Real-time Anomaly Detection for Streaming Data using Burst Code on a Neurosynaptic Processor

Qiuwen Chen, Qinru Qiu
Department of Electrical Engineering and Computer Science, Syracuse University, NY 13244, USA
Email: {qchen14, qiqiu}@syr.edu

***Abstract*—Real-time anomaly detection for streaming data is a desirable feature for mobile devices or unmanned systems. The key challenge is how to deliver required performance under the stringent power constraint. To address the paradox between performance and power consumption, brain-inspired hardware, such as the IBM Neurosynaptic System, has been developed to enable low power implementation of large-scale neural models. Meanwhile, inspired by the operation and the massive parallel structure of human brain, carefully structured inference model has been demonstrated to give superior detection quality than many traditional models while facilitates neuromorphic implementation. Implementing inference based anomaly detection on the neurosynaptic processor is not straightforward due to hardware limitations. This work presents a design flow and component library that flexibly maps learned detection network to the TrueNorth architecture. Instead of traditional rate code, burst code is adopted in the design, which represents numerical value using the phase of a burst of spike trains. This does not only reduce the hardware complexity, but also increases the results accuracy. A Corelet library, NeoInfer-TN, is developed for basic operations in burst code and two-phase pipelines are constructed based on the library components. The design can be configured for different tradeoffs between detection accuracy and throughput/energy. We evaluate the system using intrusion detection data streams. The results show higher detection rate than some conventional approaches and real-time performance, with only 50mW power consumption. Overall, it achieves 10^8 operations per watt-second.**

I. Introduction

With the blooming of machine learning and neural networks, intelligent systems have been developed for various applications such as image recognition [16], multi-media retrieval [5] and intrusion detection [17]. Many of them process streaming data in real-time and imposes high demand in accuracy and computation throughput. Real-time anomaly detection is one of these applications that continuously monitors and processes incoming data streams for patterns that do not conform to normality. It is an extremely desirable feature for improving the autonomy of today's unmanned systems or mobile devices. However, to deliver the required performance under limited power constraint is a major design challenge.

The brain has unprecedented performance and energy efficiency in cognitive tasks [10]. It is believed that perception is a procedure of probabilistic inference, and the efficiency of the biological neural system comes from its massive parallel architecture, spiking based communication and closely coupled computing and storage. Inspired by the biological structure, the IBM Neurosynaptic System with the TrueNorth architecture [12] provides a low-power platform for large-scale prototyping of Spiking Neural Network (SNN) based intelligent systems. Meanwhile, brain-inspired anomaly detection has been proposed [7] and shown to give superior detection quality than many traditional approaches. It performs inference based detection and features massive parallel structure that facilitates neuromorphic implementation. Despite the detection quality and fast processing, the system consumes high power, and migrating the model to an SNN will provide huge optimization for power efficiency. However, implementing the inference network on the TrueNorth processor is not straightforward.

A TrueNorth chip contains 4096 neurosynaptic cores, each of which has 256 axon inputs and 256 neurons. The synaptic connectivity is realized by a 256x256 crossbar. The connected core networks are encapsulated into Corelet [1] for abstraction and modular designs. While the processor provides potential to address the performance and power constraints for real-time embedded applications, challenges exist when mapping a signal processing flow onto this platform due to its hardware constraints. Firstly, each neuron (column) can only support 4 different input weights, while the weight of a connection is decided by the axon type (row). This limits all neurons who share an axon input to use the same weight rank at that row. However, most models' learned parameters are real numbers. Secondly, neurons communicate with each other using spikes, how to encode numerical value into spike trains is application specific. Thirdly, the crossbar's size of a core constrains the fan-in and fan-out of a neuron to 256. This hinders the direct mapping of big networks. Finally, some common arithmetic operations, such as division and maximum, are not as readily supported in TrueNorth as in traditional architectures.

In this work, we focus on implementing a trained inference network on TrueNorth for real-time anomaly detection [7]. Instead of rate code, which has been widely used in many other TrueNorth applications [8], burst code is used because it gives higher computation accuracy and allows very simple implementation of certain arithmetic operations, such as maximum. A Corelet library is developed, which consists of neural circuits for operations in the anomaly detection model. Our system first extract the topology and parameters of the network from the learned knowledge base. Then, it flattens and maps the network to the Corelet library components for TrueNorth configuration. With a controllable clock driver input, the network is activated by streaming data in an event-driven way. Anomaly scores are calculated in real-time by probabilistic inferences of spatial-temporal features. To our best knowledge, this is the first work that applies neurosynaptic processor to the real-time anomaly detection.

The contributions of this paper are the followings:

- We build a generic parser that transforms and maps

This work is partially supported by the National Science Foundation under Grants CCF-1337300.

inference-based anomaly detection network [6], [7] to a spiking neural network.
- A novel spike burst coding scheme is proposed for efficient representation, high accuracy and more convenient implementations. A Corelet library, *NeoInfer-TN*, is developed, which contains the neural circuit implementation of the network components for this coding.
- The adoption of bust code enables a two-phase pipelined processing for higher throughput. The throughput only depends on the spike encoding window configured to achieve a required data precision.
- A tunable accuracy factor is provided to enable tradeoff between detection accuracy and throughput.
- The network is evaluated with real-time intrusion detection data stream, and the accuracy, throughput and power performance are reported.

II. Detection Algorithm

Inference-based anomaly detection considers the anomaly as an unexpected observation in a given environmental context. It calculates the likelihood of each observation and at the same time infer the most likely one from the context. If the likelihood of the real observation is much lower than that of the expected, then an alarm is raised. The detailed detection algorithm and analysis were elaborated in Chen et. al [6], [7], so this section only sketches the computation procedure.

Cogent confabulation [9] is adopted the computing model for probabilistic inference. Cogent confabulation describes an application using a set of features (e.g. color and shape). The observed attributes of a given feature (e.g. red color, round shape) are represented as neurons, and their pairwise conditional probabilities are represented as synapses. To better organize the knowledge, neurons that represent the same features are grouped into *lexicons*, and the synapses between the neurons of two lexicons are realized as a probability matrix. The i, j^{th} entry of such matrix gives the conditional probability $p(s_i|t_j)$ between neuron s_i in the source lexicon and t_j in the target lexicon.

Whenever an attribute is observed, the corresponding neuron is activated, and its excitation is passed to the other neurons through the synapses. The excitation of a neuron t in lexicon l is calculated by summing up all incoming activations:

$$y(t) = \sum_{k \in F_l} \{ \sum_{s \in S_k} [I(s) \ln \frac{p(s|t)}{p_0}] \}, t \in S_l \qquad (1)$$

Here, F_l denotes the set of lexicons that have connections to l, and S_k is the set of neurons in lexicon k. $I(s)$ takes 0 or 1 given the activation of neuron s. p_0 is a constant selected to ensure positive weights. The excitation is essentially the log likelihood of t given the rest of the observations.

A set of lexicons are selected and referred to as *key lexicons*. They are the testing units of abnormality, representing features of the possible observations. The other lexicons that are not tested are named *support lexicons* and they represent the environmental context. Synapses are established from the support lexicons to the key lexicons. The excitations of all neurons in a key lexicon are calculated using function (1). The neuron with the highest excitation is considered the reference neuron, denoted $t_{\max}$. The *anomaly score* of a key lexicon l is computed using Equation (2).

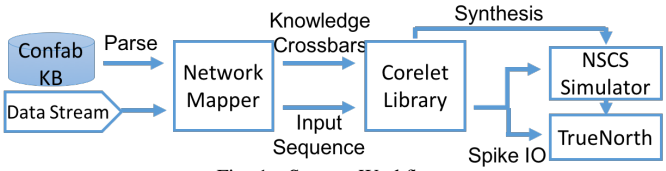


Fig. 1. System Workflow

$$\alpha_l(t) = \frac{y(t_{\max}) - y(t)}{y(t_{\max})}, t, t_{\max} \in S_l \qquad (2)$$

Here, the anomaly score is the normalized difference between the observed activation at t and the prediction $t_{\max}$. It indicates how low the sample's cogency is given the context. The scores of all lexicons are merged into the network anomaly score:

$$A(t_{l=1\ldots L}) = \frac{\sum_{l=1}^{L} \alpha_l(t_l)}{L} \qquad (3)$$

L is the number of key lexicons. t_l is the observation neuron of lexicon l. The output score is ranged in $[0, 1]$.

The efficiency of confabulation-based anomaly detection comes from a carefully structured inference network. We use the self-structuring method [7] to build the feature hierarchy in the lexicon design space, which generates a succinct network.

III. System Design

The aforementioned detection flow has four layers, (a) the support lexicon layer that collects input from the environment, (b) key lexicon layer that calculates neuron excitations using Equation (1), (c) anomaly score generator, which performs Equation (2) for each key lexicon, (d) and anomaly score accumulator, which merges all key lexicon anomaly scores using Equation (3). To convert the network into Corelets on TrueNorth, the overall workflow is shown in Fig. 1.

The network mapper reads in a trained confabulation knowledge base (KB) and maps the connections between support and key lexicons to a set of crossbar matrices considering the hardware constraints. It also maps the neuron observations to the input of the processor and synthesizes the input spikes from the given data stream. In the second step, the synthesizer maps each crossbar matrix into Corelets using our NeoInfer-TN library in the Corelet Programming Environment (CPE). It also maps the score generation and score accumulation layers into library components. At last, the network and its inputs are tested on the NSCS simulator [15] and the TrueNorth chip.

A. Network Mapping

The connection between neurons in the support and key lexicons forms a bipartite graph and can naturally be implemented as crossbar arrays, where each row corresponds to a neuron in the support lexicon and each column to a neuron in the key lexicon. However, a core can only implement crossbar up to 256x256, while the number of neurons in the support and key lexicons can reaches 3000. Matrix partition must be considered to implement one connection using multiple cores.

How to accurately represent synaptic weight in the crossbar is another challenge. Due to the hardware limitation, each column can only support 4 different weights, and all weights in

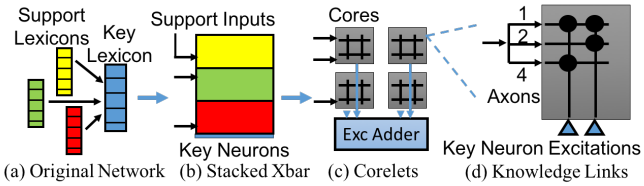


Fig. 2. Network Mapping

the same row must have the same rank in their corresponding columns. We resolve this problem by decompose each row into three rows with different weights, and the binary combination of the three rows provides flexibility in weight representation.

Fig. 2 shows each step performed by the network mapper. From the original network, large connection matrices are generated for key lexicons as shown in Fig. 2b. Each row represents a support neuron and each column represents a key neuron. As shown in Fig. 2d, for each support neuron input, 3 axon lines with corresponding strengths of 1, 2 and 4 are used. Using different combination of them, we can represent synaptic weight in the range of [0, 7]. We choose to use 3 lines because TrueNorth only supports 4 ranks in each column, and one of the ranks is used to implement an input clock, whose functionality will be explained in Section III-B. Then this large matrix is partitioned into multiple 256x256 crossbars that fits in single cores as shown in Fig. 2c. Additional Corelets are also inserted to merge the results.

To reduce the core usage by the network, we follow a "train-then-constrain" [8] approach. During the self-structuring stage [7], we limit the support connections of each key lexicon to be less than 15, and then train the knowledge base accordingly. This significantly reduces the number of required synapses with only marginal impact ($< 1\%$) on the detection quality. More detailed results will be given in Section V-B.

B. Spike Burst Coding

An important feature of the spiking neural network is that it encodes non-binary information into binary spike trains. A number of code scheme has been investigated [8], including binary code, rate code, population code and time-to-spike code. Among these schemes, rate code, which represents signal amplitude by the spiking frequency, is the most popular one and have been studied in many works [3], [13], [14].

The selection of coding scheme must lead to low-cost implementation of operations in the model and accurate representation of variables. In addition to integration and accumulation, which are common operations in many neural network models, our detection model has two special operations, maximum, which finds the maximum excitation level among all neurons in the same key lexicon; and division, which calculates the anomaly score as in Equation (2). None of the previously mentioned coding schemes give efficient implementations of these operations. The widely used rate code also suffers from sampling error, which reduces its accuracy. In this work, we propose a burst code that represents the numerical value of variables using the number of spikes that burst in a window.

Our burst code works in two phases alternatively as shown in Fig. 3: a burst neuron integrates spikes from the axons during the *input phase* and emit spikes in the *output phase*. While the neuron has a constant positive leak, the presence of

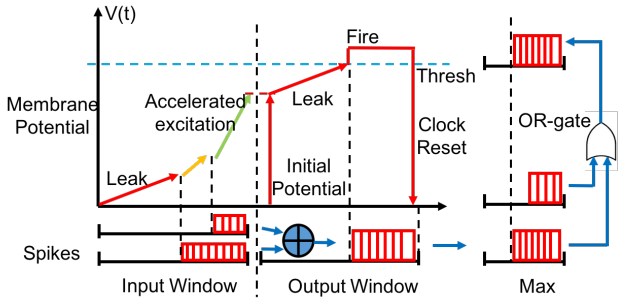


Fig. 3. Burst Code Neuron Dynamics

the input bursts accelerates the neuron's membrane potential raise, and result in a higher initial potential at the beginning of the output phase. The higher this initial level is, the faster the neuron reaches the firing threshold. It then stays on the threshold, fires until a negative input, i.e. the clock reset, is received. A binary code input can be considered a burst of only one spike, but the full input phase can be saved. The burst code is similar to a time-to-spike code in the sense that they both encode information into temporal representations. However, incidence such as bit flipping could cause larger error with time code, while the burst code is more tolerant to that since it relies on spike count rather than a single spike. Besides, summing of time codes usually requires additional neurons to translate the timing into amplitude representation. Essentially, the burst code works like the temporal version of a population code [8], but it occupies only one neuron.

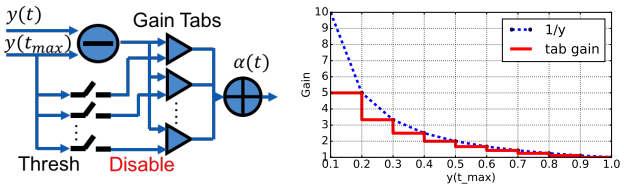
The burst code has two advantages over the more widely used rate code. Firstly, the code is more compact. For example, to represent 100 distinct values, the burst code only needs $K = 100$ ticks code window. In the case of a stochastic rate code, the spikes fire as a Bernoulli process and we use $\hat{p} = \#\text{spikes}/K$ to represent the information. The approximately normal 95% confidence interval $\hat{p} \pm 1.96\sqrt{p(1-p)/K}$ needs around $K = 10000$ to represent 0.01 granunarity. Please refer to Section V-C for the comparison of precisions of the two codes. Secondly, the burst code can max-pool values much more efficiently: a simple OR-gate will find the maximum as shown in Fig. 3, while the rate code needs a more complicated winner-takes-all (WTA) [13] circuit and scaling. We find the burst code a better option for the detection network.

C. Divider and Key Lexicon Burst Scorer

With the new encoding scheme, the hardware implementation of some arithmetic operations should be redesigned, and one example is division, which is used for the computation of Equation (2). We design a segmentation-based scorer for the lexicon anomaly score as in Fig. 4. The basic idea is to scale the excitation difference using multiple fixed-gain neurons.

While differentiating the normalized excitations $y(t)$ and $y(t_{\max})$, the later is also directed to an array of trigger neurons with different spike count thresholds. The triggers can fire to disable their corresponding gain tabs, which are linear neurons with fixed weight to threshold ratios. The gains are precomputed to approach $1/y(t_{\max})$ using Equation (4).

$$1/y(t_{\max}) \approx \sum_{i=1}^{M} \text{gain}_i I[\text{thresh}_i \geq y(t_{\max})] \qquad (4)$$



(a) Scorer Structure (b) 10-tabs Divider Response

Fig. 4. Key Lexicon Anomaly Scorer

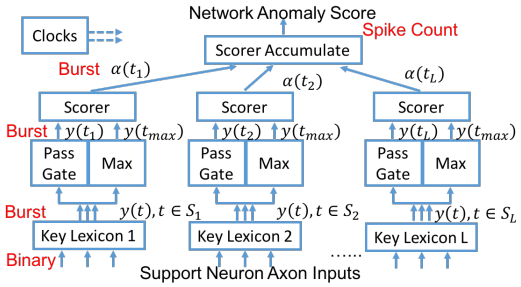


Fig. 5. Corelet Architecture

Here, M is the number of the gain tabs, and $I(.)$ is an indicator function that takes value $\{0, 1\}$. Those tabs having thresholds larger than $y(t_{\max})$ are kept on. Increasing $y(t_{\max})$ disables more tabs, results in a smaller amplifier to the signal. M decides the precision of the division. In this work, a 10-tabs scorer is used to emulate divisor with 0.1 precision as shown in Fig. 4b. The red lines indicate the scaling response of the neurons. Finally, the differential signal $y(t_{\max}) - y(t)$ passes the gain tabs and generate the anomaly score.

D. Corelet Library and Architecture

A set of Corelet configurations are designed to realize the aforementioned integration, maximization, and division operations in burst code and they form the NeoInfer-TN library. The components are instantiated to construct the detection flow as in Fig. 5. From the bottom layer up, the binary activations of the support neurons are sent to the "Key Lexicon" Corelet, who contains multiple Knowledge Crossbars and Excitation Adders to flexibly map arbitrary key lexicons. The excitations of all key neurons are computed and passed to the "Max" Corelet to find $y(t_{\max})$ using OR-gates. Also, the "PassGate" Corelet uses the actual observation t to clamp $y(t)$. The "Scorer" receives both signals and compute Equation (2) as described in Section III-C. Finally, an "Accumulator" collects the scores and represent the network-wide score of Equation 3 using spike count. The network also creates "Clocks" Corelet to control the code windows and the neuron timing.

IV. Inference Pipeline

Using the architecture constructed in Section III, we carefully design the pipeline to hide the delay and improve the throughput. This section introduces the timing of the neurons.

A. Timing for Real-time Processing

To hide the latency introduced by the burst code operations, we mesh the input window and output window as in Fig. 6. Integrations of lexicon excitations takes one window since they

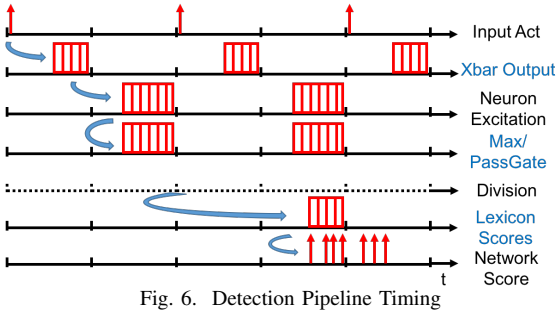


Fig. 6. Detection Pipeline Timing

are parallel additions. Lexicon scores requires two windows for divisions. In this way, the processing delay is 5 windows, and the detector handles one sample every 2 code windows (input phase and output phase). The throughput of the system does not depend on the scale of the network, but only the length of the code window.

An input clock fires at the end of the code windows to reset the neurons. We can use this signal to dynamically adjust the throughput of the pipeline. When the window size is smaller than the range of the values, the code set an lower-bound to the excitations, which is not desirable. We would like to further the code window without affecting the spike representation range. Section IV-B uses the accuracy factor to address this.

B. Accuracy Factor

When the value range is fixed, reduced window length is associated with a lower precision of the computations. This may not be feasible for classification tasks as the reduced accuracy affects the ranking of the predictions significantly. However, anomaly detection is less sensitive to a reduced precision because the relative excitations, rather than rankings, are used to score the unlikely events. Therefore, we introduce an *accuracy factor* to the neurons to make tradeoff between the throughput and the accuracy.

The accuracy factor (AF) is actually a multiplier to the neuron's positive leak, which shorten the duration that the membrane potential needs to reach the threshold. AF is applied throughout the network to make the burst code to represent a wider range, so that a smaller code window (higher throughput) can be used. The larger AF is, the less precise is the computation.

V. Evaluation

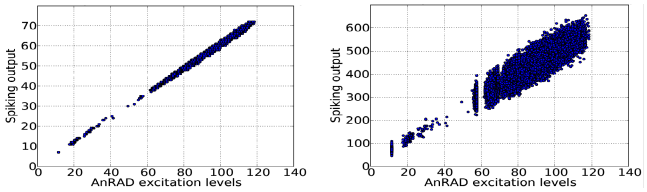
A. Experiment Setups

For evaluation, we structure and train the confabulation network using the DARPA intrusion detection dataset [11]. For each IP address pairs, traffic statistics are recorded per 300ms-frame. A random sample of 20000 frames are used for training. The test data contains about 80000 frames of 7000 normal samples, and 60000 frames of 38 attacks. The reference network and baseline results of the confabulation network are generated by the AnRAD framework [7].

The hardware development platform is IBM NS1e, which has a TrueNorth processor and its peripheral devices installed. The processor runs on 1ms/tick, therefore to represent data in the range [0, 99] we need a burst window of 100ms. There are 4096 cores, and 1 million hardware neurons on the chip,

TABLE I
NETWORK COMPLEXITY IMPACTS OF CONSTRAINT

Networks	Synapses	Cores for Key Lex	Total Cores
Original	3373K	5232	6116
Constraint	1322K	2169	2918
Reduction	60.8%	58.5%	52.3%



(a) Burst Code (b) Rate Code

Fig. 7. Excitation Correlation between Spike Code and Reference Program

which consumes 50 to 100mW during typical operation. The whole board consumes 2 to 3.5W depending on the utilization.

B. Network Construction

Using the self-structuring algorithm [7], we build the confabulation network, which is referred to as the "original" network. We also constrain the support lexicons to be less than 15 (the original network ranges from 3 to 30), and refer to the new network as the "constrained" network. Both networks have 123 key lexicons. These lexicons represent features or feature combinations, such as server, packets, etc., extracted from network traffic. With constraint, each key neuron has much less incoming synapses, and thus consume less hardware resources. From Table I, it is seen that the total number of synaptic weights is reduced by 60% by imposing the constraint on support connections. This brings the neurosynaptic core usage to less than 4096, so a single TrueNorth processor can handle the full confabulation network. Other than saving the power for additional chips and peripherals, because the spare cores can be gated, our constraint potentially reduces 50% of static power (calculation method in Section V-E).

C. Detection Accuracy

We first compare burst code and rate code for their accuracy. The window for burst code is 100 ticks, while the rate code window is 1000 ticks. A lexicon is selected from the DARPA network and has all its neuron excitations collected over 500 frames of data. Spiking outputs of both code schemes are compared with the baseline computation by the reference program. Fig. 7 is the scatter plots for both codes. The X-axis gives the baseline excitation computed by the reference program, and Y-axis denotes the spike counts at the neuron excitation pins from either code windows. Compared to rate code, our burst code shows better linear correlation with the reference computation, with a correlation coefficient of 0.998, while the rate code has 0.924. The burst code achieves higher precision with only 1/10 of the window size of the rate code. Assuming similar firing frequency (power), a burst code pipeline consumes much less energy for each detection since it can afford smaller window and thus shorter processing time.

Next we test the whole network for its performance in anomaly detection. Two configurations are tested: high accuracy with 100-tick burst window (TN-100) and high efficiency

TABLE II
DETECTION QUALITIES OF COMPARISON MODELS

Methods	SOM	RNN	Reference	TN-100	TN-10
AUROC	0.879	0.898	0.933	**0.943**	0.914

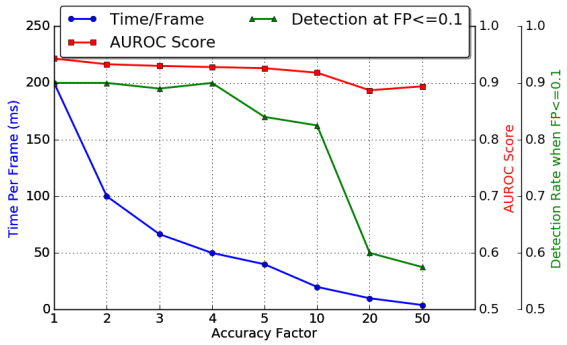


Fig. 8. Tradeoff between Quality and Speed

with 10-tick window (TN-10). The results are compared with 3 software implementation methods: self-organizing map (SOM), replicator neural network (RNN) [9] and the reference detector with full confabulation network. The metric is AUROC (Area Under Receiver Operation Curve), which measures the tradeoff between false alarm and true detection, the higher the better. Shown in Table II, TN-100 outperforms SOM and RNN by around 5%. It is even slightly better than the reference program. TN-10 provides 10X speedup over TN-100, and still generates better results compared to SOM and RNN.

D. Throughput and Accuracy Tradeoff

Computation speed is critical for real-time anomaly detection. With 100-tick windows, the spiking neural network uses 200ms for each input (alternative input/output windows). It can already achieves real-time detection as the network data stream was collected with sample intervals of 300ms. We would like to further investigate the potential performance increase for larger scaled data by trading off the accuracy.

Fig. 8 shows how the performance and detection quality vary with the change of the AF. The X-axis shows the accuracy factor. The Y-axises show the processing time (blue), AUROC score (red) and the best detection rates (green) when the false positive is less than 10%. As we increase the accuracy factor, the AUROC score only drops slightly from 0.943 (AF=1, 100-tick window) to 0.918 (AF=10, 10-tick window). However, the throughput of the system is improved significantly: with AF=10, the processing speed is reduced to 20ms/frame from the 200ms/frame base case. When the code window further shrinks to shorter than 10 ticks, the network still achieves good AUROC scores (> 0.88), but it generate so few spikes to the anomaly score that only a limited selection of thresholds can be used to tradeoff the detection and the false positives. Therefore, when we impose the false positive to be less than 0.1, the detection rate drops quickly for AF>10.

E. Power and Performance

Finally, we compare the power consumptions of different platforms. For the baselines, we have a 16-threaded program on Intel W5580 quad-core CPU, whose active power is estimated using PowerAPI [2]. Also, a CUDA program is tested

TABLE III
POWER AND PERFORMANCE OF DIFFERENT PLATFORMS

Devices	Time	Power	Energy/Sample
Xeon W5580	25.7ms	68.0W	1747.6mJ
Tesla K20	0.270ms	102.4W	27.6mJ
Jetson TK1	13.48ms	2.5W	33.7mJ
TN-10 1.0V	20ms	104.1mW	2.1mJ
TN-10 0.8V	20ms	49.22mW	0.98mJ

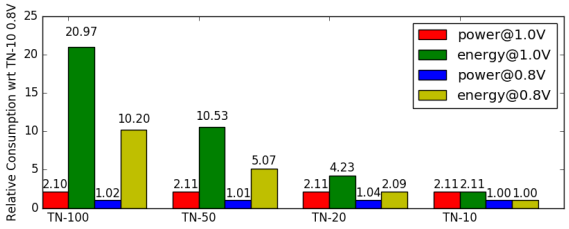


Fig. 9. Power/Energy Consumptions for Different Window Lengths

for active powers on an NVIDIA Tesla K20c workstation and the embeded GPGPU system Jetson TK1 using NVIDIA-smi and external Power meter. All baselines are supported by the AnRAD framework. The TrueNorth power is tested by the method in Cassidy et. al. [4]. To find the actual power consumption for the chip utilization, first the leakage power P_{leak} is measured when the system is idle, and then the total power P_{total} is measured with the network running. The active power is computed as $P_{\text{active}} = P_{\text{total}} - P_{\text{leak}}$. The leakage power is scaled by the number of cores actually used $P_{\text{leak\_scaled}} = P_{\text{leak}} N_{\text{cores}}/4096$. The final power is calculated as $P = P_{\text{active}} + P_{\text{leak\_scaled}}$. TrueNorth is capable of operating on 1.0V or 0.8V with the same 1ms ticks.

Table III shows the results of the baselines and the TrueNorth networks with 10-tick code windows. The spiking network is not only running faster than the CPU program, but is also 800/1700 times more energy efficient: it only consumes 2.1mJ at 1.0V and 0.98mJ at 0.8V to process each sample. Although K20 is capable of achieving a higher throughput, it runs on a much higher power and consumes up to 30X more energy compared with the TrueNorth chip. The Jetson board also consumes low power, but overall the energy consumption is still much higher than that of the spiking networks. The DARPA network has around 50000 key neurons, each of which integrates 11 support activations per sample on average. Take TN-10 0.8V for example, the power efficiency is estimated as (#neurons$\times$#supports)/(time_per_sample$\times$power) $\approx 6 \times 10^8$ operations per watt-second. Note that the cores are not fully occupied, and smaller code window offers higher efficiency.

Finally, we test the power consumptions with different accuracies. In Fig. 9, we vary the code window from 10 to 100 and report the power and energy of TrueNorth implementation normalized with respect to the power and energy of TN-10 at 0.8V voltage. Obviously, high supply voltage results in about 2X higher consumptions. All accuracy settings are basically power-neutral because the spike frequencies do not change much given the same amount of time. However, since the processing time for each frame reduces with smaller windows, the energy usage is also less at high-efficient setting TN-10.

VI. CONCLUSION

This paper presents a streaming anomaly detection network using TrueNorth neurosynaptic processor. A trained confabulation network is mapped to Corelet with our NeoInfer-TN library. The network uses an efficient burst code and features a highly concurrent architecture. The implementation achieves state-of-the-art detection precision, real-time processing and high power efficiency.

REFERENCES

[1] A. Amir, P. Datta, W. P. Risk, A. S. Cassidy, J. A. Kusnitz, S. K. Esser, A. Andreopoulos, T. M. Wong, M. Flickner, R. Alvarez-Icaza, et al. Cognitive computing programming paradigm: a corelet language for composing networks of neurosynaptic cores. In *2013 International Joint Conference on Neural Networks*, pages 1–10. IEEE, 2013.

[2] A. Bourdon, A. Noureddine, R. Rouvoy, and L. Seinturier. Powerapi: A software library to monitor the energy consumed at the processlevel. *ERCIM News*, 2013(92), 2013.

[3] L. Buesing, J. Bill, B. Nessler, and W. Maass. Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons. *PLoS Comput Biol*, 7(11):e1002211, 2011.

[4] A. S. Cassidy, R. Alvarez-Icaza, F. Akopyan, J. Sawada, J. V. Arthur, P. A. Merolla, P. Datta, M. G. Tallada, B. Taba, A. Andreopoulos, et al. Real-time scalable cortical computing at 46 giga-synaptic ops/watt with. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 27–38. IEEE, 2014.

[5] Q. Chen and Q. Qiu. Enhancing bidirectional association between deep image representations and loosely correlated texts. In *2016 International Joint Conference on Neural Networks*. IEEE, 2016.

[6] Q. Chen, Q. Qiu, H. Li, and Q. Wu. A neuromorphic architecture for anomaly detection in autonomous large-area traffic monitoring. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 202–205. IEEE, 2013.

[7] Q. Chen, Q. Wu, M. Bishop, R. Linderman, and Q. Qiu. Self-structured confabulation network for fast anomaly detection and reasoning. In *2015 International Joint Conference on Neural Networks*. IEEE, 2015.

[8] S. K. Esser, A. Andreopoulos, R. Appuswamy, P. Datta, D. Barch, A. Amir, J. Arthur, A. Cassidy, M. Flickner, P. Merolla, et al. Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores. In *2013 International Joint Conference on Neural Networks*, pages 1–10. IEEE, 2013.

[9] R. Hecht-Nielsen. *Confabulation theory: the mechanism of thought*. Springer-Verlag New York, Inc., 2007.

[10] D. C. Knill and A. Pouget. The bayesian brain: the role of uncertainty in neural coding and computation. *TRENDS in Neurosciences*, 27(12):712–719, 2004.

[11] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, et al. Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition*, volume 2, pages 12–26. IEEE, 2000.

[12] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[13] M. Oster and S.-C. Liu. Spiking inputs to a winner-take-all network. *Advances in Neural Information Processing Systems*, 18:1051, 2006.

[14] M. A. Petrovici, J. Bill, I. Bytschok, J. Schemmel, and K. Meier. Stochastic inference with deterministic spiking neurons. *arXiv preprint arXiv:1311.3211*, 2013.

[15] R. Preissl, T. M. Wong, P. Datta, M. Flickner, R. Singh, S. K. Esser, W. P. Risk, H. D. Simon, and D. S. Modha. Compass: A scalable simulator for an architecture for cognitive computing. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 54. IEEE Computer Society Press, 2012.

[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[17] Z. Zhao, K. G. Mehrotra, and C. K. Mohan. Ensemble algorithms for unsupervised anomaly detection. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 514–525. Springer, 2015.