Design and Implementation of an FPGA Architecture for High-Speed Network Feature Extraction

Sailesh Pati Ramanathan Narayanan Gokhan Memik Alok Choudhary Joseph Zambreno[†]
Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208, USA
{*ran310, memik, choudhar*}@*eecs.northwestern.edu*
[†]Electrical and Computer Engineering
Iowa State University
Ames, IA 50011, USA
zambreno@iastate.edu

Abstract

Network feature extraction involves the storage and classification of network packet activity. Although primarily employed in network intrusion detection systems, feature extraction is also used to determine various other aspects of a network's behavior such as total traffic and average connection size. Current software methods used for extraction of network features fail to meet the performance requirements of next-generation high-speed networks. In this paper, we propose an FPGA-based reconfigurable architecture for feature extraction of large high-speed networks. Our design makes use of parallel rows of hash functions and sketch tables in order to process network packets at a very high throughput. We present a detailed description of our architecture and its implementation on a Xilinx Virtex-II Pro FPGA board, and provide cycle-accurate timing results for feature extraction of input networking benchmark data. Our results demonstrate real-world throughputs of as high as 3.32 Gbps, with speedups reaching $18\times$ *when compared to an equivalent software implementation.*

1 Introduction

The goal of a Network Intrusion Detection System (NIDS) is to detect attacks on any machine within the local network by monitoring the network activity. In general, there are two different approaches taken when protecting networks using intrusion detection-based systems. The first approach, known as *signature detection*, searches for predetermined attack patterns in the network activity. The second approach, known as *anomaly detection*, looks for any sort of abnormal activity in the network flow, and then determines if the abnormal behavior is an attack. Signature detection-based methods are unable to detect new kinds of attacks, as well as those attacks that vary significantly from previously known methods. This observation has motivated a deeper study of anomaly-based NIDS mechanisms.

Anomaly detection typically involves two separate stages. In the first step, network features stored in packet headers are extracted and stored over an interval of time. In the second step, a change detection and classification algorithm is applied to this stored information in order to detect attacks. In large-scale high-speed networks, this first step in anomaly detection is the most crucial. An efficient NIDS must be able to store and classify network features without compromising on speed or loss of information. Considering the increasing size and speed of modern networks, general-purpose processors do not meet the requirements of the next generation of NIDSs. This has motivated researchers to explore the possibility of using dedicated hardware for anomaly detection systems in general [2, 3, 15] and feature extraction/classification in particular [10, 16].

Besides its use in anomaly detection, feature extraction is key to several other applications such as data mining [1], speech recognition [12], and image processing [11], among others. However, due to the clear needs of performance, we concentrate on using feature extraction for anomaly detection only. Particularly, we propose a reconfigurable architecture for feature extraction of high-speed networks, and implement this design using FPGAs. By making use of the inherent parallelism of FPGA hardware, we are able to speed up our application by a considerable amount as compared to an equivalent software implementation. Our architecture is pipelined to achieve a high throughput, making it suitable for application in multi-gigabit networks. We also make use of feature sketches to store network activity, thus minimizing the required memory resources. Our results show that the architecture is several times faster than the equivalent software implementation and offers a practical solution for feature extraction of high-speed networks.

The remainder of this paper is organized as follows. Section 2 provides a brief overview of some of the main concepts behind network intrusion detection. Section 3 motivates the need for hardware implementation of the feature extraction process. In Section 4, we describe our Feature Extraction Module (FEM) architecture and its various building blocks. Section 5 presents an example NIDS implementation, demonstrating how the FEM module can be used to detect certain types of attacks. Section 6 provides details of our implementation and presents our area and performance results. Related work is discussed in Section 7, followed by the conclusion in Section 8.

2 Intrusion Detection Overview

There are a wide variety of attacks prevalent in modern networks. Detection of each type of attack requires the monitoring of different network features. For our implementation of a FEM-based anomaly detection architecture, we focus on two major categories of intrusions common to modern day networks: *Denial-of-Service (DoS)* attacks and *port scanning* attacks. There are many different variations of DoS attacks including the SYN flood, spoofing, smurf attack, fraggle attack, and distributed DoS attack (DDoS). Although our architecture can be configured to extract features for detection of any type of the above mentioned attacks, in our study we focus only on SYN floods and DDoS attacks in a TCP/IP network. Since the implementation of FEM depends on the type of attack, we provide a description of the various attacks and the corresponding features needed to detect them.

The understanding of these types of attacks requires some basic knowledge of the 3-way handshake by which TCP connections are established. First, the source requests a connection by sending a packet with the SYN flag set to the host computer. The host then responds with a packet having both the SYN and ACK flags set. To complete this half-open connection, the source responds with a packet having the ACK flag set. Typically the host waits for a certain duration of time for the last acknowledgement from the source, after which it closes the half-open connection and the entire process has to be repeated for a new connection. A host computer in a network can handle only a finite number of connections at a time.

The *SYN flood* is a very common example of a DoS attack. In the SYN flood attack, the source (attacker) sends a series of connection requests in a short duration of time to the host (victim), filling up all of its connection handling resources. When the host responds with both the SYN and ACK flags set, the source purposefully skips sending the last ACK flag to the host. Consequently the host cannot accept any requests for a new connection as all of its connection-handling resources are consumed by the half-open connections.

In another variation of SYN flood, the source spoofs the IP address in the packet which it sends to the host to request a connection. The host responds with a SYN/ACK packet to the modified IP address which will either be non-existent or will be ignorant of the requested connection. As a result the 3-way handshake is never completed, resulting in multiple half-open connections at the host. In either case, if the host computer happens to be a server, then the entire network is affected as it becomes unable to accept any new connection requests.

In the *DDoS attack*, an attacker first gains control of several computers in the network in order to attack a server using multiple parallel DoS attacks (like the SYN flood). This attack is more difficult to detect than the standard SYN flood since it originates from multiple sources, none of which is the real source of the attack.

In the *port scanning* attack, the attacker scans for open ports on different computers on the network. During this scan the attacker sends a connection request to that particular port and determines whether the port is open from its response. Similar to the SYN flood attack, in this attack the 3-way handshake never occurs and usually ends in the second step. Since the host port's response must return to the source, the attacker cannot spoof its IP address in the initial connection request packet. Consequently, unlike in the case of a DoS attack, a port scan attacker can be traced back.

3 Application Analysis

The process of extracting features from network packets consists of two stages. The first stage involves storing the information associated with specified fields in the packet header. The fields that are stored are directly determined by the requested features. The values of these features are then computed as a function of the stored information. This is followed by a second stage in which the features are classified in order to determine relevant information about the network activity. As applied in a NIDS, this second stages uses these extracted features to detect the occurrence of an attack from outside the network.

A critical aspect of feature extraction is the speed at which it is performed. A feature extraction system that is unable to cope with the throughput of the network packet flow will result in a loss of features in some packets, thereby reducing the clarity of the overall network picture. Another interesting metric is the number of features being extracted. Extracting more features can give a more accurate picture of the network activity. With the emergence of multi-gigabit networks, it is highly essential for the first stage in a feature extraction system to have a very high throughput. Unlike in signature detection-based approaches, anomaly detection utilizes the network features gathered over a period of time to look for the possibility of occurrence of any attack. In other words, features need not be sent to the anomaly de-

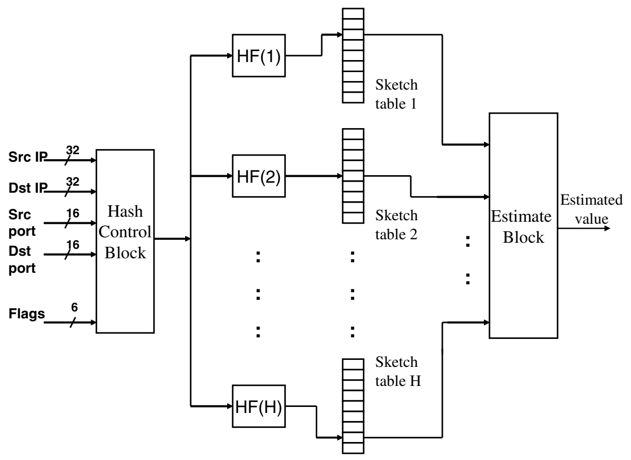

Figure 1. Feature sketch architecture

tection module for each individual packet. Consequently, the second stage of the feature extraction system can afford to have a lower throughput, even if network speed is very high.

General-purpose processors currently used in the first stage of an anomaly-based NIDS cannot meet the requirements of high-speed networks for several reasons. First, extracting multiple features from a single packet involves cycling the same data through the processor multiple times, reducing the effective throughput. Also, computation of complex features may involve extensive computation within the processor, again reducing the overall throughput given the limited computational parallelism available in general-purpose processors.

In order to overcome these limitations, we propose a hardware design to implement the first stage of the feature extraction system. We incorporate both coarse and fine-grained parallelism into the design in order to obtain a very high throughput.

4 FEM Architecture

The objective of our Feature Extraction Module (FEM) is to efficiently collect all of the necessary flow information from incoming and outgoing packets in order to detect intrusions. We make use of feature sketches to store this network information. A sketch is a probabilistic approach used to summarize large amount of information given a fixed amount of memory. Researchers have shown that sketches are able to summarize large datasets (such as network activity) with a high level of accuracy [14, 15].

The FEM consists of several feature sketches connected in parallel, with each sketch customized to store a unique feature. The general architecture of our feature sketch design is shown in Figure 1. A feature sketch consists of four important blocks: the hash control, the hash function, the sketch table, and the estimate block. The only network information which the FEM needs in order to detect DoS and port scanning attacks are the source IP, destination IP, source port, destination port, and selected flags.

The multiple parallel hash functions within each feature sketch contain a combination of the input fields. The specific combination depends on the network activity being analyzed by the FEM, and is unique for each feature sketch. The hash control block is configured as a custom input multiplexer for this purpose. For our implementation, we use Bob Jenkins' 32-bit hash function [8] because of the advantages it offers in terms of speed and fewer number of collisions.

The Jenkins hash function for hashing of three 32-bit keys (K[0], K[1], K[2]) is as shown below. The golden ratio and seeding values are random values required for the initialization of a hash function:

$A = B = golden\_ratio;$
$C = seeding\_value;$
$A = A + K[0];$
$B = B + K[1];$
$C = C + K[2];$
$mix(A, B, C);$
$mix(A, B, C);$

The mix function is defined as:

```
$mix(a, b, c)$ {
    $a = (a - (b + c)) \bigoplus (c >> 13);$
    $b = (b - (c + a)) \bigoplus (a << 8);$
    $c = (c - (a + b)) \bigoplus (b >> 13);$
    $a = (a - (b + c)) \bigoplus (c >> 12);$
    $b = (b - (c + a)) \bigoplus (a << 16);$
    $c = (c - (a + b)) \bigoplus (b >> 5);$
    $a = (a - (b + c)) \bigoplus (c >> 3);$
    $b = (b - (c + a)) \bigoplus (a << 10);$
    $c = (c - (a + b)) \bigoplus (b >> 15);$
}
```

The implementation of the Jenkins hash function is done as shown in Figure 2. The initialization stage assigns appropriate values to the keys and performs a minor part of the computation of the mix function. The rest of the mix function is implemented using three mix sub-blocks as shown in Figure 2. All the mix sub-blocks are identical in structure and differ only in the number of bit shifts performed before each XOR operation. Each mix sub-block has two pipeline stages - when added to the single pipeline stage of the initialization phase, makes the hash function a thirteen stage pipelined structure. It is important to note that in this systolic style each pipelined stage has a maximum of three logical operations, resulting in a very high clock rate and throughput.

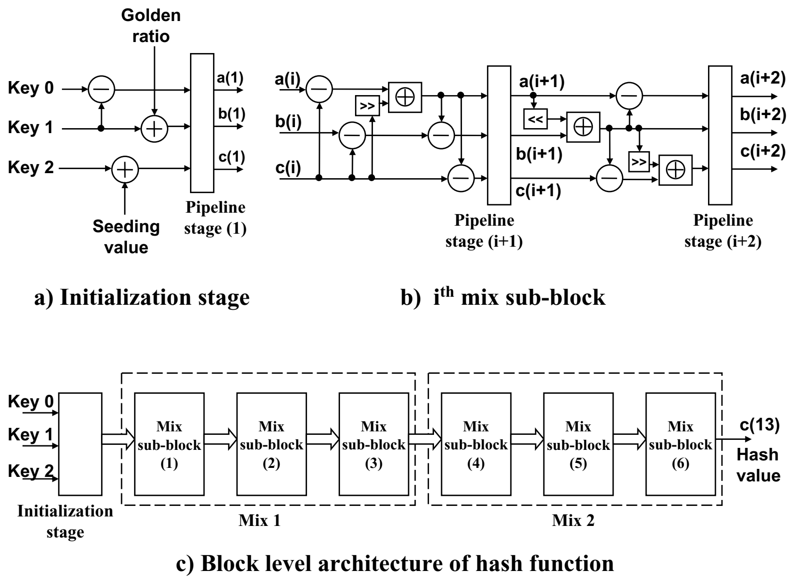

Figure 2. Hash function implementation

In the FEM, the Jenkins' hash takes as its input the source IP, destination IP, and source/destination ports. In the configuration where any of these fields are not required by a feature sketch, they are replaced by zeroes by the hash control block. Within a single feature sketch, we use multiple hash functions with different seeding values so as to minimize the effect of hash collisions. The number of hash functions within each feature sketch can be varied depending on the accuracy requirements. The sketch table is essentially a hash lookup table which stores the network flow information contained in the flags. Its size can also be varied to reduce hash collisions, depending on the NIDS accuracy requirements.

A common characteristic of the attacks described in Section 2 is that the 3-way handshake is never fully established. This 3-way handshake involves only the SYN and ACK flags. Consequently, in a given time interval the number of incomplete connections in the network is an indication of the possibility of an attack occurring in that interval. Our FEM design applies this concept when storing the network activity. If the SYN flag is set in an incoming packet, then the value in the sketch tables corresponding to that packet is incremented by one, and if the ACK flag is set that value is decremented by one. In other words, for every connection request the value in a given element of the feature sketch table is increased by one. Similarly, once that connection is fully established, that same value is decreased by one. As will be described in the following section, in our FPGA implementation of this FEM design the only attacks we have considered involve the monitoring of the SYN and ACK flags. However, the FEM implementation can be easily reconfigured to analyze network activity which involves other flags.

As can be seen in Figure 1, the estimate block selects the correct estimated value for each feature sketch from the sketch tables. Although there are various statistical estimation techniques that can be used for this purpose, we implement our estimate block as finding the minimum of the selected values from each sketch table of the feature sketch. We choose the minimum value as the estimate since it suffers the least amount from hash collisions. The output control block then selects the estimated value of only that feature sketch from which an estimate request was made. The FEM supports two functions for its operation:

1. `update` (src_ip, dst_ip, src_port, dst_port, flags)
2. `estimate` (src_ip, dst_ip, src_port, dst_port, FS_ID)

The `update` function is a write-only operation to the FEM where the flag information of the network packet is stored in the sketch tables of each feature sketch. During an `update`, the write operation is performed on all of the feature sketches of the FEM. As previously described, the value written into the sketch tables is some function of the input flags, which depends on the network feature being stored in each feature sketch. The key value in an `update` call (src_ip, dst_ip, src_port, dst_port) is used for determining the address of the sketch table where the value is to be written into.

The `estimate` function is a read-only operation where the requested estimated value from a particular feature sketch corresponding to a input key is given as output by the FEM. The key in an `estimate` call (src_ip, dst_ip, src_port, dst_port) corresponds to the sample for which the `estimate` call is made. The network feature value returned in an `estimate` call depends on the unique feature sketch identifier (FS_ID) on which the estimate is made. An estimate call to the FEM is made to retrieve any stored information and is much less frequently used than the update function.

5 An FEM-Based NIDS Application

In this section we discuss the practical application of the FEM architecture and how it can be used as the basis of a network intrusion detection system. From the application point of view, the FEM is placed within the network at some common node through which all of the network traffic passes. The FEM module's feature sketches are updated with the values in the network packets passing through that node. At any point in time, the FEM contains the up-to-date information of the network features, which can be obtained by estimating the feature sketches values with an appropriate key. This information can be used to detect attacks using an anomaly-based intrusion detection algorithm.

Consider the FEM having four feature sketches with keys and values as shown in Figure 3. As explained in the previous section, in our current implementation the FEM extracts only those port fields which are necessary to detect DoS or port scanning styles of attacks. Although each feature sketch stores only a specific feature, the overall nature of the network can be obtained by combining the information from all the feature sketches as illustrated below.

In Figure 3, FS1 contains information regarding the number of incomplete connections at any port of any computer within the network. So any (dst_ip, dst_port) pair in FS1 having a value above some threshold level is a likely victim of a SYN flood attack. Similarly, a (dst_ip) in FS2 having a high value is also a probable candidate for a SYN flood attack or it may be being scanned for any open ports. For any victim (dst_ip) in FS2, the corresponding (dst_ip, dst_port) values in FS1 give the information regarding which ports are being attacked. For any possible victim (dst_ip) in FS2, if there is a corresponding (src_ip, dst_ip) in FS4 having a high value then it can be known which computer is port scanning or performing a SYN flood attack on that particular victim.

On the other hand, if there is a possible victim (dst_ip) in FS2 but there is no corresponding (src_ip, dst_ip) in FS4 with a high value, it implies that the destination IP may be either a victim of a DDoS attack or a modified SYN flood attack in which the attacker spoofs its IP address in the packets that are sent to hinder traceback efforts. If there is a (src_ip) in FS3 having a high value with no corresponding (src_ip, dst_ip) in FS4 with a relatively high value it implies that the (src_ip) attacker may be port scanning multiple computers within the network.

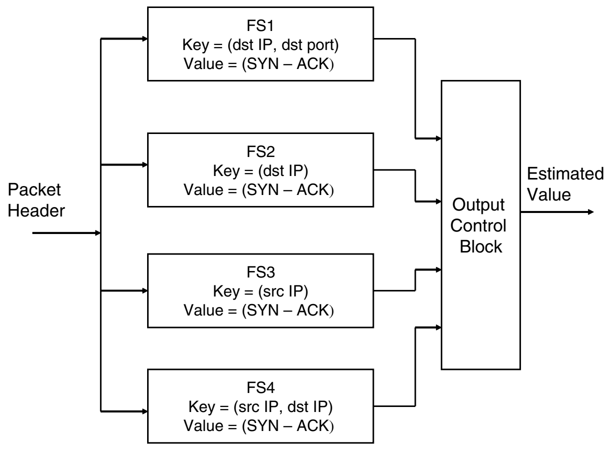

Figure 3. Sample application study

The FEM design considered above is configured so to as to extract the specific features necessary to detect certain type of attacks as mentioned earlier. However, the reconfigurability of the architecture provides the flexibility to monitor different features by simply changing the keys and the values stored in the feature sketches. For example, the size of the network traffic can be monitored by keeping track of the SYN and FIN flags. The number of extracted features can be increased by employing additional feature sketches. Since all the feature sketches are implemented with parallel pipelines, with the keys and sketch table values not affecting the critical path, any FEM configuration alteration would not affect the overall system performance.

6 Results

FPGA implementations of many diverse application domains have been shown to have significant performance improvements over a conventional software implementation. Researchers have attributed this speedup to two major factors: the inherent parallelism in FPGAs, and the inherent instruction inefficiency of standard microprocessors [6]. The majority of the logic in the FEM architecture is contained in the Jenkins' hash functions, comprised of simple arithmetic operators. Consequently, there are significant performance overheads in any software implementation, since the fetching of the instruction operands from memory consumes a large number of cycles in the CPU.

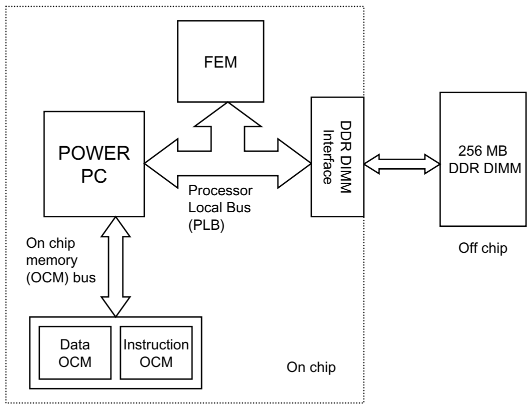

Figure 4. Experimental setup

Another aspect in favor of an FPGA implementation is that much of the arithmetic required by the FEM operates on the individual bit level, resulting in a higher computational efficiency when compared to a general-purpose processor that operates at the word level [5]. Processors sacrifice computational density for deep instruction memories which reduce their area consumption.

Parallelism in hardware is another significant factor in the speedup of this architecture. While each hash table within each feature sketch is updated sequentially in software, all are updated simultaneously in hardware. We expect the speedup to increase linearly as a function of the number of total feature sketch hash tables. Given these tradeoffs, an FPGA implementation of the FEM can meet performance requirements at the expense of area consumption.

6.1 Experimental Setup

The FEM was implemented as a single VHDL core, with a configuration file for controlling the different parameters of the FEM such as the number of feature sketches, the number of hash functions within each feature sketch, the size of the hash tables, and the keys and values for each feature sketch. We implemented the system on a Xilinx ML310 evaluation board which is an embedded development platform consisting of a Xilinx XC2VP30 FPGA and other industry-standard peripherals such as 256MB DDR RAM, 512MB compact flash card, an ethernet controller, etc. The XC2VP30 FPGA on the ML310 board belongs to the Xilinx Virtex-II Pro device family, containing 13696 slices, 136 BlockRAM modules, and 2 PPC405 processors. Xilinx Platform Studio (XPS) 8.1i was used for setting up the hardware/software system, and Xilinx ISE 8.1i was used for synthesis as well as for placement and routing.

K	N_{Bram}	N_{slices}	f_{max} (MHz)	Throughput (Gbps)
1024	68	5010	102.67	3.29
2048	72	5142	101.76	3.26
4096	84	5057	102.72	3.29
8192	100	5089	102.15	3.27
16384	136	5114	102.33	3.27

Table 1. Variation of resource utilization with K for HF=2, FS=2

The experimental setup is as shown in Figure 4. The FEM was run in an offline manner where the PPC405 was used to read in input data from the DDR DIMM and feed it to the FEM module which is connected to the PLB bus of the PPC. The OCM-connected BlockRAM stores the instructions for the PowerPC. For this setup we used the above-mentioned tools to observe the number of occupied slices (N_{slices}), number of BlockRAMs utilized (N_{Bram}) and the maximum clock frequency (f_{max}) with variation in certain parameters of the FEM such as number of feature sketches (FS), number of hash functions within each feature sketch (H) and size of the hash tables (K). Due to the inherent parallelism of the FEM design, it can take 128-bits of input in every clock cycle. For the practical implementation of our design we were limited by the PLB bus width and bandwidth, requiring us to modify the input stage so that 32 bits were processed every cycle. Consequently the effective throughput was measured as $32 \cdot (f_{max})$ in bits per second.

The FEM application was also implemented in software and run on the PPC 405 under identical conditions as the hardware. The speedup provided by hardware was measured in terms of the ratio of the number of cycles taken by the PowerPC in the hardware version (C_{hw}) to the number of cycles taken by the PowerPC in the software version (C_{sw}) to perform a single update operation. In any practical application the FEM will be updated with every single network packet but will not be estimated for every network packet. The FEM performs an estimate call much less frequently than an update call. Therefore the update function was chosen as the basis for measuring the speedup provided by hardware.

6.2 Area and Performance Results

Table 1 shows the variation in resource utilization and performance with respect to the size of a hash table (K) when FS and H are set to 2. From the trends it is observed that increasing K while keeping all other parameters constant does not result in significant variation in slice uti-

FS	H	N_{slices}	f_{max} (MHz)	Throughput (Gbps)	C_{hw}	C_{sw}
1	1	2621	102.59	3.28	433	631
1	2	3428	103.82	3.32	433	1162
1	4	5035	101.48	3.25	433	2024
2	1	3403	101.72	3.26	433	1215
2	2	5010	102.67	3.29	433	2259
2	4	8197	101.39	3.24	433	3988
4	1	4991	103.54	3.31	433	2370
4	2	8140	102.86	3.29	433	4619
4	4	13694	99.60	3.19	433	8087

Table 2. FEM area and performance summary

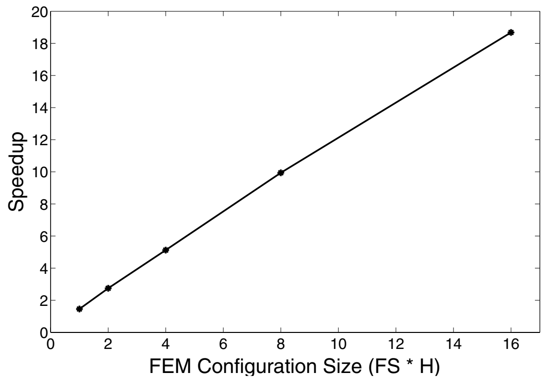

Figure 5. Hardware speedups for different FEM sizes

lization. This is because the hash tables are mapped onto BlockRAMs which can be verified by noting the increase in BlockRAM utilization when the hash table size is increased. For the different sizes of the hash table considered, the throughput is almost constant, indicating that the hash tables are not on the critical path of the FEM.

Table 2 shows the area and performance results for implementation of different configurations of the FEM on FPGA. From these results it can be observed that the number of slices utilized is directly proportional to both FS and H. This is due to the fact that an increase in FS or H directly corresponds to an additional hash function and sketch table. By considering the increments in the slices when FS and H are increased, we can see that a single hash function along with one sketch table takes up around 800 slices, which is about 6% of the total number of available slices on the Xilinx XC2VP30 FPGA. As a result, the configuration with FS=4 and H=4 occupies over 99% of the slices available in our FPGA.

From the results we see that the throughput remains more or less constant for the different configurations of the FEM. This is expected since all the feature sketches and the hash functions within each feature sketch are executed in parallel. As discussed in the previous section, the throughput values mentioned in Table 2 are limited by the PLB bus used by the PowerPC. The FEM is capable of having a throughput equal to $4\times$ that shown, provided all of the inputs are fed in parallel (i.e., it supports over 10 Gbps throughput).

As mentioned earlier, the majority of the slices are utilized by the hash functions. In the presence of area constraints, area consumption of the FEM can be reduced by replacing multiple hash functions within a feature sketch with a single hash function. Alternatively a single hash function can be used for the entire FEM. However this will be at the cost of a lower throughput. So while designing a FEM, there is a trade-off between throughput and area consumption, which can be modified optimized according to specific accuracy-performance requirements and available resources.

For performance measurements, the FEM was fed with sample datasets from the information exploration shootout organized by the Institute for Visualization and Perception Research, University of Massachusetts, Lowell [7]. Table 2 lists the average number of cycles taken by the hardware (C_{hw}) and software (C_{sw}) individually to perform a single update. The speedup provided by hardware for different configuration sizes (FS*H) of the FEM is shown in Figure 5. The results indicate that the speedup provided by hardware increases proportionately with both FS and H. As previously explained, this can be attributed to both the inherent parallelism and overhead associated with software instruction execution.

The performance results clearly show the advantage offered by hardware over software in terms of speedup - which is significant for the larger configurations of the FEM. The throughput values obtained through extensive pipelining of the design are sufficient for using the FEM in gigabit networks. The reconfigurability of the architecture and the parallelism which it offers makes it suitable for FPGA implementation.

7 Related Work

Many network applications have been implemented in hardware in order to keep up with increasing network speeds. Since the current-generation FPGAs are capable of operating at speeds up to 550Mhz, have comparable capacities to ASIC designs, and provide some flexibility in implementation, they are being actively used in developing high-speed network applications. In the most closely related work, Nguyen et al. [10] have developed a feature extraction module that utilizes multi-dimensional hashes. Although there are some similarities in the architecture, we provide a novel implementation of the underlying functions in order to obtain a higher real-world throughput while consuming significantly less area. In addition, Nguyen et al. only provide results from synthesis and do not realize their designs onto an FPGA.

FPGAs have been used in developing NIDS capable of operating up to 8 Gbps, where the network interface and intrusion detection circuitry have been integrated onto a single FPGA chip [4]. FPGAs have also been used to implement TCP/IP flow monitors [13] operating at 3 Gbps. Other network applications such as internet firewalls [9] working at 2.5 Gbps have also been implemented using FPGAs. A direct comparison of these FPGA architectures is difficult as they have different goals and are targeted towards different hardware technologies.

8 Conclusion

Feature extraction is an important component of various applications in domains such as networking, data mining, and signal processing. In this paper we propose a reconfigurable architecture for real-time feature extraction of high speed networks used for anomaly detection, and evaluate its performance using an FPGA-based hardware development platform. We show how the FEM can be used to detect various types of network attacks, and that the reconfigurability of the architecture provides the flexibility to store various network features by making minor changes to the feature sketches.

Our results clearly demonstrate that this architecture is several times faster than an equivalent software implementation (up to $18\times$). We have also observed that the relative performance of our hardware implementation improves as the number of features is increased, which is desirable for better accuracy. It is also seen that the throughput (as high as 3.32 Gbps) is unaffected by the number of features monitored, thereby making the architecture suitable for high performance network intrusion detection systems.

Acknowledgments

This work was supported in part by the National Science Foundation (NSF) under grants CCR-0325207, CNS-0720691, CNS-0406341, CNS-0551639, IIS-0613568, IIS-0536994, CCF-0541337, in part by the Air Force Office of Scientific Research (AFOSR) under grant FA9550-06-1-0152, and in part by the Department of Energy (DOE) under grant DE-FG02-05ER25691.

References

[1] NIPS 2003 Workshop on Feature Extraction. Available at http://clopinet.com/isabelle/Projects/NIPS2003, 2003.

[2] Z. Baker and V. Prasanna. Time and area efficient pattern matching on FPGAs. In *Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA)*, February 2004.

[3] C. Clark, W. Lee, D. Schimmel, D. Contis, M. Kone, and A. Thomas. A hardware platform for network intrusion detection and prevention. In *Proceedings of the Third Workshop on Network Processors and Applications (NP3)*, February 2003.

[4] C. Clark, C. Ulmer, and D. Schimmel. An FPGA-based network intrusion detection system with on-chip network interfaces. *International Journal of Electronics*, 93(6), June 2006.

[5] A. DeHon. The density advantage of configurable computing. *IEEE Computer*, 33(4), April 2000.

[6] Z. Guo, W. Najjar, F. Vahid, and K. Vissers. A quantitative analysis of the speedup factors of FPGAs over processors. In *Proceeedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, February 2004.

[7] Institute for Visualization and Perception Research, University of Massachusetts, Lowell. Contents of network intrusion collected data. Available at http://ivpr.cs.uml.edu, 2006.

[8] B. Jenkins. Hash functions and block ciphers. Available at http://burtleburtle.net/bob/hash, 2006.

[9] J. Moscola, J. Lockwood, R. Loui, and M. Pachos. Implementation of a content-scanning module for an internet firewall. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, April 2003.

[10] D. Nguyen, G. Memik, S. Memik, and A. Choudhary. Real-time feature extraction for high speed networks. In *Proceedings of the International Symposium on Field-Programmable Logic and Applications (FPL)*, August 2005.

[11] M. Nixon and A. Aguando. *Feature Extraction and Image Processing*. Elsevier, Inc., 2004.

[12] G. Saha, P. Kumar, and S. Chakroborty. A comparative study of feature extraction algorithms on ANN based speaker model for speaker recognition applications. In *Proceedings of the International Conference on Neural Information Processing (ICONIP)*, November 2004.

[13] D. Schuehler and J. Lockwood. A modular system for FPGA-based TCP flow processing in high-speed networks. In *Proceedings of the International Symposium on Field-Programmable Logic and Applications (FPL)*, August 2004.

[14] R. Schweller, A. Gupta, E. Parsons, and Y. Chen. Reversible sketches for efficient and accurate change detection over network data streams. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, October 2004.

[15] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood. Fast hash table lookup using extended bloom filter: An aid to network processing. In *Proceedings of ACM SIGCOMM*, August 2005.

[16] H. Song and J. Lockwood. Efficient packet classification for network intrusion detection using FPGA. In *Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA)*, February 2005.