Switches are Scanners Too! A Fast and Scalable In-Network Scanner with Programmable Switches

Guanyu Li
Tsinghua University

Menghao Zhang
Tsinghua & Kuaishou

Cheng Guo
Tsinghua University

Han Bao
Tsinghua University

Mingwe Xu
Tsinghua University

Hongxin Hu
University at Buffalo, SUNY

ABSTRACT

Network scanning has been a standard measurement technique to understand the network's security situations, however, probing a large-scale scanning space with existing network scanners is both difficult and slow. To address this issue, 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 to turn a switch into a practical high-speed network scanner. We conduct preliminary experiments on the open-source prototype of IMap and evaluation results show that IMap can survey all addresses (i.e., 6 Class B Addresses) and all ports of our campus network in 8 minutes, nearly 4 times faster than state-of-the-art network scanners. As an ongoing work, we plan to continuously improve the design and implementation of IMap, and hope IMap can serve as a foundation for designing next-generation terabit network scanners.

ACM Reference Format:
Guanyu Li, Menghao Zhang, Cheng Guo, Han Bao, Mingwe Xu, and Hongxin Hu. 2021. Switches are Scanners Too! A Fast and Scalable In-Network Scanner with Programmable Switches. In *The Twentieth ACM Workshop on Hot Topics in Networks (HotNets '21), November 10–12, 2021, Virtual Event, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3484266.3487368

1 INTRODUCTION

Network scanning is a typical procedure to discover active hosts, ports, and services in the network, which is mainly used by network operators/researchers for security assessment and system maintenance of a network. Enabled by tools such as Nmap [24], ZMap [13] and Masscan [22], network scanning has become a standard measurement technique to understand the security situations of the target network, even the entire Internet. Recent studies have demonstrated that network scanning can help reveal new security vulnerabilities [3, 6, 9], monitor service deployment [2, 12, 16, 26] and shed light on previously opaque distributed systems [15].

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 has been used by 18.7% of all the websites [28]. In addition, 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 complete a comprehensive scanning as *fast* as possible. Otherwise, a large number of security snapshots cannot be captured in time, and numerous security incidents may be missed [27].

However, a closer look into today's network scanners shows that they are far from being fast and scalable because of 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 deployment locations, state-of-the-art network scanners are all allocated 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

HotNets '21, November 10–12, 2021, Virtual Event, United Kingdom

ACM ISBN 978-1-4503-9087-3/21/11…$15.00
https://doi.org/10.1145/3484266.3487368

technology enabler for IMap is the emergence of programmable switches [8], 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 [7]), 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 next-generation high-speed 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 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 and make the source code publicly available [17]. Our preliminary evaluations show that IMap is nearly 4 times faster than state-of-the-art network scanners. We hope these results can inspire further works in designing next-generation terabit network scanners.

2 MOTIVATION AND OBSERVATION

2.1 Limitations of Current 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. 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 [19]. 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 and mobile devices are connecting online [4]. All these require 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 [27]. 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 is limited by two 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 [11], the throughput cannot reach more than 40 Gbps easily [20, 25, 30]. Second, in terms of deployment locations, today's network scanners are all located at the network edge. Scanning from the edge not only is limited by the upstream bandwidth of the end host, but also incurs longer scanning paths and non-negligible bandwidth waste. As a result, even the scanners can scan at higher rate (e.g., 40 Gbps), the scanning results may suffer from low hit rate because of undesirable probe/response packet drops on the end-to-end scanning paths.

2.2 Opportunities

Programmable switches [8] 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 [20]. Specifically, today's latest CPU-based network scanner, Zipper ZMap [1], could only provide a scanning rate at 14.2 Mpps and a scanning throughput at 10 Gbps. In contrast, switching ASICs can easily process a few billion packets per second, which show great potentials to be a terabit network scanner. Other hardware alternatives, such as FPGA and NPU, cannot match the performance of switching ASICs [20], 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 [7], 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 programmable switches, 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 host. Worse yet, scanning from the end host requires the end-to-end scanning path, which inevitably results in the waste of bandwidth resources and the degradation of scanning hit rate. In contrast, switches provide a unique vantage point for network scanning tasks, which is no longer plagued by the aforementioned issues. Core switches usually have huge spare bandwidths (i.e., more than 50% spare bandwidth [10]), which shows substantial potentials for network scanners to tap. This scanning vantage point is particularly valuable for next-generation high-speed network scanners.

3 IMAP DESIGN

3.1 IMap Overview

Deployment Scenario. Our scenario focuses on a network-centric deployment model, where 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 any ethical issues, as pointed out in ZMap [13]. For example, the purposes and the detailed descriptions of IMap scans should be presented clearly; the close coordination with admins of local and neighbour networks is also necessary to reduce ricks. Ideally, IMap should be built on a core network switch, which provides the scanning functionality and the routing services simultaneously. In other words, the IMap switch should first preserve the functionality of packet switching, and also behave as a high-speed network scanner when there are spare bandwidths. Note that the assumption of spare bandwidths in core switches is reasonable, and reports show that the bandwidth occupation ratio for core switches is usually less than 50% [10]. Besides, in-core-network scanners also raise 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.

Workflow. 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 [13] and Masscan [22]. 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 logics. 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 [21].

Module Overview. IMap is a high-speed in-core-network scanner, including a probe packet generation module, which is responsible to generate high-speed probe packets with random address and adaptive rate, and a response packet processing module, which processes the response packets in a correct and efficient manner. In the following part, we will describe the detailed design of these two modules.

3.2 Probe Packet Generation

The switch is designed to be a packet forwarding device, not a packet generation device, thus cannot generate probe packets without ground. Inspired by HyperTester [29], we also leverage the template-based packet generation mechanism to generate high-speed probe packets. First, the switch CPU prepares a set of template packets with initialized headers and injects them into the switching ASICs. After receiving these template packets, the switching ASICs keep looping these packets in the switch pipeline. Each packet experiences three sequential steps in the journey of switch pipeline: an *accelerator* in the ingress pipeline to accelerate the template packets to the 100 Gbps line rate, a *replicator* in the traffic manager to replicate the template packets into several switch ports, and an *editor* in the egress pipeline to modify the headers of replicated template packets into desired probe packets. 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 network conditions.

3.2.1 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, in the high-speed scanning, simply probing IP addresses in numerical order would overwhelm target networks with 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 according 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 [13] requires calculations such as multiplication and modulo, thus is not feasible in the switching ASICs.

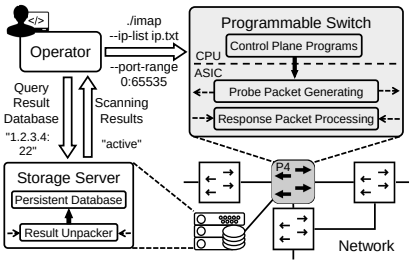

Figure 1: The workflow of IMap.

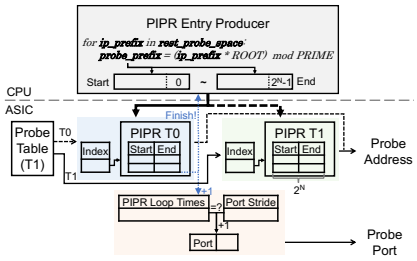

Figure 2: Random probe address.

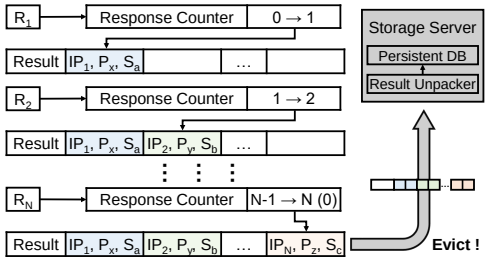

Figure 3: Response packets aggregation.

To address this problem, we leverage the flexibility of the switch CPUs 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 [13], the PIPR Entry Producer module can generate a random permutation of the probe IP ranges for any 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 random destination IP addresses. As the data plane scanning is fast, a PIPR table with the 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 a PIPR_Start array, to store the start of the probe IP range; the other is a 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 to 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 reset to 0 again and another loop starts. When PIPR_Index is looping, 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 entire PIPR table is finished, and the PIPR Entry Producer module should 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 carries a predefined flag to the switch CPUs to notify the PIPR Entry Producer module.

However, conducting a new round of PIPR table filling is a time-consuming task. According to our test on the Tofino switch [18], even with batching optimizations, filling a PIPR table with the size of 65,536 requires about 0.3 seconds. This indicates that, after a round of scanning, we have to wait for more than 0.3 seconds to start the next round of scanning, which is unacceptable for high-speed scanning. 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 switch of the state in the Probe_Table register is triggered by the finish signal of the egress to egress mirror primitive.

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 address of probe packets already has good randomness, we choose to scan ports 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 increase 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 2, which can generate probe packets to cover the scanning space completely, without overwhelming target networks.

3.2.2 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 bandwidth usage of local switch ports and 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 awareness, there are two essential factors that must be carefully considered. First, the IMap control

plane should acquire the bandwidth usages of each port in the local switch accurately. To achieve this, we introduce a counter for each port, which records the accumulated tx/rx traffic of each port. By reading the value of each counter periodically, we can estimate the current bandwidth usage of each port. Being aware of the bandwidth usage in the local switch is not enough. Sometimes even the local switch has sufficient spare bandwidths, the nearby networks may be in a poor condition. This raises our second requirement, the IMap control plane should also be able to estimate the nearby network conditions. We leave the exploration of nearby network condition estimation as our future work.

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 could 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 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 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 a certain value into the rate register to achieve the rate control.

3.3 Response Packet Processing

As an in-network scanner, the input for IMap has both normal packets and response packets. 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.

3.3.1 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 valid to the secret state accordingly. However, the switching ASICs only have limited memory, 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 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. Taking TCP as an 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 works in a similar manner, except for different packet fields. After the verifier checks the validation of the response packets, similar to ZMap [13], 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. As the switching ASICs only support fairly simple cryptography operations (e.g., CRC32), attackers may perform *chosen plaintext 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, we plan to make IMap update 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, $Key1$ 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 packet headers. We leave the detailed exploration of these security issues as our future work.

3.3.2 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 [14, 23, 29]. However, as the key set is really large in IMap (e.g., the size of the scanning address space), even only storing a 2-bit value for each key requires GB-level memory, which exceeds the memory resources of the switching ASICs (i.e., 50-100MB [23]) 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 3, 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 stores this information in one register. When

the register array is filled up, the certain response packet carries 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, 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 a N to 1 aggregation, reducing the pressure for the storage server significantly. On the side of the storage server, we use DPDK [11], a high-performance I/O framework, to parse the result packets and extract the scanning results. Finally, the scanning results are written into a persistent database.

4 PRELIMINARY EVALUATION

Implementation. We implement a prototype of IMap and make our source code publicly available here [17]. 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 PIPR table entry as 256. In the response packet processing module, we set the register array to store results temporarily with a size of 16 (i.e., N=16). The control plane part is written in ~3K lines of C code. It is responsible to initialize the data plane, receive update notifications and update entries/registers in the switch pipeline via Barefoot Runtime. To improve the efficiency of PIPR table updating, we compute all PIPR entries in advance and adopt batching APIs of Barefoot Runtime to install entries. Besides, the backend agent running on the storage server is implemented with DPDK [11], which extracts the scanning results from the aggregated response packets and writes the results into a Redis database.

Experimental setup. Our testbed consists of one 3.3 Tbps Intel Tofino switch and two Dell R730 servers. Both servers are equipped with Intel(R) Xeon(R) E5-2620 v3 CPUs and 64 GB memory, connected to the switch via 40 Gbps 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, where we can collect and analyze probe packets generated from IMap. The scanning target is configured to all ports (0-65535) of our campus network including 6 Class B IP addresses, a total of up to 25 billion scanning space. The scanning rate is set to 55 Mpps.

Performance. First, we measure the scanning throughput to demonstrate the scalability and fastness of IMap. As shown in Figure 4, with only one switch port enabled, IMap is able to generate probe packets at 40 Gbps, which is a 4 times improvement compared to Zippier ZMap [1, 31]. Besides,

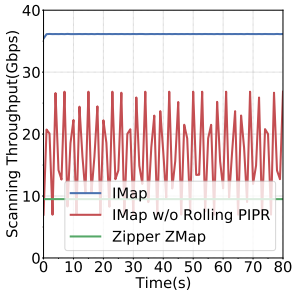

Figure 4: Scanning throughput.

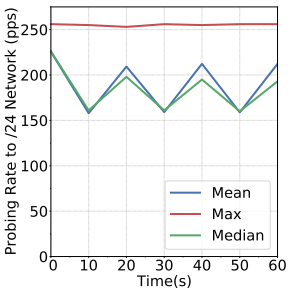

Figure 5: Probe packet pressure.

from this figure, we can also see the rolling PIPR filling optimization (§3.2) helps IMap achieve high-speed scanning continuously. With such high-performance scanning, our campus network can be fast surveyed within 8 minutes. Note that 40 Gbps is not the upper limit of IMap; instead, when we enable all ports in the IMap switch, IMap can generate probe packets at terabit line rate, potentially achieving two orders of magnitude improvement than the state of the art. Currently, we do not have such a deployment location and leave such pressure test as our future work. Second, we analyze the pressure IMap brings to edge networks to validate the effectiveness of our random probing technique. Figure 5 shows that several /24 networks in the scanning space only receive hundreds of probe packets per second even though the scanning throughput of IMap reaches as high as 40 Gbps. Such pressure brought by scanning traffic is negligible for most edge networks.

5 CONCLUSION AND FUTURE WORK

In this paper, we identify the limitations of current network scanners, and sketch the design of IMap, a fast and scalable in-network scanner with programmable switches. We conduct preliminary evaluations on the open-source prototype of IMap, and verify its functional correctness and advantages. In our ongoing work, we are planning to improve the design and implementation of the IMap prototype, deploy IMap in a large ISP, and conduct extensive experiments and measurements. We hope IMap can be the cornerstone of next-generation high-speed network scanners.

ACKNOWLEDGMENTS

We thank our shepherd, Srinivas Narayana, and anonymous HotNets reviewers for their valuable comments. This work is supported in part by the National Key R&D Program of China under Grant 2019YFB1802504, the National Natural Science Foundation of China under Grant 92038302, 61625203 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. 2014. Zippier zmap: internet-wide scanning at 10 gbps. In *8th {USENIX} Workshop on Offensive Technologies ({WOOT} 14)*. USENIX, San Diego, CA.

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

[3] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J Alex Halderman, Viktor Dukhovni, et al. 2016. {DROWN}: Breaking {TLS} Using SSLv2. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. USENIX, Austin, TX, 689–706.

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

[5] D. J. Bernstein. 2021. 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. 2015. A messy state of the union: Taming the composite state machines of TLS. In *2015 IEEE Symposium on Security and Privacy*. IEEE, IEEE, San Jose, CA, USA, 535–552.

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

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

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

[10] Cisco. 2021. 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).

[11] Intel DPDK. 2021. Learn How To Get Involved With DPDK. https://www.dpdk.org/. (2021).

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

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

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

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

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

[17] IMapScanner. 2021. IMap. https://github.com/IMapScanner/IMap.git. (2021).

[18] Intel. 2021. 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).

[19] Google Ipv6. 2021. IPv6 Adoption. https://www.google.com/intl/en/ipv6/statistics.html. (2021).

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

[21] Redis Labs. 2021. Redis. https://redis.io/. (2021).

[22] Masscan. 2021. Masscan: Mass IP port scanner. https://github.com/robertdavidgraham/masscan. (2021).

[23] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. 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*. 15–28.

[24] NMAP.ORG. 2021. Nmap. https://nmap.org/. (2021).

[25] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. 2016. NetBricks: Taking the V out of {NFV}. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 203–216.

[26] Philipp Richter, Georgios Smaragdakis, David Plonka, and Arthur Berger. 2016. Beyond counting: new perspectives on the active IPv4 address space. In *Proceedings of the 2016 Internet Measurement Conference*. ACM, New York, USA, 135–149.

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

[28] W3Techs. 2021. Usage statistics of IPv6 for websites. https://w3techs.com/technologies/details/ce-ipv6. (2021).

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

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

[31] ZMap. 2021. ZMap: The Internet Scanner. https://github.com/zmap/zmap. (2021).