Implementation Results of Bloom Filters for String Matching

Michael Attig, Sarang Dharmapurikar, and John Lockwood *
Washington University
Applied Research Laboratory
One Brookings Drive, Campus Box 1045
St. Louis, MO 63130-4899 USA
{mea1, sarang, lockwood}@arl.wustl.edu
http://www.arl.wustl.edu/projects/fpx/reconfig.htm

Abstract

Network Intrusion Detection and Prevention Systems (IDPS) use string matching to scan Internet packets for malicious content. Bloom filters offer a mechanism to search for a large number of strings efficiently and concurrently when implemented with Field Programmable Gate Array (FPGA) technology. A string matching circuit has been implemented within the FPX platform using Bloom filters. Using 155 block RAMs on a single Xilinx VirtexE 2000 FPGA, the circuit scans for 35,475 unique signatures.

1 Architecture

By using Bloom filters [1], an IDPS can be implemented that scans for tens of thousands of strings at Gigabit per second rates, all within a single FPGA. We have built a system with Bloom filters that scans Internet traffic [2]. An overview of our string matching architecture is shown in Figure 1. Packets enter the system and are processed by Internet Protocol (IP) wrappers. The data in the packet goes to the input buffer and then flows through the content pipeline. As the packet passes through the pipeline, multiple Bloom engines scan different window lengths for signatures of different lengths. Data leaves the content pipeline, flows to the output buffer, streams through the wrappers, and then packets are re-injected into the network. If a Bloom engine detects a match, a hash table is queried to determine if an exact match occurred. If the queried signature is an exact match, the malicious content can be blocked and an alert message is generated within an User Datagram Protocol (UDP) packet, informing a network administrator, an end user, or an automated process that a matching signature has been detected.

Figure 1. An overview of the Bloom filter architecture.

1.1 Bloom Engines

Each Bloom engine searches for a specified signature length. To search for multiple signature widths, multiple Bloom engines are instantiated that each operate in parallel. In one circuit, we have implemented engines to search for all signatures between 2 bytes and 26 bytes. This range al-

---

lows the circuit to scan for most of the unique strings found in the current Snort rules database [3].

Each signature window is input to the appropriate Bloom filter engine to check for membership against currently programmed signatures. Membership is checked by computing k hashes over the input signature. The result of these hash computations are used to check a bit-vector. A '1' in the hashed location indicates possible membership, and a '0' indicates non-membership. A Bloom filter signals a potential match only when all of the k hash functions indicate membership. Using multiple hash functions greatly reduces the rate of false positives reported by the first stage of the scanner.

In the second stage of the scanner, potential signature matches are sent from the various Bloom engines to an arbiter. The arbiter buffers query requests in the event that multiple queries are requested simultaneously or while the hash table is busy. We have tuned our Bloom filters so that each contains a 16,384 bit-vector. This reduces each Bloom engine's rate of generating false positives to $f = 0.0039$ and allows the optimal number of signatures that can be stored to be $n = 1419$. Together, all 25 of the Bloom filters (one for each of the lengths between 2 and 26 bytes) can scan for 35,475 signatures.

1.2 Hash Table

The hash table in the second stage of the scanner eliminates false positives generated from the Bloom filters. The hash table fetches a string record from SDRAM by computing a hash over the signature to query. After retrieving the record from memory, an exact comparison to the query string is performed. If the signature does not match the first record in SDRAM and if there was a first record, the next hash record in the chain is retrieved and checked.

While querying the hash table, the byte pipeline is paused. Assuming no hash collisions, it takes 20 clock cycles to determine whether an exact match has occurred.

2 Results

One system we implemented processes one byte per clock cycle. The circuit operates at 62.8 MHz, which provides a maximum throughput of 502 Mbps. This design uses 85% of the logic slices, 54% of the look-up tables, and 96% of the Block RAMs on the Xilinx VirtexE 2000 FPGA. A system with multiple engines is in the works that processes 32 bits per clock cycle to provide a throughput of over 2 Gbps. Because this system uses duplicate Bloom filters, the number of signatures that can be scanned for is reduced.

3 Test Environment

We have tested our system with live traffic using the configuration of Figure 2. The following steps were taken in order to configure the system for string matching. First, the software controller programmed the bitfile over the network into the FPX. Next, a software controller generated User Datagram Protocol (UDP) packets that specify which signatures to scan. Our system allows new signatures to be programmed into the system in only 30 clock cycles per signature. Two control packets are created per signature. The first is used for setting bits in the Bloom filter, and the second is used for adding the signature to the hash table. Loading 35,000 unique signatures takes 17 ms.
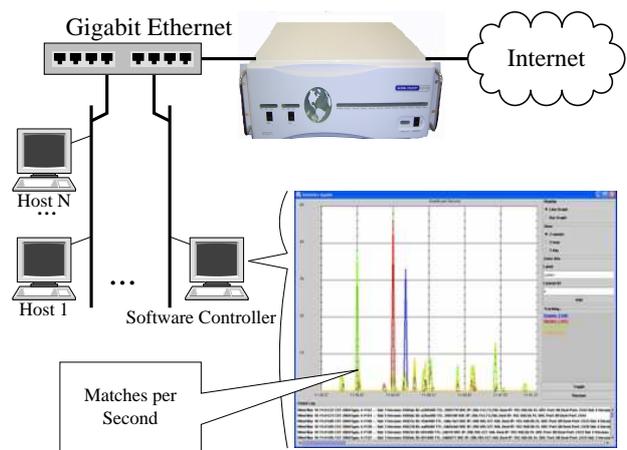

Figure 2. Configuration of our string matching system.

Internet traffic carried over Gigabit Ethernet is scanned as it passes through the chassis. If a signature is found anywhere in the packet, an UDP alert message is sent back to the software controller, where it displays the number of string matches per second and details about each event. An operation of the circuit with live traffic will be given at the demonstration night of the conference.

References

[1] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[2] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. W. Lockwood. Deep packet inspection using parallel bloom filters. In *Hot Interconnects*, pages 44–51, Stanford, CA, Aug. 2003.

[3] M. Roesch. SNORT - lightweight intrusion detection for networks. In *LISA '99: USENIX 13th Systems Administration Conference*, Seattle, Washington, Nov. 1999.