A Line Rate Outlier Filtering FPGA NIC using 10GbE Interface

Ami Hayashi[1], Yuta Tokusashi[1], and Hiroki Matsutani[1,2,3]

[1]Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan
[2]National Institute of Informatics, [3]Japan Science and Technology Agency PRESTO
{hayashi,tokusasi,matutani}@arc.ics.keio.ac.jp

ABSTRACT

As data sets grow rapidly in size and the number, an outlier detection that filters unnecessary normal information becomes important. In this paper, we propose to move the unsupervised outlier detection from an application layer to a network interface card (NIC). Only anomalous items or events are received for a network protocol stack and the other packets are discarded at the NIC. The demands for storage and computation costs at a host are thus dramatically reduced. However, because normal items are discarded at the NIC and the application layer can no longer know what is normal, in our approach, the application at the host periodically peeks at the NIC buffer. We select an outlier detection based on the Mahalanobis distance as one of the simplest algorithms. Our approach is implemented on an FPGA-based NIC that has 10GbE interfaces. The sampling frequency of the NIC buffer vs. outlier detection precision is analyzed. Real experiments using the FPGA NIC demonstrate a 14,000,000 samples-per-second throughput in performance, which is close to the 10GbE line rate.

1. INTRODUCTION

Data sets grow rapidly in size with the advances in information and communication technologies and mobile and sensing devices. Sensors that monitor physical or environmental conditions are applied to various applications, such as health-care monitoring, pollution monitoring, industrial monitoring, network logging, and server logging, and they continuously generate data. The generated data are collected via a network and stored in dedicated servers for record and analysis. The demand for storage and computing resources to manage such Big data is thus a crucial issue.

Data compression is a common solution to reduce the storage demand and mitigate a disk I/O bottleneck. It can be either lossless or lossy, and one of them is selected depending on whether dropping nonessential detail can be accepted for a given storage demand. Since compressed data must be decompressed before use, data compression incurs an extra computation cost, which introduces a space-time trade-off. A similar but different approach is an information filtering that removes nonessential information from a data stream. An extra computation cost (e.g., data decompression) is not needed to access the filtered data. The goal is to increase the signal-to-noise ratio, and it can be used instead of the data compression or prior to the data compression. For the sensing and monitoring applications, outlier detection algorithms, such as using the Mahalanobis distance, can be simply used for information filtering to identify anomalous items or events.

As sensor data collected via a network grow in size, the demands for storage and computation costs (e.g., network processing and outlier detection) are increasing. In this paper, we propose to move the unsupervised outlier detection from an application layer to a network interface card (NIC). For the sensing and monitoring applications, only anomalous items or events are received for a network protocol stack and the other packets (most data) are discarded at the NIC. The demands for storage and computation costs are thus dramatically reduced. However, because normal items are discarded at the NIC, the application layer can no longer know what is normal. To address this issue, the application periodically (but infrequently) peeks at an input buffer of the NIC and feeds back necessary information to the NIC.

In this paper, we design and implement an outlier detection based on the Mahalanobis distance for an FPGA-based NIC that has 10Gbit Ethernet (10GbE) interfaces. The outlier detection is fully pipelined and real experiments using the FPGA NIC demonstrate a 14M samples-per-second throughput, which is close to the 10GbE line rate.

The rest of this paper is organized as follows. Section 2 introduces data mining algorithms and their FPGA-based accelerators. Section 3 illustrates our outlier filtering NIC using the Mahalanobis distance and Section 4 evaluates it in terms of performance and precision. Sections 5 and 6 discuss the results and conclude this paper.

2. BACKGROUND AND RELATED WORK

2.1 Mahalanobis Distance

Various metrics can be used for an outlier detection. One of the simplest metrics is the Mahalanobis distance which represents how many standard deviations a given value is away from the average. That is, it is a weighted distance that takes into account the deviation for each feature or dependent variable.

An outlier detection using the Mahalanobis distance is expressed by the following equations.

$$\boldsymbol{\mu} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x_i} \tag{1}$$

$$\boldsymbol{\Sigma} = \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{x_i} - \boldsymbol{\mu})(\boldsymbol{x_i} - \boldsymbol{\mu})^T \tag{2}$$

$$\sqrt{(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})} > \theta \tag{3}$$

Assuming that there are m samples, each of which has n features, each sample x_i $(i = 1...m)$ is thus represented by an n-dimensional vector. Equations (1) and (2) denote the mean vector $\boldsymbol{\mu}$ and the variance-covariance matrix $\boldsymbol{\Sigma}$, respectively. In Equation (3), the left- and right-hand sides represent the

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2015) Boston, MA, USA, June 1-2, 2015.

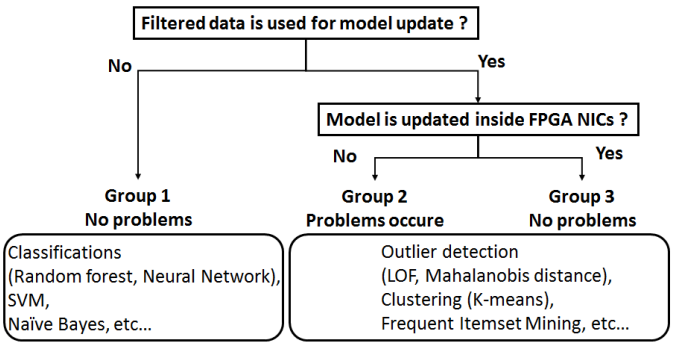


Figure 1: Classification of algorithms for filtering

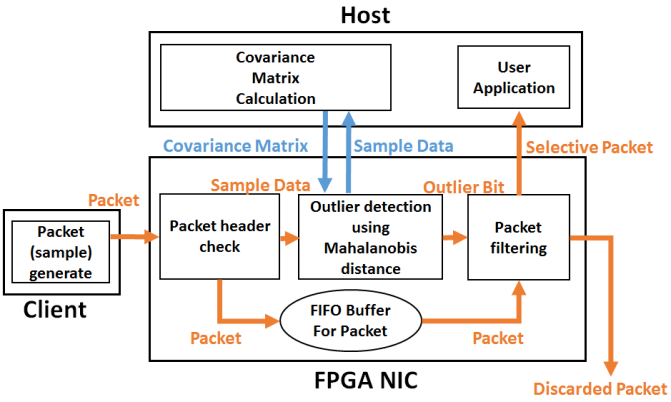


Figure 2: Outlier filtering FPGA NIC

Mahalanobis distance and the user-defined threshold value θ, respectively. If the Mahalanobis distance of a given sample is larger than θ, the sample is identified as an outlier.

The Mahalanobis distance is widely used for cluster analysis, classification, and outlier detection. In [2], a principal component analysis (PCA) is accelerated by means of FPGAs for network intrusion detection systems. More specifically, a feature extraction and the Mahalanobis distance are calculated by FPGAs for the PCA. In [4], a real-time object tracking is accelerated by FPGAs. The Mahalanobis distance is used for a color segmentation module that separates an object from a background in the system.

2.2 Other Data Mining Accelerators

Random Forest. It is a well-known machine learning algorithm that uses an ensemble of decision trees. It is applicable for classification, regression, and clustering. For the classification, decision trees independently perform a classification for given data, and then the final decision is made by the aggregated result (e.g., given by averaging or majority vote). The classification phase of the random forest can be accelerated by using FPGAs and GPUs, and both the approaches are discussed in [3]. To compensate for the deep and nonuniform nature of the decision trees, CRF (Compact Random Forest) [3] that can keep the depths of decision trees shallow and uniform is used for the FPGA-based approach.

Frequent Item-set Mining. It is a popular method for finding item-sets that frequently appear in a large-scale transaction that records a vast number of item-sets. It can be used for marketing activities, such as product placements. Eclat is a frequent item-set mining algorithm and it is accelerated by FPGAs in [6]. The design can be extended depending on available FPGA resources. The results using four FPGA boards show a 68x performance improvement compared to a software.

K-means. It is one of the most popular clustering algorithms that partition the data set into some clusters. Such a partitioning is performed to minimize the distance between each sample and the center of gravity of the cluster currently belonging. It is accelerated by FPGAs in [5]. They focus on required resources on FPGAs and propose a method to configure K-means on FPGAs using binary kb-tree or exploiting the distributed memory architecture efficiently.

3. DESIGN AND IMPLEMENTATION

3.1 Data Mining Algorithm for Filtering NIC

In our approach, normal items are discarded at NICs, and applications can no longer know these normal values. This approach affects the outlier detection algorithms. Figure 1 classifies some data mining algorithms from a standpoint of the negative impacts induced by discarding normal values at NICs.

Algorithms in Group 1 in Figure 1 perform the data selection (e.g., classifications) and do not update their data set model based on the filtered data. They work correctly even if filtered data are dropped at the NIC, because a prepared data set is preliminarily fed to build their data set model. Most supervised learning algorithms, such as Random Forest [3], are included in this group. These algorithms can be simply used for the outlier detection at FPGA NICs by combination with a packet filtering mechanism.

Algorithms in Groups 2 and 3 perform the data selection and update their data set model together. Most unsupervised learning algorithms are included in these groups and they are often used for the outlier detection. In these algorithms, because the filtered data are used for updating their data set model (i.e., learning), both the learning and data selection must be implemented inside the FPGA NIC; otherwise, their data set model cannot be updated. However, some sort of intelligence is useful for constructing the data set model in the learning phase. Actually, the data set model and some parameters can be tuned in response to various information from environments or applications. In such cases, the learning phase should be implemented at the application layer rather than the FPGA NIC, though normal values are filtered at the NIC and cannot be transferred to the application layer for learning. This paper focuses on such a dilemma. As proposed below, this problem can be simply resolved by sampling the FPGA NIC buffer from applications periodically and infrequently.

In this paper, we select an outlier detection based on the Mahalanobis distance as one of the simplest algorithms that require the filtered data for updating the data set model. Please note that the proposed strategy is widely applicable for the other data mining algorithms (especially algorithms in Groups 2 and 3) when the update of their data model is performed at application layer while filtered data are dropped at the NIC.

3.2 Outlier Filtering FPGA NIC

Figure 2 illustrates an overview of the proposed system. The data selection is performed at the FPGA NIC, while a covariance matrix and its inverse matrix used for the outlier detection are calculated by an application software and fed back to the NIC. The application periodically peeks at the input data queued for the outlier detection in the NIC via device driver APIs (e.g., read() or ioctl()) in order to calculate the covariance matrix and its inverse matrix.

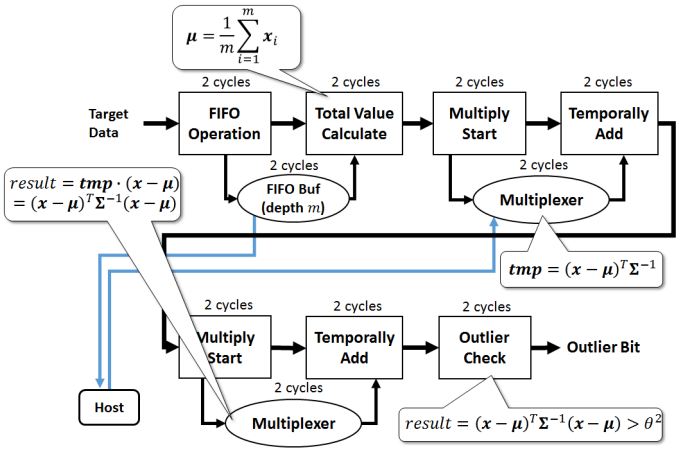


Figure 3: Outlier detection using the Mahalanobis distance

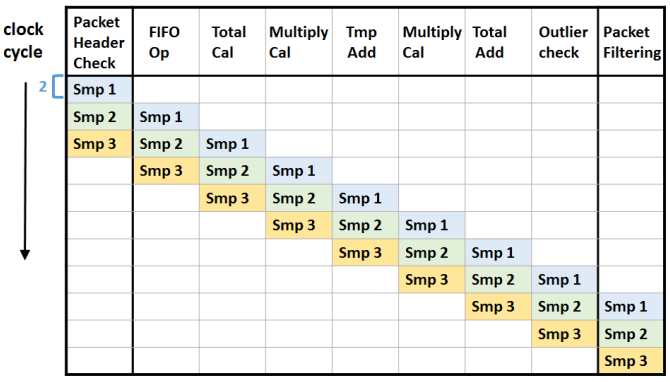


Figure 4: Packet processing pipeline

In the following, the proposed outlier detection FPGA NIC (both the FPGA NIC side and host side) is illustrated.

3.2.1 FPGA NIC Side

We use a NetFPGA-10G board as a programmable NIC. NetFPGA-10G Reference NIC design [1] combined with Xilinx 10GbE MAC IP core is used as a baseline NIC, and we implement our outlier filtering NIC by modifying the baseline.

3.2.1.1 Packet Classification Module.

Packet classification is performed at the FPGA NIC as illustrated in Figure 2. The packet classification module receives packets as 256-bit sized data due to the bitwidth of NetFPGA-10G AXI (Advanced eXtensible Interface). In the packet classification module, UDP packets with a specific destination port number are classified as sample data packets. The other packets, such as ARP (Address Resolution Protocol) packets, are all transferred to the host without outlier detection and processed by a network protocol stack as usual. The sample data packets are stored in an input FIFO buffer. Then only their sample data values (n values for n features) are extracted, and transferred to the outlier detection module. If a sample data set is classified as an outlier, the corresponding packet is transferred to the host; otherwise, the packet is discarded at the FPGA NIC.

3.2.1.2 Outlier Detection Module.

Figure 3 details the outlier detection module. An input data set is processed with the following seven steps.

1. An input data set is decomposed into values of features. These values are stored into an FIFO buffer for each feature.
2. An average of all the values stored in each FIFO buffer is calculated (Equation (1)). At the same time, a difference between the previous average value (calculated before) and this sample value is calculated.
3. An inverse matrix of a covariance matrix periodically fed by a host is multiplied by the difference value calculated in Step 2.
4. The multiplication result in Step 3 is totalized.
5. The totalized result in Step 4 is multiplied by the difference value calculated in Step 2.
6. The multiplication result in Step 5 is totalized.
7. A square of the Mahalanobis distance (calculated in Steps 3 to 6) and a user-defined threshold value are compared.

These seven steps, each of which is performed in two cycles, are fully pipelined in the FPGA NIC. Thus, sample data packets are processed in every two cycles, as shown in Figure 4. As the packet processing in the NetFPGA-10G board is operated at 160MHz, the maximum throughput is 80,000,000 samples per second. Please note that the packet classification/filtering takes two cycles in Figure 4, because we assume that a size of sample data packets is 64Byte (256-bit for each cycle). These seven steps are explained as follows.

Step 1. The feature values extracted from a single sample packet are stored in dedicated FIFO buffers. The FIFO buffer is dedicated for each feature. In this example, the data width is set to 32-bit in order to store 32-bit integer values. The FIFO buffer depth is corresponding to m in Equations (1) and (2). That is, it is the number of samples used to calculate the Mahalanobis distance. When a new value is received, the Mahalanobis distance between this value and recent data set consisting of previous m sample values is calculated. A quality of outlier detection becomes susceptible to outliers as m decreases, while a hardware amount (RAM size) of these FIFO buffers increases as m increases. These effects resulting from the FIFO buffer depth m is discussed in the evaluation section.

Step 2. All the values in each FIFO buffer are totalized and stored in a sum register. Then, the mean vector $\boldsymbol{\mu}$ (Equation (1)) is calculated based on the total value stored in the sum register. The sum register is initialized to zero by a reset signal. When a new value is enqueued to the FIFO, the value is added to the sum register. When an old value is dequeued from the FIFO, it is subtracted from the sum register. The FIFO buffer depth is set to a number which is a power of two. When m is eight, for example, the average value can be simply obtained by shifting three bits.

Steps 3 and 4. $(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma^{-1}}$ in Equation (3) is calculated as $\boldsymbol{tmp}$. The average value $\boldsymbol{\mu}$ calculated in Step 2 and an inverse matrix of a covariance matrix $\boldsymbol{\Sigma^{-1}}$ fed by a host (details are described in Section 3.2.2) are used for the calculation. Each element in the matrix is multiplied in Step 3, and the resultant elements are totalized in Step 4. Multipliers using DSP slices calculate the result in two cycles in Step 3.

Steps 5 and 6. The left side of Equation (3) is calculated as $\boldsymbol{tmp}(\boldsymbol{x}-\boldsymbol{\mu})$, where $\boldsymbol{tmp}$ was calculated in Step 4 and the mean vector $\boldsymbol{\mu}$ was calculated in Step 2. The Mahalanobis

distance is thus calculated in Step 6.

Step 7. An outlier is detected based on a square of the Mahalanobis distance calculated in Step 6 and a threshold θ^2 given by a user. Both the left and right sides in Equation (3) are squared. A 1-bit outlier detection result is transferred to the packet filtering module (denoted as Packet Filtering in Figure 2). The user or applications can adjust the threshold value θ at run-time via ioctl() interface of the NetFPGA-10G device.

3.2.2 Host Side

A covariance matrix $\mathbf{\Sigma}$ is expressed by Equation (2). m sample data (most of them should be normal values) are needed for the covariance matrix, though all the normal values are discarded at the FPGA NIC. The covariance matrix is calculated with following five steps.

1. m samples in the NIC FIFO buffer are read by a host via ioctl().
2. A covariance matrix $\mathbf{\Sigma}$ is calculated using these m values as data sets.
3. An inverse matrix $\mathbf{\Sigma^{-1}}$ is calculated.
4. Values in the inverse matrix are rounded to 32-bit integer values.
5. The integer form of $\mathbf{\Sigma^{-1}}$ is fed back to the FPGA NIC memory via ioctl().

These steps are performed at the host periodically. A frequency of these steps can be adjusted at run-time. As the frequency increases, a quality of outlier detection is improved, because the covariance matrix is built based on more fresh sample data. However, as the frequency increases, the host CPU workload and communications between host and FPGA NIC device increase. We will evaluate the frequency of these steps vs. a quality of outlier detection in Section 4 in order to address the proper frequency.

4. EXPERIMENTAL RESULTS

The proposed filtering FPGA NIC that employs the Mahalanobis distance for outlier detection is evaluated in terms of the maximum throughput, hardware amount, and outlier detection precision.

4.1 Evaluation Environment

The NetFPGA-10G board has a Xilinx Virtex-5 XC5VTX240 and four SFP+ interfaces. Our outlier detection module is operated at 160MHz. We use Xilinx ISE 13.4 for the design synthesis and implementation.

The NetFPGA-10G board with our outlier detector is mounted on a host machine (Intel Core i5-3470S @2.9GHz, 6GB RAM) via a PCI Express Gen2 x8 interface. The NetFPGA-10G machine is connected to an open-source hardwired 10GbE network tester [1] via a SFP+ copper cable.

4.2 Maximum Throughput

Assuming that each packet contains a single sample data set, the maximum throughput here is defined as the number of sample data packets processed by the proposed filtering FPGA NIC per a second. The packet length is 64Byte. The sample data packets are fed to the proposed filtering FPGA NIC in a 10Gbps line rate by the hardwired 10GbE network tester. We added a packet counter in the FPGA NIC in order to measure the number of packets processed during a certain time period at the NIC.

The maximum throughput of the proposed filtering FPGA NIC is compared to that without any outlier filtering at the NIC and host. We use the original Reference NIC design for the NetFPGA-10G board as the counterpart. In this case, all the sample data packets are transferred to the host application; thus the network protocol processing overhead is included while any outlier detection overhead is not considered.

Table 1: Maximum throughputs with and without the outlier detection

	Measured throughput
FPGA NIC w/ outlier detect.	14,208,026 packets/sec
FPGA NIC w/o outlier detect.	840,112 packets/sec

Table 1 shows the evaluation results. Figure 5 compares a measured maximum throughput of the proposed filtering NIC with a 10GbE ideal line rate. The 10GbE ideal line rate is calculated by taking into account 12Byte IFGs (Inter Frame Gaps) and 8Byte preambles. The FPGA NIC without the outlier detection receives only 5.6% of packets compared to the 10GbE line rate assuming 64Byte packets. This significant performance loss comes from 1) inefficiency of the NIC-host interface and 2) software overheads of the NIC device driver, network protocol stack, and user-space application that receives packets via a UDP socket. On the other hand, the FPGA NIC with the outlier detection processes 95.8% of packets compared to the 10GbE line rate, because it can completely eliminate the above-mentioned performance limitations. It improves the measured maximum throughput by 17.1x compared to that without the outlier detection at the NIC.

4.3 FPGA Resource Utilization

We evaluate the proposed filtering FPGA NIC in terms of the CLB and DSP utilizations by varying the number of features and the depth of the NIC FIFO buffers m.

Figures 6 and 7 show the CLB and DSP utilizations, respectively. The horizontal lines of 149,760 and 96 in these graphs denote the available resources of CLB and DSP in the NetFPGA-10G device (Virtex-5 XC5VTX240), respectively. Because the CLB utilization still has a space, the FIFO buffers that temporarily store the sample data in the FPGA NIC are implemented with CLBs (distributed RAM) for simplicity. The sample data stored in the FIFO buffers are periodically read by the host in order to calculate the covariance matrix. They can be implemented as Block RAMs if needed.

DSP48E slices are used for the multipliers. To calculate the Mahalanobis distance, a vector-matrix multiplication and a vector-vector multiplication are performed, as shown in Figure 3. Thus, $(N \times N + N)$ scalar multiplications are performed for the Mahalanobis distance, where N is the number of features. Because these scalar multiplications are performed in parallel, $(N \times N + N)$ multipliers are needed and their number increases with the square of N. Figure 7 shows that the NetFPGA-10G device can support up to four features. If sample data have more features, CLBs are used for the multipliers or a part of the multiplications are performed sequentially.

4.4 Outlier Detection Precision

The outlier detection precision of the proposed filtering FPGA NIC may be affected, because the covariance matrix is calculated based on a limited number of samples read from the FPGA NIC with a certain delay. The samples are rounded as integer values. Here, the proposed filtering FPGA NIC is compared to an ideal case, where the covariance matrix is refreshed every time when a sample is processed. In addition, the samples are represented in double precision floating-point type (not rounded).

Data sets for the experiments are generated as follows.

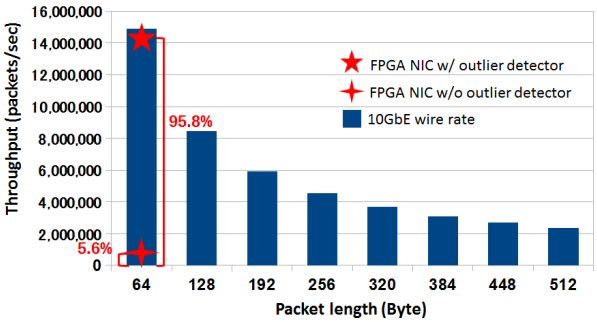


Figure 5: The measured maximum throughput vs. 10GbE ideal line rate

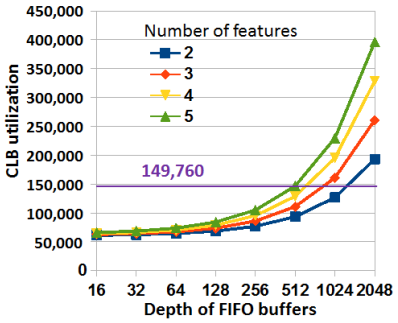


Figure 6: CLB utilization

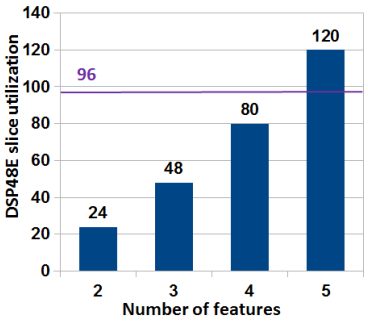


Figure 7: DSP utilization

- A data set consists of 10,000,000 samples generated based on Gaussian distribution.
- The percentage of outliers is 1% (occurrence position is random).
- The number of features is 2 and the correlation is 0.8.
- The following Gaussian distribution parameters are used.
  - Gaussian distribution $f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$
  - Normal parameter $\mu = 30, \sigma = 3$
  - Outlier parameter $\mu = 300, \sigma = 10$

Given the purpose of the filtering FPGA NIC, passing normal samples is permissible, while filtering (discarding) anomalous samples is not acceptable. Here, the misidentified rate is the percentage of normal samples incorrectly classified as outliers, under a condition that 100% or 99% of anomalous values are at least classified as outliers. A lower misidentified rate means a higher precision.

4.4.1 *Precision vs. Depth of FIFO Buffer*

The proposed filtering FPGA NIC is evaluated in terms of the outlier detection precision by varying the FIFO buffer depth m. It refreshes a covariance matrix at every 30,000 samples. In addition, it is compared to the ideal case that refreshes the covariance matrix every time using samples represented in double precision type.

Figure 8 show how the precision is affected by the FIFO buffer depth, under a condition that 100% or 99% of anomalous values are classified as outliers, respectively. The precision is lower when m is too small because the mean of each feature is easily affected by outliers. Although the proposed filtering FPGA NIC shows a quite low outlier detection precision when the FIFO buffer depth is not enough, the precision is close to the ideal case when the depth is 512 and the misidentified rate is 0% when the depth is 2,048. Please note that the NetFPGA-10G device can implement two 1,024-depth FIFO buffers as evaluated in Section 4.3. That is, for data sets that have two features, we can implement enough depth FIFO buffers that provide a comparable precision to the ideal case while keeping the 10GbE line rate performance.

4.4.2 *Precision vs. Covariance Matrix Update Frequency*

In the proposed filtering FPGA NIC, while the outlier detection is performed by the hardware, feedbacks to the outlier detection (e.g., a covariance matrix and data set model) are computed by a more flexible software. To calculate the covariance matrix, the application samples the FIFO buffer infrequently (e.g., once per 30,000 samples). Here, the proposed filtering FPGA NIC is evaluated by varying the refresh frequency of a covariance matrix, assuming the FIFO buffer depth is 1,024.

Figure 9 shows the results of the proposed filtering FPGA NIC, in which the samples used for a covariance matrix are rounded as integer values, under a condition that 100% of anomalous values must be classified as outliers. As shown, the misidentified rate is at most 1%. In addition, when samples in the double precision floating point type are used for the covariance matrix, the misidentified rate is 0% (the results are omitted due to page limitation).

4.4.3 *Precision vs. Unstable Center of Gravity*

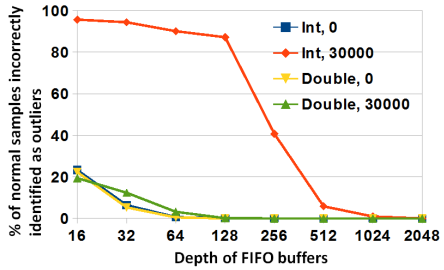
In the aforementioned experiments, the data samples were generated based on a pure Gaussian distribution. In contrast, here we evaluate the other cases, in which the center of gravity $\boldsymbol{\mu}$ of data sets (i.e., average value of one feature) is advisedly incremented over time. That is, based on the Gaussian distribution parameters listed in Section 4.4, the normal and outlier data sets are changed so that their μ values (i.e., 30 for normal and 300 for outlier) are incremented by one for every N samples, where N is 20 to 1,000. The FIFO buffer depth is set to 1,024. We assume that the sample data are rounded as integer values, and the refresh frequency of a covariance matrix is either once per 1 sample or once per 30,000 samples.

Figure 10 shows the results for unstable data sets, under a condition that 100% or 99% of anomalous values are at least classified as outliers, respectively. As N goes beyond 25, the misidentified rate becomes quite low. The precision degradation when $N < 25$ comes from the algorithm used, rather than from the refresh frequency of a covariance matrix.

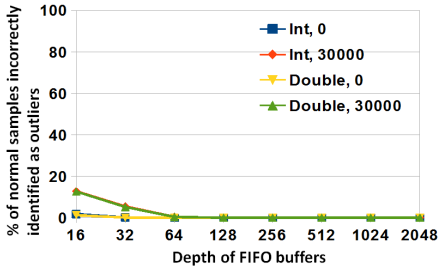
5. DISCUSSION

We selected the outlier detection using the Mahalanobis distance for the filtering FPGA NIC because of the simplicity. Although this design is well suited to a hardware implementation, it is not good enough to learn more complicated models. This fact motivates us to explore more sophisticated data mining algorithms applicable for the proposed filtering FPGA NIC, such as Local Outlier Factor (LOF) algorithm, as a future work. In this case, while a hardwired outlier detection module is implemented on the FPGA NIC, a part of the complicated learning phase is left to the software as proposed in this paper. Our approach is thus widely applicable for such unsupervised outlier detections.

Because $(N \times N + N)$ multipliers are still needed for the FPGA NIC (where N is the number of features), the sample values are rounded to integer values, in order to use faster inte-



(a) Outlier detection rate is 100%



(b) Outlier detection rate is 99%

Figure 8: Misidentified rate vs. FIFO buffer depth

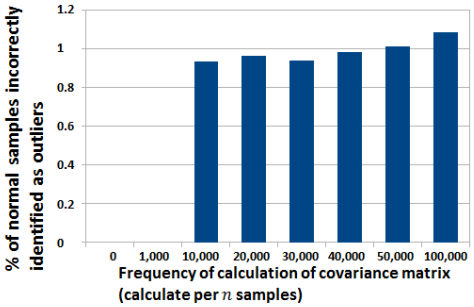
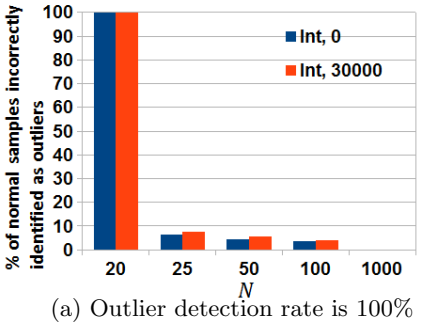


Figure 9: Precision vs. covariance matrix update frequency (integer, outlier detection rate is 100%)



(a) Outlier detection rate is 100%

(b) Outlier detection rate is 99%

Figure 10: Precision for data sets where center of gravity is unstable (data sets are changed so that the center of gravity is incremented per N samples)

ger multipliers instead of the floating point version, at the cost of precision slightly. Please note that our filtering FPGA NIC is proposed to narrow down interesting samples to be transferred to a host, in order to mitigate the memory/storage demands and software (e.g., network protocol stack) overheads. Our integer-based filtering FPGA NIC demonstrates that, under a condition that 100% of anomalous samples must be classified as outliers, most normal samples can be successfully dropped at the NIC. Given this purpose, our integer-based filtering FPGA NIC strikes an appropriate balance.

6. SUMMARY

In this paper, we proposed to move the unsupervised outlier detection from an application layer to NICs. Only anomalous data or events are received for a host and the others are discarded at the NIC. Thus, the memory/storage demands and software (e.g., network protocol stack) overheads at the receiver host are dramatically reduced. The learning phase should be implemented at an application layer for the software flexibility, though normal values are filtered at the NIC and cannot be transferred to the application layer for learning. To address such chicken-and-egg problem, the host application infrequently peeks at the NIC FIFO buffer via device driver APIs. We designed and implemented an outlier detection based on the Mahalanobis distance on NetFPGA-10G board. The host application refreshes a covariance matrix at every 30,000 samples. Real experiments demonstrated that the proposed filtering FPGA NIC processes 95.8% of packets compared to a 10GbE line rate. It also demonstrated that, under a condition that 100% of anomalous samples must be classified as outliers, most normal samples can be successfully discarded at the NIC (only 1% of normal packets are wrongly transferred to the host).

We believe that our approach, in which the outlier detection is implemented on a hardware while a part of the complicated learning phase is left to a software, would be widely applicable for various machine learning NICs. This paper addresses such a case. We are planning to implement more sophisticated data mining algorithms, such as LOF, for the FPGA NIC as a future work.

Acknowledgements A part of this work was supported by JST PRESTO.

7. REFERENCES

[1] The NetFPGA Project. `http://netfpga.org/`.

[2] A. Das, D. Nguyen, J. Zambreno, G. Memik, and A. Choudhary. An FPGA-Based Network Intrusion Detection Architecture. *IEEE Transaction on Information Forensics and Security*, 3(1):118–132, Mar. 2008.

[3] B. V. Essen, C. Macaraeg, M. Gokhale, and R. Prenger. Accelerating a Random Forest Classifier: Multi-Core, GP-GPU, or FPGA? In *Proceedings of the International Symposium on Field-Programmable Custom Computing Machines (FCCM'12)*, pages 232–239, Apr. 2012.

[4] D. Popescu, D. Patirniche, R. Dobrescu, M. Nicolae, and M. Dobrescu. Real Time Mobile Object Tracking Based on Chromatic Information. In *Proceedings of the International Conference on Remote Sensing (REMOTE'09)*, pages 13–18, Oct. 2009.

[5] F. Winterstein and S. Bayliss. FPGA-based K-means Clustering using Tree-based Data Structures. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'13)*, pages 1–6, Sept. 2013.

[6] Y. Zhang, F. Zhang, Z. Jin, and J. D.Bakos. An FPGA-Based Accelerator for Frequent Itemset Mining. *ACM Transactions on Reconfigurable Technology and Systems*, 6(1):1–17, May 2013.