An FPGA-Based Network Intrusion Detection Architecture

Abhishek Das, *Student Member, IEEE*, David Nguyen, Joseph Zambreno, *Member, IEEE*, Gokhan Memik, *Member, IEEE*, and Alok Choudhary, *Fellow, IEEE*

***Abstract*—Network intrusion detection systems (NIDSs) monitor network traffic for suspicious activity and alert the system or network administrator. With the onset of gigabit networks, current generation networking components for NIDS will soon be insufficient for numerous reasons; most notably because the existing methods cannot support high-performance demands. Field-programmable gate arrays (FPGAs) are an attractive medium to handle both high throughput and adaptability to the dynamic nature of intrusion detection. In this work, we design an FPGA-based architecture for anomaly detection in network transmissions. We first develop a feature extraction module (FEM) which aims to summarize network information to be used at a later stage. Our FPGA implementation shows that we can achieve significant performance improvements compared to existing software and application-specific integrated-circuit implementations. Then, we go one step further and demonstrate the use of principal component analysis as an outlier detection method for NIDSs. The results show that our architecture correctly classifies attacks with detection rates exceeding 99% and false alarms rates as low as 1.95%. Moreover, using extensive pipelining and hardware parallelism, it can be shown that for realistic workloads, our architectures for FEM and outlier analysis achieve 21.25- and 23.76-Gb/s core throughput, respectively.**

***Index Terms*—Feature extraction, field-programmable gate arrays (FPGA), network intrusion detection system (NIDS), principal component analysis (PCA).**

I. Introduction

TRADITIONALLY, intrusion detection techniques fall into two categories: 1) signature detection and 2) anomaly detection. Signature detection, or misuse detection, searches for well-known patterns of attacks and intrusions by scanning for preclassified signatures in TCP/IP packets. This model of network monitoring has extremely low false-alarm rates but cannot detect new attacks. String matching [10] is an example of a signature-based method for detecting suspicious payloads in packets. Rule-based intrusion detection systems such as Bro [8] or Snort [33] use known rules to identify known attacks in packet payloads, such as requests for nonexisting services, strange flag combinations, or virus attack signatures or malicious strings. A number of software and hardware implementations have been proposed in this area, some of which utilized reconfigurable hardware architectures [30], [32], [34].

The second category of intrusion detection, anomaly detection, is used to capture behavior that deviates from the norm. This method takes as input training data to build normal network behavior models. Alarms are raised when any activity deviates from the normal model. Although anomaly detection can detect new intrusions, it may suffer from false alarms, including both raised alarms for normal activity (false positives) and quiet alarms during actual attacks (false negatives). Despite this obvious drawback, the high rate of increase in new attacks and change in the pattern of old attacks has made anomaly detection an indispensable technique in NIDSs. Since network line speeds have reached Gigabit rates and anomaly detection is computation intensive, software-based techniques are inadequate. Reconfigurable hardware solutions exhibit an attractive implementation choice for anomaly detection due to their inherent parallelism, pipelining characteristics, and adaptability. As more sophisticated methods are designed, algorithms and methods can be tailored specifically to their implementation environment. Although researchers have started working on hardware implementations of data mining methods, such as the *apriori* algorithm [6], to the best of our knowledge, there are no hardware architectures specifically tailored to PCA or network anomaly detection. Fig. 1 gives an overview of a network detection system using both signature and outlier detection. A key point in this figure is the use of feature extraction modules to obtain concise information from a live packet stream.

In this paper, we propose a new architecture for building an efficient NIDS using FPGAs. Particularly, we focus on anomaly detection. Our work is comprised of a new feature extraction module (FEM) which summarizes the network behavior. It also incorporates an anomaly detection mechanism using principal component analysis (PCA) as the outlier detection method. Both these modules are implemented on reconfigurable hardware and can handle the gigabit throughput demands by modern networks. The principal advantage of our technique is the high detection rate of network intrusions.

The rest of this paper is organized as follows. Section II gives an overview of our intrusion detection architecture. Section III presents the FEM architecture and its components. Section IV describes PCA in detail. The implementation details and simulation results are illustrated in Section V. Section VI presents the

Manuscript received May 5, 2006; revised September 7, 2007. This work was supported in part by the National Science Foundation (NSF) under Grants NSF-ITR CCR-0325207, CNS-0406341, CNS-0551639, IIS-0536994, and CCR-0325207, in part by the Air Force Office of Scientific Research (AFOSR) Award FA9550-06-1-0152 and in part by the Department of Energy (DoE) under CAREER Award DE-FG02-05ER25691. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Klara Nahrstedt.

A. Das, G. Memik, and A. Choudhary are with the Electrical Engineering and Computer Science Department, Northwestern University, Evanston, IL 60208 USA (e-mail: ada829@ece.northwestern.edu; memik@eecs.northwestern.edu; choudhar@ece.northwestern.edu).

D. Nguyen is with SanDisk Corporation, Milpitas, CA 95035 USA.

J. Zambreno is with the Electrical and Computer Engineering Science Dept., Iowa State University, Ames, IA 50011-2010 USA (e-mail: zambreno@iastate.edu).

Digital Object Identifier 10.1109/TIFS.2007.916288

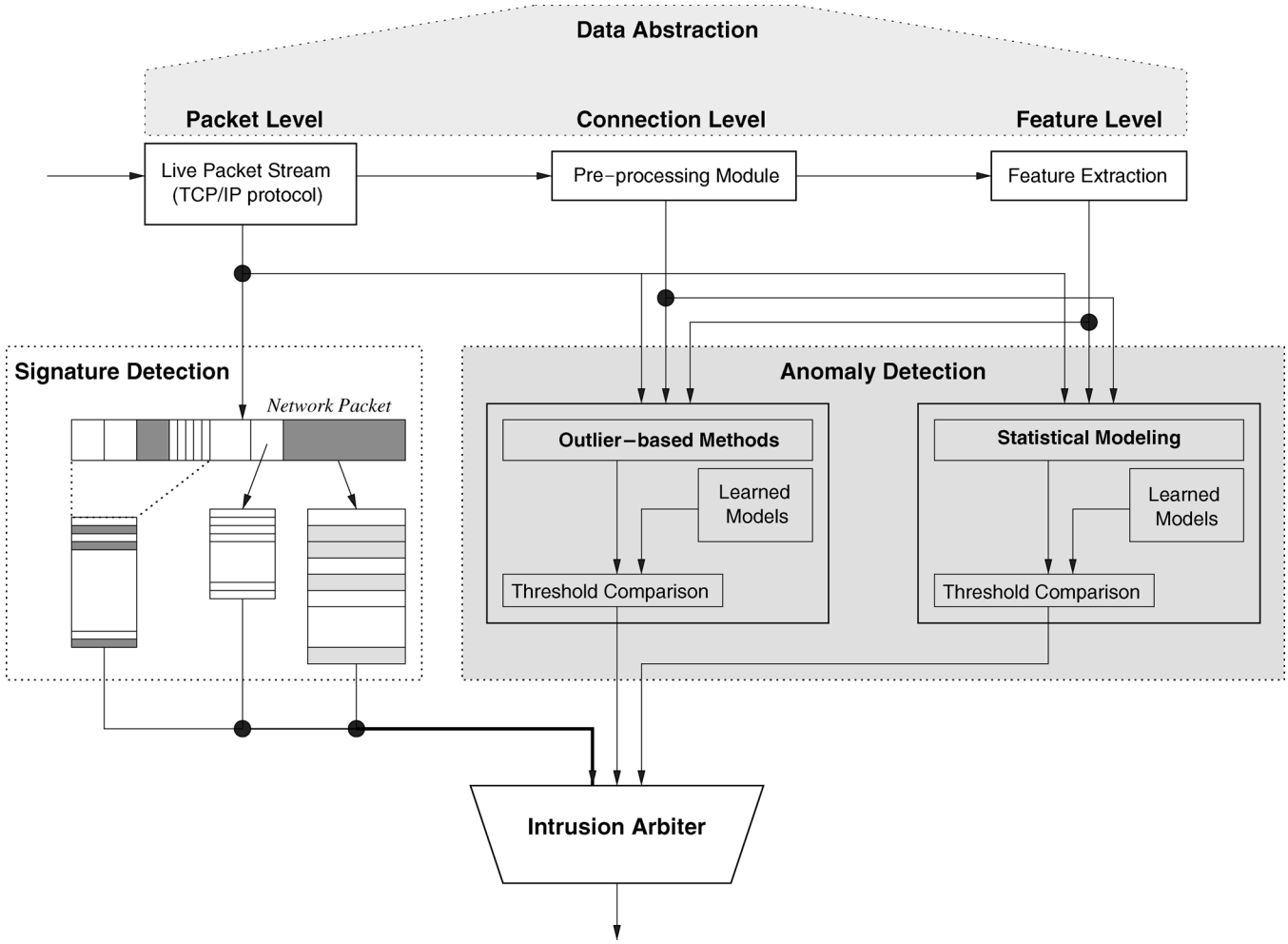

Fig. 1. General network intrusion detection framework.

related work, and Section VII concludes the paper and presents the areas of future work.

II. Intrusion Detection Architecture Overview

Our architecture for intrusion detection has two major components. First, we propose an FEM which accurately characterizes network behavior and provides an up-to-date view of the network environment [4], [28]. Extraction of packet headers alone cannot provide an accurate description of the network behavior. Depending on the application utilizing this information (e.g., rule mining, classification), different properties, such as connection duration, the number of SYN/FIN/RST flags sent, etc. should be monitored. As we will describe in the following sections, our architecture can be easily configured to gather different types of information. By utilizing the reconfigurable capabilities of FPGAs, these changes can be effectively performed. Experimental results prove that our FEM architecture is a viable alternative to expensive per-flow methods. In addition, our FEM implementation requires a constant amount of memory and achieves a guaranteed performance level, important characteristics for networking hardware design.

Second, we develop a novel architecture for intrusion detection that uses PCA as an outlier detection method. PCA is appealing since it effectively reduces the dimensionality of the data and, therefore, reduces the computational cost of analyzing new data. PCA methodology has been successfully used in signal processing, namely the Karhunen Loeve Transformation [15], and image processing for compression and restoration. In the case of the KDD Cup 1999 data, where each connection record has 41 features, we will show that PCA can effectively account for up to 50% of the variation or relative significance of the data with only five principal components. Being able to capture such a large fraction of the variation by only using a small number of features is certainly a desirable property for the hardware implementation, (i.e., such an algorithm is likely to reduce the hardware overhead significantly).

It should be noted that our techniques are considerably different from software intrusion detection mechanisms such as SNORT. The latter is an open source intrusion detection and prevention tool which combines the benefits of signature and anomaly detection methods. On the other hand, our proposed architecture will help only in anomaly detection. Hence, in order to have sound and highly efficient intrusion detection, a combination of the SNORT and our PCA architecture would be more effective. However, our module will not completely replace SNORT.

Fig. 2 illustrates the overall architecture of our intrusion detection system. In the first phase, the network header data are

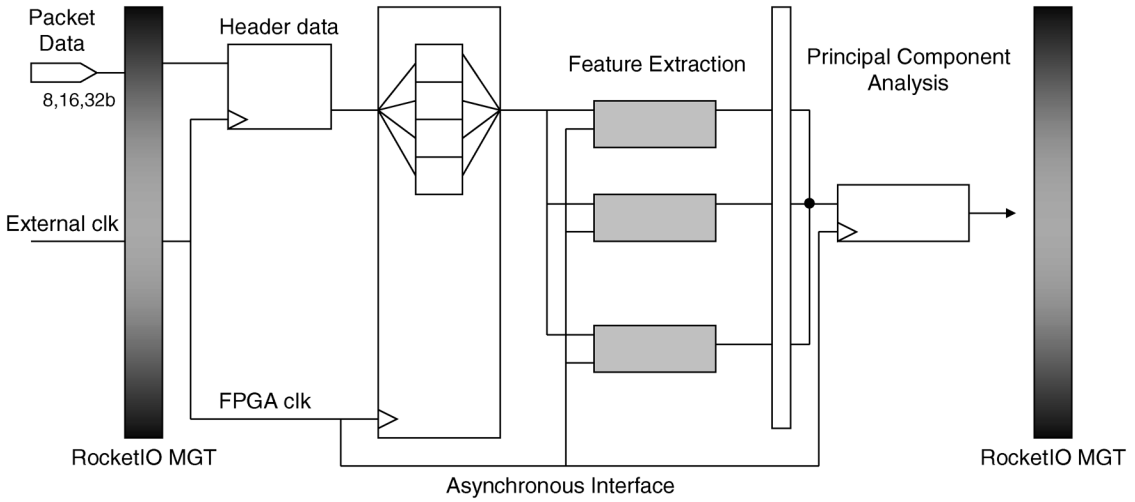

Fig. 2. FPGA architecture with feature extraction and PCA.

extracted from the packets fed into the system. The FEM, in the next phase, uses these headers to extract the temporal and connection-based characteristics of the network. We discuss FEM in detail in Section III. This information or, in other words, the network features are then processed by the anomaly detection phase, which here is done by using PCA (as shown in the figure). Formulation of PCA and its application in anomaly detection are presented in Section IV. Details regarding the framework implementation are dealt with in Section IV-E.

III. Feature Extraction Module (FEM)

Independent of the way in which anomaly detection is performed, the first step in any NIDS is flow monitoring and feature extraction. Feature extraction mines more information than conventional techniques that only monitor a small amount of features from network packets. In this section, we introduce our FEM, which characterizes network behavior within an interval of time or specified interval of connections. Network behavior represented by the FEM sufficiently reflects the current state of the network. Thus, a real-time profile of the network is always available for processing with intrusion detection schemes, such as data mining, outlier analysis, statistical methods, etc. [20].

The architecture's data-storage component models the idea of sketches [25], which are used in data-stream modeling for summarizing large amounts of information requiring a small constant amount of memory. Sketches are a probabilistic summary technique for analyzing large network streams without keeping a per-flow state that make vector projections onto other sketches to infer additional information. Our case study will show how the relationships between sketches aid in inferring additional network characteristics that are not explicitly monitored. To achieve fast execution and effective adaptation, we implement our architecture on an FPGA. The regular structure of sketches maps well onto an FPGA. We exploit the inherent parallelism in the sketch to increase throughput and obtain significant link speeds.

It is possible to model anomalous behavior associated with two general types of intrusions: 1) time based and 2) connection based. Time-based attacks cause an increase in network activity in a period of time, referred to as a "bursty attack." SYN floods are an example, where connection tables are flooded in a period of time, disabling the victim machine to service new connection requests. Connection-based attacks do not have a recognizable temporal aspect. They are sometimes referred to as "pulsing zombie attacks." Port scans may release connection requests in the span of seconds or days. Therefore, intrusion detection methods focusing on large volumes of network activity are ineffective. Our architecture can capture both connection and time-based statistics.

A. FEM Functions

FEM supports the following functions:

- UPDATE (k, v) to change the value in the sketch;
- ESTIMATE (k) to correctly retrieve a value from the sketch, where k is a key input to the hash functions and v is the feature value. The key k can be any combination of the 5-tuple fields present in TCP/IP packet headers: source IP, destination IP, source port, and destination port and protocol. The 6-b flag field, also in a packet header, assists the control logic for intelligent hashing of the 5-tuple fields depending on what network characteristics are analyzed. Note that the protocol field is redundant in our case since we are analyzing only TCP/IP packets. Since the size of the input to FEM block is 120 b, and the first four fields make up 112 b, the remaining 8-b space is kept for the protocol field. UPDATE queries are used to change the value for a key, whereas ESTIMATE queries are used to read a value back.

B. FEM Architecture

Fig. 3 highlights the main components of our architecture. It consists of a feature controller (FC), hash functions (HF), feature sketch (FS), and a data aggregate (DA). The combination of all these components provides a fast, scalable, and accurate platform from which important network characteristics can be monitored and tracked in real time. H is the number of hash functions within each FS, while K is the size of the hash tables in FS. Thus, in this figure, $\mathrm{H} = 4$. The feature controller

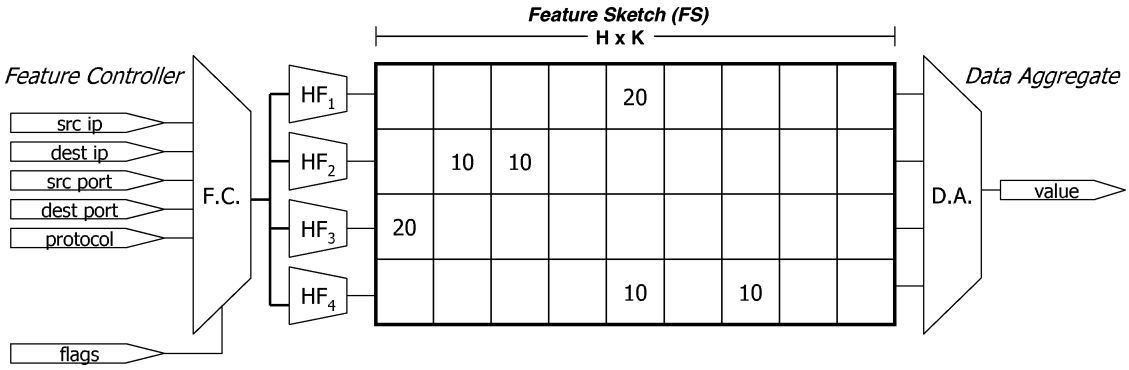

Fig. 3. FEM with one feature sketch.

(FC) coordinates the inputs to the hash functions using the flags of a packet header. The reconfigurable aspects of FPGAs make reprogramming possible to monitor a variety of network statistics. Our case study in Section III-C focuses on open connection requests originating from or incoming to hosts by utilizing the SYN and ACK flags. Other possible statistics include the number of live connections; the flow size of active connections; amount of service-related traffic; or connection-based statistics, such as the number of connections for specific services on a host. These measures would utilize the PSH (push), RST (reset), FIN (finish), and URG (urgent) flags.

The FS is an application of sketches used for data-stream modeling. It uses a constant amount of memory and has constant per-record update and reconstruction cost. Each row in the FS is accessed in parallel with different hash functions. This characteristic favors FPGAs. An FS contains H rows each of length K. When $\mathrm{H} > 1$, the accuracy of ESTIMATE queries improves. Section V-A presents the accuracy results.

Each row in the FS is addressed by a different HF. This way, the distribution of information varies for each row and the accuracy is increased. We chose the Jenkins Hash for its speed and provable scatter properties. It is implemented in various Linux kernels as a component to the IPtables connection tracking module [16], [26]. With an FPGA, all hash functions are computed in parallel. Also, by pipelining the Jenkins Hash, FEM can accept a packet on every clock cycle, thus increasing throughput.

Finally, the data aggregate (DA) component takes H values and estimates the actual value for a query. Using statistical estimation techniques, we show that ESTIMATE queries to the FS are accurate. The heuristic we implement to estimate the value of a query takes the minimum of the H values in the FS. The minimum value suffers the least from collisions. Other estimation techniques are plausible [20], but we found the minimum estimate usually gives the best results and the least hardware complexity. Minimum comparisons are performed in parallel such that this module is not on the critical path of FEM.

C. Case Study: Application of FEM on Edge Router

In this section, we present an example in which FEM is used for flow monitoring on the edge router. Fig. 4 is a simple diagram of network traffic occurring at any two nodes A and B. Node A represents outgoing traffic. The figure depicts different types of incoming traffic to node B through different ports. Port scans and SYN floods access any range of ports.

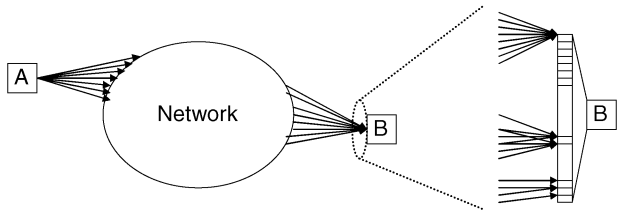

Fig. 4. Network traffic example.

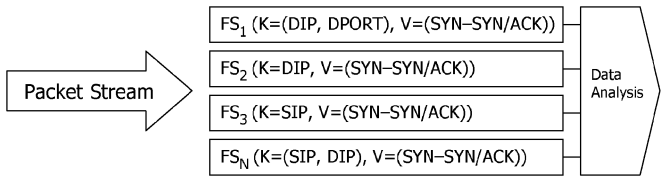

Fig. 5. Feature sketches executed in parallel.

If the FEM is placed at the host level, for example at A, the architecture is simple. Each node is aware of its location when processing network packets so the feature controller FC easily preserves connection ordering. However, when placing FEM at a router, additional logic is needed to preserve connection ordering. For example, when A and B communicate with each other, the source IP/port and destination IP/port fields in a packet are not consistent with the particular node which started the connection.

This example illustrates how to apply FEM to monitor network activity usually associated with SYN flood and port scans from a router's perspective. Each FEM consists of a number of FSs. For each FS, the key is denoted K and the feature value is denoted V. The source IP is designated SIP, destination IP DIP, source port SPORT, destination port DPORT, and protocol PROTO. The flags applicable for this case study are the SYN and ACK flags. We want to track the behavior associated with these two attacks.

First, it is known that SYN flood traffic is directed at a (DIP, DPORT) combination. Port scans are more flexible and use any combination of (DIP, DPORT). With an array of FSs, network behavior can be characterized for any given window of packets in a network stream. To monitor the behaviors of port scans and SYN floods, we propose the setup in Fig. 5.

Four FSs are accessed and updated in parallel with a stream of packets. Each FS monitors a different network characteristic. Our architecture favors FPGA implementation since the feature controller can be reprogrammed and easily placed back into the network without any modification to the core architecture. Section V-B details the FPGA implementation and performance of a FEM module with one FS. Since multiple FSs are accessed in parallel, the width of the FEM has a minor impact on performance.

FS_1 aids in SYN flood detection by monitoring the number of unserviced SYN requests for specific services. When a machine services SYN requests, it responds with a packet having the SYN and ACK flags set. For an SYN packet, a count value is incremented. For a SYN/ACK response, the count is decremented. By placing FS1 at an edge router, connection ordering relative to the DIP is easily preserved by checking the flags in the packet. All connections in FS1 are candidates for SYN floods and we denote this set as SYNFLOODset.

FS_2 monitors hosts with a large number of partially completed SYN requests. This activity indicates vertical scans or SYN floods. Notice FS_2 is a superset of FS_1. FS_2 contains all types of traffic at a particular IP. By querying both FSs with ESTIMATE, we can approximate the percentage of types of traffic at any DIP. Removing SYNFLOODset from FS_2 leaves candidates for vertical scans, VSCANset.

FS_3 observes the traffic from any SIP that causes incomplete SYN requests. This measure includes vertical, horizontal, and block scans. To differentiate this activity, FS_N is implemented to oversee the amount of traffic between any two hosts.

For an SIP ϵ FSN, if there is a DIP $\epsilon\, VSCAN_{\text{set}}$ and FS3 returns a value greater than a threshold (predetermined by other intrusion detection algorithms), we claim SIP_x is vertically scanning DIP_x. If not, SIP_x may be horizontally or block scanning on the network. Using both FS_3 and FS_N, we are able characterize additional network behavior.

The main difference between each FS is how the FC coordinates address each FS. As described, the flags SYN and ACK are used to intelligent configure each FEM. Nonetheless, our architecture is general enough to measure other network characteristics. Using SYN/FIN relationships for opening and closing network connections, it is possible to keep an FS updated with traffic flow sizes.

FEM can be employed at both the edge routers or on specific hosts. Our example contains extra logic for router implementation (connection ordering). Host implementation would actually be simpler because the perspective of network traffic is narrower.

IV. Principal Component Analysis

Once the features are extracted, the resulting values are fed into an outlier detection scheme in order to capture the attacks. Our outlier detection architecture is based on PCA. PCA is suitable for highly dimensional problems. It reduces the amount of dimensions required to classify new data. At its core, PCA produces a set of principal components, which are orthonormal eigenvalue/eigenvector pairs. In other words, it projects a new set of axes which best suit the data. In our implementation, these sets of axes represent the normal connection data. Outlier detection occurs by mapping live network data onto these "normal" axes and calculating the distance from the axes. If the distance is greater than a certain threshold, then the connection is classified as an attack. This section introduces PCA and begins with the notion of calculating distance in a high dimensional space.

A. Distance Calculation

Calculating distance from a point is a fundamental operation in outlier detection techniques. Methods include nearest-neighbor, k_{th} nearest neighbor, local outlier factor, etc. In general, the distance metric used is Euclidean distance. This is the primary calculation in the nearest neighbor approach. Let $\mathbf{x} = (x_1, x_2, \ldots, x_p)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_p)$ be two p-dimensional observations. The Euclidean distance is defined as

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})'(\mathbf{x} - \mathbf{y})}. \tag{1}$$

In (1), each feature carries the same weight in calculating the Euclidean distance. However, when features have a varied weight distribution or are measured on different scales, then the Euclidean distance is no longer adequate. The distance metric needs to be modified to reflect the distribution and importance of each field in the data. One of these metrics is known as the Mahalanobis distance

$$d^2(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})'\mathbf{S}^{-1}(\mathbf{x} - \mathbf{y}) \tag{2}$$

where $\mathbf{S}^{-1}$ is the sample covariance matrix.

B. PCA Methodology

Anomaly detection systems typically require more data than what is available at the packet level. Using preprocessing and feature extraction methods, the data available for anomaly detection is high dimensional in nature. The computational cost of processing massive amounts of data in real time is immense. Therefore, applying PCA as a data reduction tool while retaining the important properties of the data is useful. PCA works to explain the variance-covariance structure of a set of variables through a new set of orthonormal projection values which are linear combinations of the original variables. Principal components are particular linear combinations of p random variables $X_1, X_2, \ldots, X_p$. These variables have three important properties.

1) $X_1, X_2, \ldots, X_p$ are uncorrelated.
2) $X_1, X_2, \ldots, X_p$ are sorted in descending order.
3) $X_{\text{total}} = \sum_{i=0}^{p} X_i$ is the total variance that is equal to the sum of the individual variances.

These variables are found from eigenanalysis of the covariance or correlation matrix of the original variables $X_{o1}, X_{o2}, \ldots, X_{op}$ [17], [18].

Let the original data, in this case, the training data, $\mathbf{X}$ be an $\mathbf{n} \times \mathbf{p}$ data matrix of $\mathbf{n}$ observations with each observation composed of $\mathbf{p}$ fields (or dimensions) $X_1, X_2, \ldots, X_p$. In our work, we replaced the covariance matrix $\mathbf{S}^{-1}$ with the correlation matrix $\mathbf{R}^{-1}$ since many fields in the training set were measured on different scales and ranges. Using the correlation matrix more effectively represents the relationships between the data fields.

Let $\mathbf{R}$ be a $\mathbf{p} \times \mathbf{p}$ correlation matrix of $X_1, X_2, \ldots, X_p$. If $(\lambda_1, \mathbf{e}_1), (\lambda_2, \mathbf{e}_2), \ldots, (\lambda_p, \mathbf{e}_p)$ are the $\mathbf{p}$ eigenvalue-eigen-

vector pairs of the correlation matrix $\mathbf{R}$, then the ith principal component is

$$\begin{aligned} y_i &= \mathbf{e}_i'(\mathbf{x} - \overline{\mathbf{x}}) \\ &= e_i 1(x_1 - \overline{x}_1) + e_i 2(x_2 - \overline{x}_2) + \ldots \\ &\quad + e_i p(x_p - \overline{x}_p), \quad i = 1, 2, \ldots, p \end{aligned}$$

where

$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p \geq 0$;
$\mathbf{e}_i' = e_{i1}, e_{i2}, \ldots, e_{ip}$ is the ith eigenvector;
$\mathbf{x}' = (x_1, x_2, \ldots, x_p)$ is the observed data along the variables $X_1, X_2, \ldots, X_p$;
$\overline{\mathbf{x}}' = (\overline{\mathbf{x}}_1, \overline{\mathbf{x}}_2, \ldots, \overline{\mathbf{x}}_p)$ is the sample mean vector of the observation data.

The ith principal component has a sample variance λ_i and the sample covariance/correlation of any pair of principal components is equal to zero. This satisfies that PCA produces a set of independent variables. Thus, the total variance of a sample is the sum of all the variances accounted for by the principal components. The correlation between any two variables i and j is calculated by

$$\rho_{i,j} = \frac{\operatorname{cov}(X_i, X_j)}{\sigma_i \sigma_j} \tag{3}$$

where σ_x is the standard deviation of X_x over the sample data.

The principal components from the sample correlation matrix have the same properties as principal components from a sample covariance matrix. As all principal components are uncorrelated, the total variance in all of the principal components is

$$\lambda_1 + \lambda_2 + \ldots + \lambda_p = p. \tag{4}$$

The principal components derived from the covariance matrix are usually different from the principal components generated from the correlation matrix. When some values are much larger than others, then their corresponding eigenvalues have larger weights. Since the KDD cup data has many items with varying scales and ranges, the correlation matrix is utilized.

C. Applying PCA to Outlier Detection

This section outlines how PCA is applied as an outlier detection method. In applying PCA, there are two main issues: how to interpret the set of principal components, and how to calculate the notion of distance.

First, each eigenvalue of a principal component corresponds to the relative amount of variation it encompasses. The larger the eigenvalue is, the more significant its corresponding projected eigenvector should be. Therefore, the principal components are sorted from most to least significant. If a new data item is projected along the upper set of the significant principal components, it is likely that the data item can be classified without projecting along all of the principal components. In other fields, such as DSP and image compression and restoration, this is a useful property.

Second, eigenvectors of the principal components represent axes which best suit a data sample. If the data sample is the training set of normal network connections, then those axes are considered normal. Points which lie at a far distance from these axes would exhibit abnormal behavior. Outliers measured using the Mahalanobis distance are presumably network connections that are anomalous. Using a threshold value (t), any network connection with a distance greater than the threshold is considered an outlier. In our work, an outlier is implied to be an attack.

Consider the sample principal components $y_1, y_2, \ldots, y_p$ of an observation $\mathbf{x}$ where

$$y_i = \mathbf{e}_i'(\mathbf{x} - \overline{\mathbf{x}}), \quad i = 1, 2, \ldots, p.$$

The sum of squares of the partial principal component scores is equal to the principal component score

$$\sum_{i=1}^{p} \frac{y_i^2}{\lambda_i} = \frac{y_1^2}{\lambda_1} + \frac{y_2^2}{\lambda_2} + \ldots + \frac{y_p^2}{\lambda_p} \tag{5}$$

equating to the Mahanobolis distance of the observation $\mathbf{X}$ from the mean of the normal sample data set [17].

D. PCA Framework

All anomaly detections require an offline training or learning phase whether those methods are outlier detection, statistical models, or association rule mining. Many times, the mechanisms applied in the online and offline phases are tightly coupled. PCA, however, clearly separates the offline and online detection phases. This property is an advantage for hardware implementation. Another major advantage of PCA is its reduction of features. As we will show in the following sections, PCA effectively reduces the number of processed features from 40 to 8. This reduction linearly translates into area reduction in hardware and, hence, performance improvement. As a result, we can run our system at gigabit links. Fig. 6 outlines the steps involved in PCA.

In the offline phase, labeled training data are taken as input and a mean vector of the whole sample is computed. Ideally, these data sets are a snapshot of activity in a real network environment. Also, these data sets should contain only normal connections. Second, a correlation matrix is computed from the training data. A correlation matrix normalizes all of the data by calculating the standard deviation. Next, eigenanalysis is performed on the correlation matrix to extract independent orthonormal eigenvalue/eigenvector pairs. These pairs make up the set of principal components used in online analysis. Finally, the sets of principal components are sorted by eigenvalue in descending order. The eigenvalue is a relative measure of the variance of its corresponding eigenvectors. Using PCA to extract the most significant principal components is what makes it a dimensionality reducing method because only a subset of the most important principal components is needed to classify any new data.

To increase the detection rate, we use a modified version of PCA. In addition to using the most significant principal components (q) to find intrusions, we have found that it is helpful to look for intrusions along a number of least-significant components (r) as well. The most significant principal components are part of the major principal component score (MajC) and the less-significant components belong to calculating a minor principal component score (MinC). MajC is used to detect extreme

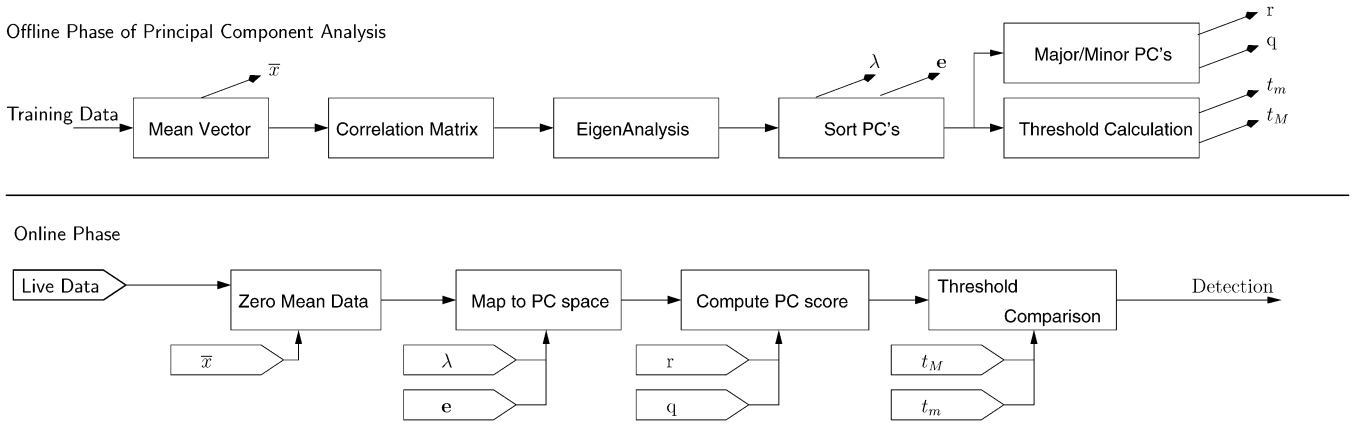

Fig. 6. PCA for network intrusion detection.

deviations with large values on the original features. These observations follow the correlation structure of the sample data. However, some attacks may not follow the same correlation model. MinC is used to detect those attacks. As a result, two thresholds are needed to detect attacks. If the principal components are sorted in descending order, then q is a subset of the highest values and r is a subset of the smallest components. The MajC threshold is denoted t_M while the MinC threshold is referred to as t_m. An observation $\mathbf{x}$ is an attack if

$$\sum_{i=1}^{q} \frac{y_i^2}{\lambda_i} > t_M \quad \text{or} \quad \sum_{i=p-r+1}^{p} \frac{y_i^2}{\lambda_i} > t_m. \tag{6}$$

The online portion takes q major principal components and r minor principal components and maps online data into the eigenspace of those principal components. There are two parallel pipelines—one for calculating the major component variability score (MajC) and one for the minor (MinC). The simulations show that adding the MinC pipeline increases the detection ability and decreases the false alarm rate of using PCA for anomaly detection. For hardware design, the most computationally expensive portion of PCA is performing eigenvector calculations and sorting. The process of calculating eigenvectors is sequential and difficult to parallelize. However, this task is part of the offline phase. We are primarily concerned with accelerating online intrusion detection using PCA. For this segment, the most important bottleneck is computing the PC score. Fortunately, this task can be parallelized as we describe in Section V-D.

E. NIDS Framework Implementation

The reprogrammability of FPGAs is an important advantage in our framework because our architecture tracks different types of network characteristics and modifies itself according to the selected features. The design can be implemented in a parameterizable ASIC which will provide better performance; however, there is no need for this extra performance improvement. Moreover, it would be extremely costly to develop a fixed architecture that tracks the same type of network characteristics. Using RocketIO multigigabit transceivers (MGT) available on new Virtex FPGA chips, it is possible to stream packets straight into the FPGA without suffering any slowdown. The RocketIO transceivers can be used in conjunction with gigabit Ethernet ports. As packets stream through the MGT in 1-,2-, or 4-B chunks, a state machine is used to extract the related header fields from the packet. The logic in the state machine is comprised of layer 2 (data-link layer) protocols and managing offsets to extract certain data fields of variable length (8, 16, 48 b).

The feature extraction modules hash tuples of values (e.g., SIP,DIP, SPORT, DPORT), but the packet data are streamed through the FPGA in specified chunks. Therefore, an asynchronous interface between the MGT and the feature extraction modules is required. These interfaces are highlighted on the system overview shown in Fig. 2. Once all of the required header fields are ready, handshaking is commenced and the data tuple is shipped off to the next state of processing.

In some rare cases, the number of cycles to determine whether a packet is malicious may exceed the time (in cycles) to process the whole just before it is routed. In this situation, the packet may get routed even before the intrusions are detected. However, this type of race condition will never occur in our architecture since it takes more cycles to process a complete packet than it takes to extract header fields (which are usually at the head of the packet) and ship them to the feature extraction module, which is a stall-free pipelined design. After a data tuple is finished being processed in the feature extraction module, it has the option of being characterized by using PCA or having its data sent out of the FPGA in the form of UDP packets to be processed using other software-based methods.

At the feature extraction/PCA boundary, there is no need for an asynchronous interface because the feature extraction modules run in parallel and, hence, FEM and PCA modules can be directly tied to each other. Therefore, they all complete at the same time and the results can be sent to the PCA in parallel as a feature vector. However, if the hash functions for different feature extraction modules are not identical, then an asynchronous interface is required.

Altogether, it is possible to implement this outlier detection framework featuring PCA on an FPGA using RocketIO transceivers. It is possible to also send data out the MGT as UDP

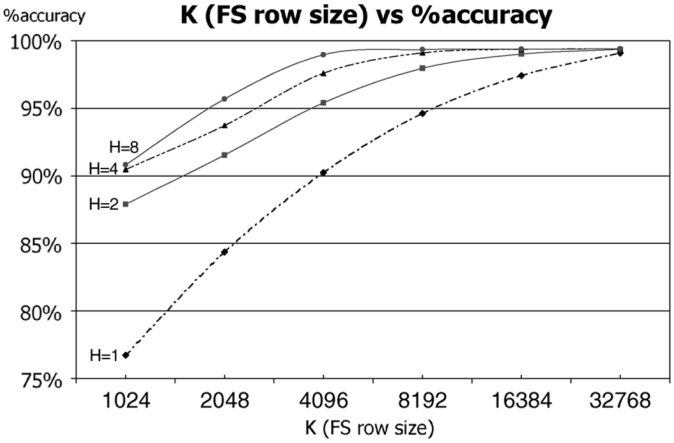

Fig. 7. Effect of FS row length(K) on accuracy.

packets for additional software processing. The interfaces between each level of processing allows for the insertion of more advanced feature extraction methods or other outlier detection algorithms.

V. Results

This section is divided into two main subsections presenting the FPGA implementation and simulation results of the FEM and PCA framework, respectively. First, we investigate the accuracy of using feature sketches by testing different FS sizes. For implementing FEM, we arbitrarily chose six days of traces from the 1999 DARPA Intrusion Detection Evaluation [24]. Half of the traces contain labeled attacks and the other half do not. Nonetheless, FS should accurately represent the network environment. Then, we examine the performance of our PCA architecture for outlier detection. We use the modified PCA for three reasons: to increase the detection rate, to decrease the false alarm rate, and because it can be tailored to FPGA design. The following subsections discuss them in detail.

A. Simulations Results for FEM

In this section, we investigate the accuracy of using feature sketches by testing different FS sizes. There are no known benchmarks specifically compiled for feature extraction, so we arbitrarily choose six days of traces from the 1999 DARPA Intrusion Detection Evaluation [24].

We simulate an FS ($\mathrm{K} = (\mathrm{SIP}, \mathrm{DIP}, \mathrm{DPORT}, \mathrm{SPORT})$, $\mathrm{V} = (\mathrm{SYN} - \mathrm{SYN/ACK})$). Our test on the FS is more intensive because more connections are simultaneously tracked. By virtue of design, FS is constantly updating; so we stream in 24 h of network activity and query the FS afterwards to compare the FS estimate with exact per-flow results.

Fig. 7 presents the accuracy of using an FS. H represents the number of rows in the FS and K represents the size of each row. The accuracy is measured as the percentage of precisely estimated flows (i.e., where the estimated value is equal to the actual value) out of all flows in the DARPA traces. The results of all six days are averaged together. For multiple hash function results ($\mathrm{H} > 1$), we use the Jenkins Hash with different seed values.

TABLE I
Constant Total $\mathrm{K} = 16384$ Entries

H	K	Accuracy
1	16384	97.4238%
2	8192	97.9699%
4	4096	97.6100%
8	2048	95.6835%

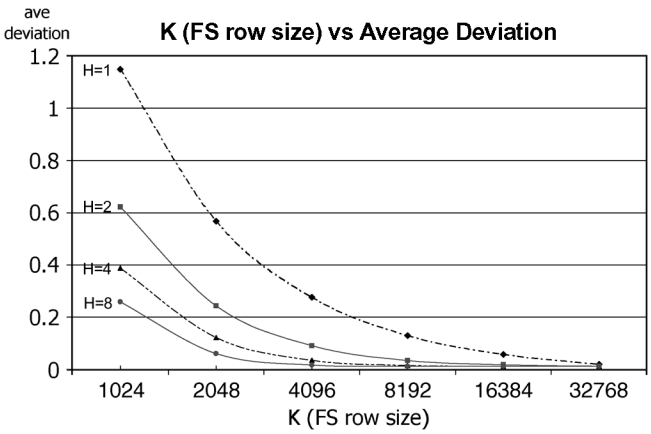

Fig. 8. Effect of FS row length (K) on average deviation.

When keeping K constant and increasing H, the accuracy improves. For example, with $\mathrm{H} = 1$, $\mathrm{K} = 2048$, the accuracy is 84.3%. With $\mathrm{H} = 2$ and $\mathrm{K} = 1024$, the accuracy increases to 87.8%. The 3.4% difference equates to 5586 more precisely estimated flows of the total 164 276 flows. However, in most cases, increasing K boosts accuracy more than increasing H. This is attributed to hash function limitations, such as poor scattering or lack of variability between different hash functions, or unavoidable collisions in small row size K (e.g., $\mathrm{H} = 8$, $\mathrm{K} = 1024$).

Table I represents an example of this behavior. The accuracy improves when increasing the number of rows until $\mathrm{H} = 8$, at which point the small K value limits the accuracy. Overall, however, the FS data structure ably satisfies accuracy demands. In Section V-B, we investigate how increasing H changes throughput and FPGA performance.

Fig. 8 reports another measure of the effectiveness of feature sketches, the average deviation of estimations from exact per-flow results. Clearly, increasing H improves estimation of, in this case, SYN-SYN/ACK values. This trend persists for other network behavior measures. As in Fig. 5, the gap between $\mathrm{H} = 1$ and $\mathrm{H} = 2$ is the largest. It shows that our datasets result in mostly two collisions. This fact favors more balanced FS configurations versus a one-row FS where collisions adversely affect the accuracy.

B. FPGA Implementation of FEM

FEM was implemented on a Xilinx VirtexII xc2v1000 chip. This member of the Virtex II family contains 5120 slices and 40 16-kb block random-access memory (RAM) modules. We used Synplify Pro 7.2.1 for logic synthesis and the Xilinx ISE 5.2i suite for placement and routing. For our hash function, the Jenkins Hash was extensively pipelined to operate at 270.6 MHz.

TABLE II
FEATURE SKETCH PLACE AND ROUTE

H	K	slices	Freq(MHz)	Throughput(Gbps)
1	8192	628	167.5	18.42
2	4096	1263	202.6	22.29
4	2048	2543	216.6	23.82
1	16384	634	169.3	18.62
2	8192	1265	190.1	20.99
4	4096	2543	193.2	21.25
1	32768	643	113.6	12.50
2	16384	1274	135.4	14.89
4	8192	2543	152.3	16.76

Table II contains the performance and area metrics for FEM implemented for edge routers. The performance results are similar for host-level implementation since the additional logic in the feature controller (FC) is not on the critical path of the FEM. We test configurations for $\mathrm{H} = 1$, 2, and 4. Throughput, clock frequency, and slices are reported for three overall row sizes $\mathrm{K} = 8192$, 16384, and 32768. The throughput value is calculated from the 5-tuple data {source IP, destination IP, source port, destination port, protocol} and the 6-b flag field is used to configure the FC.

It is clear that for a given memory size, increasing H increases throughput because it reduces the memory size and, hence, reduces the access times. Similarly, for a constant H, reducing the total memory amount (K) also increases the throughput. Among the simulated configurations, the best throughput of 23.81 Gb/s is achieved for $\mathrm{H} = 4$ and $\mathrm{K} = 2048$. However, note that this configuration has a relatively low accuracy of 94.1%. Hence, when one considers the "$\mathrm{accuracy} \times \mathrm{throughput}$" product, the best configuration is $\mathrm{H} = 4$ and $\mathrm{K} = 4096$, which can extract information at 21.25 Gb/s.

Note that the increase in the number of slices is mostly a result of using multiple hash functions in parallel. Replicating the hash functions allows higher throughput and frequency at the expense of area. If there are area constraints, however, one could use one hash function implementation for multiple FS rows, providing the values to each of them at consecutive cycles. This would result in decreased throughput but also a reduced area requirement. Since the Jenkins Hash is pipelined, mapping a hash function to multiple rows would not introduce extra-long delays.

In conclusion, the simulations show that feature sketches are effective data structures for network behavior characterization. The simulation results demonstrate the gains in accuracy and estimation ability of feature sketches. FPGAs take advantage of multiple FS rows to satisfy gigabit throughput demands. Consequently, feature sketches, the main components of FEM, are attractive data structures for FPGAs to exploit parallelism.

C. Simulation Results for PCA

To measure the effectiveness of PCA, we use data sets from the KDD Cup 1999 repository [19], used for the Third International Knowledge Discovery and Data Mining Tools Competition. Both training and testing data sets are provided. The training data sets contain records of network connections labeled either as normal or attack. Each connection record is made up of 41 different features related to the connection. The 41 features are divided into three categories: basic features of TCP connections, content features of the connection, and traffic features which are derived using a 2-s time window to monitor the relationships between connections. The traffic-level features encompass the same service and same host information, such as the number of connections in the past 2 s that have the same destination host as the current connection. These three categories correspond to the levels of data abstraction outlined in Fig. 1. We only use 28 of the 41 features in our study. Seven features among the remainder were symbolic and the rest of the features contained only zero values. Thus, the remainder of the features have no impact on the behavior characteristic of the data set.

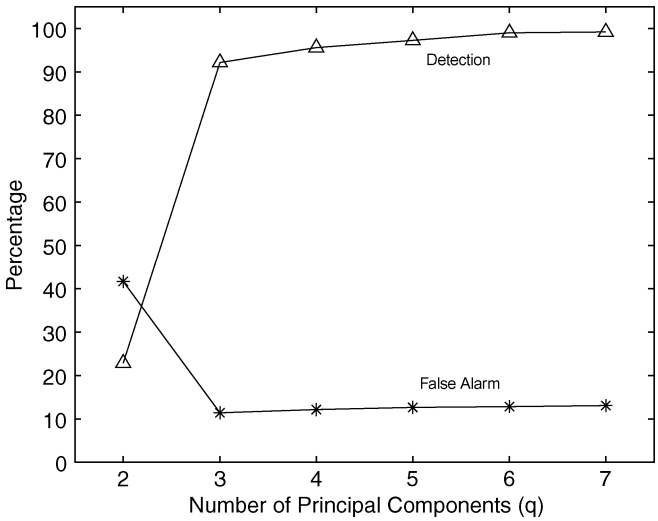

Fig. 9. Detection and false alarm rate versus q.

For these simulations, no distinctions were made between different types of attacks. Either a connection record exhibited normal properties or its behavior was noticeably different from the behavior expected from PCA. In this way, PCA was applied as an independent statistical mechanism for anomaly detection.

For the training phase of PCA, we use a provided 10% subset data file. This subset was stripped of all attack connections and chunks of 5 000 normal connections were extracted as training data for PCA. We used five training files (kddc25, kddc40, kddc55, kddc70, kddc75), each having 5 000 normal connections. For the testing phase, we used a provided test file which contained 10% of all testing data. In the first round of tests, we averaged together the detection rates of five training sets against the 10% testing set. The testing set had 311 029 connections out of which 250 436 were attack connections. Even though the ratio of attack to normal connections is high, Fig. 9 shows that PCA detects a high percentage of attacks with a low false alarm rate. In this graph, the MajC pipeline uses between one and seven principal components. With $q = 3$, PCA is already able to detect 92.2% of attacks, whereas with $q = 7$, PCA can detect 99.2% of attacks. The results indicate PCA effectively reduces the dimensionality of multivariate problems. Our experiments contained 28 original variables but only seven principal components are required to detect 99.2% of the attacks. In other words, 21 less variables are needed to classify any new data point which reduces the computational cost significantly.

TABLE III
VARIATION (%) VERSUS Q

filename	q=2	q=3	q=4	q=5	q=6	q=7
kddc25	27.57	37.54	45.36	59.41	59.13	64.42
kddc40	32.06	40.85	47.63	53.75	58.79	63.72
kddc55	25.06	33.43	40.79	47.48	53.75	59.38
kddc70	31.52	42.46	52.22	58.75	65.13	69.74
kddc75	33.28	43.33	51.86	58.20	64.07	68.09

TABLE IV
DETECTION AND FALSE ALARM RATES UTILIZING MinC AND MajC PIPELINES

Training File	Test File	q (Major)	r (Minor)	Detection	False Alarm
kddc25	test1	3	–	75.56%	10.27%
		3	4	98.19%	5.22%
		3	5	99.99%	2.25%
		5	–	91.94%	10.52%
		5	4	97.81%	5.22%
		5	5	99.95%	2.28%
		7	–	99.91%	9.55%
kddc70	testB	3	–	62.60%	12.50%
		3	4	99.98%	3.28%
		3	5	99.90%	2.12%
		5	–	83.23%	14.23%
		5	4	98.50%	6.39%
		5	5	99.96%	2.91%
		7	–	98.74%	5.21%
kddc55	test4	3	–	92.89%	12.11%
		3	4	99.50%	4.18%
		3	5	99.86%	2.01%
		5	–	96.21%	11.21%
		5	4	98.19%	5.17%
		5	5	99.96%	1.95%
		7	–	99.93%	10.50%

Table III shows how the variation of the data depends on q, the number of principal components. For example, with $q = 2$, about 30% of the data variation is captured. Intuitively, increasing the number of principal components places a tighter bound on the amount of variation that PCA can account for.

Table IV shows the impact of adding the MinC pipeline to PCA. The testing data sets in this table each have between 100 000 to 125 000 network connections randomly extracted from the testing data. The training files are the same ones used in Table II. Unlike the 80.5% attack distribution in the full testing set, these data sets contain 30% to 35% attacks for our experiments.

In addition to classifying network connections based on their major principal component score (MajC), another score based on the minor principal components (MinC) was calculated [31]. The number of major principal components (q) accounts for the majority of the data's correlation structure while the minor principal components (r) account for a small portion of the variation. This way, when attacks are detected, there is additional information if attacks do not conform to the normal correlation structure. For this study, we used minor components with eigenvalues less then 0.20.

In most cases, adding the MinC pipeline for network connection boosts the detection rate and decreases the false alarm rate. In the case of $(q, r) = (3, 5)$, the average detection rate in Table IV is 99.92% and the average false alarm rate decreases to 2.13%. In some cases, (3, 5) performs better than (5, 5). This is due to the random distribution of attacks and normal connections. By including more components, we may actually miss an attack, because different configurations will have different threshold values. Some normal attacks may seem like attacks and vice-versa. However, in general, we see that increasing q increases the detection rate.

For completeness, we also include the case of $(q, r) = (7, \mathrm{x})$ indicating no MinC pipeline. This configuration increases the detection rate but the average false alarm rate is 8.42%. So by watching for anomalous behavior along two subsets of the principal components, attacks can be recognized along two different sections of the same correlation structure and thereby increase the detection rate and decrease the false alarm rate.

D. FPGA Implementation of PCA

For the FPGA implementation, a single principal component score pipeline was implemented to study the impacts of parallelizing PCA. The target device XC2VP100 (speed grade: -5) was chosen from the Xilinx Virtex II Pro family [38]. This is one of the larger platforms of the family containing 444 18 $\times$ 18 block multipliers, 44 096 slices, and 444 18-kb block SelectRAM+ blocks. The Virtex-II Pro platform FPGAs also provide up to two PowerPC 405 32-b RISC integrated cores on a single device.

Synplify Pro 7.2 was used for synthesis and Xilinx ISE 5.2i for place and route statistics. To examine the area and performance of PCA in real time, we implement the online portion of the principal component score pipeline (PCSP) as shown in Fig. 10. We implement PCSP having an input data $\mathbf{X}$ with four and eight 32-b data fields. Also, we vary the number of principal components to calculate a principal component score between four and eight. This workload is feasible for a real-world implementation of PCA. In our simulations, each input data contained 28 fields for which we extracted 2 to 7 principal components.

There are many levels of parallelism to exploit in the PSCP pipeline. They are depicted in the dashed line boxes in Fig. 10. First of all, subtracting the mean vector $\overline{x}$ from the input data is done in parallel. If each data tuple has $\mathbf{p}$ fields $[\mathbf{X} = (x_1, x_2, \ldots, x_p)]$, then $\mathbf{p}$ operations are performed in parallel. The next phase for PCA is calculating the partial component scores (parC). The element-by-element multiplication, using fixed-point arithmetic, is performed in parallel. This operation maps the new data along each principal component axis. The first summation is specific to calculating the Mahalanobis distance of the new data from the axes. This is accomplished with an adder tree that scales with the depth of the adder tree $[\log_2(p)]$. The result is then squared and divided by the eigenvalue of the ith principal component. The next step is the summation of all parC scores using another adder tree. This scales logarithmically with the number of principal components (q or r) designated. Finally, the principal component score is compared with a threshold value (t_M or t_m) determined in offline processing. If both the MajC and MinC pipelines are used, then both threshold values will be used. The MajC and MinC pipelines have the exact same design as in Fig. 10. The only difference in the two are the thresholds used (t_M versus t_m), the number of principal components used (q versus r).

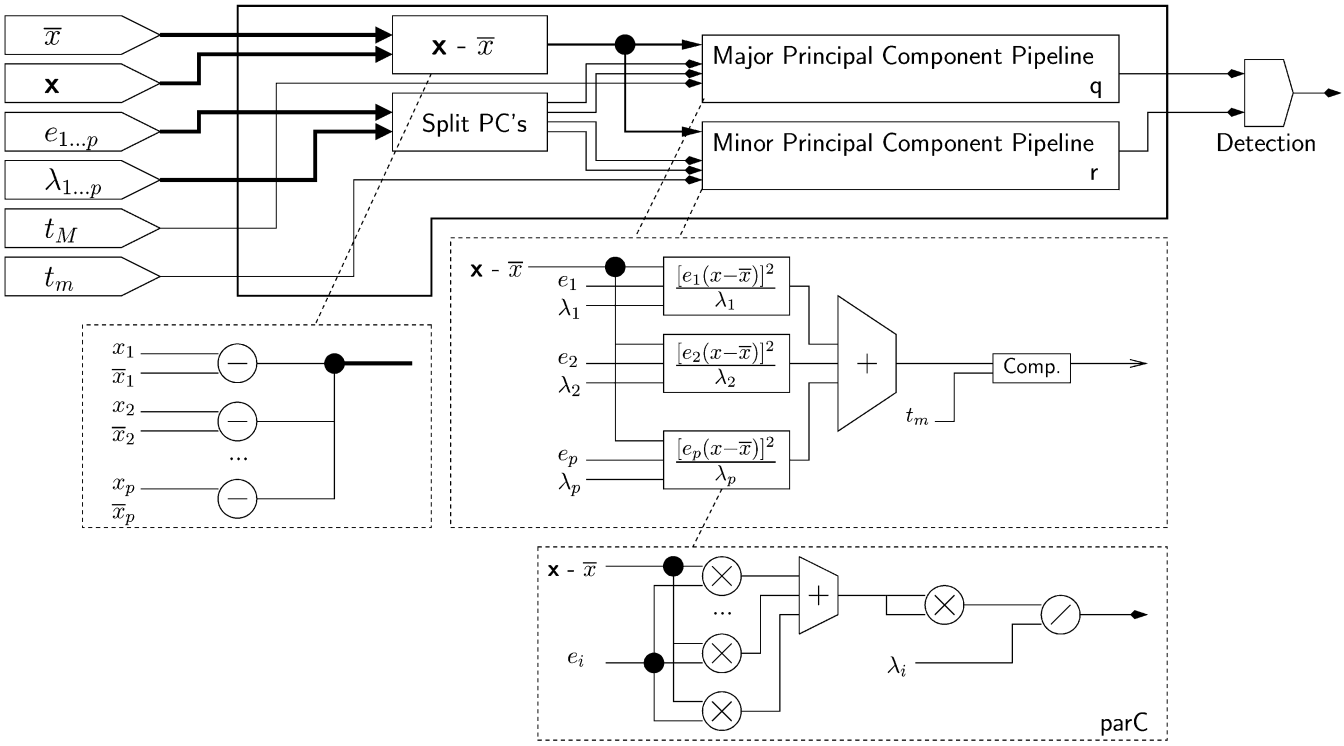

Fig. 10. PCSP pipeline for FPGA.

TABLE V
PIPELINE STAGES BREAKDOWN [$z = \max(q, r)$]

Operation	# pipeline stages
a = **x** - $\overline{x}$	1
b = e_i(**x** - $\overline{x}$)	4
c = $\sum_{i=0}^{p} b_i$	$\log_2(p)$
d = c^2	4
e = $\frac{d}{\lambda}$	34
f = $\sum_{i=0}^{q} e_i$	$\log_2(z)$
outlier = f $\leq$ threshold	1
Total # pipeline stages	44 + $\log_2(p)$ + $\log_2(z)$

Either or both of MajC and MinC pipelines detect intrusions using one correlation model from the PCA. Attacks are detected on two portions of the correlation structure. As the simulations show, this method increases the detection rate and decreases the false alarm rate. From a hardware perspective, the choice of q and r affects the number of pipeline stages required. In Table V, which shows the number of pipeline stages needed for each operation in PCSP, $\max(q, r)$ can be substituted in for z in the table.

It should be noted that any change in the MajC/MinC attributes would require the FPGA to be updated. During these updates, the operation of the FPGA will be halted. However, such changes are extremely rare, and may take place once in a day, month, or even year. Note that the eiegenvectors and eigenvalues can be changed dynamically since their sizes do not change for the given configuration.

E. FPGA Performance of PCA

Table VI shows the place and route statistics with Xilinx ISE 5.2i. We examine the throughput possible with different configurations of the PSCP pipeline. The number of principal components (q) is the number of parallel parC's to sum up on the FPGA. The number of fields (p) is the number of 32-b fields used to calculate the throughput of the PSCP pipeline. The #mult field is the number of 18 $\times$ 18-b block multipliers used.

Parallelism is utilized at four levels: at subtracting the mean vector, element by element multiplication, summation of q parC's, and the summation in calculating the parC score. The summation tasks have the most impact on throughput since those tasks take a variable amount of pipeline stages. Figs. 11 and 12 show the impact on FPGA throughput for varying the number of principal components (q) or varying the size of the input data (p).

In Fig. 11, each line corresponds to data with eight, six, or four fields of 32 b each (P8, P6, P4). The lines from top to bottom are P8, then P6, and P4 at the bottom. First, it is clear that for any q, increasing the workload (p) increases the throughput. For example, with $q = 4$, P8 has the highest throughput at 23.76 Gb/s, then P6 with 16.87 Gb/s, and finally P4 with 10.47 Gb/s. This is due to exploiting the parallelism in the summation to calculate a parC score. The best throughput results are with q $= 1$; however, this is not a good configuration in terms of detection quality. A more relevant load would be with $q = 4$, where P8 shows how to achieve the highest throughput. This is partially attributed to full usage of adder trees in the pipeline. When an adder tree is fully populated, registers are used when no operations are taking place on a partial sum. Increasing q after this point, decreases the link capacity as the buses involved in the pipeline begin to be very wide and the cost of communication and coordination leverages the advantages of parallel design. Another characteristic to note in Fig. 11 is that, in general, increasing q for any constant workload p decreases the throughput. This is explained by the adder tree at which the parC scores are summed up. At this point, the 32-b values have increased to 128-b values and the large amount of bandwidth

TABLE VI
XILINX ISE 5.2I PLACE-AND-ROUTE STATISTICS (XC2VP100)

# q	p (#fields)	#mult.	slices	freq (MHz)	Data Throughput (Gbps)	Pipeline Stages	Overall Latency (us)
1	4	32	2605	134.59	17.23	47	.3492
1	6	40	2979	133.32	25.60	48	.3600
1	8	48	3225	136.07	34.83	48	.3527
3	4	96	7679	67.55	8.65	48	.7105
3	6	120	8669	73.96	14.20	49	.6625
3	8	144	9275	78.77	20.17	49	.6221
4	4	128	10152	81.81	10.47	48	.5867
4	6	160	11450	87.88	16.87	49	.5576
4	8	192	12236	92.82	23.76	49	.5279
6	6	240	17140	67.57	12.97	49	.7251
6	8	288	18286	75.83	19.41	50	.6593
8	8	384	24208	56.50	14.46	50	.8849

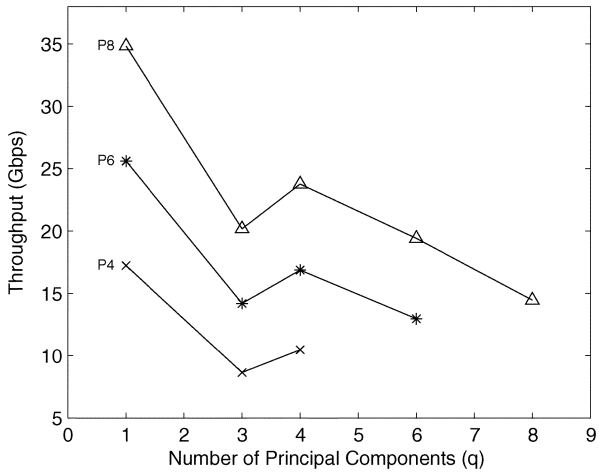

Fig. 11. Throughput versus $q(p = \mathrm{k})$.

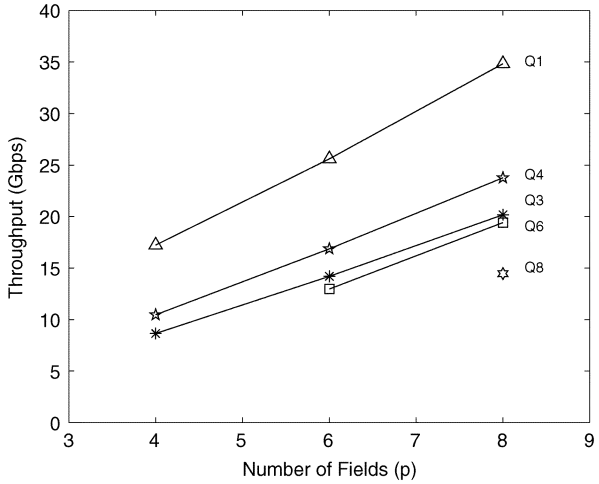

Fig. 12. Throughput versus $p(q = \mathrm{k})$.

and coordination may inhibit the advantages of parallelization. Also, the number of stages required to perform the summation also varies by $\log_2(q)$. For example, with $q = 1$, there is no need for an adder tree. For $q = 2$ to $q = 4$, two pipeline stages are required.

In Fig. 12, the throughput is analyzed for a constant q (Q1, Q3, Q4, Q6, Q8) to compute a principal component score (MajC or MinC) and varying the workload with p ranging from four to eight fields. In other words, we keep the number of pipeline stages in the second summation (see Table V) constant. The second summation is an intense operation from a bandwidth perspective. As the data travel through the pipeline, an initial 32-b value gets multiplied twice and becomes 128 b wide. We use this pipeline configuration to preserve the accuracy in fixed-point arithmetic. The overall throughput increases moving from Q3 to Q4 but then decreases afterwards in Q6 and Q8. The reason for this behavior is the large bandwidth of the second summation.

When increasing the workload p, the throughput increases linearly. With $q = 1$, increasing p from 4 to 8 increases the throughput from 17.23 to 34.83 Gb/s. For $q = 4$, the throughput increases from 10.47 to 23.76 Gb/s. We see that increasing p exploits the pure parallelism in subtracting the mean vector and the element-by-element multiplication. In Fig. 11, those two levels of utilizing parallelism are held constant and, thus, do not affect the trend in throughput. Again, it seems that having an adder tree accepting four inputs performs the best. Nonetheless, in maximizing the precision with fixed-point operations on 32-b data, this design still delivers high throughput levels that are possible to support more than 10-Gb/s link speeds. Our implementation handles feature processing with PCA and supports a data throughput of 23.76 Gb/s for $(q, r) = (4, 8)$. These results indicate that our design can be effectively used for network intrusion detection using features as input. In addition, since it achieves a high throughput level, it can be used for packet-by-packet processing (e.g., with 40-B packets, the same configuration can support up to 29.70 Gb/s).

Overall, our design for PCA exploits parallelism on multiple levels. First, MajC and MinC scores are calculated in parallel. Second, within each pipeline, element-by-element matrix operations execute concurrently. The structure and layout of FPGAs lend well to matrix operations. And third, subtracting the mean from any new data tuple is performed outside the pipeline. The results from this operation are distributed to the MajC and MinC pipelines in a data parallel manner. Clearly, FPGAs are well suited for our implementation of PCA. For a representative workload, our implementation outputs at a link speed of 23.76 Gb/s; enough to support gigabit line rates.

Note that we have not considered FPGA vulnerability in our design. Although NIDS hardware is susceptible to external attacks, FPGA boards are harder to attack than the rest of the system, since they are programmed locally.

VI. Related Work

Previous works show that many networking applications have found their way into hardware implementations [39]. For example, FPGAs have been used in developing platforms for the experimentation of active networks [12] for services, such as detection of denial-of-service (DoS) attacks, real-time load balancing for e-commerce servers, real-time network-based speed recognition servers for v-commerce, etc. Also, high-speed front-end filters and security-management applications for ATM firewalls have found their way onto FPGAs to reduce performance penalties at the IP level [23]. Also, the TCP/IP splitter [29] has been implemented as part of the field-programmable port extender (FPX) project to perform flow classification, checksums, and packet routing, but this implementation is limited to 3-Gb/s monitoring. Prior research has been conducted in this area to develop a flow-size monitor similar to FEM [21]. However, the design was not capable of being updated when connections were completed. This limitation prevented achieving an accurate representation of the network. Nguyen *et al.* have proposed a hardware architecture for FPGAs that is effective in capturing network characteristics and can handle gigabit throughput [28]. Other studies [14] agree that per-flow methods will not suffice and propose intelligent algorithms and multistage filters using multistage hash tables to increase accuracy over Cisco's NetFlow (which uses sampling to characterize network traffic).

In anomaly detection, two prominent methods are used. The first type is based on a set of rules or specifications of what is regarded to as normal behavior while the second type learns the behavior of a system under normal operation. The first relies on manual intervention and is essentially a short extension of signature detection-based IDSs. Rule-based intrusion detection systems, such as Bro [8] or Snort [33], use known rules to identify known attacks, such as requests for nonexistent services or virus attack signatures or malicious strings in packet payloads. Anomaly detection systems, such as ALAD [22], SPADE [35], and NIDES [3] compute statistical models for normal network traffic and generate alarms when there are large differences from the normal model. These methods apply statistical tests to determine whether the observed activities greatly deviate from the normal profile. These statistical-based schemes assume some sort of multivariate distribution of data points. The Canberra technique is another multivariate statistical metric for anomaly detection. This method does not suffer from assumptions about data distribution [13]. Yet this technique does not perform well unless network attacks occur in bunches. In the case of port scan malicious activity, which occurs over a long period of time, the Canberra technique may not be as effective as it would be for SYN flood and DoS attacks.

Many reconfigurable architectures have been implemented for intrusion detection. Baker and Prasanna were able to implement a version of the Apriori [2] algorithm using systolic arrays [6] and also look into efficient pattern matching [5] as a signature-based method. Sidhu and Prasanna also implemented a pattern matching architecture for FPGAs [32]. Attig and Lockwood proposed a framework for rule processing on FPGAs [4]. Many packet processing architectures for FPGA have been implemented. The scope of these applications ranges from string matching, payload processing, packet classification, and TCP flow processing [11], [30], [34].

The Karhunen–Lo'eve Transform, which uses the concepts of PCA, has been mapped to FPGAs in the past [15] for use with multispectral imagery suitable for remote-sensing applications. However, this application does not decouple the eigenanalysis step from the main PCSP pipeline which we accelerate for network intrusion detection. Former works show that PCA has been used as an outlier detection method in NIDSs [27].

Other systems employ outlier detection methods, such as local outlier factor (LOF) [7], which is a density-based approach, or kth nearest neighbor by calculating distances between connections. Aggarwal and Yu [1] studied the behavior of projections from a data set to find outliers. Lazarevic, *et al.* performed a survey on multiple outlier detection schemes, including nearest neighbor and LOF on the 1998 DARPA data set [9] and showed that LOF performs favorably as a density-based approach [21]. The 1998 DARPA data set is composed of tcpdump traces over four weeks of simulated network activity including network attacks. Each connection record in the 1998 DARPA data set is comprised of many variables extracted using tcpdump [36] and tcptrace [37] utilities.

Shyu, *et al.* [31] uses PCA on the KDD CUP 1999 data set [19]. We use a similar PCA methodology as Shyu and, in our tests, we verify that PCA is an effective outlier detection method for network intrusion detection. PCA compares favorably to LOF, nearest-neighbor, and the Canberra metrics and achieves high detection rates with low false alarm rates. For any false alarm rates, PCA is still detects up to 99% of attacks. However, these studies did not consider hardware implementations of the algorithms or their applicability to hardware implementation.

VII. Conclusion

Future generation network intrusion detection systems will most likely employ both signature detection and anomaly detection modules. Anomaly detection methods process a large amount of data in order to recognize anomalous behavior or new attacks which signature detection cannot. The previous work mostly concentrated on accelerating signature detection techniques. However, hardware implementations of anomaly detection methods have not been proposed. Some reasons include the complexity and high computational cost associated with these algorithms. In any case, current software methods fail to keep up the high-link speeds. Signature detection can be performed live, but live anomaly detection requires a comprehensive picture of the network environment. Our feature extraction module provides this functionality using feature sketches, which map well onto reconfigurable hardware. Many network behavior parameters can be monitored using our architecture by making small modifications to the design. These characteristics include flow size, number of open connections, number of unserviced connection requests, etc. For the intrusion detection part, we have used PCA as an effective way of outlier analysis. PCA is particularly useful because of its ability to reduce data dimensionality into a smaller set of independent variables from which new data can be classified. We used a modified version of PCA

to scan for strange behavior on two regions of a single correlation structure. As a result, PCA can detect up to 99% of attacks and only suffer a 1.95% false alarm rate for the KDD Cup 1999 data sets. A general hardware architecture was proposed for FEM and the online portion of PCA and implemented on the Xilinx Virtex-II family of FPGAs. In order to increase the throughput of the whole system, pipelining and inherent parallelism FPGAs were used extensively. Parallelism was exploited for many element-by-element matrix operations and summations were achieved through adder trees. For a constant number of principal components (q), increasing the data input size (p) also increased the throughput of the system. Our architecture for FEM shows a high percentage (97.61%) accuracy with a very low estimation error (0.0365 packets), thus making the overall throughput as high as 21.25 Gb/s for a 16-K entry FEM. The PCA part of our design clocks at 92.82 MHz on a Virtex-II Pro and achieves 23.76-Gb/s data throughput and would be able to support 29.07 Gb/s for 40-B packets.

ACKNOWLEDGMENT

The authors would like to thank Prof. S. Öğrenci Memik for her helpful comments and suggestions pertaining to this work.

REFERENCES

[1] C. C. Aggarwal and P. S. Yu, "Outlier detection for high dimensional data," presented at the ACM SIGMOD Conf., Santa Barbara, CA, May 2001.
[2] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *Proc. ACM SIGMOD Int. Conf. Management Databases*, 1993, pp. 207–216.
[3] D. Anderson, T. Lunt, H. Javits, A. Tamaru, and A. Valdes, "Detecting unusual program behavior using the statistical components of NIDES," May 1995.
[4] M. E. Attig and J. Lockwood, "A framework for rule processing in reconfigurable network systems," presented at the IEEE Symp. Field-Programmable Custom Computing Machines, Napa, CA, Apr. 2005.
[5] Z. K. Baker and V. K. Prasanna, "Time and area efficient pattern matching on FPGAs," presented at the ACM Int. Symp. Field-Programmable Gate Arrays (FPGA), Monterey, CA, 2004.
[6] Z. K. Baker and V. K. Prasanna, "Efficient hardware data mining with the apriori algorithm on FPGAs," presented at the IEEE Symp. Field Programmable Custom Computing Machines, Napa, CA, 2005.
[7] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," presented at the ACM SIGMOD Conf., Dallas, TX, May 2000.
[8] Bro, Bro Intrusion Detection System 2002.
[9] DARPA Intrusion Detection Evaluation. [Online]. Available: http://www.ll.mit.edu/IST/ideval. 1998
[10] S. Dharmapurikar, M. Attig, and J. W. Lockwood, "Design and implementation of a string matching system for network intrusion detection using FPGA-based bloom filters" 2004.
[11] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. W. Lockwood, "Deep packet inspection using parallel bloom filters," presented at the Symp. High Performance Interconnects, Stanford, CA, Aug. 2003.
[12] A. Dollas, D. Pnevmatikatos, N. Asianides, S. Kawadias, E. Sotiriades, S. Zogopoulos, and K. Papademetriou, "Architecture and applications of PLATO, a reconfigurable active network platform," presented at the IEEE Symp. Field-Programmable Custom Computing Machines, Rohnert Park, CA, 2001.
[13] S. M. Emran and N. Ye, "Robustness of Canberra metric in computer intrusion detection," presented at the IEEE Workshop on Information Assurance and Security, West Point, NY, 2001, U.S. Military Academy.
[14] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," presented at the ACM SIGCOMM Conf. Applications, Technologies, Architectures, Protocols for Computer Communication, Pittsburgh, PA, 2002.
[15] M. Fleury, B. Self, and A. C. Downton, "A fine-grained parallel pipelined Karhunen-Loeve transform," presented at the Int. Parallel and Distributed Processing Symp., Nice, France, Apr. 2003.
[16] B. Jenkins, Jenkins, Hash Functions and Block Ciphers.
[17] J. D. Jobson, *Applied Multivariate Data Analysis, Volume II: Categorical and Multivariate Methods*. New York: Springer-Verlag, 1992.
[18] I. T. Jolliffe, *Principal Component Analysis*. New York: Springer-Verlag, 2002.
[19] KDD Cup 1999 data. [Online]. Available: http://www.kdd.ics.uci.edu/databases/kddcup99/kddcup-99.html. Aug. 1999
[20] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch based change detection: Methods, evaluation, and applications," presented at the ACM SIGCOMM Internet Measurement Conf., Miami, FL, 2003.
[21] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava, "A comparative study of anomaly detection schemes in network intrusion detection," presented at the SIAM Conf. Data Mining, Minneapolis, MN, May 2003.
[22] M. V. Mahoney and P. K. Chan, "Learning nonstationary models of normal network traffic for detecting novel attacks," presented at the ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, AB, Canada, Jul. 2002.
[23] J. McHenry, P. W. Dowd, F. A. Pellegrino, T. M. Carrozzi, and W. B. Cocks, "An FPGA-based coprocessor for ATM firewalls," presented at the IEEE Symp. FCCM, Napa, CA, Apr. 1997.
[24] MIT Lincoln Laboratory, DARPA Intrusion Detection Evaluation.
[25] S. Muthukrishnan, Data streams: Algorithms and applications 2003.
[26] NetFilter/IPtables: Firewalling, NAT and Packet Mangling for Linux 2.4.
[27] D. Nguyen, A. Das, G. Memik, and A. Choudhary, "A reconfigurable architecture for network intrusion detection using principal component analysis," presented at the IEEE Symp. Field-Programmable Custom Computing Machines, Napa, CA, Apr. 2006.
[28] D. Nguyen, G. Memik, S. Memik, and A. Choudhary, "Real-time feature extraction for high speed networks," presented at the Int. Conf. Field Programmable Logic and Applications, Monterey, CA, 2005.
[29] D. V. Schuehler and J. W. Lockwood, "TCP splitter: A TCP/IP flow monitor in reconfigurable hardware," presented at the Hot Interconnects 10 (HotI-10), Stanford, CA, 2002.
[30] D. V. Schuehler, J. Moscola, and J. W. Lockwood, "Architecture for a hardware-based, TCP/IP content-processing system," *IEEE Micro.*, vol. 24, no. 1, pp. 62–69, Jan./Feb. 2004.
[31] M. Shyu, S. Chen, K. Sarinnapakorn, and L. Chang, "A novel anomaly detection scheme based on principal component classifier," in *Proc. IEEE Foundations New Directions of Data Mining Work., in Conjunction With 3rd IEEE Int. Conf. Data Mining*, 2003, pp. 172–179.
[32] R. Sidhu and V. Prasanna, "Fast regular expression matching using FPGAs," presented at the IEEE Symp. Field-Programmable Custom Computing Machines, Rohnert Park, CA, Apr. 2001.
[33] SNORT, SNORT: The open source network intrusion detection system 2002.
[34] H. Song and J. W. Lockwood, "Efficient packet classification for network intrusion detection using FPGA," presented at the Int. Symp. Field-Programmable Gate Arrays, Monterey, CA, Feb. 2005.
[35] SPADE, Stealthy Portscan Intrusion Correlation Engine 2002.
[36] Tcpdump Utility. [Online]. Available: http://www.tcpdump.org.
[37] Tcptrace Utility. [Online]. Available: http://www.jarok.cs.ohiou.edu/software/tcptrace/index.html.
[38] Xilinx Virtex-IIPro-Datasheet [Online]. Available: http://www.direct.xilinx.com/bvdocs/publications/ds083.pdf.
[39] V. Yegneswaran, P. Barford, and J. Ullrich, "Internet intrusions: Global characteristics and prevalence," presented at the ACM SIGMETRICS, San Diego, CA, 2003.

Abhishek Das (S'06) received the B.Tech. (Hons.) degree in computer science and engineering from the Indian Institute of Technology, Kharagpur, India, in 2005 and is currently pursuing the Ph.D. degree in electrical engineering and computer science at Northwestern University, Evanston, IL.

Currently, he is a Graduate Research Assistant in the Center for Ultra Scale Computing and Information Security at Northwestern University. In 2007, he was with the Digital Enterprise Group (DEG) at Intel Corporation, Hillsboro, OR, where he worked on platform security of Intel's business platforms. His research interests include power-aware and reliable computer architecture, reconfigurable computing, and hardware–software co-design.

David Nguyen received the B.S. degree in computer engineering from Northwestern University, Evanston, IL, in 2002 and the M.S. degree in computer engineering from Northwestern University in 2005.

Currently, he is with SanDisk Corp. Milpitas, CA, a consumer electronics storage solutions company specializing in NAND flash-storage technology, focusing on modeling and evaluating new system controller features as well as NAND memory features. Previously, he was with Sandia National Laboratories, working in the network security group designing FPGA-based network security solutions in 2005. His research interests are in application-specific architectures with an emphasis on embedding security and driving performance and reliability in NAND flash storage. He is the author of various conference publications in reconfigurable network security.

Mr. Nguyen was the recipient of the Walter P. Murphy Fellowship in 2002.

Joseph Zambreno (M'02) received the B.S. degree (Hons.) in computer engineering in 2001, the M.S. degree in electrical and computer engineering in 2002, and the Ph.D. degree in electrical and computer engineering from Northwestern University, Evanston, IL, in 2006.

Currently, he is an Assistant Professor in the Department of Electrical and Computer Engineering at Iowa State University, Ames, where he has been since 2006. His research interests include computer architecture, compilers, embedded systems, and hardware/software co-design, with a focus on run-time reconfigurable architectures and compiler techniques for software protection.

Dr. Zambreno was a recipient of a National Science Foundation Graduate Research Fellowship, a Northwestern University Graduate School Fellowship, a Walter P. Murphy Fellowship, and the Electrical Engineering and Computer Science Department Best Dissertation Award for his Ph.D. dissertation "Compiler and Architectural Approaches to Software Protection and Security."

Gokhan Memik (M'03) received the B.S. degree in computer engineering from Bogazici University, Istanbul, Turkey, in 1998 and the Ph.D. degree in electrical engineering from the University of California at Los Angeles (UCLA) in 2003.

Currently, he is the Wissner-Slivka Junior Chair in the Electrical Engineering and Computer Science Department of Northwestern University. He was with Bimtek, a startup company that provided Internet solutions, from 1997 to 2000, and BlueFront Defenses, a startup company that designs hardware-based network security solutions, from 2000 to 2002. His research interests are in computer architecture with an emphasis on networking hardware design and physical-aware microarchitectures. He is the author of two book chapters and more than 70 journal and refereed conference publications in these areas. He is also the co-author of *NetBench* and *MineBench*, two widely used benchmarking suites for networking and data-mining applications, respectively.

Dr. Memik has been in the program committees of 20 workshops and conferences, was the Co-Chair for the Advanced Networking and Communications Hardware Workshop (ANCHOR) held in conjunction with ISCA between 2004 and 2006, and is the Program Co-Chair for the 2007 International Symposium on Microarchitecture (MICRO). He was the recipient of the Department of Energy CAREER Award in 2005, Searle Teaching Excellence Fellowship in 2004, Henry Samueli Excellence in Teaching Award in 2002, and the Henry Samueli Fellowship in 2001.

Alok Choudhary (F'05) received the B.E. (Hons.) degree from the Birla Institute of Technology and Science, Pilani, India, in 1982, the M.S. degree from the University of Massachusetts, Amherst, in 1986, and the Ph.D. degree in electrical and computer engineering from the University of Illinois, Urbana-Champaign, in 1989.

From 1989 to 1996, he was on the faculty of the Electrical and Computer Engineering Department at Syracuse University, Syracuse, NY. Currently, he has is a Professor of Electrical and Computer Engineering at Northwestern University, Evanston, IL, where he also holds an adjunct appointment with the Kellogg School of Management in the Marketing and Technology Innovation Departments, Northwestern University. His research interests are in high-performance computing and communication systems, power-aware systems, information processing, and the design and evaluation of architectures and software systems.

Prof. Choudhary's career has been highlighted by many honors and awards, including the National Science Foundation Presidential Young Investigator Award, an IEEE Engineering Foundation award, an IBM Faculty Development award, and an Intel Research Council Award.