IMap: Fast and Scalable In-Network Scanning with Programmable Switches

Guanyu Li, *Tsinghua University;* Menghao Zhang, *Tsinghua University; Kuaishou Technology;* Cheng Guo, Han Bao, and Mingwei Xu, *Tsinghua University;* Hongxin Hu, *University at Buffalo, SUNY;* Fenghua Li, *Tsinghua University*

https://www.usenix.org/conference/nsdi22/presentation/li-guanyu

This paper is included in the Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation.

April 4–6, 2022 • Renton, WA, USA

978-1-939133-27-4

Open access to the Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation is sponsored by

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

IMap: Fast and Scalable In-Network Scanning with Programmable Switches

Guanyu Li[⋆], Menghao Zhang[⋆†], Cheng Guo[⋆], Han Bao[⋆], Mingwei Xu[⋆], Hongxin Hu[◦], Fenghua Li[⋆]
[⋆]*Tsinghua University* [†]*Kuaishou Technology* [◦]*University at Buffalo, SUNY*

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 and scalable 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 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. Leveraging IMap, we also discover several potential security threats in our campus network, and report them to our network administrators responsibly.

1 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 [39], ZMap [14] and Masscan [33], 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 [3, 6, 10], monitor service deployments [2, 13, 20, 42] and shed light on previously opaque distributed systems [19], 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 proceeded to the stage of large-scale deployment, and reports show that IPv6 is used by 18.7% of all the websites [47]. Along with the adoption of 5G networks, more and more Internet-of-Things (IoT) devices and mobile devices are connecting online [4]. 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 [46]. 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. 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. 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.

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 pro-

grammable switches [9], which offer unprecedented programmability and flexibility without sacrificing performance. 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 [8]), 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 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. As an in-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 high-speed network scanner. We implement a prototype of IMap in an Intel Tofino switch [23], and make the source code publicly available [22]. 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 Addresses), 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. 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 following contributions in this paper:

- We analyze the limitations of current network scanners, and identify the opportunities brought by programmable switches (§2).
- We propose IMap, a fast and scalable 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 (§3, §4).
- We implement an open-source prototype of IMap, and conduct extensive testbed experiments and campus network deployments to show advantages of IMap (§5, §6).

Finally, we make some discussions in §7, describe related works in §8, and conclude this paper in §9.

2 Motivation and Observation

2.1 Limitations of Current Network Scanners

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 35% of its users access Google via IPv6 [24]. 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 [4]. 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 [46]. 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, with Zippier ZMap [1], 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 [45]. 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 [12], the throughput cannot reach more than 40 Gbps easily [25,41,50]. 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 the scanners are capable of scanning at 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 (§6.2). Not surprisingly, because of these fundamental limitations, since the publication of Zipper ZMap [1], 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 [7, 15, 16, 21, 36].

2.2 Opportunities by Programmable Switches

Programmable switches [8, 9] bring unprecedented opportunities to address the limitations of current network scanners.

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 [25]. Specifically, today's latest CPU-based network scanner, Zipper ZMap [1], 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 [25], thus not promising for a high-speed network scanner.

Flexibility to support 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 [8], 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. As a result, diverse scanning tasks can be implemented in the programmable switch, which would potentially be the foundation of next-generation high-speed network scanners.

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 [11]), 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.

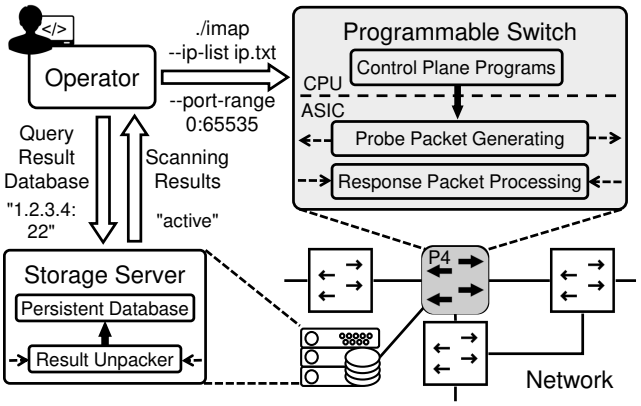

Figure 1: The workflow of IMap.

3 IMap Overview

3.1 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 [14]. 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% [11]). 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 their networks, which we believe is an irresistible trend in the foreseeable future [40, 43, 44]. 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.

3.2 Workflow and Design Requirements

IMap is designed to be a high-speed, easy-to-use network scanner, so the usage of IMap is similar to traditional network scanners, such as ZMap [14] and Masscan [33]. As shown in Figure 1, operators should first specify the scanning address spaces and scanning port ranges beforehand. Then IMap control plane programs parse these configurations and issue the parsed parameters into the IMap packet processing logic. After that, IMap data plane programs generate high-speed probe packets and process response packets accordingly. Finally, 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 [29]. In the design, implementation and deployment of IMap, we identify several different design requirements that must be satisfied to make IMap a practical high-speed network scanner, especially in terms of probing packet generation and response packet processing:

Space-complete and rate-adaptive probe packet generation(§4.1). 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.

Correct and efficient response packet processing(§4.2). With regards 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.

4 IMap Design

4.1 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 [49], we also leverage the template-based packet generation mechanism to generate high-speed probe packets. As shown in Figure 2, the switch CPU first prepares a set of template packets with initialized headers, and injects them into switching ASICs. 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. After 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.

(1) **Accelerator.** The accelerator is located at the ingress pipeline, and it 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.

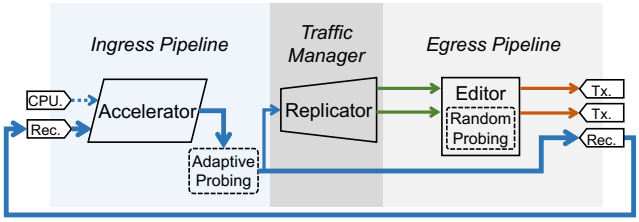

Figure 2: Probe packet generation of IMap.

(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 to modify 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 line rate in multiple egress ports. Nevertheless, to be a practical 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.

4.1.1 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 probing IP addresses in numerical order would overwhelm the target networks with the scanning traffic, which may produce inconsistent probing results and incur complaints from the target networks. To avoid this, IMap should be able to scan the addresses accord-

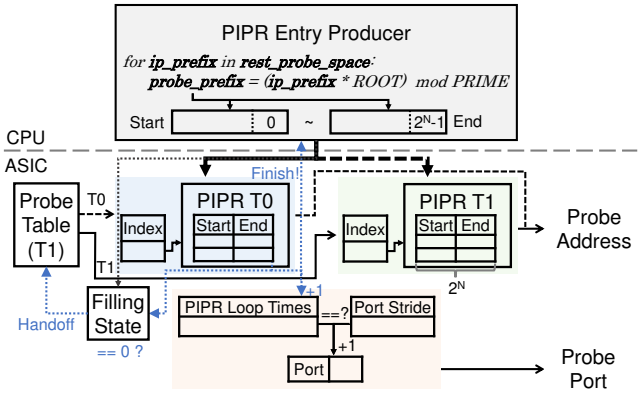

Figure 3: Random probe address.

ing to a 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 [14] requires calculations such as multiplication and modulo, thus is not feasible in the switching ASICs.

To address this problem, 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 [14], 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 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 [23], even with the batching optimization, filling a PIPR table with 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 PIPR_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 PIPR_Table register.

Until now, 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 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 3, which achieves to generate

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

4.1.2 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 can be 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 $v_{IMap} \in V$ and its ports $P_{IMap} = \{p\}$ connect to the network with links $\{e_p\} \subset E$. The maximal scanning rate for port p is c_{e_p}, which is the bandwidth capacity of the link e_p. According to the routing table on v_{IMap}, we can partition the scanning space S by P_{IMap} in advance so that each port p corresponds to a routing-aware sub-scanning space $s_p \subset S$. Besides, we can estimate the extra load $d_{p,e}$ on each link e caused by full-rate probe packets of s_p. This can be done by configuring IMap to send probe packets of 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 $d_{p,e}$ when the scanning rate is c_{e_p} [28, 31]. 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:

$$max \quad \sum_{p \in P_{IMap}} \alpha_p c_{e_p} \tag{1}$$

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

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 for one single Autonomous System (AS) network; for inter-AS networks, as different networks belong to different administrative domains and they 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 rate register, which is used to make the aforementioned threshold configurable from the control plane. In the ingress pipeline, the rate register resides in the front of the timestamp register, and the control plane programs can fill the certain value into the rate register to achieve the rate control.

4.2 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.

4.2.1 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 is corresponding 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 similar to SYN cookies [5]. Rather than maintaining states in the switching ASICs, we encode the secret state into the mutable fields of each probe packet. The fields should

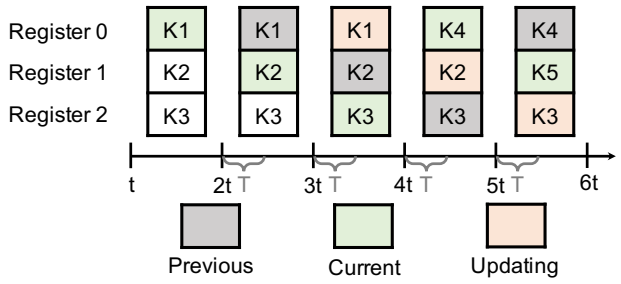

Figure 4: Key update procedure of IMap.

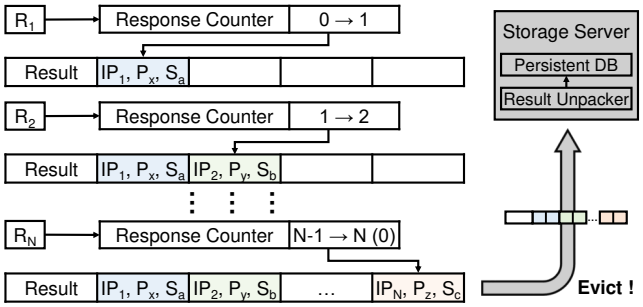

Figure 5: Response packets aggregating in IMap.

have recognizable effects on fields of the corresponding response packets. Specifically, for TCP scanning, we choose the source port and initial sequence number; for ICMP, we use the ICMP identifier and sequence number; for UDP, we use the 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 [14], 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* [48]. 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, simply updating the *Key* would result in inconsistent scanning results. For example, *Key*1 is updated to *Key*2 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 of time. 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 4, where T denotes the max time interval between any probe packet and the corresponding response packet. The editor will encode 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.

4.2.2 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 [17, 35, 49]. 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 [35]) 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 5, IMap designs 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 counter in the ingress pipeline. Upon an incoming response packet, the counter 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 counter 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 [12], 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.

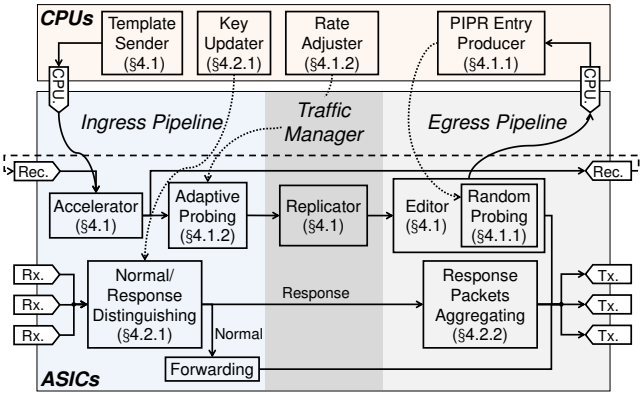

Figure 6: Component layout of IMap.

5 Implementation

We implement a prototype of IMap, and make our code publicly available here [22]. Figure 6 illustrates the component 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. 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.

The control plane part is written in ∼3K lines of C code. It is responsible to initialize the data plane, inject template packets, receive update notifications, update entries/registers in the data plane and interact 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 [18] 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 §4.1.2. 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 and the data plane applies Xorshift [32] 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 [29] database.

6 Evaluation

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

- What is the overall effectiveness of IMap to conduct in-network scanning (§6.2)?

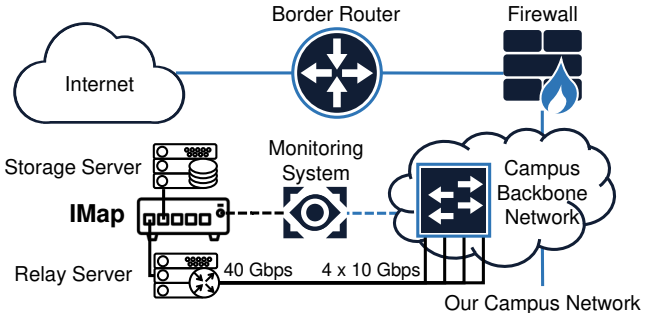

Figure 7: Deployment of IMap.

- Can IMap generate high-speed probe packets with random address and adaptive rate (§6.3)?
- Can IMap process response packets correctly and efficiently (§6.4)?
- How helpful is IMap in understanding our campus network's security situations (§6.5)?

6.1 Experimental Setup

IMap setup. Our testbed is composed of one 3.3 Tbp/s Intel Tofino switch (Edgecore Wedge 100BF-32X) and two Dell R730 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. 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 7. 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.

Baselines. We use two state-of-the-art network scanners as baselines in our experiments, i.e., Zipper ZMap [1] (Z-ZMap for short) and Masscan [33]. 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 officially supported on the project homepage of these baseline scanners. We adopt the fastest configuration recommended in Z-ZMap [1] 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.

Scanning Task. Since TCP SYN scanning is one of the most representative probes among single-packet probes, we use TCP SYN scanning to evaluate IMap in our experiments. The scanning target is configured to some or all ports (0∼65535)

Table 1: 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

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.

6.2 Overall Effectiveness

Scanning accuracy. In order 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 8 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 verify baseline scanners experience the decreased hit rate with the higher scanning rate due to the drop of probe/response packets [1].

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 9. IMap gains 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 IMap can conduct high-speed scanning accurately and efficiently.

Fastness and scalability. 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 the midnight to minimize the impact for our campus network and report the averages in Table 1.

[1]It includes time to send probe packets as well as a fixed 5-second delay after all probes are sent, during which scanners wait for late response packets.

Table 2: Switch resource utilization.

	Computational			Memory	
Resource	Tables	ALUs	Gateways	SRAM	TCAM
Usage	42.86%	45.84%	18.75%	20.83%	0.69%

The results show IMap is able to generate 55.6 million probe packets per second (close to 40 Gbps linespeed), which is 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 1 also shows IMap can complete scanning tasks much faster than the other scanners, which can help operators capture network security snapshots much more quickly.

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 2 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 less than half of computational resources, one-fifth of SRAM, and negligible TCAM, still leaving enough resources for the concurrent execution of traditional forwarding behaviors [35]. Leveraging high-end switches with more hardware resources (e.g., Edgecore Wedge 100BF-65X), the resource usage of IMap can be much lower. Besides, 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 [26, 51].

6.3 Probe Packet Generation

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 10, 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 address randomization of IMap achieves slightly worse statistical properties than Z-ZMap because IMap employs the PIPR table, but 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 11 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.

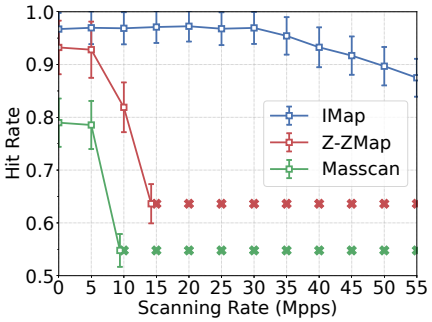

Figure 8: Hit rate vs. Scanning rate.

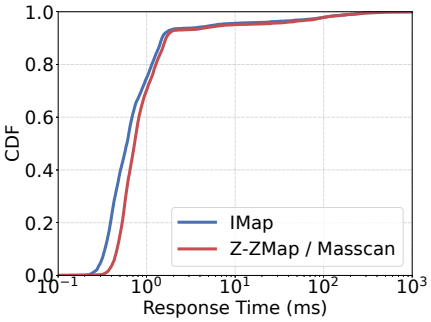

Figure 9: Response time of probe packets.

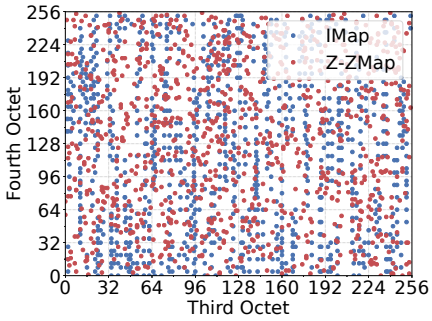

Figure 10: Distribution of generated probe addresses.

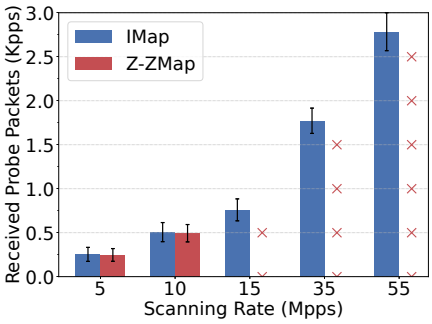

Figure 11: Network pressure for access networks.

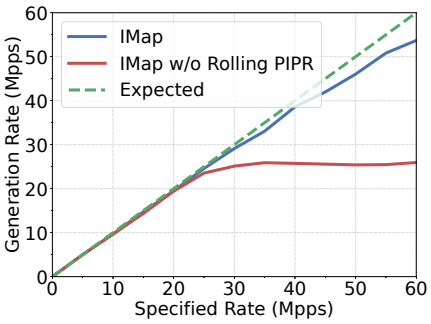

Figure 12: Generation rate of probe packets.

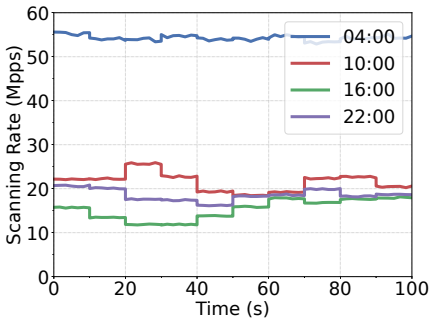

Figure 13: Adaptive scanning rate.

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 12, the error gradually increases with the rising of the scanning rate, but it is always limited within 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 scanning. Besides, from this figure, we can also see that rolling PIPR filling optimization (§4.1) 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 in Figure 13. 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.

6.4 Response Packet Processing

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 `t`. To find a suitable value for `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 14, 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 [38]. In this case, IMap is protected from chosen plaintext attacks with `t` smaller than 1.3 seconds. Then we choose several different `t` for IMap and scan all ports of our campus network to seek how `t` affects probing. Figure 15 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`t`. The results manifest that, under a common attacker, 0.3s∼1.3s are all applicable choices for `t` in our campus network.

Response aggregating. To testify the efficiency of 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 16 and 17 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.

6.5 Analysis of Scanning Results

High-speed scanning of IMap has enabled the faster snapshots of the network. Therefore, we conduct an experiment where IMap continuously scans all addresses in our campus

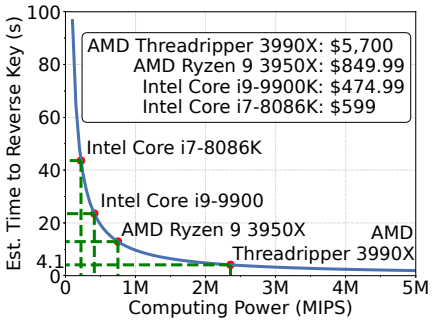

Figure 14: Reverse time for *Key*.

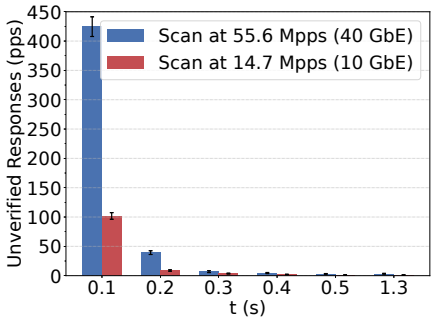

Figure 15: Impact of `t` to probing.

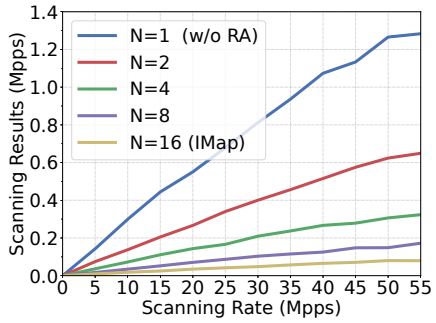

Figure 16: Packet rate of scanning results.

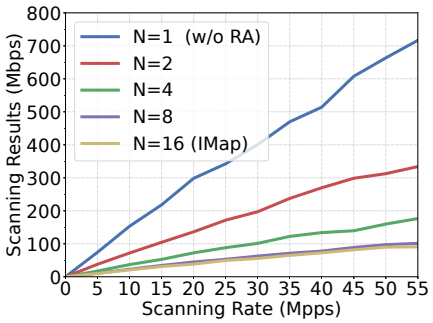

Figure 17: Throughput of scanning results.

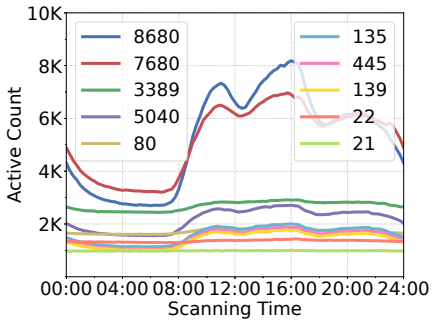

Figure 18: Activity of ports over one day.

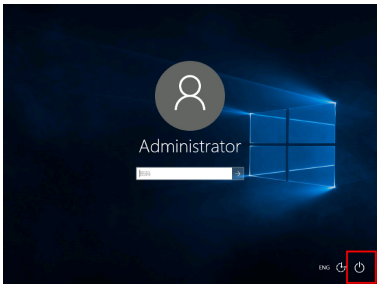

Figure 19: RDP screenshot of an vulnerable host.

Table 3: 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 4: 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%

network on all TCP ports. This experiment lasts for a week and the 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.

Protocol adoption. We first compute the average count of active and inactive hosts for each port in all time periods. Table 3 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 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 3 displays a really different sorting from that of the Internet in ZMap [14]. For example, the most active port in our campus network is 8680, which is used by WeChat, one of the most popular messaging App, 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 18, 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.

Potential risks. Among the top 10 ports, 135 (DCE/RPC) and 3389 (RDP) catches our attention because they are known for information leakage. Considering their popularity, we investi-

gate the exploitability of their vulnerabilities in our campus network. As shown in Table 4, 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 19 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.

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 is 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.

7 Discussion

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 of time, 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.

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.

8 Related Works

Our work is highly related to the following topics.

Network scanners. Many network scanners have been developed to conduct network scanning tasks. Nmap [39] is optimized for small network segments with a wide variety of probing techniques. IRLscanner [30], ZMap [14], Masscan [33] and Zipper ZMap [1] 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.

IPv6 scanning. Numerous research works have been devoted to improving the IPv6 scanning efficiency by optimizing the scanning space algorithmically. Entropy/IP [15] employs information entropy to segment the addresses in the hitlist and generate target addresses based on the relationship between different fragments. 6Gen [36] and Entropy-Clustering [16] extend the scope of prefix space for Entropy/IP and discover seed address fingerprint with clustering analysis. 6hit [21] 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.

Programmable switches. Recently programmable switches have been used as accelerators for various applications in networking [17, 35, 37], distributed systems [25, 26] and security [27, 34, 51], 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 [49], which shows how to design a high-speed network tester with programmable switches. However, HyperTester neither illustrates how to generate probe packets with random address and adaptive rate, nor how to process response packets correctly and efficiently. IMap addresses these unique challenges, and thus turns a switch into a practical high-speed network scanner.

9 Conclusion

In this paper, we identify the limitations of current network scanners, and introduce IMap, a fast and scalable 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 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.

Acknowledgments

We thank our shepherd, Wenting Zheng, and anonymous NSDI reviewers for their valuable comments. We would also like to thank Shicheng Wang and Xingjian Zhang from Tsinghua University for joining some discussions of this paper. This work is supported in part by the National Natural Science Foundation of China under Grant 61625203, 61832013 and 61872426, and Tsinghua University-China Mobile Communications Group Co.,Ltd. Joint Institute. Menghao Zhang and Mingwei Xu are the corresponding authors.

References

[1] David Adrian, Zakir Durumeric, Gulshan Singh, and J Alex Halderman. Zippier zmap: internet-wide scanning at 10 gbps. In *8th USENIX Workshop on Offensive Technologies (WOOT 14)*, San Diego, CA, 2014. USENIX.

[2] Johanna Amann, Oliver Gasser, Quirin Scheitle, Lexi Brent, Georg Carle, and Ralph Holz. Mission accomplished? https security after diginotar. In *Proceedings of the 2017 Internet Measurement Conference*, pages 325–340, New York, USA, 2017. ACM.

[3] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J Alex Halderman, Viktor Dukhovni, et al. Drown: Breaking tls using sslv2. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 689–706, Austin, TX, 2016. USENIX.

[4] AVSystem. 5g iot: What does 5g mean for iot? https://www.avsystem.com/blog/5g-iot/, 2021.

[5] D. J. Bernstein. Syn cookies. https://cr.yp.to/syncookies.html, 2021.

[6] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A messy state of the union: Taming the composite state machines of tls. In *2015 IEEE Symposium on Security and Privacy*, pages 535–552, San Jose, CA, USA, 2015. IEEE, IEEE.

[7] Robert Beverly, Ramakrishnan Durairajan, David Plonka, and Justin P Rohrer. In the ip of the beholder: Strategies for active ipv6 topology discovery. In *Proceedings of the Internet Measurement Conference 2018*, pages 308–321, 2018.

[8] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[9] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. *ACM SIGCOMM Computer Communication Review*, 43(4):99–110, 2013.

[10] Stephen Checkoway, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J Bernstein, Jake Maskiewicz, Hovav Shacham, and Matthew Fredrikson. On the practical exploitability of dual ec in tls implementations. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 319–335, San Diego, CA, 2014. USENIX.

[11] Cisco. Best practices in core network capacity planning white paper. https://www.cisco.com/c/en/us/products/collateral/routers/wan-automation-engine/white_paper_c11-728551.html, 2021.

[12] Intel DPDK. Learn how to get involved with dpdk. https://www.dpdk.org/, 2021.

[13] Zakir Durumeric, David Adrian, Ariana Mirian, James Kasten, Elie Bursztein, Nicolas Lidzborski, Kurt Thomas, Vijay Eranti, Michael Bailey, and J Alex Halderman. Neither snow nor rain nor mitm... an empirical analysis of email delivery security. In *Proceedings of the 2015 Internet Measurement Conference*, pages 27–39, New York, USA, 2015. ACM.

[14] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Zmap: Fast internet-wide scanning and its security applications. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 605–620, Washington, D.C., USA, 2013. USENIX.

[15] Pawel Foremski, David Plonka, and Arthur Berger. Entropy/ip: Uncovering structure in ipv6 addresses. In *Proceedings of the 2016 Internet Measurement Conference*, pages 167–181, 2016.

[16] Oliver Gasser, Quirin Scheitle, Pawel Foremski, Qasim Lone, Maciej Korczyński, Stephen D Strowes, Luuk Hendriks, and Georg Carle. Clusters in the expanse: Understanding and unbiasing ipv6 hitlists. In *Proceedings of the Internet Measurement Conference 2018*, pages 364–378, 2018.

[17] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*, pages 357–371, 2018.

[18] Gurobi. The fastest mathematical programming solver. http://www.gurobi.com/, 2021.

[19] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 205–220, Bellevue, WA, 2012. USENIX.

[20] Ralph Holz, Johanna Amann, Olivier Mehani, Matthias Wachs, and Mohamed Ali Kaafar. Tls in the wild: An internet-wide analysis of tls-based protocols for electronic communication. In *Symposium on Network and Distributed System Security (NDSS)*, San Diego, CA, USA, 2016. Internet Society.

[21] Bingnan Hou, Zhiping Cai, Kui Wu, Jinshu Su, and Yinqiao Xiong. 6hit: A reinforcement learning-based approach to target generation for internet-wide ipv6 scanning. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[22] IMapScanner. IMap. https://github.com/IMapScanner/IMap.git, 2021.

[23] Intel. Intel Tofino: P4-programmable Ethernet switch ASIC that delivers better performance at lower power. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html, 2021.

[24] Google Ipv6. Ipv6 adoption. https://www.google.com/intl/en/ipv6/statistics.html, 2021.

[25] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. Netchain: Scale-free sub-rtt coordination. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 35–49, Renton, WA, USA, 2018. USENIX.

[26] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 121–136, 2017.

[27] Qiao Kang, Lei Xue, Adam Morrison, Yuxin Tang, Ang Chen, and Xiapu Luo. Programmable in-network security for context-aware byod policies. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 595–612, 2020.

[28] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. Semi-oblivious traffic engineering: The road not taken. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 157–170, 2018.

[29] Redis Labs. Redis. https://redis.io/, 2021.

[30] Derek Leonard and Dmitri Loguinov. Demystifying service discovery: implementing an internet-wide scanner. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 109–122, 2010.

[31] Hongqiang Harry Liu, Srikanth Kandula, Ratul Mahajan, Ming Zhang, and David Gelernter. Traffic engineering with forward fault correction. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, pages 527–538, 2014.

[32] George Marsaglia et al. Xorshift rngs. *Journal of Statistical Software*, 8(14):1–6, 2003.

[33] Masscan. Masscan: Mass ip port scanner. https://github.com/robertdavidgraham/masscan, 2021.

[34] Roland Meier, Petar Tsankov, Vincent Lenders, Laurent Vanbever, and Martin Vechev. Nethide: Secure and practical network topology obfuscation. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 693–709, 2018.

[35] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 15–28, 2017.

[36] Austin Murdock, Frank Li, Paul Bramsen, Zakir Durumeric, and Vern Paxson. Target generation for internet-wide ipv6 scanning. In *Proceedings of the 2017 Internet Measurement Conference*, pages 242–253, 2017.

[37] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 85–98, 2017.

[38] Gabriel Nivasch. Cycle detection using a stack. *Information Processing Letters*, 90(3):135–140, 2004.

[39] NMAP.ORG. Nmap. https://nmap.org/, 2021.

[40] Tian Pan, Nianbing Yu, Chenhao Jia, Jianwen Pi, Liang Xu, Yisong Qiao, Zhiguo Li, Kun Liu, Jie Lu, Jianyuan Lu, et al. Sailfish: accelerating cloud-scale multi-tenant multi-service gateways with programmable switches. In

Proceedings of the 2021 ACM SIGCOMM 2021 Conference, pages 194–206, 2021.

[41] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. Netbricks: Taking the v out of nfv. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 203–216, 2016.

[42] Philipp Richter, Georgios Smaragdakis, David Plonka, and Arthur Berger. Beyond counting: new perspectives on the active ipv4 address space. In *Proceedings of the 2016 Internet Measurement Conference*, pages 135–149, New York, USA, 2016. ACM.

[43] SDXcentral. At&t picks barefoot networks for programmable switches. https://www.sdxcentral.com/articles/news/att-picks-barefoot-networks-programmable-switches/2017/04/, 2021.

[44] SDXcentral. Barefoot scores tofino deals with alibaba, baidu, and tencent. https://www.sdxcentral.com/articles/news/barefoot-scores-tofino-deals-with-alibaba-baidu-and-tencent/2017/05/, 2021.

[45] The ZMap Team. The zmap project. https://zmap.io/, 2021.

[46] Vybint. 15 alarming cyber security facts and stats. https://www.cybintsolutions.com/cyber-security-facts-stats/, 2021.

[47] W3Techs. Usage statistics of ipv6 for websites. https://w3techs.com/technologies/details/ce-ipv6, 2021.

[48] Sophia Yoo and Xiaoqi Chen. Secure keyed hashing on programmable switches. In *Proceedings of the ACM SIGCOMM 2021 Workshop on Secure Programmable network INfrastructure*, pages 16–22, 2021.

[49] Dai Zhang, Yu Zhou, Zhaowei Xi, Yangyang Wang, Mingwei Xu, and Jianping Wu. Hypertester: high-performance network testing driven by programmable switches. *IEEE/ACM Transactions on Networking*, 2021.

[50] Menghao Zhang, Jun Bi, Kai Gao, Yi Qiao, Guanyu Li, Xiao Kong, Zhaogeng Li, and Hongxin Hu. Tripod: Towards a scalable, efficient and resilient cloud gateway. *IEEE Journal on Selected Areas in Communications*, 37(3):570–585, 2019.

[51] Menghao Zhang, Guanyu Li, Shicheng Wang, Chang Liu, Ang Chen, Hongxin Hu, Guofei Gu, Qi Li, Mingwei Xu, and Jianping Wu. Poseidon: Mitigating volumetric ddos attacks with programmable switches. In *the 27th Network and Distributed System Security Symposium (NDSS 2020)*, 2020.