A Scalable Network Port Scan Detection System on FPGA

Tejasvi Anand, Yagnesh Waghela and Kuruvilla Varghese
Centre for Electronics Design and Technology (CEDT), Indian Institute of Science
Bangalore, India, 560012
Email: edkuru@cedt.iisc.ernet.in

***Abstract*—With ever increasing network speed, scalable and reliable detection of network port scans has become a major challenge. In this paper, we present a scalable and flexible architecture and a novel algorithm, to detect and block port scans in real time. The proposed architecture detects fast scanners as well as stealth scanners having large inter-probe periods. FPGA implementation of the proposed system gives an average throughput of 2 Gbps with a system clock frequency of 100 MHz on Xilinx Virtex-II Pro FPGA. Experimental results on real network trace show the effectiveness of the proposed system in detecting and blocking network scans with very low false positives and false negatives.**

I. Introduction

Internet started as an interconnection of trusted hosts, but as the network grew, malicious users also became a part of it. These malicious users, hackers and computer worms, try to compromise remote hosts by exploiting the loopholes present in the services running on them. Port scanning, a reconnaissance operation, is performed to identify such vulnerable services. Based on their methods of probing, port scans can be categorized as vertical port scan, horizontal port scan and stealth scan [1], [2]. According to the last available US-CERT's quarterly report [3], among the reported incidents, the total number of scan probes and attempted access in FY09 Q1 is 73%. Increasing bandwidth, rise in the number of network users, and the advent of new services have made matters worse. Hence, there is a need to reliably detect network scans without affecting the quality of service.

We can divide the ongoing research effort to detect network scans into two broad categories. They are anomaly based detection and signature or rule based detection. In anomaly based detection systems [1],[4]-[7], a system is first trained with the normal network behavior, and when deployed, any deviation from the learned behavior is marked as an anomaly. A clear advantage of anomaly detection systems is in detecting new attacks [8] and detecting the modifications of the old ones. However, a big challenge in these systems is to accommodate the long term changes in the network traffic behavior.

In the domain of rule based scan detection systems, few popular systems exist [9],[10]. Because of the rule based approach, these systems are not very effective in detecting attacks that spans over multiple packets such as network scans and denial of service attacks. Another major limitation of these algorithms is that they store all the states of the connections passing through them, making them non-scalable with increasing network speed. A middle approach between rule based and anomaly based techniques is described in [11],[13]-[15]. These techniques look for abnormal behavior with preconfigured threshold limits. However, these techniques need to store the state of all connections which makes them non-scalable. Using similar approach, [16],[18] have proposed a scalable architecture which can detect port scans and few other network attacks. The work in [16] uses Partial Configurable Filters (PCF) to detect port scans by analyzing imbalance in TCP connections; any source sending packets with more SYN flags than FIN flags is detected as a scanner. These architectures, however efficient, can only detect limited attacks which have some imbalance in the flow of control packets as in Distributed Denial of Service (DDoS) and port scans. The issue of detecting stealth scans remains unaddressed.

In this paper, we propose an algorithm and a scalable architecture to detect network scans in an aggregated fashion. Our algorithm detects and mitigates port scans, based on unique characteristics exhibited by the scanners. It also detects stealth scans with a wide range of inter-probe period. The flexible architecture makes it a suitable candidate for designing a scalable Network Intrusion Detection System (NIDS). Architectural decisions to reduce area and to increase the throughput are also discussed in this paper. The proposed Scan Detection System is implemented on Xilinx Virtex-II Pro FPGA (XC2VP30). Results from test done with captured real trace is used to validate the efficacy of the proposed system. The implementation of the proposed architecture on FPGA targets TCP scans, but UDP and ICMP scans can also be detected with a similar approach.

The rest of the paper is organized as follows. Overview of the architecture is presented in Section II. Section III discusses characteristics of a typical scanner and the proposed scan detection algorithm. Details of FPGA implementation is presented in Section IV. Section V describes salient features and design decisions of the proposed system. Results obtained from the real trace is presented in Section VI. Section VII concludes the paper.

II. Architecture Overview

A conceptual view of the complete Network Intrusion Detection System and block diagram of the proposed Scan

978-1-4577-1740-6/11/$26.00

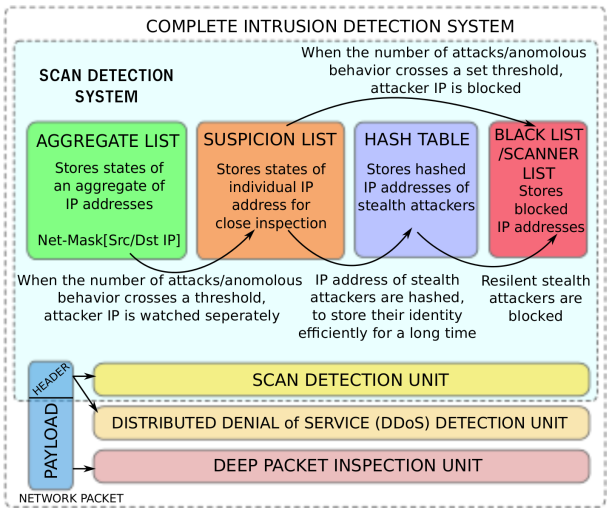

Fig. 1. A conceptual view of Network Intrusion Detection System

Detection System's architecture are shown in Fig. 1. *Aggregate List*, *Suspicion List*, *Hash Table*, and *Black List* or *Scanner List* form the basic building blocks and store all the required information. This information is processed by various attachable units, to perform desired operation. In this paper only the Scan Detection System (SDS) is discussed. However, the aim of showing NIDS architecture is to emphasize the fact that with minimal modifications to various lists, Deep Packet Inspection and Distributed Denial of Service Attack Detection units could be easily added to develop a complete network security solution.

Based on programmable subnet masks, as in IP routing tables, *Aggregate List* stores information for a group of IP addresses. This group of IP addresses are monitored as a whole for any malicious activities. Malicious activities such as port scans, Denial of Service (DoS), worms and virus attacks spans over multiple packets. Depending on the application programs running on a network client, benign IP addresses could also sometimes show such malicious behavior. Because of this reason, a group of IP addresses is watched for the occurrence of attack or malicious packets. Once the number of such packets crosses a pre-configured threshold, the IP address in the subnet, which is responsible for the last attack packet, is tagged as suspicious. To avoid spoofing which may happen when an attacker IP address deliberately makes benign IP address to appear as scanner IP address to Scan Detection System, aggregate of destination IP addresses are also stored in the *Aggregate List*. This list helps in ensuring that the reply to request packets originate from legitimate IP addresses.

Once the suspicious IP address is identified, its activity is monitored separately in the *Suspicion List*. Size of the *Suspicion List* increases linearly with the number of connections made by suspicious IPs. After confirming the suspicious IP address as an attacker or a scanner, it is added to the *Black List* or *Scanner List*. Any packet originating from and going to IP addresses in the *Black List* are dropped. *Hash Table* keeps track of the stealth scanners. This strategy of first identifying suspicious IPs and check them individually achieves a very small false positive and false negative, and hence the quality of service does not degrade. Ability to store minimal information while identifying suspicious IP addresses makes this architecture highly scalable.

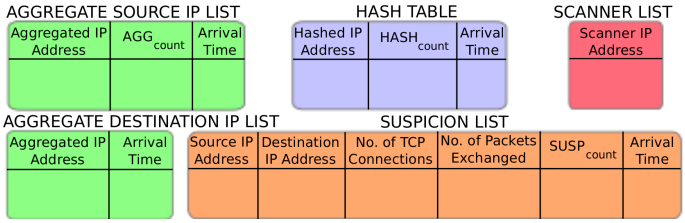

Fig. 2. Contents of various hardware lists

III. Properties of a Scanner and Scan Detection Algorithm

Scanners, unlike benign users, exhibit abnormal behavior. This section discusses such behavior and the proposed algorithm which helps in detecting network scans.

A. Properties of a Scanner IP Addresses

Following are the characteristics exhibited by scanners:

1. If a scanner IP address sends a SYN packet to a closed port on a remote host, as per TCP-IP protocol, the remote host replies by a packet with RST+ACK flags set. Since, a scanner attempting a scan, has only an a priori knowledge of open ports on a remote host, it receives a lot of TCP packets with RST+ACK flags set.

2. To perform OS finger printing on the remote host, scanner IP address sends packets with invalid flags set in the TCP packet header [17].

3. While performing TCP-SYN scan, scanners leave the TCP connection half open [12],[19], resulting in incomplete three way handshake. In this case, number of packets exchanged between scanner IP address and victim IP address is two. Sometimes scanner IP addresses tear the half open connection by a RST packet, making number of packets exchanged equal to three.

4. Scanner IP address initiates a connection and completes three way handshake, but does not transfer any data. This is the most common "claim and hold" attack known as Naptha. Though this is a class of Denial of Service (DoS) attack, port scanning could also be performed this way. In this case, the number of packets exchanged between the scanner and victim are three. Some scanners even close the connection with a normal FIN packet and receive FIN+ACK in response, making the number of packets exchanged equal to five.

If the remote host has an inbuilt protection, then for any SYN request on its closed port, it may choose to remain silent or occasionally reply with a packet with RST+ACK flags set. This approach could cause an imbalance in the number of SYN request and RST or FIN reply packets in the network [16]. In our algorithm, this imbalance is not used to detect

scans. However, the proposed architecture has the flexibility to accommodate these behavioral changes in detecting scans.

B. Algorithm

The contents of *Aggregate List*, *Suspicion List* and *Hash Table* are shown in Fig. 2.

The algorithm is as follows:

1. Source and destination IP address of every incoming packet is checked across various lists. Action on this incoming packet is decided depending on the list in which this IP is found.

2. If the source/destination IP address is found in the *Scanner List*, the packet is dropped.

3. Next, *Hash Table* is checked for the match. If hash of this IP exists, *Hash Table* is updated and an entry of this IP is made in the *Suspicion List*.

4. If the IP is found in the *Suspicion List*, then based on the packet's TCP header, the *Suspicion List* is updated. Similarly, *Aggregate List* is updated if the packet's IP is found in the *Aggregate List*.

5. If, however, the IP is not found in any of the lists, then based on the subnet mask, a new entry is made in the *Aggregate List*.

6. In the *Aggregate List*, if an IP address shows first two properties of the scanner (receives RST+ACK or send invalid flags), AGG_{count} of the corresponding subnet is incremented. When AGG_{count} reaches a configurable threshold limit AGG_{TH}, the last IP address which caused this increment is added to the *Suspicion List*.

7. In the *Suspicion List*, the suspicious IP address is checked for all four characteristics of the scanner, and $SUSP_{count}$, associated with that IP address, is incremented accordingly. When $SUSP_{count}$ reaches a configurable threshold limit $SUSP_{TH}$, the suspicious IP address is added to the *Scanner List*.

8. Entries in the *Aggregate List*, *Suspicion List* and *Hash Table* which are inactive for a configurable time interval are timed-out and purged from the list. This operation helps in the controlling the size of these lists.

9. Suspicious IP address which are timed-out from the *Suspicion List* are hashed in the *Hash Table*. Their corresponding $SUSP_{count}$ is added to $HASH_{count}$. When $HASH_{count}$ reaches a configurable threshold limit $HASH_{TH}$, the last IP address hashed is tagged as a scanner, and is added to the *Scanner List*. This helps in detecting stealth scanners. Section V discusses stealth scan detection in more detail.

IV. FPGA Implementation of the Scan Detection System

The Scan Detection System is implemented on Xilinx Virtex-II Pro XC2VP30 FPGA as an embedded system, as shown in Fig. 3. The Scan Detection System Core is attached to the On-Chip Peripheral Bus (OPB) as a peripheral. A hard-core 32 bit PowerPC processor is used to initialize Ethernet Media Access Controllers (EMAC) and configure the Scan Detection System Core, by writing configuration inputs such as threshold values. Two Xilinx soft-core EMACs attached to

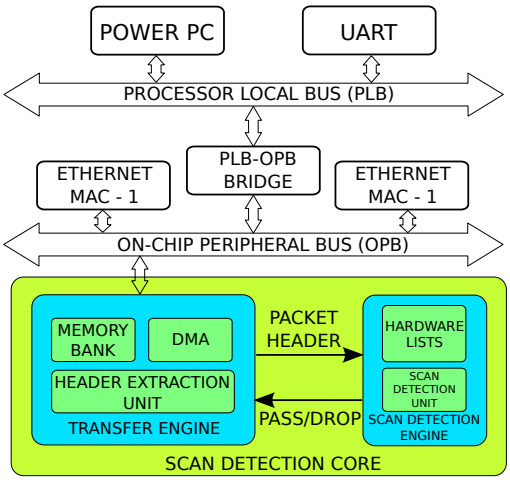

Fig. 3. Block diagram of hardware implementation of the Scan Detection System on Xilinx Virtex-II Pro

OPB, perform the Ethernet MAC operations. Once, a complete Ethernet frame is available in one of the MAC's FIFOs, Direct Memory Access (DMA) unit, in Transfer Engine, transfers the complete frame to one of the empty memory banks. These memory banks are implemented using FPGA's Block RAM (BRAM) modules.

Header extraction logic copies the header and stores it in the header extraction registers and signals the Scan Detection Unit to start the operation. Header Extraction Unit and Memory Banks are part of the Transfer Engine. *Aggregate List*, *Suspicion List*, *Hash Table*, *Scanner List* and the Scan Detection Unit are part of the Scan Detection Engine. Depending on the decision of the Scan Detection Unit, the buffered frame is either copied from the memory bank to the transmitting FIFO of MAC, or it is dropped.

V. Architecture and Features of the Scan Detection System

This section elaborates detailed architecture, the key features of the proposed Scan Detection System and discusses architectural design decisions.

A. Small Area

Various lists shown in Fig. 2 is implemented using Content Addressable Memory (CAM) for fast search operation. CAM is a special type of memory in which data is given as an input and address of the locations where input data matches the stored data is given as output. CAM performs parallel search operation across all the locations and produces result within one or two clock cycles. It is implemented using Configurable Logic Blocks (CLBs) of FPGA. When CAM is compared with the dedicated BRAM modules available on FPGA, it is relatively costly because of two reasons. Firstly, when the width of CAM increases, it consumes a large number of CLBs which otherwise could have been used for implementing logic circuits. Secondly, when a large number of CLBs are used to implement CAM, because of large interconnect delay,

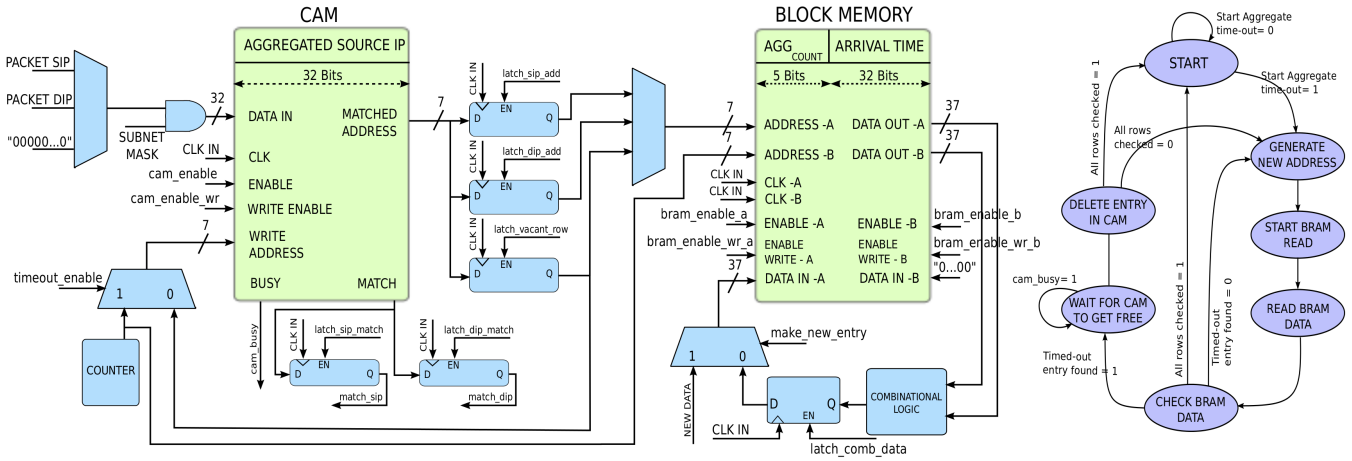

Fig. 4. Hardware implementation of the *Aggregate List* and *Time-out-purge* state machine

time required for read and write operations increases. On the other hand, depending on the depth, search operation in BRAM modules requires multiple clock cycles. To address this problem, CAMs were designed to store only IP addresses, and these IP addresses act as a pointer to the BRAM where the rest of the information is stored. This approach requires few extra clock cycles but it saves precious CLBs. Fig. 4 shows the detailed implementation along with the essential control signals. Source IP and Destination IP addresses of the incoming packet is input to the CAM through *DATA IN* port for search and write operation. Matched address is given to the BRAM through *ADDRESS -A* port. This is used to update Arrival time, AGG_{count} or to make a new entry in the BRAM. Data path from *DATA OUT-A* to *DATA IN -A* forms the update logic. Counter is used to access all the entries in the BRAM to check for the time-out condition and purge the data from CAM and BRAM.

B. High Throughput

Storing limited data in CAMs also helps in reducing the read and write cycle time resulting in high system clock frequency. For a given frequency, the number of cycles required to process a packet's header decides the throughput of the system. Therefore, an efficient control logic is desirable to perform this processing in limited cycles. As a common practice in designing complex systems, signal exchange among state machines at the same level of hierarchy is avoided. However, In the proposed system, to save few clocks cycles, signals were exchanged between state machines at same and at different hierarchy.

Every entry in the *Aggregate List*, *Suspicion List* and *Hash Table* have time-out counters, and time-out condition of all the entries are checked periodically. The duration when this check is triggered is a configurable parameter for all the three lists. When the time-out condition of a particular entry in a list is being checked, read or write operation on that list, for incoming packets, cannot be performed. The number of cycles required by the time-out state machine depends on the depth of the list, and is relatively large as compared to the number of cycles required to process a packet. Stalling the normal packet processing operation during time-out operation affects the system's throughput. To overcome this problem, the time-out control logic, associated with every list in the architecture, is designed to operate in the cycle stealing mode. In this mode, whenever a normal operation is not being performed in a particular clock cycle, the time-out control logic kicks-in and checks every entry for time-out condition and updates or purges it accordingly. Read and write operation for purging an entry in the BRAM is done by *Time-out-purge* state machine, shown in Fig. 4, on port-B and hence does not require any synchronization with other operations. Deletion in CAM and BRAM is synchronized using the busy signal of the CAM - *cam_busy*. Hence, apart from storing limited data in CAMs which reduced read and write cycle time, bypassing control hierarchy and cycle stealing operation increase the overall system's throughput.

C. Detection of Stealth Scanners

In order to avoid detection, stealth scanners scan the network with a very large inter-probe period. Assuming the stealth scanner is present with other active benign IPs in the subnet, active benign IPs make sure that the arrival time in the subnet entry in the *Aggregate List* is updated on a regular basis. As a result the subnet entry will not be purged and AGG_{count} will cross AGG_{TH} due to the scanner's behavior. Consequently, the stealth scanner present in the *Aggregate List* will be tagged as suspicious.

In the *Suspicion List*, if the inter-probe period of scan is large, stealth scanner's entry is timed-out and the scanner IP address is purged from the *Suspicion List* and it is hashed to the *Hash Table*. The $SUSP_{count}$ associated with that IP address is added to the corresponding $HASH_{count}$. Next time, if the scanner IP address sends a scan probe with invalid flags set, or receives a RST+ACK packet, then it is directly

TABLE I
XC2VP30-FPGA RESOURCE UTILIZATION

Logic Utilization	Used	Available	Utilization
No. Slices	10227	13696	74%
No. Slice Flip Flops	8347	27392	30%
No. of 4 input LUTs	15213	27392	55%
No. BRAMs	134	136	98%

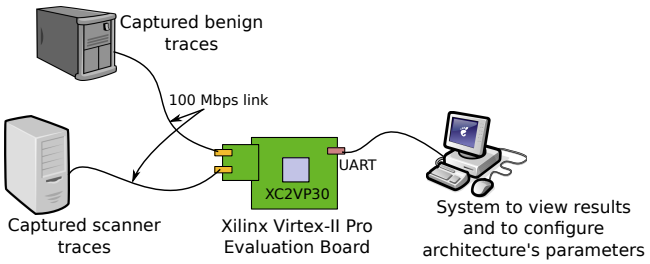

Fig. 5. Test setup to replay captured packets and evaluate the Scan Detection System

added to the *Suspicion List* and the corresponding $HASH_{count}$ is incremented by one. If the above mentioned action is repeated, the $HASH_{count}$ associated with the stealth scanner's IP address will eventually cross the configurable threshold and the scanner IP address will be added to the *Scanner List*. The Time-out period of the *Hash Table* is large compared to the *Aggregate List* and the *Suspicion List*. This ensures that the hashed entry of the stealth scanner is not purged even if it remains inactive for a long duration of time. With the appropriate values of thresholds and time-out periods, scanners with arbitrarily long inter probe period could be detected.

D. Scalability and Flexibility

In the proposed architecture, scalability is achieved by keeping information of a group of IP addresses in a single row of the *Aggregate List*. This group of IP addresses are grouped together through a configurable subnet mask which need not be contiguous. With this flexibility, the *Aggregate List* can be populated in a very controlled way. For example, a group of untrusted users could be grouped together in a separate group. With little modification, a new column can be added to the *Aggregate List* which stores the AGG_{TH} value for the corresponding row. This means that for the i^{th} row, AGG^i_{count} must cross the AGG^i_{TH} for the scanner IP to be added in the *Suspicious List*. By keeping a low value of AGG_{TH} for rows having untrusted hosts, fast scanner detection could be achieved.

Scan detection rate can also be traded with capacity of the *Aggregate List*. For example, if the subnet mask is changed to accommodate more IP addresses in each subnet of the *Aggregate List*, the possibility of finding more than one scanner IP address in that particular subnet will also increase. As a result, AGG_{count} of that particular subnet must build up above AGG_{TH} for two or more times so as to add scanners to the *Suspicion List*. This repeated build up of AGG_{count} could lead to a possible delay in detecting a scanner.

VI. TEST SETUP AND RESULTS

FPGA implementation of Scan Detection Core achieved an average throughput of 2 Gbps with a system clock of 100 MHz. This calculation is made by observing the average cycles required by the Scan Detection Core to process a packet. Throughput of the entire system, limited by the slow On-Chip Peripheral Bus (OPB) and Soft EMAC cores, is 100 Mbps. OPB and soft EMAC were chosen because 100 Mbps speed is sufficient to test the system on our network as a proof of concept. However, this speed limitation can easily be overcome by using Gigabit EMAC cores and a dedicated bus. Table I shows the resource utilization of Xilinx Virtex-II Pro XC2VP30 FPGA having 128 rows each in *Aggregate List* and *Suspicion List*, 256 rows in *Hash Table* and 64 rows in *Scanner List*. BRAM usage of 98% also includes the memory of Power PC.

Performance of system was evaluated by testing it against live network traces. We collected 30 minutes trace from one network gateway of our Institute using Tcpdump [20]. This benign trace, with a diverse range of traffic, consists of more than 200 active benign IP addresses having more than 3000 TCP flows. This trace was then engineered by assigning new source IP and destination IP addresses to every TCP flow. This modification was done to make the captured benign trace to look like 3000 active benign IP addresses. An alternative way could be to collect 10 such 30 minute traces and then different IP addresses could be assigend to each of these 30 minute traces to get 2000 different active benign IP addresses.

A trace of 38 scanners scanning hosts inside the institute network was captured separately by running network scan tool NMAP [19] with default vertical SYN scan option. The number 38 does not carry any special significance, and was chosen to simulate a network condition with more than 1% scanners. Captured traces, of both benign users and scanners, help in ensuring similar network condition while testing the Scan Detection System with different configurable parameters. Subnet mask in the *Aggregate List* is chosen to divide all the benign IP addresses, passing through Scan Detection System, into 64 subnets. Scanner IP addresses are assigned randomly to these subnets.

A. Evaluation for Normal Scans

To study the effect of configurable parameters on the accuracy of detection i.e. false positives and false negatives, captured traces were replayed using Tcpreplay [21] with different set of configurable parameters in a test setup, as shown in Fig. 5. Fig. 6 shows the results obtained with different set of AGG_{TH} and $SUSP_{TH}$. For these set of tests, $HASH_{TH}$ was set to 8. Results show that the false positive decreases rapidly as AGG_{TH} and $SUSP_{TH}$ are increased. From the results we can also observe that there is no false positive when both AGG_{TH} and $SUSP_{TH}$ are set to a high value. During the complete duration of testing, all scanner IP addresses were detected by

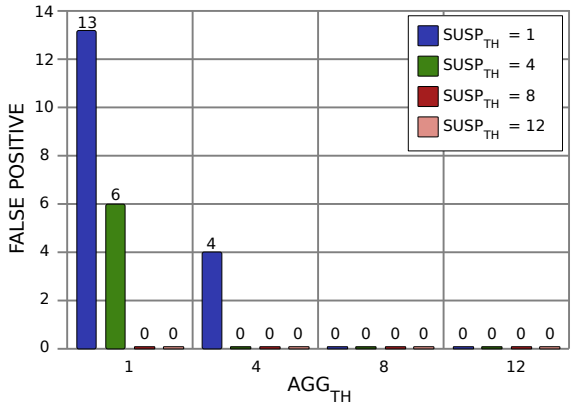

Fig. 6. Effect of AGG_{TH} and $SUSP_{TH}$ on the number of False Positives.

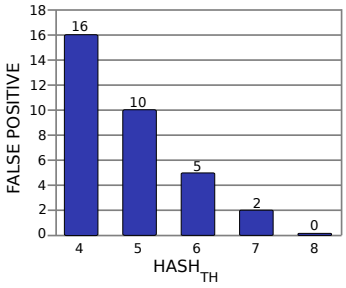

Fig. 7. Effect of $HASH_{TH}$ on the number of False Positives

the system and their traffic was blocked before they could complete their respective scans.

B. Evaluation for Stealth Scans

To test the effectiveness of the system in detecting stealth scans and to study the impact of $HASH_{TH}$, a trace consisting of more than 500 benign users, generated from the captured 3000 TCP flows, and 20 stealth scanners was given to the Scan Detection System. The trace of benign IPs was engineered such that every benign IP either receive a RST+ACK packet or have an invalid flag set in one of its packet's TCP header. This manipulation of benign IP flow was done to ensure that few benign IPs land in the *Suspicion List* and eventually in the *Hash Table*. This helps in studying the impact of $HASH_{TH}$ on the number of false positive. For this experiment, AGG_{TH} and $SUSP_{TH}$ values were kept at 5. Fig. 7 shows the observed false positives as the $HASH_{TH}$ is increased. False negative in this experiment was zero, i.e. all stealth scanner IP addresses were detected. From these results it can be observed that as the configurable threshold values are increased, the false positive decreases.

VII. Conclusion and Future Work

This paper introduced a flexible, scalable architecture and a new algorithm to detect and block network port scans. Scalability is achieved by storing information of a subnet rather than keeping states of individual flows. Accuracy is achieved by identifying scanner IP address from its unique network characteristics and separately analyzing suspicious IP addresses. Ability to detect stealth scans is achieved by hashing suspicious IP addresses with long inter-probe period. FPGA implementation of the proposed architecture has achieved an average throughput of 2 Gbps with 100 MHz system clock. We obtained satisfactory results during tests with real network traces. An appropriate follow-up to the work presented here is to perform rigorous analytical investigation into the set of configurable parameters, to achieve optimal performance.

Acknowledgment

The authors would like to thank Xilinx, Inc. for their support.

References

[1] S. Staniford, J. A. Hoagland, and J. M. McAlerney, "Practical automated detection of stealthy portscans," *Journal of Computer Security*, vol. 10, no. (1-2), 2002.
[2] Vinod Yegneswaran *et al.*, "Internet intrusions: Global characteristics and prevalence," In Proceedings of ACM SIGMETRICS, 2003.
[3] US-CERT. [Online]. Available: http://www.us-cert.gov/press_room/trendsanalysisQ109.pdf
[4] J.E. Dickerson *et al.*, "Fuzzy intrusion detection," in *Proc. IFSA World Congress and 20th NAFIPS International Conference*, Vancouver, BC , Canada, pp. 1506-1510, Jul 2001.
[5] W. El-Hajj *et al.*, "On Detecting Port Scanning using Fuzzy Based Intrusion Detection System," in *Proc. International Wireless Communications and Mobile Computing Conference*, Crete Island, pp. 105-110, Aug 2008.
[6] C. Gates *et al.*, "Scan Detection on Very Large Networks Using Logistic Regression Modeling," in *Proc. 11th IEEE Symposium on Computers and Communications (ISCC-06)*, pp. 402-408, Jun 2006.
[7] C. Leckie and R. Kotagiri, "A probabilistic approach to detecting network scans," in *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS-02)*, pp. 359-372, Aug 2002.
[8] A. Das, D. Nguyen, J. Zambreno, G. Memik, and A. Choudhary, "An FPGA-Based Network Intrusion Detection Architecture," *IEEE Transactions on Information Forensics and Security*, vol. 3, no. 1, pp. 118-132, Mar. 2008.
[9] L.T. Heberlein *et al.*, "A network security monitor," in *Proc. IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, USA, pp. 296-304, May 1990.
[10] SNORT. [Online]. Available: http://www.snort.org/
[11] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23-24, pp. 2435-2463, 1999.
[12] K. Shah *et al.*, "Feasibility of detecting TCP-SYN scanning at a backbone router," in *Proc. American Control Conference*, vol. 2, pp. 988-995, Jun 2004.
[13] Jung Jaeyeon *et al.*, "Fast portscan detection using sequential hypothesis testing," in *Proc. IEEE Symposium on Security and Privacy*, pp. 211-225, May 2004.
[14] Zhang Yu and Fang Binxing, "A Novel Approach to Scan Detection on the Backbone," in *Proc. International Conference on Information Technology: New Generations*, Las Vegas, Apr 2009.
[15] Avinash Sridharan, T. Ye, and Supratik Bhattacharyya, "Connectionless port scan detection on the backbone," in *Proc. 25th IEEE International Performance, Computing, and Communications Conference (IPCC-06)*, Phoenix, Apr 2006.
[16] R. R. Kompella, S. Singh, and G. Varghese, "On Scalable Attack Detection in the Network," *IEEE/ACM Trans. Neworking*, vol. 15, no. 1, pp. 14-25, Feb. 2007.
[17] NMAP - Remote OS Detection, [Online]. Available: http://nmap.org/book/osdetect.html
[18] Shijin Kong, Tao He, Xiaoxin Shao, Changqing An, and Xing Li, "Scalable Double Filter Structure for Port Scan Detection," in *Proc. International Conference on Communications (ICC-06)*, Instanbul, pp. 2177, June 2006.
[19] The art of ports scanning, [Online]. Available: http://nmap.org/nmap_doc.html
[20] Tcpdump, [Online]. Available: http://www.tcpdump.org/
[21] Tcpreplay, [Online]. Available: http://tcpreplay.synfin.net/