g., `start -ip-list ip.txt -type TCP -port-range 0:65536`). These configurations and parameters are parsed and issued into the IMap packet processing logic. During the scanning procedure, IMap data plane programs generate high-speed probe packets and process response packets accordingly, where the scanning results, i.e., the information extracted from the response packets, are written into a persistent database, such as a Redis in-memory data store [36]. After one scanning task finishes, operators can change and specify another scanning task as they desire. In the design, implementation and deployment of IMap, we identify several different design requirements that must be satisfied to make IMap a practical switch-based high-speed network scanner, especially in terms of probing packet generation and response packet processing:

1) Space-Complete and Rate-Adaptive Probe Packet Generation(§IV-A): In terms of probe packet generation, there are two key requirements in switch-based high-speed scanning. First, IMap should be able to cover the desired scanning space (i.e., |address space| × |port space|) completely, without duplications and omissions. This is a basic functional requirement for a network scanner. Second, packet switching is the first-class citizen of the switch, therefore, IMap should be able to conduct network scanning tasks without affecting normal network routing functionality. As the spare bandwidth of the network is dynamic, we need a network-aware method to generate high-speed probe packets with adaptive rate.

2) Correct and Efficient Response Packet Processing (§IV-B): With regard to response packet processing, we also have to fulfill two requirements. First, switches are also responsible for normal packet forwarding, therefore, the input packets for the switch-based scanner have both normal packets and response packets. As a result, the scanner should be able to distinguish normal packets and response packets correctly. Second, response packets cannot be steered to servers directly, as it may saturate the bandwidth of the storage servers and overwhelm the writing capability of the database. The scanner should have an efficient response packet processing approach to reduce the server-side pressure.

IV. IMap Design

To fulfill the requirements above, we design the high-speed in-network scanner IMap. Compared with our previous version of IMap [1], current IMap supports the reconfiguration of scanning tasks at runtime, without incurring switch downtime. The core of IMap includes a probe packet generation module, which is responsible for generating high-speed probe packets with random address and adaptive rate, and a response packet processing module, which processes response packets in a correct and efficient manner. We will describe the detailed design of these two modules below.

A. Probe Packet Generation

Switch is designed to be a packet forwarding device, not a packet generation device, thus cannot generate probe packets without ground. Inspired by HyperTester [37], we also leverage the template-based packet generation mechanism to generate high-speed probe packets. As shown in Figure 3, at runtime, when operators specify a scanning task, the switch CPU first sets the Working_State register in the switch pipeline to 1, and then prepares a set of template packets based on the scanning type, e.g., TCP SYN packets, ICMP echo request packets, or UDP packets, to be injected into switching ASICs. Meanwhile, the Scanning_Type register in the switch pipeline is also set to the corresponding value, i.e., 0 for TCP SYN scans, 1 for ICMP echo request scans, and 2 for UDP scans, which is used to control the processing logics for these three types of probe/response packets. Our tests manifest 50k template packets are enough for line-rate scanning and the injection takes 15 ms, causing negligible loads on the switch CPU. Upon receiving these template packets, switching ASICs keep looping these packets in the switch pipeline, where each packet experiences three sequential steps: an *accelerator* to accelerate the template packets to 100 Gbps line rate, a *replicator* to replicate the template packets into several switch ports, and an *editor* to edit the headers of replicated template packets into desired probe packets. After the scanning task finishes, the switch CPU sets the Working_State register to 0 to drain this type of template packets (the drain period should last for a least a few seconds for safety). And then operators

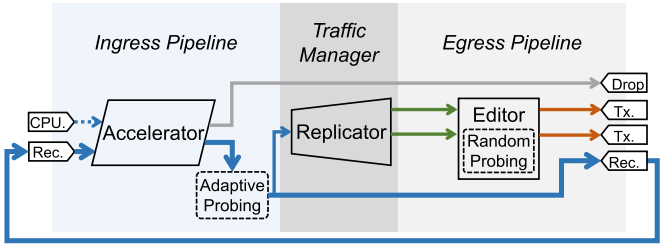


Fig. 3. Probe packet generation of IMap.

can specify another type of scanning task and start scanning again. In the following part, we will present the design of the accelerator, replicator and editor in detail.

1) Accelerator: The accelerator is located at the ingress pipeline, and its working state is determined by the Working_State register. When a scanning task is specified and the Working_State register is set to 1, the accelerator keeps looping the template packets by injecting these packets into the *recirculate port*. The recirculate port is a special port in the switch pipeline, where the injected packets are sent back to the ingress pipeline immediately. Therefore, after injecting a set of template packets to fill the switch pipeline, we get a 100 Gbps line-rate stable packet source for the replicator. When the scanning task is finished and the Working_State register is set to 0, the accelerator marks these template packets as dropping to drain the switch pipeline.

2) Replicator: The replicator is located at the traffic manager, which mainly takes the template packets from the accelerator as input and replicates these packets into a given port set with the *packet replication engine*. The packet replication engine is a hardware component in the traffic manager, which is widely supported by today's programmable switches. By configuring a set of ports for multicast from the control plane, incoming packets will be replicated and forwarded to the given port set in parallel. The original template packets from the accelerator will continue to be recirculated across the switch pipeline, to ensure line-rate stable packet source for the replicator, and the replicated template packets would go through the editor for further processing.

3) Editor: The editor resides in the egress pipeline, and it is responsible for modifying the replicated template packets into the desired probe packets. As long as the packet headers can be parsed by programmable switches, the headers can be set to given values, e.g., constants, or values from registers. To turn replicated template packets into probe packets, some header fields (e.g., destination IP address, destination port) need modification via the editor, while other fields (e.g., protocol type, source IP address) are inherited from the template packets which are created by the switch CPU initially.

With the steps above, we obtain continuous probe packets at 100Gbps line rate in multiple egress ports when the Working_state is 1. Nevertheless, to be a practical switch-based high-speed network scanner, IMap should be able to generate probe packets to cover the scanning space (i.e., |address space| × |port space|) completely, and adapt the scanning rate according to the network conditions.

4) Random Probe Address: To cover the scanning address space completely, an intuitive way is to scan from the start IP address to the end IP address one by one. Nevertheless, simply

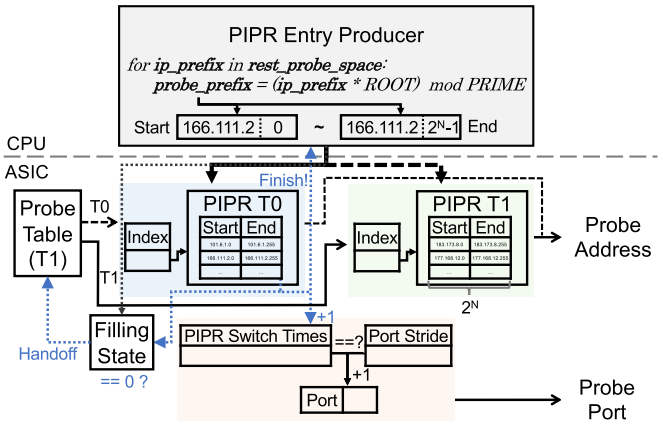


Fig. 4. Random probe address.

probing IP addresses in numerical order would risk overloading the target networks with the scanning traffic, which may incur complaints from the target networks and have larger possibilities of dropping probe/response packets. In the case of a distant transient network failure, the numerical probing order would produce inconsistent results. To avoid these problems, IMap should be able to scan the addresses according to a random permutation of the address space, without duplications and omissions. However, the switching ASICs only have limited programmability and memory resources, which cannot support complex calculations or maintain massive states. The address generation approach in ZMap [3] requires calculations such as multiplication and modulo, thus is not feasible in the switching ASICs.

To address this problem, we observe that although the switching ASICs do not have the ability to generate a random permutation of the address space, the flexibility of the switch CPU has such an ability. Therefore, we leverage the flexibility of the switch CPU to supplement the switching ASICs to generate line-rate address-random probe packets. In the editor of the switching ASICs, we design a Probe IP Range (PIPR) table based on register arrays. In the switch CPUs, we have a PIPR Entry Producer module. Using the address generation method similar to ZMap [3], the PIPR Entry Producer module can generate a random permutation of the probe IP ranges for a given address space. After the PIPR Entry Producer module fills part of the generated probe ip ranges into the PIPR table, probe packets can iterate through the PIPR table to obtain the random destination IP addresses. As the data plane scanning is pretty fast, a PIPR table with an entry size of 1 will be scanned quickly, so we store a probe ip range in each entry of the PIPR table. To implement this, our PIPR table consists of two register arrays: one is named as PIPR_Start array, which is used to store the start of the probe ip range; the other is named as PIPR_End array, to store the end of the probe ip range. Before the PIPR table, we have a PIPR_Index register, which is used to index the PIPR table. The initial value of the PIPR_Index register is set as 0 by the control plane; upon an incoming probe packet, the value of PIPR_Index increases by 1, until the size of the PIPR table; after that, the PIPR_Index is assigned as 0 again and another loop starts. For the PIPR_Start array, upon each incoming packet, the corresponding PIPR_Start register increases by 1, until the

PIPR_End register. When the value of the last PIPR_Start register is equal to the value of the last PIPR_End register, the scanning for the current PIPR table is finished, and the PIPR Entry Producer module is supposed to fill a new round of probe ip ranges into the PIPR table. To send the finish signal to the control plane, we leverage the *egress to egress mirror* primitive in the switch pipeline, which can carry a predefined flag to the switch CPU to notify the PIPR Entry Producer module.

However, conducting a new round of PIPR table filling is a time-consuming task. According to our tests on the Intel Tofino switch [18], even with the batching optimization, filling a PIPR table with a size of 65,536 requires about 0.3 seconds. This indicates that, after a round of scanning, the data plane has to wait for at least 0.3 seconds to start the next round of scanning. This is unacceptable for high-speed scanning, as it compromises the scanning rate significantly. To resolve this problem, we introduce two PIPR tables and PIPR_Index registers. When one PIPR table is being scanned, the other PIPR table is being filled with the next round of probe ip ranges. To make the two PIPR tables handoff seamlessly, we design a Probe_Table register in the first stage of the egress pipeline, which is switched between 0 and 1, and controls the flow of probe packets. The switching of the Probe_Table register is triggered by the finish signal of the egress to egress mirror primitive. Definitely, to achieve continuous probe packets, there is a mathematical relation that the PIPR table size, the PIPR table filling time, and the scanning rate must satisfy. Supposing the size of the PIPR table is N, the difference between each PIPR_Start register and PIPR_End register (i.e., the size of a PIPR table entry) is L, the PIPR table filling time is T seconds, the total scanning rate R (packets per second) should satisfy that $R \leq \frac{N \times L}{T}$. However, there are still a few extreme scenarios where the actual PIPR table filling time is longer than the expected T, e.g., caused by the congestion of the switch CPU or the control channel. It means the inequality is not held and the PIPR table is being read before fully filled. To deal with such cases, we add a Filling_State register before the Probe_Table register to indicate whether the PIPR table filling is finished. It is set to 1 when the control plane begins to fill and set to 0 when the control plane finishes the filling. The finish signal of the egress to egress mirror primitive will check whether the Filling_State register is 0 before it switches the Probe_Table register.

Until now, leveraging Algorithm 1, we have achieved that a large scanning address space is divided into several pieces and filled into the PIPR tables, enabling high-speed, continuous probe packets with random addresses. However, the designs above only consider one port scenario, which should be extended to support a port range scenario, e.g., scanning from port 22 to port 80. Since the scanning address already has good randomness, we choose to scan the port one by one. However, updating the Port register from the control plane would bring about race conditions, as the high-speed probe packets are already looping in the switch pipeline. To address this, we design a port self-increment mechanism in the data plane. As the control plane knows in advance the number of times the scanning address space needs to loop in the PIPR table, we design a Port_Stride register in the switch pipeline, which is filled with the number of loop times by the control plane. Every time the scanning of one PIPR table finishes, the corresponding PIPR_Switch_Times counter increases by 1, until the value of the Port_Stride register. Then, the Port register increases by 1 and the counter is set as 0 again. With all the mechanisms above, the final design of our random probe address is described in Figure 4, which achieves to generate

Algorithm 1 Random Address Generation Algorithm

```
// Control plane
Generate a random permutation for the given probe address space;
Set Probe_Table as 0;
Set PIPR0_Index and PIPR1_Index as 0;
Fill PIPR0_Start and PIPR0_End;
while the rest probe address space is not 0 do
    if Receive finish signal then
        extract Probe_table from the finish signal;
        Set Filling_State as 1;
        if Probe_Table is 0 then
            Fill PIPR1_Start and PIPR1_End;
        else
            Fill PIPR0_Start and PIPR0_End;
        Set Filling_State as 0;
// Data plane
for each incoming template packet do
    if Probe_Table is 0 then
        if PIPR0_Index is size_of_PIPR_table then
            PIPR0_Index ← 0;
        else
            PIPR0_Index ← PIPR0_Index + 1;
        PIPR_Index ← PIPR0_Index;
        metadata.end ← PIPR0_End[PIPR_Index];
        packet.dstIP ← PIPR0_Start[PIPR_Index];
        PIPR0_Start[PIPR_Index] + +;
    else
        if PIPR1_Index is size_of_PIPR_table then
            PIPR1_Index ← 0;
        else
            PIPR1_Index ← PIPR1_Index + 1;
        PIPR_Index ← PIPR0_Index;
        metadata.end ← PIPR1_End[PIPR_Index];
        packet.dstIP ← PIPR1_Start[PIPR_Index];
        PIPR1_Start[PIPR_Index] + +;
    if packet.dstIP is metadata.end and
     PIPR_Index is size_of_PIPR_table and
     Filling_State is 0 then
        Probe_Table ← 1 − Probe_Table;
        Generate finish signal with Probe_Table;
```

address-random probing packets to cover the scanning space completely, without overwhelming target networks.

5) Adaptive Probe Rate: To avoid affecting the normal packet routing functionality of the network, IMap desires a network-aware method to generate high-speed probe packets with adaptive rate. The "adaptive" here has two kinds of meanings. First, the control plane of the IMap switch should be aware of the nearby network conditions for further scanning rate adjustment. Furthermore, the IMap data plane should have a rate-adjusting interface, which can receive commands from the control plane to accurately adjust the scanning rate.

To be control plane aware, IMap should be able to adjust the scanning rate according to the network conditions. We formulate the scanning rate adjustment problem as follows. The scanning network is modeled to a graph $G = (V, E)$, where V and E are sets of forwarding devices and directed links between devices. Note that link $e = (v_i, v_j)$ is directed, and (v_i, v_j) and (v_j, v_i) are different links. Each link $e \in E$ has a capacity c_e and its current load is represented with l_e. We assume there exists a monitoring system in the network, so l_e can be obtained with the port bandwidth usage of the devices connected by e periodically. IMap is deployed in device $v_{IMap} \in V$, where its switch ports $P_{IMap} = \{p\}$ connect to the network with links $\{e_p\} \subset E$ and the maximal scanning rate at port p is $c_{e_p} - l_{e_p}$. According to the routing table and spare bandwidth at v_{IMap}, we partition the scanning space S by P_{IMap} in advance so that each port p corresponds to a sub-scanning space $s_p \subset S$. Our scanning space partition follows the following principles. (1) Since each switch port p corresponds to a certain routing table, for the address space that is included in the routing table, the address space is assigned to the switch port p directly. (2) For the remaining address space that is not included in any routing table, the address space is partitioned based on the spare bandwidth at port p (i.e., $c_{e_p} - l_{e_p}$). Ports that have large spare bandwidth would obtain more remaining address space and vice versa, as this would accelerate the scanning speed significantly. After that, for each sub-scanning space s_p, we can estimate the extra load $d_{p,e}$ on each link e caused by full-rate probe packets $c_{e_p} - l_{e_p}$. This is achieved by configuring IMap to send probe packets in sub-scanning space s_p with a specific tag on port p at low rate, then using the monitoring system to detect the load caused by the traffic with the given tag, and finally inferring $d_{p,e}$ for $c_{e_p} - l_{e_p}$ with equal proportion [38], [39]. Such partition and estimation should be repeated to adapt to routing dynamics when the routing tables in the scanning network change drastically. Then the scanning rate adjustment problem can be solved based on the Linear Programming (LP), as follows:

$$max \sum_{p \in P_{IMap}} \alpha_p (c_{e_p} - l_{e_p}) \tag{1}$$

$$s.t.\ \forall e \in E : l_e + \sum_{p \in P_{IMap}} \alpha_p d_{p,e} \leq \beta c_e \tag{2}$$

where $0 \leq \alpha_p \leq 1$ denotes the rate throttling parameter and $0 \leq \beta \leq 1$ denotes the maximum bandwidth occupation ratio. α_p is the output of this formulation and β is set by administrators to make the network robust for burst traffic. Equation (1) indicates that the objective is to maximize the total scanning rate on all ports. And Equation (2) states the extra load brought by IMap can not overwhelm any link in the network. Given $\{\alpha_p\}$, the control plane can determine the scanning rate for each port. Note that our current design fits one single Autonomous System (AS) network; for inter-AS networks, as different networks belong to different administrative domains and are not willing to share confidential information (e.g., network topology, network utilization), it is extremely difficult to design an inter-AS network-aware rate adjustment approach accurately. IMap is mainly designed for the single-AS network scanning, and only provides a best-effort probing service for inter-AS network scanning tasks.

To make the scanning rate of IMap adjustable, we add a *throttle* in the switch pipeline, which can be adjusted from the control plane dynamically. Located in the ingress pipeline, the throttle is used to determine when the replicator could replicate the template packets. In general, the switching ASICs can provide a per-port 100 Gbps packet processing capability, thus enabling nanosecond-level (e.g., ∼6 nanoseconds for 64-byte packets) timestamp for each incoming packet. Our throttle consists of two registers in the switch pipeline. The first one is named as a Timestamp register, which is used to record the timestamp of the last template packet that is successfully replicated and sent out to the editor. For every incoming template packet, we calculate the difference between the timestamp of the current packet and the timestamp recorded in the Timestamp register. Upon the difference exceeds a certain threshold, we pass the template packet to the replicator and update the recorded timestamp. The second one is named as a Period register, which is used to make the aforementioned threshold configurable from the control plane. In the ingress pipeline, the Period register resides in the front of the Timestamp register, and the control plane programs can fill a certain value into the Period register to achieve the rate control.

B. Response Packet Processing

As an in-network scanner based on the core switch, IMap is also responsible for forwarding normal packets, e.g., packets from other routers and switches in the network. IMap should be able to distinguish normal packets and response packets correctly. Meanwhile, since the throughput of response packets may be large, IMap should be able to efficiently process the response packets to avoid saturating the storage server.

1) Distinguishing Normal/Response Packets: To distinguish response packets from normal packets, one approach is to maintain a secret state for each probe packet, and then verify whether the response packet corresponds to the secret state accordingly. However, the switching ASICs only have limited memory resources, which cannot maintain massive secret states.

To resolve this, we design a stateless connection mechanism inspired by SYN cookies [40]. Rather than maintaining states in the switching ASICs, we encode the secret state into the mutable fields of each probe packet. The fields should have recognizable effects on fields of the corresponding response packets. Specifically, for TCP scanning, we choose the source port and sequence number; for ICMP, we use the ICMP identifier and sequence number; for UDP, we use the

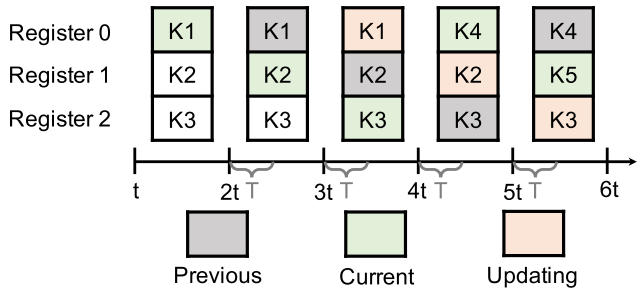


Fig. 5. Key update procedure of IMap.

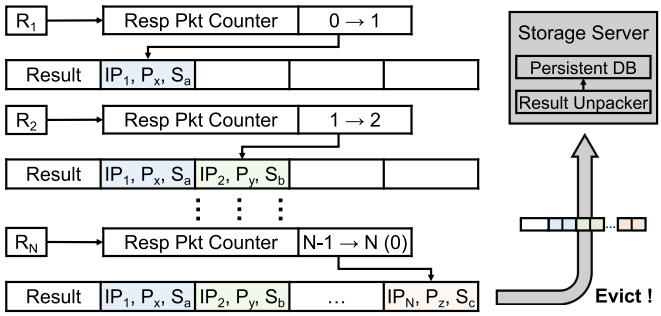


Fig. 6. Response packets aggregating in IMap.

source port. Take TCP as a concrete example, in the egress pipeline, when IMap sends a probe packet, the editor sets $SrcPort$ as $hash(Key, Proto, SrcIP, DstIP)$, and $SeqNo$ as $hash(Key, Proto, SrcIP, DstIP, SrcPort, DstPort)$, where Key is a secret key maintained in the register of the switching ASICs. Accordingly, in the ingress pipeline, IMap has a *verifier*, which checks the $DstPort$ and $AckNo$ to determine whether the received packet is a valid response to the probe packet. ICMP scanning and UDP scanning work in a similar manner, except for different packet fields. After the verifier checks the validation of the response packets, similar to ZMap [3], IMap also responds a TCP RST packet to each SYN-ACK packet to close the TCP connection.

One potential issue with the method above is the security of the verifier. Currently the hash functions supported in the switching ASICs (e.g., CRC32) are relatively simple, which are not true cryptographic functions and are vulnerable to *chosen plaintext attacks* [41]. As a result, attackers may perform such attacks to restore the Key, and deliberately inject forged response packets to pollute the scanning results. To further enhance the security of the verifier and enable pollution-free scanning results, IMap updates the Key every t seconds. This can reduce the damage caused by compromised secret keys to a large extent: even if an attacker somehow manages to obtain the current key, such knowledge will become useless after at most t seconds.

However, naively updating the Key would result in inconsistent scanning results. For example, $Key1$ is updated to $Key2$ after IMap sends the probe packet. Soon the response packet arrives, the verifier determines this packet is invalid as the current key cannot obtain a correct validation for its packet headers. To address the inconsistency issue described above, IMap stores the last key used for a certain period. More specifically, IMap maintains three keys (i.e., the previous key, the current key, and the next key) at any given time. Every t seconds, IMap rotates a slot index from 0 to 2, and the key in $slot_i$ is used for the hash function. Each key can stay in a slot for at most $3t$ seconds; after $3t$ seconds, the key is updated by the control plane. A concrete example is shown in Figure 5, where T denotes the max time interval between any probe packet and the corresponding response packet. The editor also encodes the 2-bit slot index of the key into the header fields of the probe packet, and the fields should also be added in the corresponding response packet within this connection. Currently, we encode it into the source port for TCP/UDP and the identifier for ICMP. Based on the slot index, the verifier can conduct the validation correctly.

2) Aggregating Response Packets: To avoid saturating the storage server, IMap desires an efficient response packet processing approach. One intuitive approach is to use hash mechanisms [37], [42], [43]. However, as the key set is really large in IMap (e.g., the size of the scanning address space), even only storing 2-bit value for each key requires GB-level memory, which exceeds the memory resources of the switching ASICs (i.e., 50-100MB [43]) significantly.

To resolve this problem, instead of seeking to store all the keys/values, we adopt a response packet aggregation mechanism that is compatible with the current switching ASICs. More specially, as shown in Figure 6, IMap designs an aggregator, i.e., a dedicated N-size register array, to temporarily store the scanning results. For each incoming response packet, IMap extracts its source IP, source port and state (i.e., active or inactive), and records the information in one register. When the register array is filled up, the corresponding response packet packs all the results from the register array, and goes to the storage server. To determine which register stores which result, we implement a Resp_Pkt_Counter register in the ingress pipeline. Upon an incoming response packet, the Resp_Pkt_Counter register increases by 1. The information extracted from the i-th packet will be stored in the i-th register. The N-th response packet will trigger the replication and be sent to the switch port connected with the storage server, packing and carrying all the results from the register array. Meanwhile, the Resp_Pkt_Counter register is reset as 0 and another aggregation loop starts. With this approach, IMap achieves an N to 1 aggregation, reducing the pressure for the bandwidth of the storage server significantly. In the side of the storage server, we use DPDK [23], a high-performance I/O framework, to parse the result packets and extract the scanning results. Finally, the scanning results are stored in a persistent database.

Note that across the switch pipeline, the packet processing logics for the three scanning types (i.e., TCP SYN scans, ICMP echo request scans, and application-specific UDP scans) are compiled into the switching ASICs in advance, and its runtime packet processing logic is determined by the metadata read from the Scanning_Type register.

V. Implementation

We implement a prototype of IMap, and make our code publicly available here [19]. Figure 7 illustrates the component

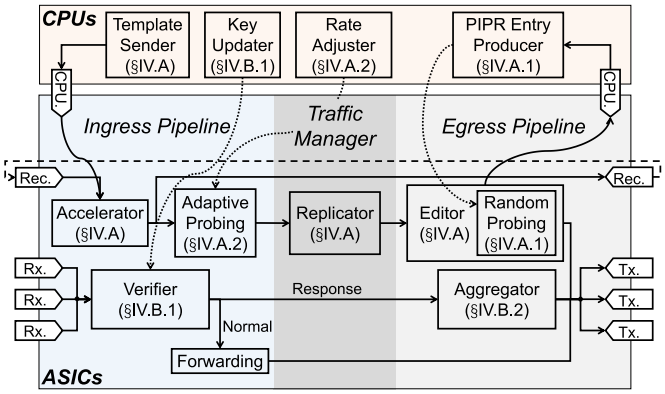


Fig. 7. Component layout of IMap.

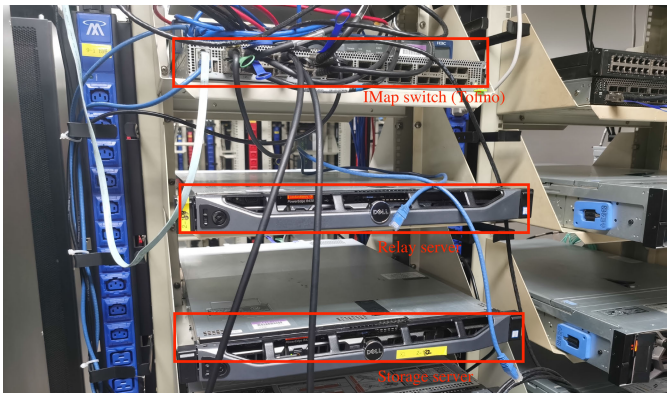


Fig. 8. A real-world setup of the IMap switch, the relay server and the storage server.

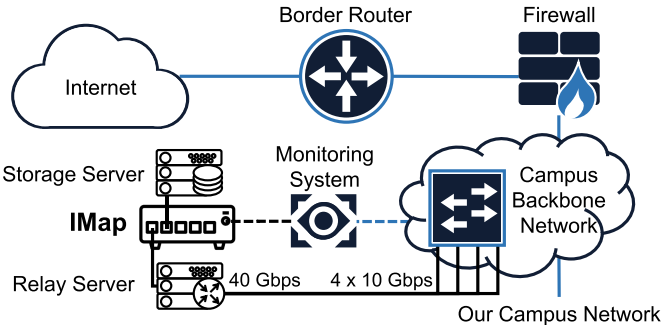


Fig. 9. Deployment of IMap.

layout of IMap on the data plane switching ASICs and the control plane switch CPUs.

The data plane part is implemented with ∼2K lines of P4-16 code for the Intel Tofino ASIC, with an increment of ∼300 lines of codes compared with the previous IMap [1]. In the probe packet generation module, we set the size of PIPR tables as 65536 and the size of one PIPR table entry as 256. In the response packet processing module, we utilize CRC32 as the *hash* function, allocate a 64-bit register for each Key, and set the size of the register array to store results temporarily as 16. In addition, IMap should also be able to process the ARP request packets correctly. Under our scenario, IMap is built on a switch, so sometimes other devices would ask for the MAC address of the IMap switch.

The control plane part is written in ∼3K lines of C code, with an increment of ∼500 lines of codes compared with the previous IMap [1]. It is responsible for initializing the data plane, injecting template packets, receiving update notifications, updating entries/registers in the data plane and interacting with the campus monitoring systems. In the probe packet generation module, we set β in Equation (2) as 0.8 to accommodate to traffic bursts, and solve the LP problem with the Gurobi [44] toolboxes. Since the routing tables in our campus network are pretty stable, we only estimate the extra load $d_{p,e}$ on each link by full-rate probe packets once, with the approach in §IV-A.5. In the response packet processing module, to reduce the risk of suffering from chosen plaintext attacks, the control plane generates a random Key every `t=1` second applies Xorshift [45] as the random number generator.

Besides, the backend agent running on the storage server is implemented with DPDK, which extracts the scanning results from the aggregated response packets and writes the results into a Redis [36] database.

VI. Evaluation

In this section, we evaluate IMap via testbed experiments and real-world deployments to answer the questions below:

- Can IMap conduct network scanning effectively (§VI-B)?
- Can IMap generate high-speed probe packets with random address and adaptive rate (§VI-C)?
- Can IMap process response packets correctly and efficiently (§VI-D)?
- How helpful is IMap in understanding our campus network's security situations (§VI-E)?

A. Experimental Setup

1) IMap Setup: Our testbed is composed of one 3.3 Tbp/s Intel Tofino switch (Edgecore Wedge 100BF-32X) and two Dell R430 servers. Both servers are equipped with Intel(R) Xeon(R) E5-2620 v3 CPUs and 64 GB memory, and connected to the switch via 40 GbE Intel XL710 NICs. In particular, one server runs as the storage server and the other server runs as the relay node to bridge IMap with our campus network. Figure 8 shows a real-world setup for the testbed above. With the relay server, we can collect and analyze the probe packets from IMap and the response packets to IMap accurately. Working with the network administrator of our campus network, we deploy IMap to connect to one backbone switch in our campus network, as shown in Figure 9. Due to security and reliability considerations, we are not allowed to replace the backbone switch with the IMap switch. The network conditions are obtained from the monitoring systems in our campus network, according to which IMap adjusts its scanning rate correspondingly. Since TCP SYN scan is one of the most representative single-packet probes, we use TCP SYN scan to evaluate IMap in most of our experiments. The scanning target is configured to some or all ports (0∼65535) of our campus network including 6 Class B addresses, a total of up to 25 billion scanning space, which is nearly 6 times larger than Internet-wide single-port scanning space.

2) Baselines: We use two state-of-the-art network scanners as baselines in our experiments, i.e., Zipper ZMap [21] (Z-ZMap for short) and Masscan [4]. They are deployed on a Dell R430 server located at the network edge, which is equipped with a 10 GbE Intel 82599ES NIC to connect to our campus network. Note that 10 Gbps is the maximum capacity

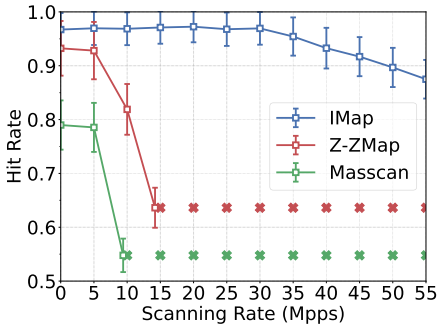


Fig. 10. Hit rate vs. Scanning rate.

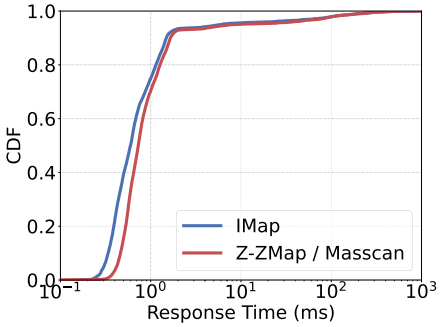


Fig. 11. Response time of probe packets.

officially supported on the project homepage of these baseline scanners. We adopt the fastest configuration recommended in Z-ZMap [21] for baseline scanners to achieve the best scanning capability, e.g., we install "PF_RING ZC" NIC driver to support the high-speed scanning of Z-ZMap and Masscan.

B. Overall Effectiveness

1) Scanning Accuracy: To determine whether IMap can perform high-speed scanning and obtain accurate scanning results in our campus network, we examine whether the scanning rate, i.e., the rate of probe packets sent from IMap, has any effect on the hit rate, i.e., the fraction of responsive probed hosts (responding with SYN-ACK or RST in this case). We experiment by using IMap and baseline scanners to scan port 80 of our campus network and normalize the experimental results. Figure 10 shows that IMap is capable of handling scanning at up to 55 Mpps without obvious hit rate degradation. In contrast, baseline scanners such as Z-ZMap and Masscan can neither reach a high scanning rate, nor achieve a comparable hit rate (at least 1.5 times gap). These benefits are brought by the in-network deployment location and performant switch implementation. Our results also show that baseline scanners experience a decreased hit rate with the higher scanning rate due to the drop of probe/response packets on the end-to-end scanning path [21].

2) Vantage Point: To demonstrate the advantage of IMap in employing in-network scanning, we probe all addresses in our campus network on port 80 and measure the latency between sending a probe packet and receiving the response packet from active hosts. We also conduct the same measurement on two baseline scanners at the same time. The CDF of the results are shown in Figure 11. IMap gains a much shorter round-trip response time for over 90 percent of hosts than state-of-the-art scanners. This is benefited from the fact that IMap is deployed in the core network and probe/response packets take a shorter path, 2-4 hops compared with 4-8 hops of end-to-end scanning. It also indicates the less bandwidth waste to the network and the smaller possibilities of dropping probe/response packets, which promises that IMap can conduct high-speed scanning accurately and efficiently.

TABLE I
SCANNING RATE AND SCANNING COMPLETION TIME

Scanner	Scanning Rate	Time for 1000-Ports Scan[1]	Time for All-Ports Scan
IMap	55.6 Mpps	12 seconds	8 minutes
Z-ZMap	14.2 Mpps	35 seconds	33 minutes
Masscan	9.4 Mpps	51 seconds	50 minutes

TABLE II
SWITCH RESOURCE UTILIZATION

	Computational			Memory	
Resource	Tables	ALUs	Gateways	SRAM	TCAM
IMap [1]	42.86%	45.84%	18.75%	20.83%	0.69%
Reconfigurable IMap	53.68%	56.75%	29.64%	31.32%	0.79%

3) Overall Functionality: To illustrate that IMap is fast and scalable in network scanning, we measure the scanning rate and scanning completion time of IMap and baseline scanners. Port 0-999 and all ports of our campus network are chosen as the scanning tasks respectively. For each scanner, we repeat both tasks for 10 trials at midnight to minimize the impact on our campus network and report the averages in Table I. The results show IMap is able to generate 55.6 million probe packets per second (close to 40 Gbps linespeed), which is a 4 times improvement compared with Z-ZMap and Masscan. Note that 40 Gbps is not the upper limit of IMap; instead, when we enable all ports of the switch, IMap can generate probe packets at terabit line rate. Currently, we cannot replace the core switch with IMap to conduct such a pressure test, which is left for our future work. Besides, Table I also shows IMap can complete scanning tasks much faster than the other scanners, which can help operators capture network security snapshots much more quickly. Moreover, to demonstrate IMap's capability in supporting runtime reconfiguration, we change the scanning task from TCP SYN scans to ICMP echo request scans with switch CLI at runtime. Our reconfigurable IMap does not need to reload a new P4 program nor incur switch downtime, and the procedure of scanning task re-specification happens smoothly. In contrast, to support the scanning task re-specification above, the previous IMap [1] results in seconds of switch downtime, during which IMap cannot function as a switch correctly.

4) Resource Overhead: To evaluate the resource consumption of IMap, we focus on its resource usage of our test switch, which is a low-end switch with pretty limited resources. Table II displays the average hardware resource utilization of IMap across all stages of the switch. As we can see, even with such a low-end switch, IMap takes up about half of computational resources, one-third of SRAM, and negligible TCAM, still leaving enough resources for the concurrent execution of traditional forwarding behaviors [43]. Leveraging high-end switches with more hardware resources (e.g., Edgecore Wedge 100BF-65X), the resource usage of IMap can be much

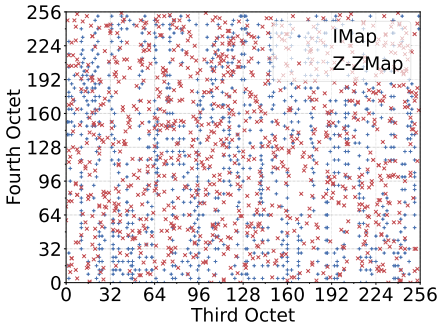


Fig. 12. Distribution of generated probe addresses.

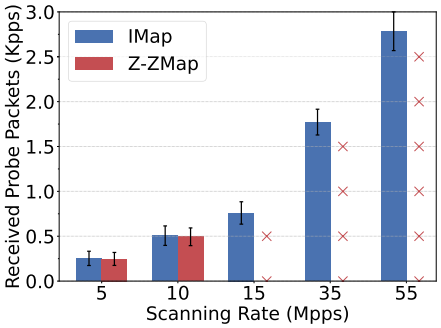


Fig. 13. Network pressure for access networks.

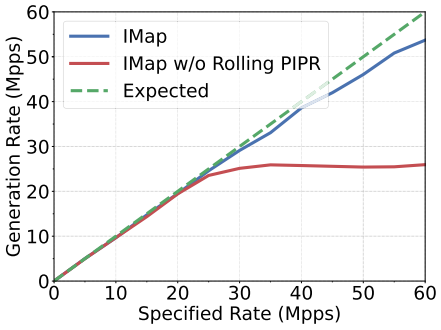


Fig. 14. Generation rate of probe packets.

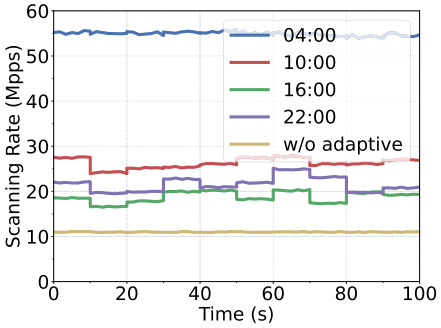


Fig. 15. Adaptive scanning rate.

lower. Besides, compared with the previous IMap [1], our reconfigurable IMap occupies a bit more hardware resources. This is because to support the reconfiguration of different scanning tasks at runtime, our reconfigurable IMap puts the processing logics of three types of probe/response packets into the switching ASICs in advance. Moreover, the resource utilization of a switch does not have any obvious effect on its forwarding performance. This is because as long as the compiled P4 program that integrates IMap and the forwarding functionality can be fitted into the switching ASICs, the switch is guaranteed to process and forward packets at terabit line rate [46], [47].

C. Probe Packet Generation

1) Random Probing: To validate the randomness of probe addresses generated by IMap, we first explore the distribution of the first 1000 addresses selected by IMap and Z-ZMap when they are probing port 80 of our campus network. Considering our campus network only contains class B addresses, as shown in Figure 12, we only keep the last two octets of the IP address and map them to the x and y coordinates, respectively. Based on the results, we can see that the addresses of IMap show a pattern of several blurred vertical lines to some extent, which results from the scanning orders from one PIPR_Start register and the corresponding PIPR_End register, while for Z-ZMap, the addresses show no statistical pattern. Although the address randomization of IMap achieves slightly worse statistical properties than Z-ZMap, we believe it is still good enough to avoid overwhelming the destination networks. To verify this, we analyze the pressure IMap brings to access networks. Figure 13 indicates several vital access networks in our campus network only receive thousands of probe packets per second even though the scanning rate of IMap reaches as high as 55 Mpps. Such additional packet overhead is negligible for most edge networks.

2) Adaptive Probing: To evaluate the adaptability of scanning rate of IMap, we first quantify the rate control accuracy of IMap by comparing the rate specified by the runtime parameter with the actual rate of probe packets sent from IMap. As shown in Figure 14, the error gradually increases with the rising of the scanning rate, but it is always limited to 5% even when the scanning rate of IMap reaches 55 Mpps. Such error mainly comes from the restricted accuracy of the packet rate in the recirculate port and can be manually corrected in the real-world scanning. Besides, from this figure, we can also see that rolling PIPR filling optimization (§IV-A) helps IMap achieve high-speed scanning continuously. Then we investigate whether IMap can adjust its scanning rate according to network conditions. We conduct scanning on our campus network with IMap at different time, and record the rate of probe packets. Since the monitoring system reports the campus network conditions every 10 seconds, and the LP problem can be solved within 3 seconds for our campus network, we make IMap update the scanning rate every 10 seconds to adapt to the change of the network conditions. To show the comparison with IMap without the rate-adaptation technique, we have several conversations with the network administrators of our campus network and confirm the largest conservative probe rate they can accept. This probe rate is an estimation of the spare bandwidth in our campus network based on their experiences, which should have no obvious effect on the normal packet routing functionality. All the result of the probe rate is shown in Figure 15. As we can see from this figure, with the rate-adaptation technique, IMap can adapt to the network conditions effectively and take full advantage of the spare bandwidth in the network, achieving much higher efficiency. Besides, compared with the previous IMap [1], the probe efficiency of our reconfigurable IMap improves 3-10% in our campus network.

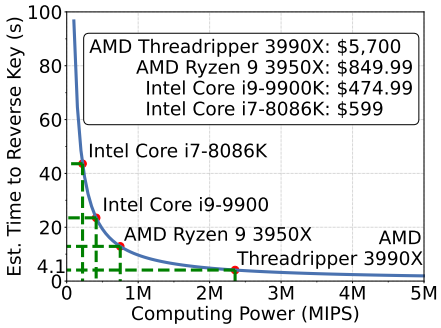


Fig. 16. Reverse time for Key.

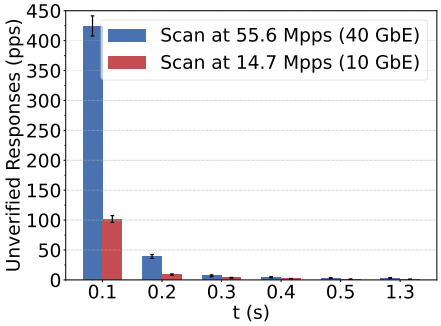


Fig. 17. Impact of $\mathtt{t}$ to probing.

D. Response Packet Processing

1) Secure Verifying: The security of the response verifier is guaranteed by the dynamic key updating technique in IMap, whose efficiency is decided by the parameter $\mathtt{t}$. To find a suitable value for $\mathtt{t}$, we first simulate the relationship between the computing power and the time required to reverse Key used by the hash function in IMap. As we can see from Figure 16, it takes about 4 seconds for high-end CPUs and more than 20 seconds for mainstream CPUs to locate the real Key using the stack algorithm [48]. In this case, IMap is protected from chosen plaintext attacks with $\mathtt{t}$ smaller than 1.3 seconds. Then we choose several different $\mathtt{t}$ for IMap and scan all ports of our campus network to seek how $\mathtt{t}$ affects probing. Figure 17 presents the number of response packets received by IMap but not pass the verifier during each scan, which occurs when the response time is beyond $3\mathtt{t}$. The results manifest that, under a common attacker, 0.3s∼1.3s are all applicable choices for $\mathtt{t}$ in our campus network.

2) Response Aggregating: To testify the efficiency of the response packet aggregation mechanism, we configure IMap to scan the campus network at different rate, and monitor the response traffic that is sent from the switch to the storage server. Figure 18 and 19 display the packet rate and throughput of such traffic with or without aggregating response packets respectively. It can be seen that the aggregation enables a 93.8% reduction in RX rate and an 86.1% reduction in RX throughput for the storage server, which efficiently protects it from being saturated by massive response traffic.

E. Analysis of Scanning Results

High-speed scanning of IMap has enabled faster snapshots of the network. Therefore, we conduct an experiment where IMap continuously scans all addresses in our campus network on all TCP ports. This experiment lasts for a week and the

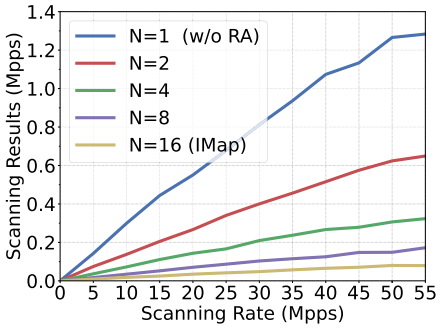


Fig. 18. Packet rate of scanning results.

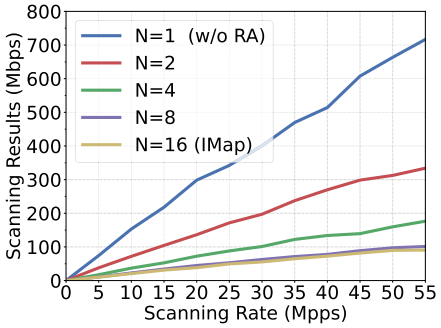


Fig. 19. Throughput of scanning results.

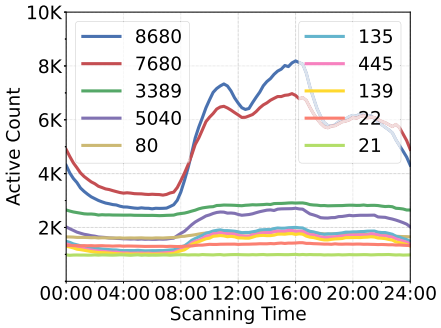


Fig. 20. Activity of ports over one day.

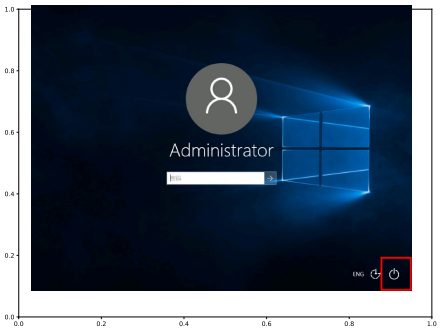


Fig. 21. RDP screenshot of a vulnerable host.

scanning results in the Redis database are persisted into the disk with a tag of time after each scan is over. In order to explore potential applications of IMap, based on the scanning results, we attempt to track the adoption of common protocols and discover new potential risks and security incidents in our campus network.

1) Protocol Adoption: We first compute the average count of active and inactive hosts for each port in all periods. Table III lists the top 10 open ports we observed and reveals several interesting findings. First, as the proportion of online devices in our campus network (∼5%) is far lower than that

TABLE III
TOP 10 ACTIVE TCP PORTS OF OUR CAMPUS NETWORK

Port	Service	Active Rate	Active Count	Inactive Count
8680	WeChat	1.34%	5271	12555
7680	Windows Update Delivery	1.33%	5211	12065
3389	RDP (Remote Desktop)	0.69%	2693	11659
5040	Windows Deployment Service	0.55%	2176	11709
80	HTTP	0.44%	1722	4841
135	Microsoft DCE/RPC	0.41%	1607	11081
445	Microsoft-DS	0.38%	1499	10592
139	NetBIOS Session	0.36%	1422	10559
22	SSH	0.34%	1354	4831
21	FTP	0.25%	983	10918

TABLE IV
EXPLOITABILITY OF VULNERABILITIES TO 135 AND 3389

Port	Vulnerability	Exploitability
135	Leak the host name, OS version, timestamp	100%
	Leak all NICs and IPs	99.6%
	Leak all RPC services	98.8%
3389	Leak the host name, OS version, timestamp	81.3%
	Leak the login screen	35.4%
	Remote shutdown*	20.2%

of the Internet, the active rate of the port is also lower than that of the Internet. Besides, we notice IMap just receives a small number of response packets from some sensitive ports, like ports 22 and 80, and we speculate the reason is that many systems filter such probe packets via the host firewall. Finally, we find Table III displays a really different sorting from that of the Internet in ZMap [3]. For example, the most active port in our campus network is 8680, which is used by WeChat, one of the most popular messaging Apps, and the second one is 7680, which is occupied by Windows to distribute system updates. We also observe a surprising number of open ports associated with file/device sharing over the network, such as ports 139, 445, and 5070. We attribute these differences to that more personal devices than servers are connected to our campus network. Furthermore, we then analyze the active rate variation of the top 10 ports by time over one day. As we can see in Figure 20, the active rate of some ports, e.g., 8680 and 7680, exposes an obvious diurnal pattern while that of other ports does not change significantly over time. This is because the former are usually opened by personal devices while the latter are opened by servers.

2) Potential Risks: Among the top 10 ports, 135 (DCE/RPC) and 3389 (RDP) catch our attention because they are known for information leakage. Considering their popularity, we investigate the exploitability of their vulnerabilities in our campus network. As shown in Table IV, 100% of the 135-opened hosts and more than 80% of the 3389-opened hosts are at the risk of information leakage. Moreover, the 3389-opened Windows hosts are also vulnerable to being shutdown remotely due to the misconfiguration from their users. For instance, Figure 21 is the screenshot from one of such vulnerable hosts, showing that users are allowed to perform shutdown operations on the RDP login screen. Even though the firewall of our campus network bans external access to internal hosts' sensitive ports including 135 and 3389, we believe these vulnerabilities still pose a high risk to our campus network and may be exploited by attackers. We have contacted our network administrators, and they confirmed these risks and issued a notice to remind teachers and students to check their configurations.

3) Botnets Detection: We also implement several alarm scripts triggered when the scanning results satisfy some conditions. One of them is used to detect botnets by monitoring whether the active count of certain ports surges in the last scan. During our experiment, we did find a fast increase of 48101-opened hosts and suspected it was caused by a Mirai botnet. We reported such an issue to the network administrators immediately and they finally determined it is just an experiment on Mirai conducted by a security lab. Even we dodged a bullet, it still reflects the potential of IMap in fast revealing security incidents with high-speed scanning, which cannot be obtained in time by existing network scanners like Z-ZMap and Masscan.

VII. DISCUSSION

A. Scanning Results V.S. Deployment Locations

From a network perspective, different switches have diverse network utilization, topological connection relations and access restrictions. As a result, the deployment locations of IMap affect the scanning results inevitably. Furthermore, we can also coordinate multiple switches to deploy IMap for cooperative scanning, which can achieve a higher scanning rate and hit rate. For any given network, there must be optimal distributed deployment locations in a given period, which can achieve the highest scanning rate and hit rate. We leave the deep exploration of optimal distributed deployment locations in a given network as our future work.

1) Relationship With Application-Layer Scanners: Currently, IMap only supports single-packet scanning, including TCP SYN scans, ICMP echo request scans, and application-specific UDP scans, and does not support complex application-layer protocols (e.g., TLS handshakes) directly. However, similar to ZMap, IMap can serve as a foundation to obtain the responsive host list from the given port, e.g., port 443 for TLS protocol. Based on this list, operators can use application-layer scanners to collect advanced information, e.g., a custom certificate fetcher to retrieve TLS certificates. In a word, IMap can narrow down the scanning space for application-layer scanners significantly.

VIII. RELATED WORKS

Our work is highly related to the following topics.

A. Network Scanners

Many network scanners have been developed to conduct network scanning tasks. Nmap [2] is optimized for small network segments with a wide variety of probing techniques. IRLscanner [49], ZMap [3], Masscan [4] and Zipper ZMap [21] are designed for Internet-scale scanning, mainly with a single-packet probing paradigm. IMap is very similar to ZMap and Masscan in the scanning methodology, but with different implementation targets and deployment locations, thus achieving orders of magnitude scanning capability improvement.

B. IPv6 Scanning

Numerous works have been devoted to improving the IPv6 scanning efficiency by optimizing the scanning space algorithmically. Entropy/IP [28] employs information entropy to segment the addresses in the hitlist and generate target addresses based on the relationship between different fragments. 6Gen [29] and Entropy-Clustering [31] extend the scope of prefix space for Entropy/IP and discover seed address fingerprint with clustering analysis. 6hit [27] adopts a reinforcement learning based target generation method to improve the probing efficiency. As a high-speed scanning system, IMap is completely orthogonal to these algorithmic works. And the scanning space generated from these algorithms can be set as the input of IMap to further improve the scanning efficiency.

C. Programmable Switches

Recently programmable switches have been used as accelerators for various applications in networking [42], [43], [50], distributed systems [24], [46] and security [47], [51], [52], [53], and these applications achieve far better performance with lower costs than their software counterparts running on commodity servers. The closer work to ours is HyperTester [37], which shows how to design a high-speed network tester with programmable switches. However, HyperTester illustrates neither how to generate probe packets with random address and adaptive rate, nor how to process response packets correctly and efficiently, nor how to support runtime reconfiguration when changing testing tasks. IMap addresses these unique challenges, and thus turns a switch into a practical high-speed network scanner.

IX. Conclusion

In this paper, we identify the limitations of current network scanners, and introduce IMap, a fast, scalable and reconfigurable in-network scanner with programmable switches. We devise a set of techniques and optimizations, i.e., an address-random and rate-adaptive probe packet generation mechanism, and a correct and efficient response packet processing mechanism, to turn a switch into a practical runtime-reconfigurable high-speed network scanner. We implement an open-source prototype of IMap and conduct extensive evaluations to show the advantages of IMap compared with current network scanners. We hope IMap can serve as the foundation of next-generation terabit network scanners.

References

[1] G. Li et al., "IMap: Fast and scalable in-network scanning with programmable switches," in *Proc. NSDI*, 2022, pp. 667–681.

[2] NMAP.ORG. (2021). *Nmap*. [Online]. Available: https://nmap.org/

[3] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast internet-wide scanning and its security applications," in *Proc. 22nd USENIX Secur. Symp.* Washington, DC, USA: USENIX, 2013, pp. 605–620.

[4] Masscan. (2021). *Masscan: Mass IP Port Scanner*. [Online]. Available: https://github.com/robertdavidgraham/masscan

[5] N. Aviram et al., "DROWN: Breaking TLS using SSLv2," in *Proc. 25th USENIX Secur. Symp.* Austin, TX, USA: USENIX, 2016, pp. 689–706.

[6] B. Beurdouche et al., "A messy state of the union: Taming the composite state machines of TLS," in *Proc. S&P*. San Jose, CA, USA: IEEE, 2015, pp. 535–552.

[7] S. Checkoway et al., "On the practical exploitability of dual EC in TLS implementations," in *Proc. 23rd USENIX Secur. Symp.* San Diego, CA, USA: USENIX, 2014, pp. 319–335.

[8] J. Amann, O. Gasser, Q. Scheitle, L. Brent, G. Carle, and R. Holz, "Mission accomplished? HTTPS security after DigiNotar," in *Proc. IMC*, New York, NY, USA, 2017, pp. 325–340.

[9] Z. Durumeric et al., "Neither snow nor rain nor MITM... an empirical analysis of email delivery security," in *Proc. IMC*, New York, NY, USA, 2015, pp. 27–39.

[10] R. Holz, J. Amann, O. Mehani, M. Wachs, and M. A. Kaafar, "TLS in the wild: An Internet-wide analysis of TLS-based protocols for electronic communication," in *Proc. NDSS*. San Diego, CA, USA: Internet Society, 2016.

[11] P. Richter, G. Smaragdakis, D. Plonka, and A. Berger, "Beyond counting: New perspectives on the active IPv4 address space," in *Proc. IMC*, New York, NY, USA, 2016, pp. 135–149.

[12] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, "Mining your Ps and Qs: Detection of widespread weak keys in network devices," in *Proc. 21st USENIX Secur. Symp.* Bellevue, WA, USA: USENIX, 2012, pp. 205–220.

[13] W3Techs. (2022). *Usage Statistics of IPv6 for Websites*. [Online]. Available: https://w3techs.com/technologies/details/ce-ipv6

[14] AVSystem. (2021). *5G IoT: What Does 5G Mean for IoT?* [Online]. Available: https://www.avsystem.com/blog/5g-iot/

[15] Vybint. (2021). *15 Alarming Cyber Security Facts and Stats*. [Online]. Available: https://www.cybintsolutions.com/cyber-security-facts-stats/

[16] P. Bosshart et al., "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 99–110, Oct. 2013.

[17] P. Bosshart et al., "p4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[18] Intel. (2021). *Intel Tofino: P4-Programmable Ethernet Switch ASIC that Delivers Better Performance at Lower Power*. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html

[19] IMapScanner. (2021). *IMap*. [Online]. Available: https://github.com/IMapScanner/IMap.git

[20] Google IPv6. (2022). *IPv6 Adoption*. [Online]. Available: https://www.google.com/intl/en/ipv6/statistics.html

[21] D. Adrian, Z. Durumeric, G. Singh, and J. A. Halderman, "Zippier ZMap: Internet-wide scanning at 10 Gbps," in *Proc. WOOT*. San Diego, CA, USA: USENIX, 2014.

[22] The ZMap Team. (2021). *The ZMap Project*. [Online]. Available: https://zmap.io/

[23] I. DPDK. (2021). *Learn How to Get Involved With DPDK*. [Online]. Available: https://www.dpdk.org/

[24] X. Jin et al., "NetChain: Scale-free sub-RTT coordination," in *Proc. NSDI*. Renton, WA, USA: USENIX, 2018, pp. 35–49.

[25] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, "NetBricks: Taking the V out of NFV," in *Proc. OSDI*, 2016, pp. 203–216.

[26] M. Zhang et al., "Tripod: Towards a scalable, efficient and resilient cloud gateway," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 570–585, Mar. 2019.

[27] B. Hou, Z. Cai, K. Wu, J. Su, and Y. Xiong, "6Hit: A reinforcement learning-based approach to target generation for Internet-wide IPv6 scanning," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, May 2021, pp. 1–10.

[28] P. Foremski, D. Plonka, and A. Berger, "Entropy/IP: Uncovering structure in IPv6 addresses," in *Proc. Internet Meas. Conf.*, Nov. 2016, pp. 167–181.

[29] A. Murdock, F. Li, P. Bramsen, Z. Durumeric, and V. Paxson, "Target generation for Internet-wide IPv6 scanning," in *Proc. Internet Meas. Conf.*, Nov. 2017, pp. 242–253.

[30] R. Beverly, R. Durairajan, D. Plonka, and J. P. Rohrer, "In the IP of the beholder: Strategies for active IPv6 topology discovery," in *Proc. Internet Meas. Conf.*, Oct. 2018, pp. 308–321.

[31] O. Gasser et al., "Clusters in the expanse: Understanding and unbiasing IPv6 hitlists," in *Proc. Internet Meas. Conf.*, Oct. 2018, pp. 364–378.

[32] Cisco. (2021). *Best Practices in Core Network Capacity Planning White Paper*. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/routers/wan-automation-engine/white_paper_c11-728551.html

[33] T. Pan et al., "Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 194–206.
[34] SDXcentral. (2021). *AT&T Picks Barefoot Networks for Programmable Switches*. [Online]. Available: https://www.sdxcentral.com/articles/news/att-picks-barefoot-networks-programmable-switches/2017/04/
[35] SDXcentral. (2021). *Barefoot Scores Tofino Deals With Alibaba, Baidu, and Tencent*. [Online]. Available: https://www.sdxcentral.com/articles/news/barefoot-scores-tofino-deals-with-alibaba-baidu-and-tencent/2017/05/
[36] Redis Labs. (2021). *Redis*. [Online]. Available: https://redis.io/
[37] D. Zhang, Y. Zhou, Z. Xi, Y. Wang, M. Xu, and J. Wu, "Hypertester: High-performance network testing driven by programmable switches," *IEEE/ACM Trans. Netw.*, vol. 29, no. 5, pp. 2005–2018, Oct. 2021.
[38] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *Proc. ACM Conf. SIGCOMM*, Aug. 2014, pp. 527–538.
[39] P. Kumar et al., "Semi-oblivious traffic engineering: The road not taken," in *Proc. NSDI*, 2018, pp. 157–170.
[40] D. J. Bernstein. (2021). *SYN Cookies*. [Online]. Available: https://cr.yp.to/syncookies.html
[41] S. Yoo and X. Chen, "Secure keyed hashing on programmable switches," in *Proc. ACM SIGCOMM Workshop Secure Program. Netw. Infrastruct.*, Aug. 2021, pp. 16–22.
[42] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 357–371.
[43] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "SilkRoad: Making stateful layer-4 load balancing fast and cheap using switching ASICs," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 15–28.
[44] Gurobi. (2021). *The Fastest Mathematical Programming Solver*. [Online]. Available: http://www.gurobi.com/
[45] G. Marsaglia, "Xorshift RNGs," *J. Stat. Softw.*, vol. 8, no. 14, pp. 1–6, 2003.
[46] X. Jin et al., "NetCache: Balancing key-value stores with fast in-network caching," in *Proc. 26th Symp. Operating Syst. Princ.*, Oct. 2017, pp. 121–136.
[47] M. Zhang et al., "Poseidon: Mitigating volumetric DDoS attacks with programmable switches," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020.
[48] G. Nivasch, "Cycle detection using a stack," *Inf. Process. Lett.*, vol. 90, no. 3, pp. 135–140, May 2004.
[49] D. Leonard and D. Loguinov, "Demystifying service discovery: Implementing an Internet-wide scanner," in *Proc. 10th ACM SIGCOMM Conf. Internet Meas.*, Nov. 2010, pp. 109–122.
[50] S. Narayana et al., "Language-directed hardware design for network performance monitoring," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 85–98.
[51] R. Meier, P. Tsankov, V. Lenders, L. Vanbever, and M. Vechev, "NetHide: Secure and practical network topology obfuscation," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 693–709.
[52] G. Li et al., "NETHCF: Enabling line-rate and adaptive spoofed IP traffic filtering," in *Proc. IEEE 27th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2019, pp. 1–12.
[53] Q. Kang, L. Xue, A. Morrison, Y. Tang, A. Chen, and X. Luo, "Programmable in-network security for context-aware BYOD policies," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 595–612.

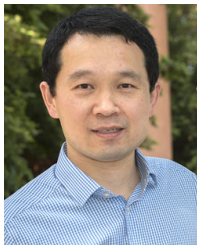
Menghao Zhang (Member, IEEE) received the B.S. and Ph.D. degrees in computer science from Tsinghua University in 2016 and 2021, respectively. He is currently an Associate Professor with the School of Software, Beihang University. His research interests include programmable networks, high-performance networks, networked systems, and network security.

Guanyu Li received the B.S. degree from the School of Compute Science and Technology, Huazhong University of Science and Technology, in 2018, and the Ph.D. degree from the Institute for Network Sciences and Cyberspace, Tsinghua University, in 2023. His research interests include software-defined networking, network function virtualization, and cyber security.

Cheng Guo received the B.S. degree in computer science from Tsinghua University in 2021, where he is currently pursuing the M.Sc. degree with the Institute for Network Science and Cyberspace. His research interests include programmable networks and high-performance networks.

Han Bao received the B.S. degree in computer science from Tsinghua University in 2022, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology. His research interests include programmable networks, datacenter networks, and network security.

Mingwei Xu received the B.S. and Ph.D. degrees from Tsinghua University. He is currently a Full Professor with the Department of Computer Science and Technology, Tsinghua University. His research interests include computer network architecture, high-speed router architecture, and network security.



Hongxin Hu (Member, IEEE) received the Ph.D. degree in computer science from Arizona State University, Tempe, AZ, USA, in 2012. He is currently an Associate Professor with the Department of Computer Science and Engineering, University at Buffalo, SUNY. His current research interests include security, privacy, networking, and systems.

Fenghua Li received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, in 2004. He is currently an Associate Professor with Tsinghua University. His research interests include network management, wireless networks, and network architecture.