Associative Memory Based on Sparse-Clustered Network with Selective Decoding for Internet Packet Classification

Scott Dickson Dagondon

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

August 2016

A thesis submitted to McGill University in partial fulfilment of the requirements for the degree of Master of Engineering.

2016/08/05

Abstract

To enforce Internet security protocols and define Quality of Service (QoS), Internet Service Providers (ISPs) need to identify the applications that are consuming network bandwidth. Distinguishing between safe and malicious traffic aids in network intrusion detection and interception. Likewise, categorizing applications into classes aids in traffic management for better service. ISPs use Internet Packet Classification (IPC) to categorize packets into flows (traffic sharing IP addresses, ports, and protocol), and thereby the classes generating them. Traditional IPC based on port numbers and payload pattern recognition are no longer effective because current applications can dynamically change port numbers and cipher their contents. Recent machine learning (ML) IPC have speed-bounded accuracy, and complex implementation due to the need to track packet sizes and order of arrival. This work proposes a new IPC approach that uses associative memory (AM) based on sparse-clustered network with selective decoding (SD-SCN). Unlike ML approaches, this solution takes bits extracted directly from the flow ID as input, which greatly reduces system complexity. It achieves 99.3% accuracy, consumes only 44 Mbits of memory, and runs 775 times faster than the state-of-the-art FPGA-implemented approach, which uses Support Vector Machines.

Abrégé

Afin de renforcer les protocoles de sécurité de linternet et de définir la qualité de service, les fournisseurs du service dinternet (FSI) ont besoin d'identifier les applications qui consomment la bande passante du réseau. La distinction entre les trafics sûrs et merveilleux, aide à la détection ainsi quà linterception des intrusions du réseau. De même, la catégorisation des applications dans des classes, aide à la gestion du trafic aux fins dun meilleur service. FSI utilisent le paquet de classification dinternet (PDI) pour classer les paquets en flux (traffic de partage des adresses IP, les ports et protocoles), ainsi les classes d'application les génèrent. LPDI traditionnel basé sur les numéros de port et la reconnaissance des formes de la charge utile, nest plus efficace. La capacité des applications actuelles à changer dynamiquement les numéros de port et de chiffrer leur contenu a déjà contourné cette technologie. Les récents apprentissage machine (AM) PDI ont une précision de vitesse limitée et une mise en uvre complexe en raison de la nécessité de suivre la taille et l'ordre d'arrivée de paquets. Ce travail propose une nouvelle approche PDI qui utilise la mémoire associative basée sur un réseau clairsemé-regroupé avec un décodage sélectif. Contrairement aux approches AM, cette solution prend les bits extraits directement à partir de l'ID du flux comme entrée, ce qui réduit considérablement la complexité du système. Elle atteint 99,3% de précision, ne consomme que 44 Mbits de mémoire, et fonctionne 775 fois plus rapide que létat de l'approche de lart, mis en uvre sur le FPGA, qui utilise Machine à vecteurs de support.

Acknowledgments

This research is funded by the Natural Sciences and Engineering Research Council of Canada. I would like to extend my sincerest gratitude to my thesis supervisor, Prof. Brett H. Meyer, for cultivating the idea that research is more than just about grand ideas. Behind every publishable work are iterations of hypotheses, simulations, and analyses. Prof. Meyer helped me understand that surviving graduate studies warrants recognizing the implicitly colossal amount of labour that is involved not only in developing ideas, but in proving why they are valid, and why the world should care. I would also like to thank Prof. Warren J. Gross for his expertise on associative memories, and valuable guidance he has given throughout the course of my research. Prof. Gross has a natural ability to communicate a complex topic such as sparse-clustered networks into a more easily comprehensible material; this is evident in all his talks and published works. I would also like to thank Prof. Vincent Gripon and Prof. Naoya Onizawa for taking the time to talk to me about their works on associative memory, and sharing their knowledge about the topic. I would also like to acknowledge The Telecommunication Networks Group at Universitá di Brescia in Italy for providing the network traces that were used to train and test the designs that are presented in this work.

My move from Manila to Montreal has been seamless; thanks to thoughtful family and friends. My parents have been very supportive in ways I cannot begin to count. When I was younger, they were just mom and dad. Today, to me, they are two human beings with memories, visions, and politics, who have unfathomable affection for me and for each other. I appreciate their presence and love greatly. I also want to thank my old friends back in the Philippines who listened when I talked about my plans of leaving my job of over five years to go back to school; these are friends that I intend to keep around for a long time. I want to thank my new friends here in Montreal who are constantly teaching me new things each day. I want to thank my roommate, Jacques Fortier, for the free event passes at Parc Jean-Drapeau; these cultural events have helped me live and appreciate Montreal regardless of the season. I also want to thank my other roommate, Marc Bourgeois, who works as a sommelier and has, in his words, “never paid for a bottle of wine in over 20 years”; I appreciate the free lessons on food, wine, and classical music, which served as welcome breaks and rewards for hitting research milestones. I want thank my friends from different parts of the world, who have introduced me to different cultures, food, and

languages. Like me, they (or their parents before them) have decided that they are strong enough to conquer Montreal winter. And like me, they value and invest in our relationship as much as they value hard work and success. Lastly, it took three brilliant friends from three different French-speaking countries to translate my academic abstract en Français; I want to acknowledge Thomas Lariviere (France), Mounir El Houssaini (Morocco), and my roommate Jacques (Canada) for their patience and hard work.

Contents

List of Figures

List of Tables

List of Acronyms

ISP	Internet Service Provider
QoS	Quality of Service
VoIP	Voice-over-Internet Protocol
P2P	Peer-to-Peer
SLA	Service-Level Agreement
IPC	Internet Packet Classification
ML	Machine Learning
SVM	Support Vector Machines
SCN	Sparse-Clustered Network
SD	Selective Decoding
AM	Associative Memory
RAM	Random Access Memory
LSM	Link Storage Module
LD	Local Decoder
GD	Global Decoder
SPM	Serial Pass Module
FPGA	Field-Programmable Gate Array

Chapter 1

Introduction

Internet Service Providers (ISPs) and network administrators deal with Quality of Service (QoS) and security concerns on a daily basis. QoS defines the assurance that a set of applications may only suffer a sufficiently low delay, or packet loss, which is usually accomplished by reserving bandwidth or buffer space through some form of packet prioritization [1]. Different applications have different QoS requirements; for instance, Voice-over-IP (VoIP) applications are more sensitive to packet losses and end-to-end delay, and therefore require some level of priority over applications, which are more resilient to the same type of network issues, such as peer-to-peer (P2P) file sharing [2][3]. In order to provide continuous service, ISPs and network administrators must ensure that resources required by specific applications — especially the ones deemed critical — are readily available. These types of service guarantees and the corresponding consequences for not meeting service obligations are included in the Service-Level Agreement (SLA), which is the contract between ISPs and end users [4]. Defining a reasonable and competitive QoS, therefore, is decidedly important because this is how ISPs can optimize bandwidth and maximize profit. Moreover, and perhaps more importantly, a well-defined QoS translates to better network management and service, which help retain customers. Likewise, network administrators strive to attain a well-defined QoS in order to avoid network downtime, which, depending on the criticality of the system that the network supports, may be exceptionally undesirable.

Detecting and intercepting malicious traffic is also a crucial part of the responsibility of ISPs and network administrators. Malicious codes or "malware" refer to any application that is specifically created to cause an unexpected and unwanted event on a user's com-

puter, or a server [5]. Some forms of malware try to extort money from users of infected computers by encrypting their files and demanding payment in exchange for the decryption key [6]. It has been reported that between June 2014 and March 2015, losses due to this type of malware, duly termed as "ransomware", have totalled to more than $1.1 million [7]. Other forms of malware steal personal identifications such as names, phone numbers, and addresses, which could be used to impersonate a person for fraudulent purposes [6]. In order to enforce Internet security protocols, and similarly define QoS, ISPs and network administrators must identify the types of applications that are consuming network bandwidth. For security, the ability to distinguish between safe and malicious applications is useful for network intrusion detection. For QoS, the ability to categorize applications into classes is useful for better traffic management and fair service pricing. To address these security and QoS concerns, ISP and network administrators use real-time Internet Packet Classification (IPC) [8]. IPC is the process of categorizing packets into *flows*, and thereby their generating applications. A flow refers to a group of packets that use the same source and destination IP addresses, ports, and protocol. Flows with QoS constraints can be subsequently prioritized, while malicious traffic can be dropped; as such, IPC functions are often attached to packet forwarding engines such as routers [9].

Traditional methods of IPC include application detection based on port numbers, and payload pattern recognition [10]. Methods based on port numbers rely on the fact that conventionally, most applications use a specific set of well known ports for communication. On the other hand, methods based on recognizing patterns on the packet's payload rely on the fact that most packets contain application-specific signatures. These approaches however are no longer effective because current applications can dynamically change their port numbers and/or cipher their contents [11]. More recent IPC approaches use machine learning (ML) algorithms. These approaches generate a model based on features extracted from packet headers [10][12]. The main problem with ML-based approaches, however, is that the models are often large, complex, and therefore slow — many of the features of two flows belonging to the same class may be uncorrelated (*e.g.*, source IP), increasing classification complexity — and high packet throughput limits the time available for any single classification decision. Moreover, recent studies on this area focus on improving algorithm accuracy, ignoring the cost (*e.g.*, classification speed, memory) associated with real-time hardware implementation [13][14][15][16]. One of the more recent ML-based solutions developed for FPGA implementation, which uses Support Vector Machines (SVM), is able to

attain a very high ($\sim$ 100%) classification accuracy by using a minimum of 1000 support vectors. This high number of support vectors imposes a high classification delay, and limits the solution to networks with packet arrival speed of $\leq$ 350 Kpackets per second [11]. Beyond this rate, classification fails and accuracy drops to zero. Furthermore, like most ML solutions in literature, SVM-based approaches are only able to attain high accuracy by taking multiple packet sizes per flow as features, which requires a flow builder. A flow builder stores flow IDs, monitors flow expiries, and uses a hash function for accumulating the required number of packet sizes belonging to the same flow into specific addresses in a cache prior to classification [11][17][18][19]. This packet size dependence prolongs classification, and increases system complexity and cost.

The formidable task of selecting an IPC approach to address the issues on security and QoS is aggravated by the fact that we live in a high-speed big data world where as much as 2.5 quintillion bytes of data are created everyday [20], and data transmissions run at speeds as high as 10 Gb/s [21]. An effective IPC algorithm therefore is one that can classify a high number of flows with maximum accuracy, and minimum memory requirement and classification delay. This work proposes a new method of IPC using a Sparse Clustered Network (SCN) with Selective Decoding (SD). SD-SCN is a highly scalable flavour of associative memory (AM), capable of storing a large number of messages, and employing less complex hardware for match retrieval compared to earlier SCN implementations [22][23][24][25]. An SD-SCN is made up of clusters, each representing a component of the full binary message to be stored. The SD-SCN-based IPC solution proposed in this work takes a class-labelled flow ID, *i.e.*, the concatenation of the binary values of the IP addresses, ports, and protocol characterizing a flow, and the class of application generating it, as input. By using the flow ID directly, this solution removes the need for a flow builder, thereby reducing system cost and complexity. Naive IPC with SD-SCN, however, achieves poor accuracy and limited design flexibility: each class-tagged flow ID is long (112 bits), confined to a single network configuration that consumes 12 Mbits of on-chip memory; and, many subsections of the flow ID are expected to take on a limited number of values (*e.g.*, source IP), limiting the ability of SD-SCN to differentiate flows, resulting to accuracy as low as 15%.

This work demonstrates that when the input message is modified by (a) ignoring flow ID bits that tend not to change, and (b) artificially increasing the number of (application) classes, the IPC performance of SD-SCN is substantially improved: the solution is able to process one flow in 4.1 nanoseconds, and achieve 99.3% accuracy, while only using 44.3

Mbits of memory. Compared to the state-of-the-art FPGA-implemneted SVM approach, the solution proposed in this work is able to achieve about the same accuracy at 775 times faster classification speed, making it suitable for networks with packet arrival speed up to 244 Mpackets per second. Moreover, SD-SCN-based IPC uses at least 5.8 times less RAM, which is made possible by removing the 256-Mbit cache for packet size tracking that comes with the flow builder [11][17][18][19].

Chapter 2

Background

2.1 Sparse-Clustered Network with Selective Decoding

An associative memory (AM) is a form of memory that generates a match to a given input by searching in parallel components of previously learned data, instead of using explicit addresses like in conventional memory. This makes AM ideal for applications such as recommender systems [26], which provide recommendations for items like books and music, and data mining [27], where search operations can be initiated given partial input. SD-SCN is a highly scalable implementation of AM, which offers high storage capacity and efficiency; it is able to use less complex hardware for data retrieval by removing the matrix multiplication and max-functions of early SCN implementations [22][23][24][25].

2.1.1 Data Learning

During learning, an input message of width K is broken into c k-bit sub-messages, and one sub-message is assigned to one cluster. Figure 2.1 shows how SD-SCN maps a K-bit message (data to be stored) into c clusters of l binary neurons. In this case, $K = 16$, $c = 4$, and $k = K/c = 4$. Note that K, c, and k are all integers. This means that an input of width K can only be supported by a finite set of SD-SCN configurations, which is defined by the different integer c-and-k combinations. The number of neurons in a cluster, l, is calculated using:

$$l = 2^k = 2^{K/c} \tag{2.1}$$

where k is the number of bits per sub-message.

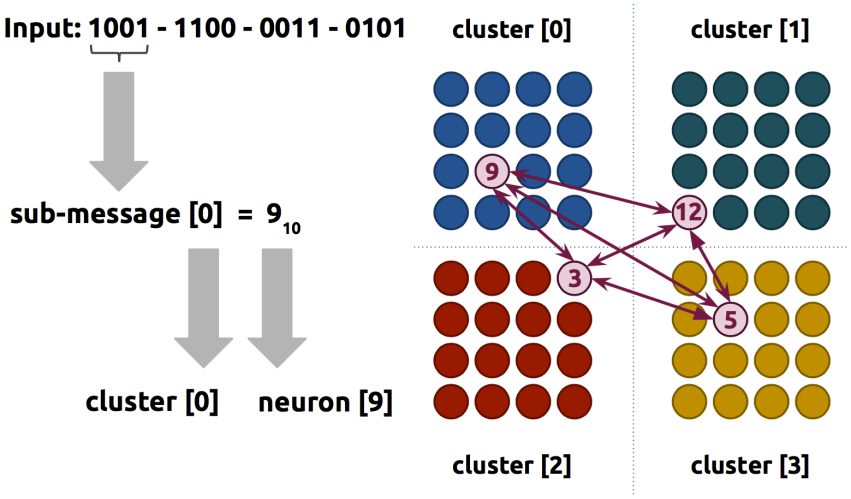

Fig. 2.1 Input message mapping during SD-SCN learning.

In each cluster, c_i, the neuron with index j that is equal to the integer equivalent of the input sub-message is activated during learning. A cluster in the network can have index $i = 0, 1, ..., c-1$; a neuron in the i^{th} cluster, $l_{(i,j)}$, can have an index $j = 0, 1, ..., l-1$. The c active neurons (one from each cluster) are then linked. All the unidirectional links from one neuron in one cluster, to all the other active neurons in the other clusters, are stored in memory. Since there are c clusters, there are a total of $c \times (c-1)$ links for each learned message. The complete set of links for one learned message is called a "clique". SD-SCN uses one RAM block to store all the possible unidirectional links between all neurons in a cluster pair, which means that there are also a total of $c \times (c-1)$ RAM blocks in the SD-SCN network.

Figure 2.2 shows the pairing of cluster c_0 to the rest of the clusters in the network. Each time an input message is introduced into the network, the first sub-message is assigned to cluster c_0, the second, to cluster c_1, and so on. The outgoing links from c_0 to a pair cluster are stored in one RAM block. During learning, when the complete message is provided to the network, each cluster (and sub-message) will always be associated to the rest. This means that for cluster c_0 (first sub-message), associations to clusters c_1 (second sub-message), c_2 (third sub-message), ..., and c_{c-1} (final sub-message) will always exist. Maintaining these one-way associations, and using dedicated RAM blocks for links storage, is the reason why, during retrieval, SD-SCN can generate a match even when the input is

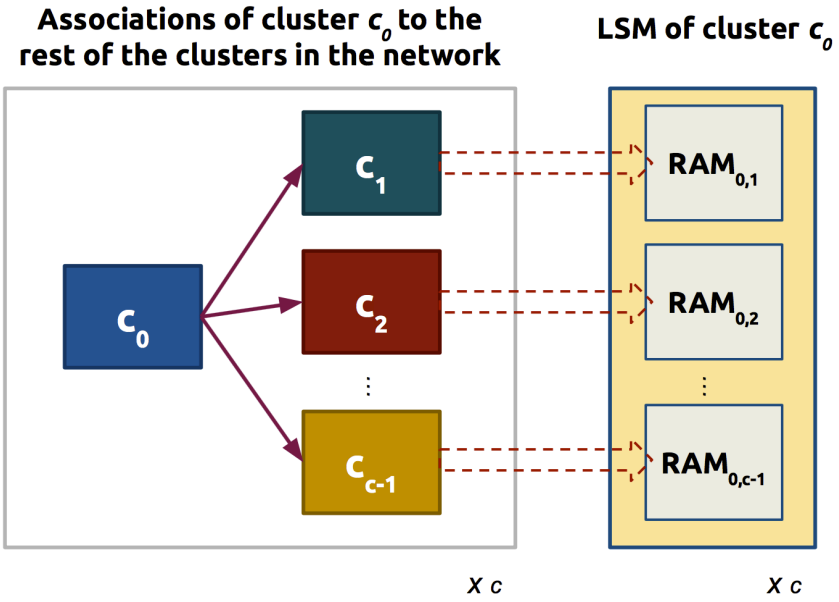

Fig. 2.2 Inter-cluster links storage into the LSM.

partially erased.

2.1.2 Data Retrieval

In hardware, an SD-SCN cluster is made up of the logic that is used to decode the neuron that represents the input sub-message, and the memory blocks that contain the previously learned associations between sub-messages, *i.e.*, the unidirectional links from a cluster's local neurons to the neurons of the rest of the clusters in the network. Figure 2.3 shows the block digram of an SD-SCN cluster. As discussed in the previous section, since each sub-message is associated to $c-1$ other sub-messages, an SD-SCN cluster, which represents one sub-message, uses $c-1$ RAM blocks to store these associations. During data retrieval, if the input sub-message to a cluster, $\mathbf{input\_read}_i$, is specified, SD-SCN uses the cluster's local decoder (**LD**) to activate neuron $l_{(i,j)}$, which represents the sub-message. The network then proceeds to check if this neuron, and hence, sub-message, is part of a clique, *i.e.*, full message, that was learned in the past. To do this, the global decoder (**GD**) must ensure that $l_{(i,j)}$ is linked to at least one neuron in each of other $c-1$ clusters in the network. First, the LD forwards the k-bit index of neuron $l_{(i,j)}$ to the LSM. This k-bit signal represents the address of the local "source" neuron, which is sent to the $c-1$ RAM blocks, which

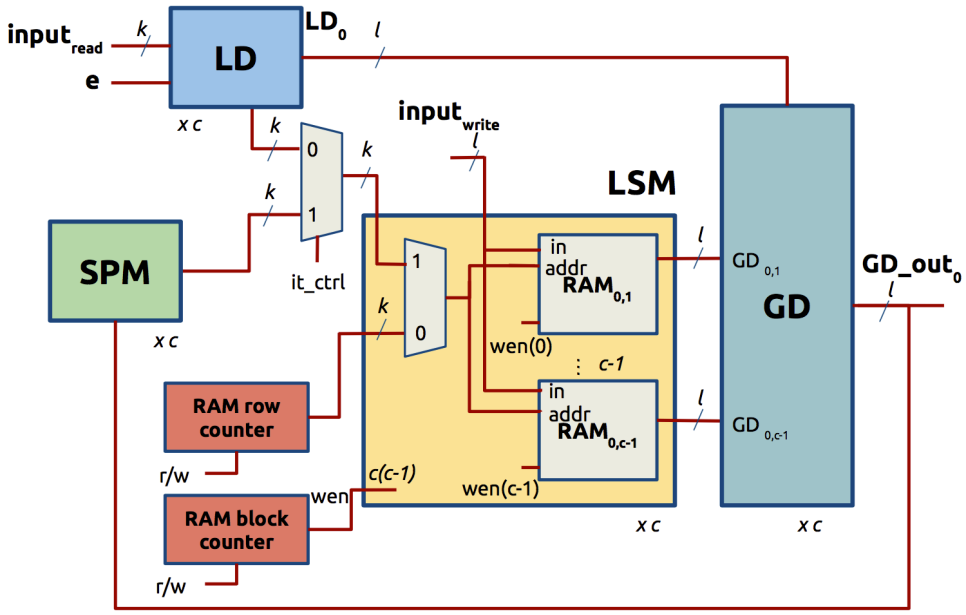

Fig. 2.3 Block diagram of an SD-SCN cluster.

contain the previously learned associations. In turn, the $c-1$ RAM blocks forward l-bit signals to the GD. Each l-bit signal represents the associations of the source neuron to the neurons of a cluster pair, and each bit in the signal represents an index of a "destination" neuron in that cluster. For instance, the l-bit output of RAM block $\mathbf{RAM}_{0,1}$ contains the associations of the input to cluster c_0, *i.e.*, neuron $l_{(i,0)}$, to all the neurons in cluster c_1. The RAM blocks, therefore, can be viewed as a matrix where the the rows, referenced by the k-bit signal from the LD, represent the source neurons, and the columns, the destination [24]. The output of a RAM block is the entire row of data specified by the LD; each bit in the output represents the association between the source neuron and a destination neuron. The RAM row and RAM block **counters** generate the appropriate indexes representing the source neuron and cluster pair for links checking.

The l-bit signal from a RAM block to the GD has a "1" at bit locations that represent indexes of destination neurons that are linked to the source neuron. Specifically, a "1" signifies that the local neuron and the corresponding destination neuron were activated at the same time in the past because they represent two (of the c) components of a learned full message. If at least one link exists between the local (source) neuron and an *active* neuron in each of the (destination) cluster pair, *i.e.*, there is a '1' at bit locations representing the

active neurons in the other clusters, then neuron $l_{(i,j)}$ becomes the output of the i^{th} cluster's GD, $\mathbf{GD\_out}_i$. To illustrate, if a network has four clusters, $c = 4$, each with 16 neurons, $l = 16$, a 16-bit zero-trailing signal of "10100..." between the topmost RAM block of c_0, $\mathbf{RAM}_{0,1}$, and its GD means that a unidirectional link exists from the local neuron, *i.e.*, the first sub-message, to the first ($i = 0$) and third ($i = 2$) neurons in c_1. If the decimal equivalent of the local neuron is 12, this means that during learning, (at least two) full messages following these patterns were introduced to the network:

1. "$12 - 0 - .. - ..$"
2. "$12 - 2 - .. - ..$"

If the 16-bit output of the lower two RAM blocks, $\mathbf{RAM}_{0,2}$ and $\mathbf{RAM}_{0,3}$, are "01000..." and "01001...", respectively, then the following full messages were introduced to the network during learning:

1. "$12 - 0 - 1 - 1$"
2. "$12 - 0 - 1 - 4$"
3. "$12 - 2 - 1 - 1$"
4. "$12 - 2 - 1 - 4$"

If the active neurons in c_1, c_2, and c_3, are $l_{(1,2)}$, $l_{(2,1)}$, and $l_{(3,4)}$, respectively, then neuron $l_{(12,0)}$ remains active because it is part of a learned full message that includes all the other active neurons. Finally, $l_{(12,0)}$ becomes the output of c_0, $\mathbf{GD\_out}_0$.

If the sub-message is *not* specified, *i.e.* "erased" – the input is *incomplete* – the erased flag, input **e**, is raised, and the LD activates *all* the local neurons in the cluster (Figure 2.4). In hardware, this is accomplished by sending an all 1s l-bit output, $\mathbf{LD}_j$, to the GD. This l-bit output tells the GD that any one of the local neurons can be the match. The role of the GD is then to track the number of "candidate" local neurons that has a complete set of $c-1$ unidirectional links to the neurons in the $c-1$ other clusters, which may be specified, or erased as well. During the first iteration, the GD forwards the information to the Serial Pass Module (**SPM**), which sends the indexes of the candidate the neurons, one per cycle, to the LSM. The LSM forwards the address of the candidate neuron being evaluated to the GD. We call the candidate neurons "ambiguities". Each ambiguity that does not have a

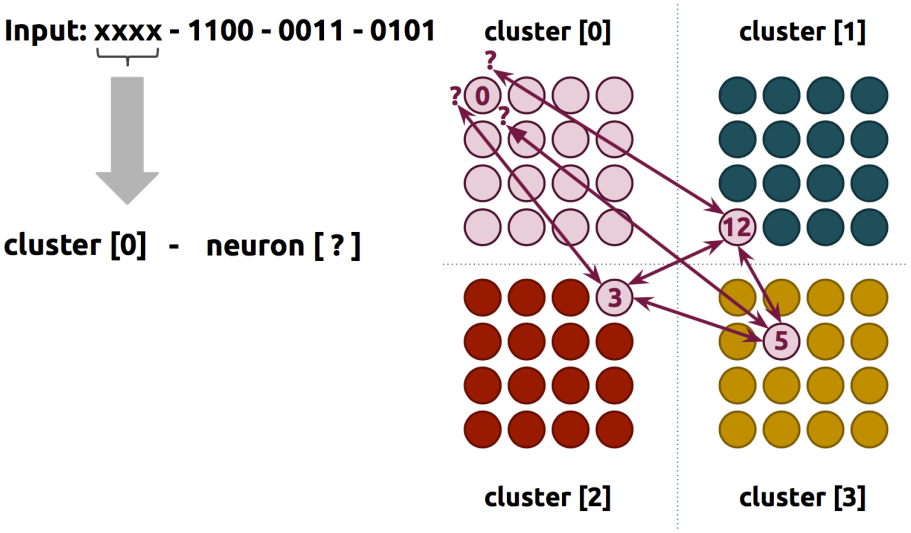

Fig. 2.4 Message decoding during SD-SCN retrieval.

complete $c-1$ set of outgoing links to the active neurons in the other clusters are deactivated by the GD. The goal of an SD-SCN cluster is then to eliminate the ambiguities one by one, until only one is left, at which point a match is generated and sent out as the final output, GD_out$_i$. If the network cannot narrow down the number of ambiguities in a cluster to one, an error is declared. Because the GD of an erased cluster ultimately decides which neuron remains active and becomes the final output, erased clusters are referred to as the *output clusters*. Clusters that are specified, on the other hand, which activate exactly one neuron per input, are called the *input clusters*.

2.2 SD-SCN Accuracy

SD-SCN accuracy refers to the ability of the network to generate a match given a full or partially erased input. We say that a network exhibits high accuracy if has a low error rate. Accuracy, therefore, can be seen as to the ability of the network to narrow down the number of ambiguities in an erased cluster to exactly one, and is calculated using:

$$accuracy = \frac{M_{read} - error}{M_{read}} = 1 - \frac{error}{M_{read}} \tag{2.2}$$

where *error* is the number of times an error is raised out of M_{read} read executions performed by the network. M_{read} is also the number of input messages, partial or complete, provided to the network during data retrieval phase. Each time an error is raised because the ambiguities cannot be resolved, the number successful match generation drops, and so does accuracy. Given a constant message width, K, total number of learned messages, M, and number of erased clusters, c_e, SD-SCN accuracy is determined mainly by the number of available links stored in memory, which is controlled by two factors: input distribution, and number of neurons per cluster, l.

2.2.1 Input Distribution

A non-uniform input refers to a set of learned messages where the values at certain bit locations change more frequently than others. The works in [28] and [29] showed that with everything else constant, error rate increases significantly with a non-uniform input. The correlation between input messages during learning results to a re-use of specific sets of links, instead of maximizing the total available links in the network, which makes distinguishing between stored messages difficult during retrieval.

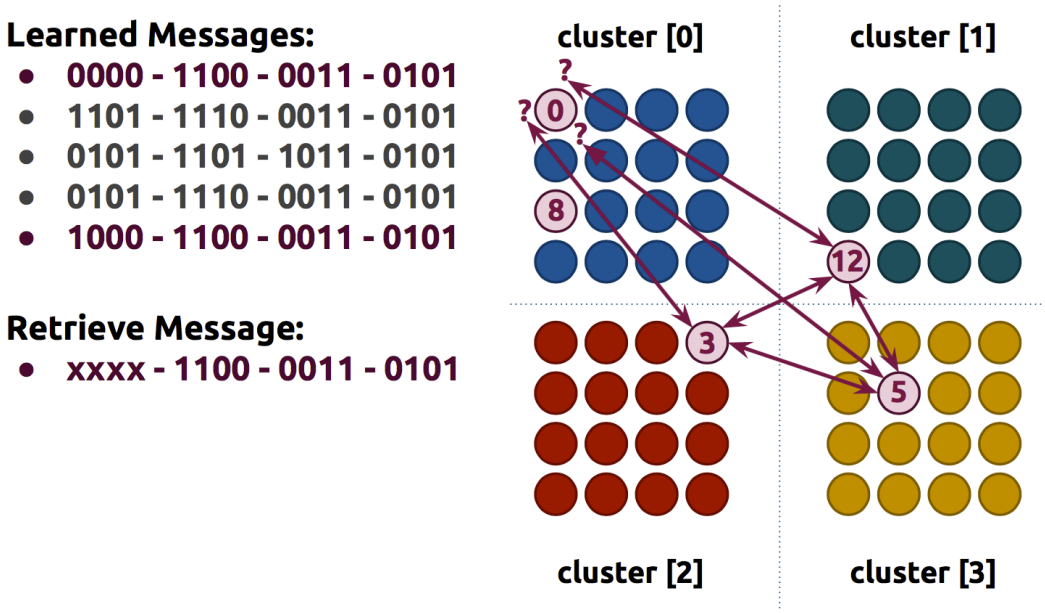

Fig. 2.5 Effect of learning non-uniform input to SD-SCN accuracy.

Figure 2.5 shows an example of a non-uniform input and its effect on neural activation.

Notice that the learned messages have a lot of non-switching bits. In fact, the fourth sub-message is the same for all messages, which means that throughout learning, only a single neuron was activated in cluster c_3. To retrieve a match to the input "xxxx$-1100-$ $0011-0101$", where the first sub-message is erased, the neurons in the specified clusters are first activated. The specified sub-messages are represented by the numbered neurons in clusters c_1, c_2, and c_3. As discussed in the previous section, the LD activates all the neurons in the erased cluster, and the GD deactivates ambiguities that are not linked to *all* the active neurons from the other clusters. In this case, the numbered neurons in cluster c_0, $l_{(0,0)}$ and $l_{(0,8)}$, are the remaining active ambiguities because both represent sub-messages that are associated to the active neurons in the specified clusters. Specifically, these active ambiguities represent the first sub-message of the first and last items on the list of learned messages, respectively. Notice that both full-messages share the same second, third, and fourth sub-messages, and only differ on first sub-message. Since the number of ambiguities in cluster c_0 cannot be narrowed down to exactly one, an error is raised, and overall accuracy drops. The inevitable reuse of neurons due to a non-uniform input results to the storage of cliques that use repeated sets of outgoing links. If an input is uniform, the bits in the full message change value at an almost equal degree throughout learning. This results to the activation of more neurons per cluster, which means more varied sources of outgoing links per clique. Increasing the unique sets of outgoing links stored in memory, by making sure that input distribution is uniform, is essential in order to increase network accuracy.

2.2.2 Neurons per cluster

The neurons represent the source and destination of the links that are stored in memory during the learning phase. Therefore increasing l increases the maximum theoretical number of links that the network can use to learn M messages. From Equation 2.1, we know that we can increase l by setting a small value for c, *i.e.*, using a network configuration with fewer clusters. The works in [25] and [30] showed that, with everything else equal, a network with a smaller l, and hence a bigger c, has a lower probability of successfully eliminating ambiguities because a fewer number of neurons translates to a fewer number of links that the network can use to differentiate previously stored messages.

2.3 SD-SCN Memory Requirement

SD-SCN uses one RAM block for the unidirectional links between neurons of one cluster to another. The total number of RAM blocks in the network is equal to the total number of cluster pairs, which is $c \times (c-1)$. A RAM block acts like a matrix where the rows represent indexes of the source neurons, and the columns, the destination. The size of each RAM block therefore is just equal to the product of the total number of neurons in the source and destination clusters:

$$\mu_{block}(l_{src}, l_{dst}) = l_{src} \times l_{dst} \tag{2.3}$$

where l_{src} and l_{dst} are the number of neurons in the source and destination clusters, respectively. Assuming the clusters in the network have uniform sizes, Equation 2.3 can be written as:

$$\mu_{block}(l) = l^2 \tag{2.4}$$

and the total memory requirement is just the size of a RAM block multiplied by the number of cluster pairs:

$$\mu_{total}(c, l) = c \times (c-1) \times (l)^2 \tag{2.5}$$

Equation 2.5 suggests that total memory, μ_{total}, increases quadratically with l. Since l is indirectly proportional to c, as shown in Equation 2.1, this also means that total memory, μ_{total}, decreases with an SD-SCN configuration that uses a fewer number of clusters.

SD-SCN clusters do not necessarily have to be equal in size. It was introduced in Section 2.1.2 that specified clusters are referred to as input clusters, and erased clusters as the output. This reference has to do with the role of the clusters in specific SD-SCN applications. For instance, if SD-SCN is applied to packet routing, the input can be the IP address of the packet, which can broken down into c_{in} clusters, and the output can be the routing rule to be applied [31]. If there are 256 routing rules and only one output, then c_{out} must have $l_{out} = 256$ neurons, and $k_{out} = \log_2(l_{out}) = 8$ bits, *regardless* of the number of c_{in} and l_{in}. For SD-SCN configurations where the input and output clusters don't share a common l, and hence k, the full input width, K, can be expressed as:

$$K = c_{in} \times k_{in} + c_{out} \times k_{out} \tag{2.6}$$

Similarly, Equation 2.5 can be expressed as a function of l_{in} and l_{out}:

$$\begin{aligned}\mu_{total}(c_{in}, c_{cout}, l_{in}, l_{out}) = c_{in} \times (c_{in} - 1) \times (l_{in})^2 \\ +2 \times c_{out} \times c_{in} \times (l_{in} \times l_{out})\end{aligned} \tag{2.7}$$

where l_{in} and l_{out} are the neurons in the input and output clusters, respectively, The first term in Equation 2.7 computes the sum of RAM block sizes involved in c_{in}-to-c_{in} pairings, whereas the second term computes that of c_{in}-to-c_{out} pairings.

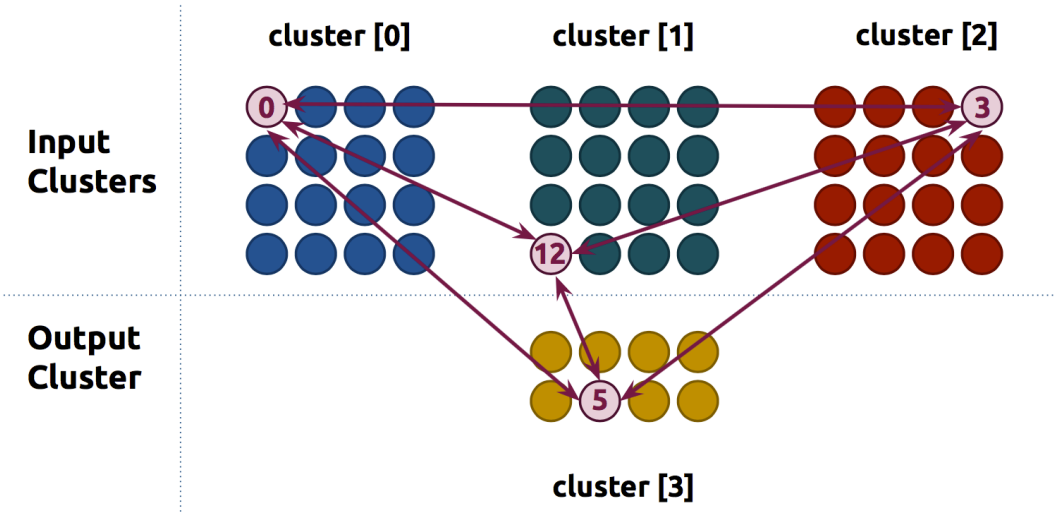

Fig. 2.6 Input message mapping in SD-SCN with $l_{out} < l_{in}$.

Figure 2.6 shows an example of how input message mapping occurs in an SD-SCN that uses input and output clusters that do not share l. Notice that the output cluster uses $l_{out} = 8$, which means that it takes a sub-message that is made up of $k_{out} = \log_2(l_{out}) = 3$ bits. During retrieval, this system receives three specified sub-messages, which are assigned to input clusters c_0, c_1, and c_2; the erased flag is raised for output cluster c_3. At the end of the decoding process for each message query, the lone active ambiguity in cluster c_3 becomes the SD-SCN output.

2.4 SCN Access Delay

The work in [24] models access delay as a function of the maximum number of serial accesses to the RAM block, *i.e.*, the maximum number of ambiguities during the first iteration. The first iteration refers to the complete execution of the LD and GD processes given specified and/or erased sub-messages. Specifically, access delay is computed using:

$$access\ delay\ = 2 + (\beta + 1) \times (it - 1) \tag{2.8}$$

where β is the number of ambiguities during the first iteration, and it is the number of iterations needed to generate a match. β is measured using simulation, whereas it is a function of the number of erased clusters, c_e. For $c_e = 1$, $it = 1$. That is, given one erased cluster, it only takes one iteration to generate a result; any ambiguities at the end of the first iteration cannot be resolved by performing further iterations because the number of active neurons from the other clusters hasn't changed, and so there's no new information, *i.e.*, links, that the GD can use to produce a new outcome.

On the other hand, if $c_e = 2$, the first iteration, $it = 1$, covers the (1) activation of the specified neurons in the input clusters and all the neurons in the (erased) output clusters (LD process), *and* (2) the deactivation of the ambiguities in both erased clusters, serially and simultaneously (GD process). As previously discussed, an ambiguity is deactivated if it is not connected to $c - 1$ active neurons (specified or ambiguous) at the time that it is being evaluated. Since an ambiguity in one erased cluster helps determine if an ambiguity in the other should stay active or not, and since the same ambiguity can be active and then deactivated within $it = 1$, a second iteration, $it = 2$, is needed to check if the active ambiguities declared during the first iteration are still active, *i.e.*, are still connected to at least one active neuron from each of the other clusters in the network. If $c_e = 1$, a second iteration is *not* necessary because the status of the ambiguities in the lone erased cluster only depends on the specified neurons from the input clusters, which are never going to be deactivated, *i.e.*, GD is not executed for the specified input sub-messages. $it \geq 2$ only involves the complete execution of the GD on the instantaneous number of ambiguities; LD is only executed at the start of $it = 1$, when sub-messages that make up a query are introduced into the network.

2.5 Packet Classification

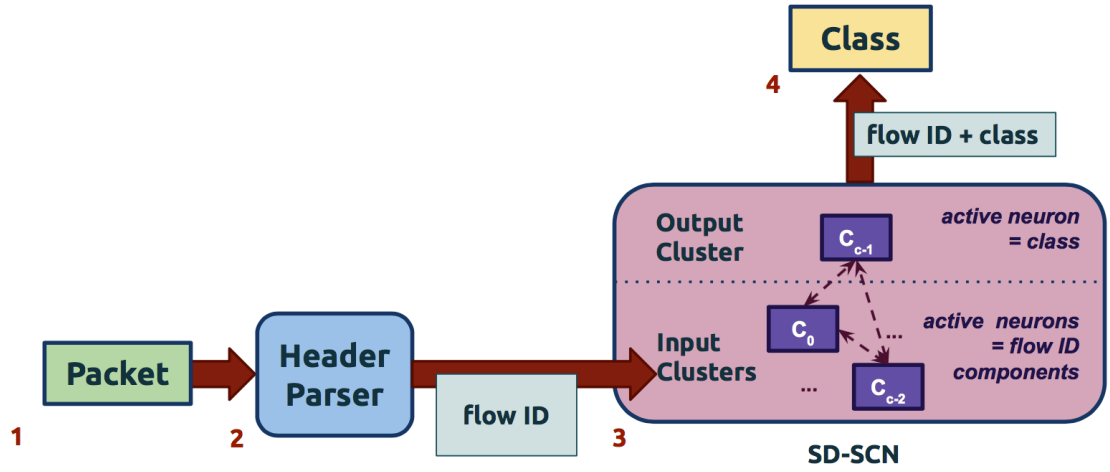

Fig. 2.7 IPC process using SD-SCN.

In general, IPC is the preprocessing necessary for discriminating and controlling packets transmitted over the Internet based on a set of predefined rules. As such, IPC is a crucial component in network operations such as routing, packet filtering, and traffic accounting [32]. For the purpose of defining QoS and security protocols, IPC is used to identify the generating classes (applications) of captured packets based on the information they carry. These useful information range from port numbers, packet sizes, and payload patterns [10][12]. Packets that carry the set of information that is associated with a particular class get routed based on the rule assigned to that class. For instance, packets classified to be generated from a streaming application get the appropriate resources set by network administrators. Likewise, packets classified to be generated by malicious applications can be dropped, *i.e.*, denial of service.

In order to use SD-SCN for IPC, it is first necessary to define the input source and structure, as well as viable network configurations, that will maximize classification accuracy, and minimize memory requirement and classification delay. Since IPC processes flows, the baseline input message to the SD-SCN-based IPC solution is the class-tagged flow ID (112 bits wide), which refers to the concatenated binary values of: source ($32b$) and destination ($32b$) IP addresses, source ($16b$) and destination ($16b$) port numbers, protocol ($8b$), and the generating class ($8b$) of captured packets. During learning, the input to the SD-SCN is the

full class-tagged flow ID bits, which will be mapped into the available clusters. Similar to the works in [11] and [12], this work will only consider packets transmitted in UDP and TCP, and classify flows into eight classes: *Web*, *Stream*, *RC*, *P2P*, *Mail*, *IM*, *Download*, and*Game*. During retrieval, the input is the flow ID bits, *i.e.*, the class bits are erased (see Figure 2.7). The network then searches through its stored links to find a clique that includes the neurons representing the specified flow ID bits, and a neuron in the output cluster representing a class . Like most implementations of SD-SCN, the output – in this case the class bits – will have a *dedicated cluster*. The flow ID bits, on the other hand, which will be specified during learning and retrieval, will be mapped into the input clusters. The following chapter will discuss how to develop and test an SD-SCN that performs IPC using a real network trace, which is made up of $80,000$ flows associated with the different classes, obtained from a campus network in University of Brescia [33].

Chapter 3

Optimizing SD-SCN for IPC

3.1 Problem Statement

The fundamental challenge in using SD-SCN for IPC is defining the input message, in much the same way that ML-based approaches need to identify the set of features that produce the best classification accuracy. Chapter 2 details the effect of a non-uniform input distribution to SD-SCN accuracy; the reuse of specific sets links during learning limits the ability of the network to distinguish between stored messages during retrieval. Additionally, the width of the input message, K, limits the possible values of c, and hence l, which in turn affects the accuracy and size of SD-SCN. Input characterization, therefore, is a very crucial step in optimizing SD-SCN for IPC since it singularly controls both the performance and cost of resulting designs. In order to bound the design space, and similarly minimize cost, this work proposes the use of the readily available information from the class-labelled flow ID, *i.e.*, source and destination IP addresses, ports, and protocol, as input to the IPC-optimized SD-SCN. This choice makes it possible to design an IPC that does away with packet sizes and order of arrival, thereby eliminating the flow builder.

Figure 3.1 highlights the part of the state-of-the-art ML-based IPC and this work proposes to replace with SD-SCN. Eliminating the flow builder eliminates the logic and cache used in tracking the status of flows, and sizes of packets and order of arrival. Given the SD-SCNs sensitivity to the distribution and width of the input, this work's first-order goal, therefore, is to identify which of the $K = 112$ bits in the baseline input allows for an SD-SCN design that *maximizes accuracy*. Similar to the work in [11][17], this work aims to develop an IPC that consumes *minimum memory* to be implementable on an FPGA. Addi-

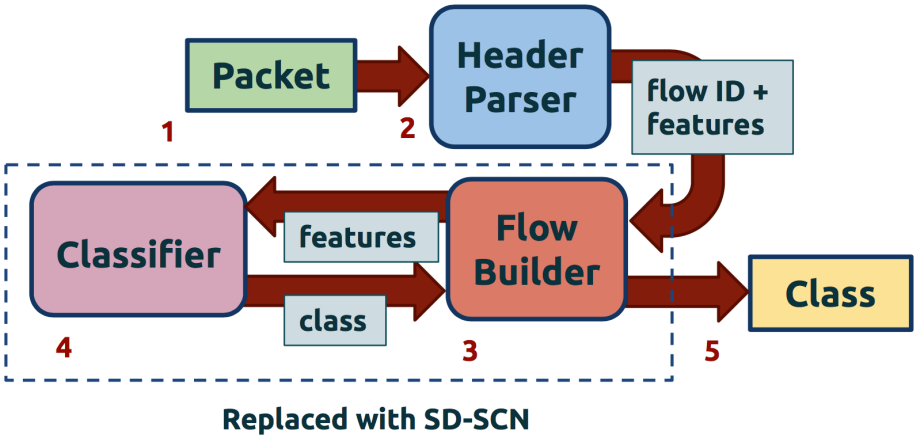

Fig. 3.1 Comparison of ML and SD-SCN solutions to IPC.

tionally, since IPC systems are normally attached to network routers, it is imperative that the IPC-optimized SD-SCN has a *minimum classification delay*, so as not to introduce a bottleneck in packet routing. Considering that the core of SD-SCNs retrieval process relies on accessing previously stored links, we will model classification delay using access delay figures from CACTI [34], which takes RAM block sizes as input, among other parameters (discussed in detail in Chapter 4). We know based on Equation 2.4 that l, and hence k, defines RAM block sizes. The biggest SD-SCN RAM block size that can be implemented on an FPGA (2 MB) corresponds to a design that uses $k = 12$ bits per cluster. On the other hand, the smallest SD-SCN RAM block size that CACTI can model (128 B) corresponds to a design that uses $k = 5$ bits per cluster. For these reasons, and in order to simplify modelling, this work proposes to constrain the design space to $k = [5, 12]$. Coincidentally, SD-SCN configurations that use $k \leq 5$ employ smaller RAM blocks ($\leq$ 128 B), which have a higher overhead to useful memory ratio compared to the alternative, rendering them inefficient for high-speed operations such as IPC. The limit on k is the reason why the class, which occupies a dedicated cluster as mentioned in Section 2.5, is formatted using one-hot encoding. That is, even though the eight classes can be identified using only three bits (*i.e.*, $2^3 = 8$), we use eight to satisfy: $5 \leq kout \leq 12$. Given a network trace of M flows, this work's *final goal* is to identify which of the $K = 112$ class-tagged-flow-ID bit indexes should make up the final input, in order to design an SD-SCN that satisfies the k limit, achieves maximum accuracy, and consumes minimum memory and classification delay.

3.2 Maximizing Cluster Utilization

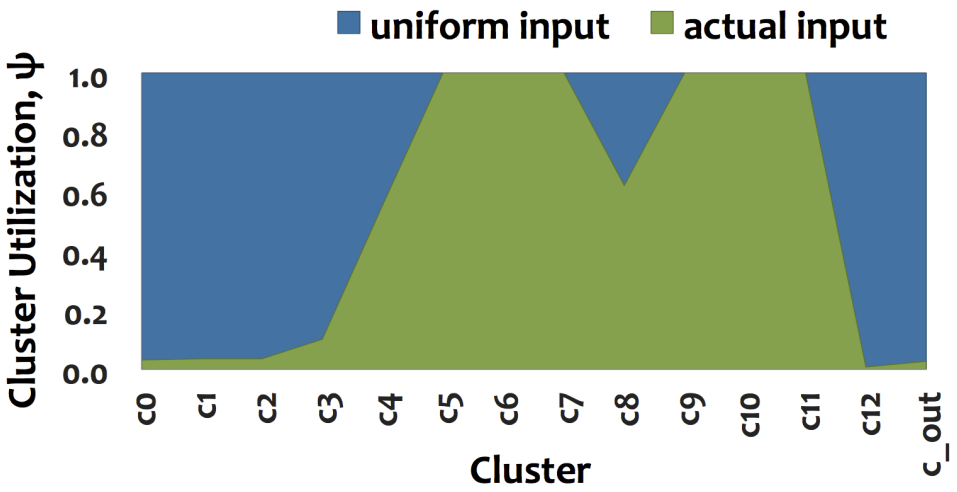

Fig. 3.2 Clusters utilization comparison of uniform (randomly generated) and real (actual network trace) inputs.

Uniformly distributed input produces high SD-SCN accuracy because it maximizes the available inter-cluster links during learning, which allows the network to more easily differentiate between stored messages during retrieval (Chapter 2). In the context of IPC, a uniform input refers to a network trace that is able to use all available neurons at virtually equal frequency during learning. Since an active neuron is simply the integer equivalent of a binary sub-message, this work argues that an input set is more likely able to maximize a design's available inter-cluster links if the values of its sub-messages vary, *i.e.*, "switch", frequently throughout the learning process. Similarly, the probability that the value of a sub-message switches increases as the value of its individual bits switch from learning one input to the next. A constantly switching sub-message has a higher probability of utilizing most of the available neurons in a cluster during learning simply because each time a new – or more specifically, unique – sub-message is received, a new neuron is activated. A cluster that receives a limited number of unique sub-messages will activate a limited number of unique neurons, and therefore will result to a limited number of unique links, since the neurons represent the link sources and destinations. In contrast, a cluster that receives a significantly higher number of unique sub-messages utilizes more of its available neurons, and will result to more unique links being stored in memory. This work refers to the ra-

tio of activated neurons to total neurons in a cluster as cluster utilization, which can be calculated using:

$$\psi_i = \frac{l_i^*}{l_i} \tag{3.1}$$

where ψ_i is the i^{th} cluster utilization; l_i^* is the total neurons activated during learning, and l_i is the total available neurons in cluster c_i. A uniformly distributed input, which produces higher accuracy by maximizing the available inter-cluster links of a specific SD-SCN design, causes high values of ψ in all clusters.

Figure 3.2 compares the ψ values of the real network trace [33] made up of 80k 112-bit class-tagged flow IDs used in the simulations presented in this work, and an equivalent uniform input, which is generated by obtaining 80k unique random samples from range $[0,\ 2^{112} - 1]$. The graph shows 14 clusters since, following the k-limit, a 112-bit input can be broken down into 14 sub-messages, each with a width of $k = 8$ bits. A $\psi = 1$ means that all available neurons in cluster c_i have been activated *at least once* during learning; this is the ideal scenario because a higher number of unique active neurons translate to a higher number of unique links in memory, which maximizes accuracy as described in Section 2.2. Notice that the actual network trace has some very poor ψ values, specifically for clusters c_0 , c_1 , c_2 , c_3 , c_{12} and c_{out}. On the other hand, the uniform input has $\psi = 1$ all throughout. This work proposes the use of clusters utilization, ψ, as a measure of SD-SCN-configuration-aware input distribution uniformity. While there are several statistical approaches in quantifying general input distribution [35][36], ψ is more SD-SCN-specific, which correlates directly to the number of active neurons (links) acquired during learning, and its effect to accuracy. It will be shown in the Results section that an input set and SD-SCN configuration combination that results to high ψ values achieves high accuracy.

3.3 Bit Activity

Following the assumption that an input set with constantly switching sub-messages, and hence constantly switching bits, produces high values of ψ, this work hypothesizes that, if we only consider flow ID bit indexes that have an almost equal frequency of switching, *i.e.*, eliminate those that barely switch, we are generating an input set that mimics a uniform input's high values of ψ, which results to high classification accuracy. This work proposes a metric called *bit activity factor*, α, which measures the frequency of switching of each flow

ID bit in a given network trace and can be computed using:

$$\alpha(n) = 1 - \left| \frac{ones_n - \frac{M}{2}}{\frac{M}{2}} \right| \tag{3.2}$$

where $ones_n$ is the number of times out of the total number of flows, M, that bit n is a 1. Essentially, Equation 3.2 measures how close bit n is to maximum switching, which is 50% of the total number of flows in the network trace. An $\alpha(n)$ close to 1.0 means that bit n is 1 (or 0) half of the time during learning. On the other hand, $\alpha(n)$ close to 0.0 means that bit n is either 1 (or 0) most of the time.

Figure 3.3 shows the bit activity factors of 80k 112-bit class-tagged flows captured on a campus network [33]. It can be seen that the bits from the IP source have a lower α than the bits from fields such as IP destination, and the source and destination ports. The reason for this is that the analyzed network trace is a collection of outgoing flows from a local area network to the Internet, and therefore will naturally have a narrower list of (local) IP addresses compared to the destination, which includes IP addresses for web, messaging, mail servers, etc. It can also be seen that the activity factors for the port bits are higher because a single device normally uses multiple ports for different applications. The protocol field only represents two different values: UDP (represented by bit string "0001000″), and TCP ("00000110″) with non-overlapping "1s″, which explains the minimum switching. The eight class bits represent the eight classes, and the bit with the highest *alpha* represents the class with the highest occurrence in the trace, which in this case is Web, represented by class bit index $n = 7$.

Measuring α is the first attempt at developing a metric that can be used as a basis for SD-SCN input bit selection, to improve accuracy, and at the same time, minimize memory requirement by reducing K. In order to overcome the non-uniformity in the input to SD-SCN, previous works have employed a combination of techniques that include adding random clusters and/or bits [28], and using compression codes such Huffman coding [29]. The former increases the input width, K, further, which incurs the penalty of added memory. For a system that already takes 112 bits of baseline input, this approach can be limiting (discussed further in Chapter 5). The latter converts input to variable length messages, and uses random bits to *fill* in freed space in order to maximize input differentiability; the obvious disadvantage of which is the added cost of coding and decoding. Since the goal of

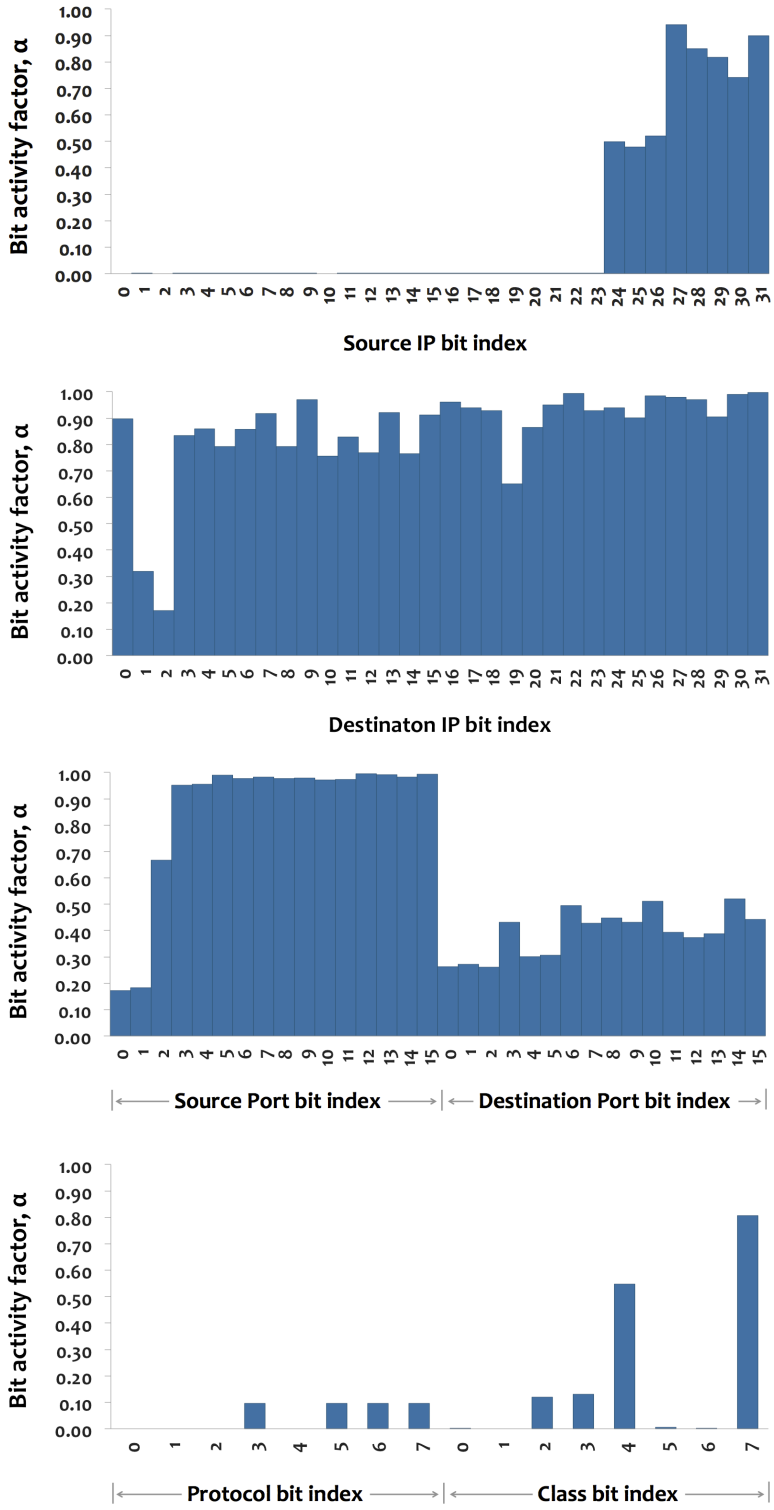

Fig. 3.3 Bit activity factor, $\alpha(n)$ for $n = 0, ..., K-1$, of a real network trace of 80k $K = 112$-bit flows.

this work is to develop an IPC with minimum delay and memory, selecting fewer useful bits on the basis of α, instead of introducing additional data *and* processing is a more logical approach.

3.4 Bit Activity Thresholding

To filter out bit locations n with low α, this work proposes using *bit activity threshold*, Th, as the parameter that determines a final input message of width K for SD-SCN. For instance, a $Th = 0.42$ means that we want to generate an input message that is made up of flow ID bits with $\alpha \geq 0.42$. Lowering Th increases the width of the input message, K. Conversely, increasing Th narrows K, which, as shown in Chapter 2, lowers total memory requirement, μ_{total}, assuming constant c. An input with narrower K produced by increasing Th encompasses higher switching bits. This work hypothesizes that increasing Th results in an input set that is made up of shorter messages of highly switching bits, which produce high clusters utilization, ψ_i for $i = 0, ..., c-1$, similar to those of a true uniform input set, which produces higher accuracy compared to the alternative. Higher ψ values, along with a design that uses high l, translates to higher numbers of activated unique neurons per cluster during learning. The higher number of unique active neurons translates to a higher number of unique links in memory, which maximizes SD-SCN accuracy, as discussed in Section 2.2.

It is worth noting that there are ranges of Th where the number of complying bit indexes, and therefore the resulting K, do not change. This work proposes sweeping Th from 0 to 1 in decimal increments, d, to generate unique values of K. The choice of d depends on the desired sensitivity to the differences in α. Using the same real network trace and setting $d = 0.01$, a total of 21 unique Th-defined K values are generated; the first change in K is at $Th = 0.17$, where the message width is truncated from the baseline 112 bits to 80 bits. The dropped bit indexes include: IP source bit indexes below $n = 24$, and all the protocol bits. It is also worth mentioning that the output bits will be excluded from the Th check. That is, the class bits, regardless of their activity factors, will always be included in the final SD-SCN input message because they represent the field that we are interested in recovering later.

Another benefit of using Th to generate the final input width, K, is that it allows for more design options, which are not possible using the baseline class-tagged flow ID bits

alone. These additional designs refer to the different c-based configurations for one value of K. Specifically, for every $K(Th)$, there exists at least one cluster-size-dependent SD-SCN configuration as long as $(K - kout)$ **mod** $c_{in} = 0$. That is, a cluster-size-dependent configuration exists as long as the input message bits, excluding the eight-bit wide class bits (k_{out}), can be divided equally into c_{in} input clusters. Again, $k_{out} = 8$ is factored out from the c_{in} calculation because, as mentioned in Section 2.5, the output will always have its own cluster. As an example, $K(0.17) = 80$ bits can be implemented as a seven-cluster SD-SCN with $c_{in} = 6$ and $c_{out} = 1$. Each input cluster will have $k_{in} = 12$ bits: $k_{in} \times c_{in} + k_{out} \times c_{out} = 12b \times 6 + 8b \times 1 = 80$ bits. Alternatively, it can be implemented as a nine-cluster ($k_{in} = 9b$), 10-cluster ($k_{in} = 8b$), or 13-cluster ($k_{in} = 6b$) design. In contrast, the baseline class-tagged flow ID with $K(0) = 112$ bits, can only be implemented using $c = 14$ ($c_{in} = 13$, $c_{out} = 1$) and $k_{in} = 8b : K = 8b \times 13 + 8b \times 1 = 112$ bits. An alternative solution to the lack of design flexibility in the baseline input width, $K = 112$ bits, is to the pad the input with zeroes or random bits (introduced briefly in the previous section). This approach is the opposite of bit activity thresholding since instead of reducing the number of bits, it introduces new bits into the input, and hence new clusters into the design. Padding with zeroes allows K to grow and c_{in} and k_{in} to take new values. Using the same real network trace, the highest accuracy that can be achieved with this approach is 88.19%. This is produced by padding four zeroes, $p_{zero} = 4$, to the baseline input, producing a new input width, K', equal to $K + p_{zero} = 112 + 4 = 116$ bits, which can be implemented as a 10-cluster SD-SCN that uses $k_{in} = 12$ bits, and consumes 1.2 Gbits of memory. Zero-padding, however, does not change the contents of the input, which means that for all the new c-and-k combinations produced by increasing p_{zero}, the original clusters retain their old ψ values, and the new cluster(s) that receive the zero-padding will always only activate neuron zero out of all the available neurons, thereby resulting to a poor new cluster(s) utilization, ψ. With no real improvement in the ψ values, only l, and hence k, controls accuracy. Chapter 5 details the effect of zero-padding to accuracy and cluster utilization.

Padding with random bits addresses the lack of cluster utilization, ψ, control in the zero-padding approach. However, in order to overcome the effect of the poor ψ values of the original input set (Figure 3.3), the amount of random-bit padding, p_{rand}, must be significantly high. Using the same new input width as that of the zero-padding approach, $K' = 116$, but replacing the zeroes with four randomly generated padding bits, $p_{rand} = 4$, simulation confirms that there is virtually no improvement in accuracy. Setting $p_{rand} = 16$

increases accuracy from 88.19% to 90.0%, $p_{rand} = 28$ to 91.2%, and $p_{rand} = 40$ to 92.1%. The obvious major disadvantage of random bit-padding is the high penalty in memory for a very minimal improvement in accuracy. In fact, sweeping p_{rand} from four to 64 bits, and evaluating the performance of resulting SD-SCN designs that use $l_{in_{max}} = 2^{k_{in_{max}}} = 2^{12}$ shows an average improvement in accuracy of only 1% per 371 Mbits of memory. Chapter 5 provides more details on the effect of random-bit-padding to accuracy and the corresponding cost in resulting SD-SCN designs.

In order to avoid incurring additional memory in attempting to improve accuracy, the bit activity tresholding metric, Th, is developed in order to widen the design space by reducing K, instead of introducing new bits into the input. Th increases input message differentiability by removing the bits that are common, *i.e.*, least switching, from the entire input set. Section 2.2 details that given K, accuracy depends on input distribution, and the number of neurons per clusters, l. The goal of this work, therefore, is to identify the SD-SCN designs with the best classification accuracy and speed and lowest memory requirement by setting two parameters:

1. Th, to maximize ψ_i for $i = 0, 1, ...c_{in} - 1$, and reduce K. The former maximizes accuracy, while the latter establishes a trade-off between μ_{block} and access delay

2. l_{in} given $K(Th)$, which establishes a trade-off between accuracy and μ_{block}, and hence access delay

3.5 Optimizing Accuracy with XOR

Since bit activity thresholding excludes the output (class) bits – we can only retrieve information that we have learned – the final input set will always have a degree of correlation that is defined by the low output cluster utilization, ψ_{out}. The output cluster always uses $k_{out} = 8$ bits (and neurons) to represent the eight application classes generating the 80k flows in the real network trace. Therefore the maximum output cluster utilization is always only $\psi_{out} = 8/l_{out} = 8/2^{k_{out}} = 8/2^8 = 8/256(3.125\%)$. In order to increase ψ_{out}, this work proposes activating dummy neurons in the output cluster during learning, instead of constantly reusing the predefined eight neurons that represent the eight classes. Specifically, an XOR operation is performed on the real k_{out}- bit output sub-message (the class) and

the corresponding k_{in}-bit sub-message of the input cluster with the highest utilization, $\max_{0 \leq i \leq c_{in}-1}(\psi_i)$. There are two reasons why this approach is taken to increase ψ_{pit}:

1. The XOR function is chosen because it outputs a "1″ 50% of the input cases, which is ideal in order to maximize the variability of the resulting dummy neurons; using AND will have a 75% bias towards "0″, and OR will have a 75% bias towards "1″.

2. The k_{in}-bit sub-message from the c_{in} with the highest utilization as the second input to the XOR operation because this reduces the possibility of producing the same set of dummy neurons over and over, which is precisely the problem that this optimization approach is trying to solve in the first place.

In order to recover the real class bit during retrieval, the XOR operation can be reversed simply by performing XOR on the dummy output neuron, identified by the GD process, and the specified sub-message from the highest utilization cluster. This makes sense because XOR follows this property: A XOR B = C, A XOR C = B. The max-utilization cluster can be identified by measuring the ψ values of a given network trace and SD-SCN configuration. In hardware, the output cluster utilization optimization can be achieved by attaching XOR gates to the output terminal of the output cluster, placed adjacent to the pre-identified max-utilization cluster. The obvious cost of performing XOR optimization is that the size of the output cluster must now be at least equal to the size of the input clusters. That is, the condition $l_{out} \leq l_{in}$ must be satisfied so that the output cluster always contains the neuron representing the result of the XOR operation. This cost however is only incurred for designs that start with an output cluster that is smaller than the input clusters, *i.e.*, the number of bits (and therefore neurons) in the class is lower than the number of bits in the input cluster ($k_{out} \leq k_{in}$).

Chapter 4

Results

4.1 Accuracy

The fundamental assumption in this work is that changing K by controlling Th facilitates in finding an input set, and hence an SD-SCN design, that maximizes clusters utilization, ψ_i for $i = 0, ...c_{in} - 1$, which in turn maximize classification accuracy. As mentioned in Section 2.2, two factors control accuracy: (a) input distribution, which this work measures using ψ values (Figure 3.2), and (b) the number of neurons per input cluster, l_{in}, and hence the number of bits per input-cluster sub-message, k_{in}, (Equation 2.1). Therefore, we expect designs that use $k_{in_{max}}$, i.e., $k_{in} = 12$, and measure high ψ values for a given input set to achieve maximum accuracy. As discussed in Section 3.4, bit activity thresholding generates a new input set, which is made up of bit indexes, $n = 0, ..., K - 1$, from the original input with $\alpha(n) \geq Th$; the k_{out} bits in the output sub-message is excluded from thresholding since they represent the class, which SD-SCN needs to learn for later retrieval. In order to prove the hypothesis that bit activity thresholding maximizes ψ values, and hence accuracy, the performance of the designs produced by increasing Th is simulated using three different inputs sets:

1. input generated using this work's proposed method, i.e., choosing $K(Th)$ bits that satisfy $\alpha \geq Th$
2. input generated using $K(Th)$ randomly chosen bits
3. input generated using the opposite of this work's method, i.e., choosing the $K(Th)$

bits with the lowest α.

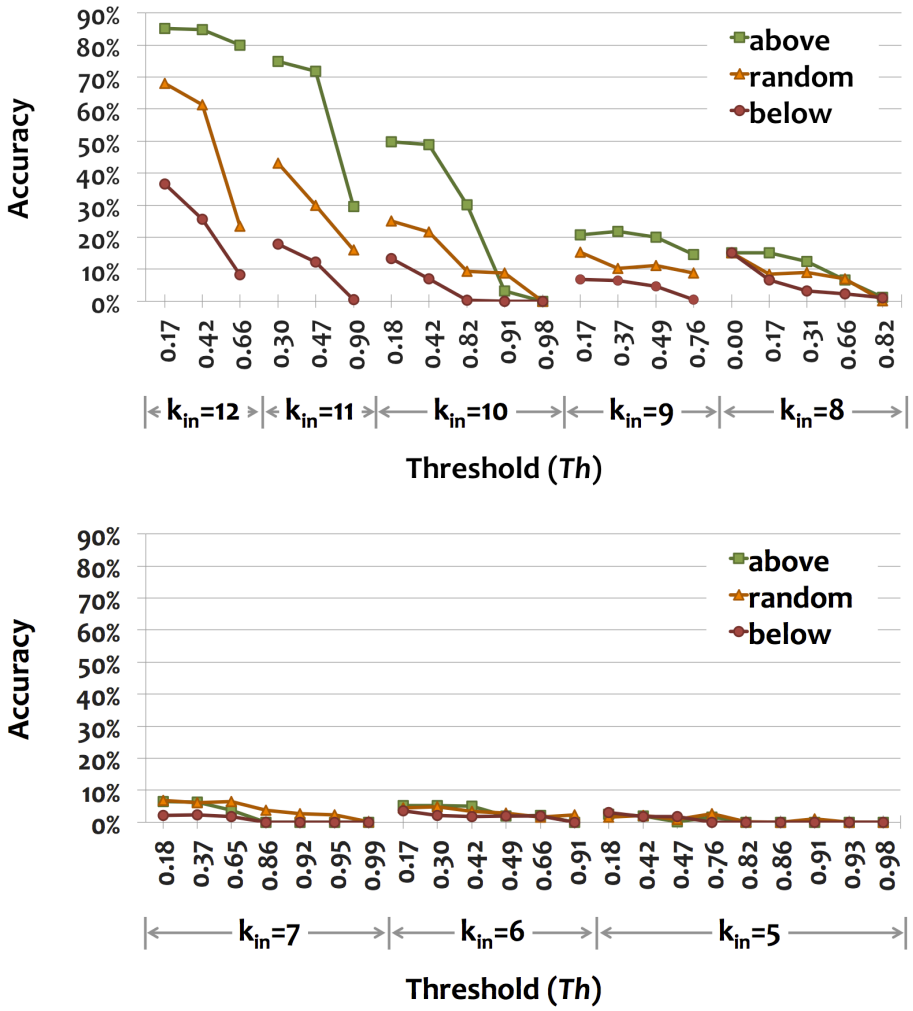

Fig. 4.1 Accuracy of $K(Th)$-controlled designs using actual (real network trace) input sets produced by choosing bits (1) randomly, (2) above, and (3) below Th.

The results are shown in Figure 4.1, which presents data in decreasing k_{in} and increasing Th. Each point in the x-axis refers to a single design generated using thresholding. There are two significant observations that can be made based on the results: (a) first, the theoretical effect of k_{in}, and hence l_{in}, to accuracy is very apparent the higher the number neurons per cluster, l_{in}, and hence k_{in}, the higher the number of learned links, and the higher the accuracy (Section 2.2). Accuracy drops toward zero as k_{in} approaches $k_{in_{min}}$,

i.e., $k_{in} = 5$. (b) Second, in virtually all combinations of Th and k, the input generated using the proposed method, i.e., choosing bits with $\alpha \geq Th$, always produces the highest accuracy, with two exceptions: (b.1) the input set used when $Th = 0.0$ ($k_{in} = 8$), i.e., zero-percent thresholding, is identical for all three input conditions, which explains why accuracy is also identical. (b.2) The only other significant point where bit activity thresholding approach did not produce maximum accuracy is at $Th = 0.91$ ($k_{in} = 10$). Randomly generated input produced 9% accuracy, whereas the proposed approach only produced 3%. This point, however, refers to a design that takes an input of width $K(0.91) = 36$ bits, which means a total of $K(0.0)K(0.0.91) = 11236 = 76$ bits have been dropped, and only 36 bits are used to differentiate 80k flows. While increasing Th removes the bits common to most of the flows in the input set, setting it to a really high value also removes the bits that differentiate them. This is evident in each k_{in} section in Figure 4.1, where almost always the first and lowest Th value produces the maximum accuracy; further increasing Th only causes accuracy to drop since doing so eliminates the highly-switching bits which maximizes flow differentiability. The data presented in Figure 4.1 proves that controlling Th and l together fulfils the goal of finding an SD-SCN design with maximum accuracy. Through simulation, this designed is identified to take an input of $K(0.17) = 80$ bits, with $k_{in} = 12$ bits, and $c_{in} = 6$.

This work proposed in Section 3.2 that clusters utilization, ψ_i for $i = 0, 1, ..., c-1$, can be used as a measure of SD-SCN-aware input uniformity to predict accuracy; that is, accuracy can be maximized by controlling Th in order to produce an input set that mimics the high ψ values of a true uniform input. To prove this, the clusters utilization of the input sets and designs from the previous experiment are measured; and in fact, the most accurate input set and design combination produces the highest clusters utilization. Figure 4.2 shows the comparison of the clusters utilization of the most accurate thresholding-generated design from Figure 4.1, using the three previously described input sets. Note that the ψ values from the input generated using our proposed method, i.e. *above* Th, are the ones closest to the all-ones ψ of a true uniform input. The input generated using *random* bits is second, and the input generated using bits *below* Th, which has the lowest set of ψ values, is the last. This trend is true for the rest of the designs; the higher the ψ values, the higher the accuracy. The results prove that clusters utilization, ψ_i for $i = 0, ..., c-1$, are a sufficient measure of SD-SCN-aware input distribution uniformity, which consequently predicts SD-SCN accuracy.

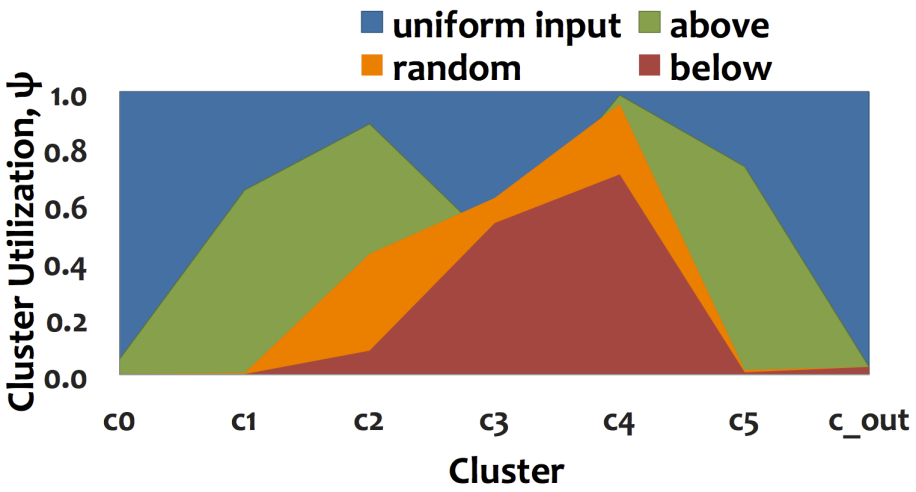

Fig. 4.2 Clusters utilization comparison of uniform and real (actual network trace) input sets produced by choosing $K(0.17) = 80$ bits: (1) randomly, (2) above, and (3) below $Th = 0.17$.

The accuracy figures presented in this section are based on simulations performed the real network trace of 80k unidirectional flows. Unidirectional flows make a sufficient input dataset because fundamentally, the goal of IPC is to detect the presence of a class of application in the network. As stated in Section 2.5, a flow is always associated with a class during learning, and so a successful classification alone results in the detection of an application that is running in the network. Moreover, since communication over the Internet is almost always two-way, the presence of an outgoing flow guarantees a returning flow, both of which are associated with the same class. While the direction of the flows affects input distribution – as evident by the wide variations in the values of α presented in Figure 3.3 – and therefore SD-SCN accuracy, their mere inclusion in the dataset is sufficient to guarantee that the condition (i.e., frequency distribution of the application classes) of the network is accurately represented in the IPC analysis.

4.2 Memory

Since Th controls K, and therefore the implementable number of neurons per input cluster, l_{in}, and the input clusters, c_{in}, it can be said that total memory requirement, μ_{total}, is also a function of Th. For this reason, and the fact that the SD-SCN IPC implementation

proposed in this work only uses $c_{out} = 1$, Equation 4.1 can be rewritten:

$$\begin{aligned}\mu_{total}(c_{in}, K(Th)) = c_{in} \times (c_{in} - 1) \times \left(\frac{K(th) - k_{out}}{c_{in}}\right)^2 \\ +2 \times c_{in} \times \left(\frac{K(th) - k_{out}}{c_{in}} \times 2^{k_{out}}\right)\end{aligned} \tag{4.1}$$

where $k_{out} = 8$, as before. Equation 4.1, and the effect of Th to accuracy as shown in the previous section, indicates that it is possible to use Th to find an appropriate accuracy-memory trade-off for the SD-SCN implementation of IPC. From Figure 4.1, the top three designs with accuracy $\geq 80\%$ use $k_{in_{max}}$ (or $l_{in} = 2^{12} = 4,096$), and have the following configurations:

1. $K(0.17) = 80$, with 85.1% accuracy at 73.4 MB;
2. $K(0.42) = 68$, with 84.9% accuracy at 50.4 MB;
3. $K(0.66) = 56$, with 80.0% accuracy at 31.5 MB.

This means that it is possible to save as much as 23 MB in memory by moving from design (1) to (2), while losing only 0.2% of accuracy. The pareto-optimal solutions based on total memory requirement and accuracy will be discussed in the following section.

4.3 Optimization

The low maximum possible output cluster utilization, $\psi_{out} = 8/256$, which is caused by the class bits being having a dedicated (output) cluster, limits maximum possible accuracy that can be achieved by adjusting parameters Th and l_{in}. In order to improve ψ_{out}, this work proposes replacing the limited output cluster neurons with dummy neurons produced by an XOR operation on the original class bits and the sub-message from the cluster with the highest ψ (Section 3.5). Figure 4.3 compares the pareto-optimal plots in total memory requirement versus error rate of the baseline solution, which uses Th and l adjustments only, and the optimized XOR-reinforced solution. Memory requirement for both solutions are calculated using Equation 4.1, and error rate (1− **accuracy**) is measured through simulation using the real network trace described previously. The most obvious difference between the two types of results is that for the baseline solution, more memory, i.e., a

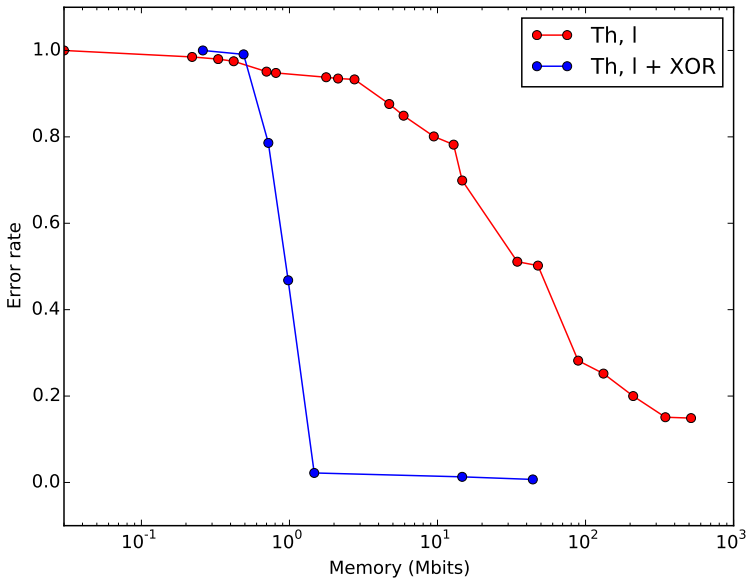

Fig. 4.3 Pareto-optimal plots in error rate and memory of designs produced by (1) Th-and-l_{in} adjustments only, and (2) with the added XOR-optimization (dummy neurons).

bigger network design (higher l_{in}), is required to get a smaller error rate (higher accuracy); specifically, the smallest error rate, 0.149 (or 85.1% accuracy), is achieved with a design that uses 516 Mbits of memory, with $K(0.17) = 80$ bits, $c_{in} = 6$, $l_{in} = l_{in_{max}} = 4,096$, and $l_{out} = 2^8 = 256$. The XOR-optimized solution, on the other hand, is able to attain the smallest error rate, 0.007 (99.3% accuracy), for a design that only uses 44 Mbits of memory, with $K(0.42) = 68$ bits, $c_{in} = 6$, and $l_{out} = l_{in} = 1,024$ (since $k_{in} = 10b > k_{out} = 8b$). These results are expected, since, in order to overcome the effect of the low ψ_{out} and maximize accuracy, the baseline solution must use more links, and therefore neurons, in the input clusters, which requires more memory; moreover, as shown in Figure 4.3, for each k_{in}, the lowest Th, and hence highest K, produces the highest accuracy (lowest error rate), which translates to higher μ_{total} (Equation 4.1). In contrast, the XOR-optimized solution is able to avoid a low ψ_{out} by activating dummy neurons in c_{out}, which means that in order to produce a design with low error rate, k_{in} doesn't have to be set at maximum value to maximize the number of learned links from the input clusters, and make up for a low output cluster utilization. The baseline solution has $\psi_{out} = 0.0273$, whereas the XOR-optimized

solution has $\psi_{out} = 1.0$.

4.4 Classification Delay

This work defines *classification delay*, δ, as the amount of time the SD-SCN takes to generate a match to a given input. Since the core of SD-SCN's retrieval process relies on accessing previously stored links in the RAM block through the LSM (Section 2.1.2), this work models classification delay as a function of the access delay of the RAM blocks. Equation 2.8 models access delay as a function of β, the maximum number of ambiguities in the erased cluster during the first iteration, which consequently represents the number of serial accesses to a RAM block, measured through simulation. Accordingly, this work's model for classification delay is:

$$\delta(\theta, \beta) = \theta \times \beta \tag{4.2}$$

where θ is the access delay to a RAM block, which is a function of RAM block size calculated using Equation 2.3. For configurations where the input and output clusters don't have the same size, i.e. $l_{in} \neq l_{out}$, the size of bigger ram block is used, i.e., $\max\left[\mu_{block}(l_{in}, l_{in}), \mu_{block}(l_{in}, l_{out})\right]$, to obtain θ, which is estimated using [34] using the following assumptions: μ_{block} uses 32-bit addressing, single bank, single read/out port, and 45 nm CMOS technology. Equations 2.3 and 2.4 indicate that that the size of a RAM block depends on the *size* of the cluster, which is defined by its number of neurons, l. Since l is inversely proportional to the number of clusters, c, (Equation 2.1), Equation 4.2, therefore, suggests that SD-SCN designs with fewer input clusters, i.e., lower c_{in}, will have a higher classification delay by virtue of the use of more l_{in}, which translates to bigger RAM blocks that have higher access delays.

The number of ambiguities after the first iteration, β, which is captured during simulations, can be viewed as a measure of the network's ability to easily narrow down the number of candidate neurons in the erased cluster, given the links obtained during learning. In this sense, it can be said that β is also a function of input distribution, and l. The more uniform the input data (high ψ values), and the more neurons per cluster the network uses (high l_{in} and l_{out}), the more links there are in memory that can be used for eliminating ambiguities. Since this work has proven that (a) controlling Th and using maximum l_{in} (by using maximum k_{in}) and l_{out} (by activating dummy neurons in c_{out}), maximizes accuracy –

Table 4.1 Memory requirement and classification delay of pareto optimial XOR-assisted designs.

Th - k_{in}	Accuracy	Memory μ_{total} (Mbits)	β	Classification delay, δ (ns)
0.42 - 10	99.3%	44.0	2	4.102
0.37 - 9	98.7%	14.7	3	4.608
0.49 - 6	97.8%	1.5	3	3.960
0.86 - 7	53.2%	0.98	5	6.6
Clustering [13]	$\sim$ 95.0%	–		–
I-SVM [14]	$\sim$ 97.0%	–		–
I-RF [15]	$\sim$ 97.0%	–		–
I-EM [16]	$\sim$ 96.0%	–		–
SVM-FPGA [17]	$\sim$ 100.0%	$\sim$10.0		3178

both parameters maximize the number of stored links – it is expected that β will be lower for a design with high accuracy; similarly, δ is expected to be higher for a design that uses high k_{in} and/or k_{out}, *and* have a high β. Table 4.1 shows the performance and cost of the top most accurate XOR-optimized designs from Figure 4.3. compared with the state-of-the-art-solutions. As predicted, β is lower for designs with high accuracy. Increasing the differentiability of the flows through thresholding, and maximizing clusters utilization ψ by using high l_{in} and/or l_{out}, increases accuracy by *minimizing* β or the number of flows that "look alike". Following Equation 4.2, the results also show that δ is higher for bigger designs (higher l_{in}, and hence k_{in}) that have a higher β. In comparison, using the same real network trace, the SVM-FPGA [17] solution consumes 3178 ns to classify a flow. The high classification delay is due to the use of 8000 support vectors to achieve $\sim$ 100% accuracy; at data rates $\geq$ 320 Kpackets/second, accuracy drops from 100% to 0%. The rest of the existing ML-based solutions only measured accuracy.

Chapter 5

Pitfalls

Using bit-activity thresholding, together with XOR optimization, facilitates in finding an SD-SCN configuration that is capable of achieving 99.3% classification accuracy with a classification delay of 4.10 nanoseconds per flow, and 44.040 Mbits of memory requirement (Chapter 4). Before arriving at these remarkable results, this work considered a few other approaches, which are based on hypotheses that did not test as exceptionally. This chapter details these approaches and explains why they are unable to measure as notably in one (or more) of the IPC metrics, which are accuracy, total memory requirement, and classification delay.

5.1 Padding the Input

5.1.1 Zero-padding

One of the most seemingly obvious work around solutions to the hard limit in the number of different SD-SCN configurations that can support the baseline input with width $K = 112$ bits is to increase K via zero-padding. As mentioned in Chapter 2, there is a finite set of SD-SCN configurations that can support an input of width K; this set is defined by the integer-valued combinations of c-and-k. Following the $k = [5, 12]$ constraint introduced in Chapter 3, $K = 112$ bits (108-bit wide flow + 8-bit wide class) can only be supported by an SD-SCN with configuration: $c = 14$, where $c_{in} = 13$ and $c_{out} = 1$, and $k_{in} = 8$. As before, the number of bits in the single dedicated output cluster is constant, i.e., $k_{out} = 8$ (eight classes). Using Equation 2.6, the values of parameters c_{in}, k_{in}, c_{out}, and k_{out} check

out: $K = c_{in} \times k_{in} + c_{out} \times k_{out} = 13 \times 8b + 1 \times 8b = 112$ bits. Since the values of c_{in} and k_{in} depend on K, padding zeroes to the *MSB* end of the input will widen the design space by allowing K to grow in order to evaluate new configurations other than the one defined by $c_{in} = 13$ and $k_{in} = 8$, *without* modifying the contents of the input.

Table 5.1 Performance and cost of SD-SCN designs processing zero-padded input.

new K ($K^{'}$)	total padding (p_{zero})	input bits per cluster (k_{in})	input clusters (c_{in})	accuracy ($1 - error\ rate$) %	total memory Mbits	classification delay (δ) ns
112	0	8	13	15.07	11.9	6.60
113	1	7	15	7.79	4.4	5.85
116	4	12	9	88.19	1,226.8	8.20
118	6	11	10	67.87	388.0	8.97
120	8	8	14	15.07	13.8	6.60
122	10	6	19	6.27	2.0	6.57

Table 5.1 shows the accuracy, total memory requirement, mu_{total}, and classification delay, δ, of the different SD-SCN configurations produced by incrementing the zero-padding parameter, p_{zero}, from 0 to 10. For this round of simulations, the same set of real network trace [33] is used as that in the preceding simulations presented in the previous chapters. The new input message width, $K^{'}$, follows the form: $K^{'} = p_{zero} + K$, where $K = 112$ bits, the baseline input width. Zero-padding is done on the MSB end of the class-tagged flow ID bits in order to maintain the original contents of the input. Note that some values of p_{zero}, specifically $p_{zero} = 2,\ 3,\ 5,\ 7,\ 9$, are not present in the table because they produce values of $K^{'} - k_{out}$ that cannot be evenly distributed among their respective number of input clusters, c_{in}. Additionally, only the highest possible value of k_{in} for each $K^{'}$ is shown in the table. As discussed in Section 2.2, for a given input set of width K, a configuration that uses maximum k produces maximum l, which, when all the other parameters are equal (i.e., number learned messages, M, and erased clusters, c_e), produces maximum accuracy. Table 5.1, therefore, aims to show the *maximum* accuracy, and the corresponding penalty in memory and classification speed, that can be achieved through zero-padding, which this work hypothesizes as a way of bypassing the c_{in}-and-k_{in} limit to the baseline input width, $K = 112$ bits,

It can be seen that maximum accuracy of 88.2% is achieved using $p_{zero} = 4$ for a

configuration with $k_{in} = 12$ and $c_{in} = 9$, with over 1.2 Gbits of memory usage. Through simulation, the classification speed of this configuration is estimated to be 8.20 nanoseconds per flow (Equation 4.2). In contrast to the the optimized solution presented in Chapter 4, zero-padding results to 12% more classification error, uses 27× more memory, and takes twice the time in generating a result. It makes sense that the configuration with the highest k_{in} value has the highest accuracy since, as discussed in detail in the previous chapters, accuracy is controlled by input distribution and l, where $l = 2^k$ (Equation 2.1). Zero-padding increases K in order to widen the design space and find configurations that use a high l_{in} value, which are expected to achieve higher accuracy. This hypothesis proves to be accurate as simulation results presented in Table 5.1 show that accuracy goes up as the value of k_{in} increases.

While zero-padding manages to increase accuracy by increasing l_{in}, it lacks control over the production of an input set that closely mimics a uniform input distribution, i.e., measured ψ values do not approach the ideal value, which is 1.0. This is due to the simple fact that zero-padding does not change the original content of the input data. That is, given a configuration defined by $K^{'}$, c_{in}, and k_{in}, sub-message '00001', for instance, is still just '1'– both of which point to the same neuron in the newly created cluster that results from the increase in the width of K due to zero-padding. If p_{zero} is high enough such that the number of zero-padding bits fits in a single cluster, i.e., the first sub-message becomes a string of zeroes, learning will always only activate neuron zero in cluster c_0 for all flows. This means that zero-padding does *not* introduce any real new associations (links) during learning, nor does it improve cluster utilization. It's sole benefit is that it allows the network to be configured to use maximum l_{in}. Any correlation that exists in the input prior to the introduction of zero-padding still exists.

Figure 5.1 shows the measured clusters utilization of the zero-padded actual input using $p_{zero} = 8$ ($K^{'} = 112 + 8 = 120$), and those of the baseline input ($K = 112$), and a uniform input, generated in the manner similar to that in Section 3.2. Note that the (baseline) actual input does not have a ψ value for cluster $c_0^{'}$; the design that supports the baseline input only has 14 clusters, and $c_0^{'}$ represents the newly created cluster due to zero-padding. The ψ values of the original clusters of the baseline input completely overlap those of the zero-padded input, and the ψ value for the newly added cluster is very minimal, i.e., $\psi_{c_0^{'}} = 1/2^{k_{in}} = 1/256 = 0.004$. In fact, continuously adding zeroes to the baseline input in $p_{zero} = 8$ increments will not improve the ψ values of the original 14 clusters, but will

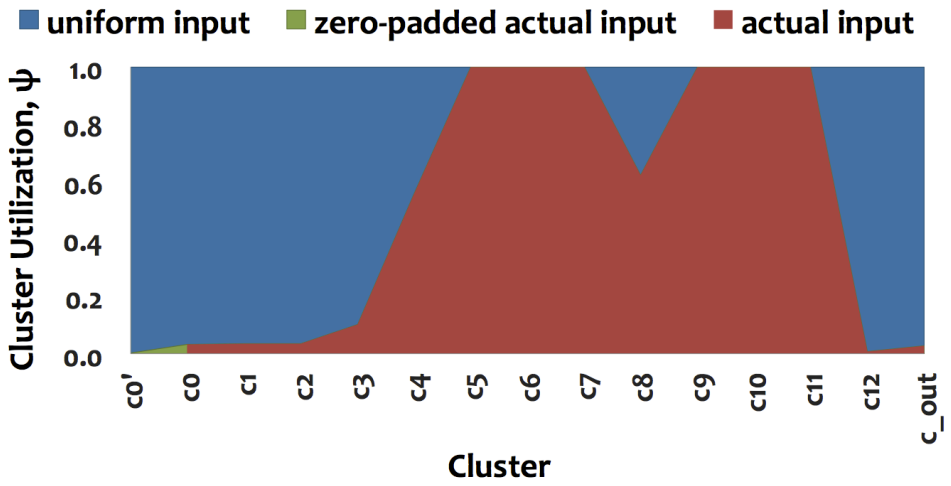

Fig. 5.1 Clusters utilization comparison of zero-padded ($p_{zero} = 8$), uniform, and baseline real ($K = 112$) inputs.

only create new clusters with utilization equal to $\psi_{c_0'} = 0.004$. This is true for any value of p_{zero}. That is, any and all new clusters created in $K' = p_{zero} + K(0)$ will have a cluster utilization of $\psi_{c_0'} = 0.004$, and the original clusters from the baseline input will retain their old ψ values.

Since zero-padding can only control accuracy via l_{in}, and since the maximum value of l_{in} is predetermined by by the $k_{in} = [5, 12]$ limit, maximizing accuracy by incrementing p_{zero} is expected to hit a ceiling. Since the generated inputs using zero-padding do not show any clusters utilization improvements, configurations with the same l_{in} (and therefore k_{in}), which are produced by continued increment of p_{zero}, are bound to have exactly the same accuracy. Notice in Table 5.1 that configurations produced by $p_{zero} = 0$ and $p_{zero} = 8$ both use $k_{in} = 8$, and have the same accuracy (15.07%). This means that adding 8 bits of zero-padding to the baseline input in order to arrive at a network configuration that uses the same k_{in} but more clusters – specifically, one cluster more than the baseline – results in an increase of memory requirement, from 11.9 to 13.8 Mbits, with no improvement in the original clusters utilization, ψ, and therefore results to no improvement in accuracy whatsoever. Continuously incrementing p_{zero}, therefore, after hitting the desired value of k_{in} and l_{in} is moot. To verify this hypothesis, we can keep incrementing p_{zero} and simulating the performance of configurations that use the same k_{in}. Since the goal is to maximize accuracy, the simulations can use $k_{in} = 12$, the maximum value for k_{in} that

produces maximum l_{in}. The results are shown in Figure 5.2. As p_{zero} is increased, memory requirement increases to accommodate the added bits. However, since new input sets failed to mimic the high ψ values of the uniform input, and l_{in} is constant – the two parameters that control accuracy – accuracy is stuck at 88.2%.

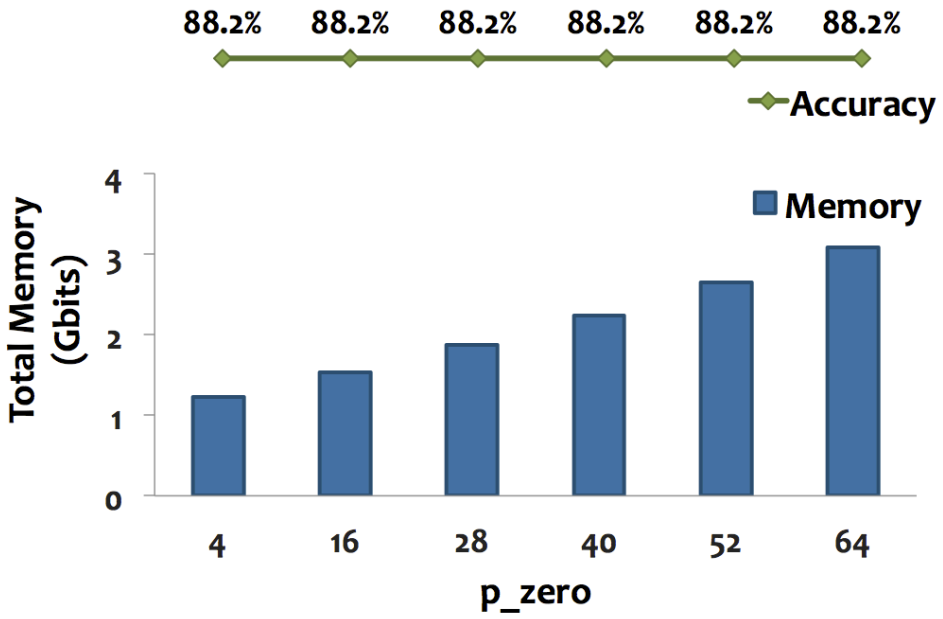

Fig. 5.2 Accuracy and total memory requirement of SD-SCN designs processing *zero*-padded input clustered into sub-messages of width $k_{in} = 12$.

5.1.2 Random bit-padding

In order to address the lack of control in producing an input set that mimics the high ψ values of a uniform input in the zero-padding approach, an alternative would be to use randomly generated bits for the MSB padding to the baseline class-tagged flow ID bits. Using the same configurations presented in the previous subsection, Table 5.2 shows the accuracy, memory requirement, and classification speed of SD-SCN when the random bit padding parameter, p_{rand}, is incremented, in place of p_{zero}. Notice that the accuracy numbers in Table 5.1 barely changed from those in the zero-padding approach. In fact, the very minimal improvement in accuracy starts to appear at higher values of p_{rand}. Specifically, only the last two configurations, $p_{rand} = 8$ and $p_{rand} = 10$, have an accuracy improvement of 0.10% and 0.01%, respectively. In theory, random bit-padding introduces ones and zeroes into the

Table 5.2 Performance and cost of SD-SCN designs processing *random-bit*-padded input.

new K (K')	total padding (p_{rand})	input bits per cluster (k_{in})	input clusters (c_{in})	accuracy ($1 - error\ rate$) %	total memory Mbits	classification delay (δ) ns
112	0	8	13	15.07	11.9	6.60
113	1	7	15	7.79	4.4	5.85
116	4	12	9	88.19	1,226.8	8.20
118	6	11	10	67.87	388.0	8.97
120	8	8	14	15.17	13.8	6.60
122	10	6	19	6.28	2.0	6.57

input, which helps activate new neurons and improve cluster utilization, ψ. However, in order to *maximize* its effect, i.e., change the input distribution of the original 80k 112-bit input set, p_{rand} must be set to a significantly high value. Based on the results of simulation presented in Table 5.1, introducing $p_{rand} = 10$ bits to an input set made up of 80k $112-$bit correlated messages has little effect in improving accuracy, or the number of ambiguities during the first iteration, β, which is why classification delay also did not improve. Of course, total memory figures are similar for both zero-padding and random-bit-padding approaches since the same set of network configurations are used in both simulations.

Random bit-padding can be viewed as introducing *redundant clusters* into the network. That is, new data is deliberately introduced into the network – which forces the creation of new clusters – and associated with each original input message in order to help diversify and increase the number of links associated with each learned clique. This is true for the zero-padding approach as well, except the new clusters that are created are empty, which explains why increasing p_{zero} further while keeping k_{in} constant only consumes more memory as more clusters are crated, but does not improve accuracy since there's no improvement in clusters utilization (Figure 5.1). Since random-bit padding creates non-empty clusters and introduces new links and neurons during learning, an improvement in accuracy is expected as p_{rand} is increased continuously at the expense of more memory.

Figure 5.3 shows the accuracy and memory requirement of SD-SCN configurations that use $k_{in} = 12$ and produced by continuously increasing p_{rand}. Note that unlike Figure 5.3, accuracy *does* improve as the amount of padding increases. This is because the random bits that are padded into the input increase and/or change the number of activated neurons

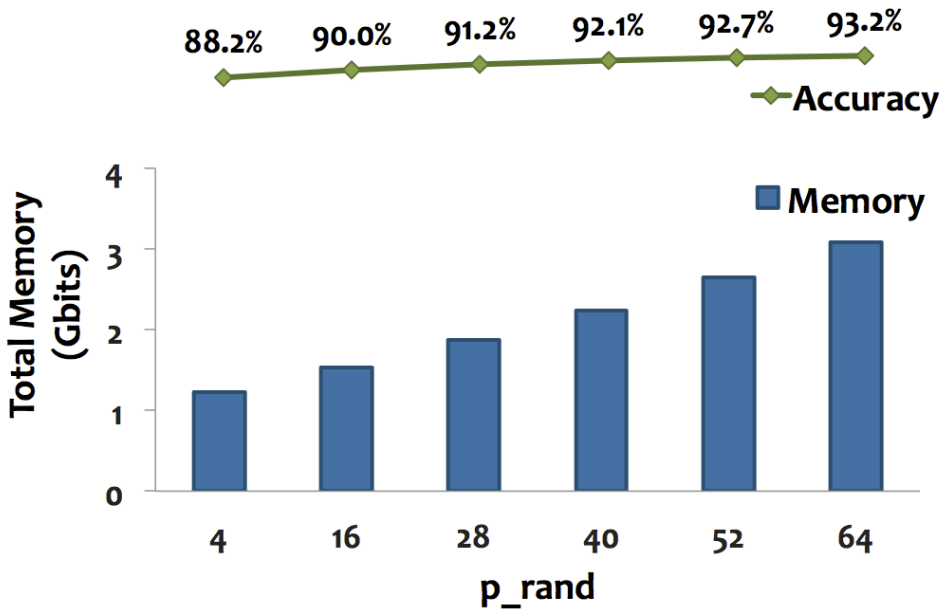

Fig. 5.3 Accuracy and total memory requirement of SD-SCN designs processing random-bit-padded input clustered into sub-messages of width $k_{in} = 12$.

for each learned message during learning. It is also worth noting that adjacent SD-SCN configurations using the same k_{in}, $K^{'}(k_{in})_{m-1}$ and $K^{'}(k_{in})_{m}$, are $(p_{rand_{m-1}})^2$ apart. That is, the amount of random bit-padding needed to find the nearest configuration with the same amount of input bits per cluster, k_{in}, is equal to the square of the current p_{rand}. As such, the obvious cost of random bit-padding is increased memory consumption. The main problem however, is that the improvement in accuracy is too low compared to the penalty in memory. In fact, the average improvement in accuracy for configurations shown in Figure 5.3 is only 1% per 371 Mbits of memory. This accuracy improvement is too low, especially when compared against the optimized solution presented in Chapters 2 and 3 where a virtually perfect accuracy is achieved using only 44 Mbits of memory.

Aside from the increased memory requirement and poor accuracy improvement, another disadvantage of using random-bit padding is the added system complexity and cost associated with the implementation of an algorithm that generates and *remembers* the random bits assigned for each input message. Assigning the same randomly generated bits to all the input messages has the same affect as adding zeroes; that is, the correlation between messages remain and the added bits do not help distinguish the messages apart. In order

to produce a new input set with differentiable entries, an efficient algorithm for generating random bits and associating them to the original input must be developed. In order to retrieve the output, the same set random bits must be recalled during search. Assigning the wrong random bits to search input during retrieval increases the risk of an error, which pulls down accuracy. Since the goal of this work is to develop an IPC that does not introduce a bottleneck in packet routing, using an approach that introduces additional coding/decoding functions with associated delay is counter-productive.

5.1.3 Limitation of input-padding

The biggest disadvantage of padding the input in order to widen the design space is that it ignores the basic principle of *input selection* for maximum accuracy. As discussed in the previous chapters, even machine learning-based approaches have to hand-pick the features that will help develop a model that maximizes the separability of the classes in order to maximize accuracy [10][12]. Taking the full 112 bits as the input requires a large SD-SCN network, which translates to higher memory requirement. Moreover, doing so also accepts the existing correlation between input messages, which causes lower classification accuracy and speed. By not taking advantage of the fact that not all bits in the input are going to be useful for classification, resulting SD-SCN designs in an input-padding approach (zero or random bits) are almost always bigger, slower, and more error-prone, especially when dealing with real (correlated) data.

5.2 Input Component Permutation

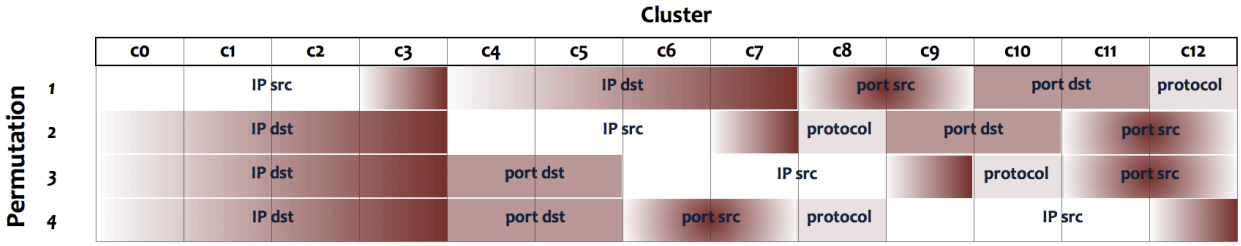

Fig. 5.4 Sample permutation of baseline actual input ($K = 112$) over $c_{in} =$ 13 input clusters.

The main goal of the design space exploration performed in this work is to find an

SD-SCN configuration that performs IPC with maximum accuracy at minimum cost. As mentioned in the previous chapters, the sizes of the RAM blocks in the LSM, together with their corresponding access delays, put a constraint on the size of the SD-SCN that can be used for real-time IPC. The ideal configuration consumes a small enough total memory to be implementable on an FPGA, thus the k-limit (Section 3.1). Additionally, accuracy, as discussed previously, is maximized when the input distribution is normal, i.e., the messages have little to no correlation, and the network uses a high number of neurons per cluster, l. Controlling the value of l is straight forward; given an input set of K-bit messages, targeting maximum accuracy can begin simply with setting a maximum possible value for l by setting the smallest possible value for c (Equation 2.6). Changing the distribution of the input, on the other hand, is more challenging. Chapter 3 proposes a method of selecting only the bits in the input with high activity factor, α, in order to maximize clusters utilization, ψ for $i = 0, 1, ..., c_{in} - 1$. The idea is that if each bit in the input switches a lot, a new neuron will be activated each time a new message is learned. This forces the input to mimic a uniformly distributed input set, which activates *all* available neurons at near-equal frequencies throughout learning instead of only a select few, and causes high ψ values in all the input clusters. Correlated input activates a fewer number of unique neurons during learning, which makes distinguishing between learned messages more difficult during retrieval. Before the XOR-assisted bit-activity thresholding approach presented in Chapter 3 was developed, this work attempted to control cluster utilization, ψ, and therefore input distribution, through the permutation of the order of the components of the input message.

The input to SD-SCN for IPC is the class-tagged flow ID. As discussed in the previous chapters, this input is 112 bits long and is made up of *six* components: source ($32b$) and destination ($32b$) IP addresses, source ($16b$) and destination ($16b$) ports, protocol ($8b$), and the class ($8b$). In the real network trace [33] used as the input set for the simulations in this work, some of the components change values more than others. For instance, since the input set is made up of outgoing flows, there are fewer source IP addresses than there are destination; a single application, e.g. Mail, Web, etc., can take on multiple IP addresses. Similarly, a single application can generally use multiple ports, and so there even much fewer source IP addresses compared to source and destination port numbers. These characteristics of the input set are described in Chapter 3 and illustrated by the activity factors presented in Figure 3.3. The input component permutation approach hypothesizes

that, given a network configuration, there is a specific order of the input components that results to a new input set that mimics a uniform input's high ψ values and maximizes accuracy. Reordering the input components redistributes the component bits among the available clusters, and the order that maximizes clusters utilization produces the highest accuracy.

Since the class bits are assigned to the dedicated output cluster, the different ways of rearranging the order of the $n = p = 5$ remaining input components is ${}_nP_k = \frac{n!}{(n-k)!} = 120$. The resulting clusters utilization, ψ_i for $i = 0, 1, ..., c_{in} - 1$, for each way of ordering the input components can be calculated using Equation 3.1. Since the baseline input of width $K = 112$ bits can only be supported by a network using $c_{in} = 13$ and $k_{in} = 8$ (Section 3.4), the best ordering of the input components therefore is the one that produces clusters utilization, ψ values, that are closest to the maximum possible value, similar to those of a uniform input (Figure 3.2), over the $c_{in} = 13$ input clusters. The goal of this approach, then, is to find out if it's possible to attain the high clusters utilization values of a uniform input, by moving the components of real (correlated) input around in order to maximize accuracy.

Figure 5.4 shows four possible arrangements of the baseline input components across the the $c_{in} = 13$ input clusters. The input components are shown as gradient blocks where the *white* areas represent poor bit activity levels, and the *red* areas represent the opposite. For each permutation (row), the clusters that coincide with the red areas have ψ values close to 1.0 (high activity), whereas the ones with white areas have ψ close to 0.0 (low activity). The hypothesis of this approach is that there is a single permutation of the input components that will maximize clusters utilization. However, as illustrated by the sample permutations in Figure 5.4, maximizing clusters utilization in this manner is theoretically impossible. Moving the input components with wider red areas to fill low-activity clusters only does the opposite to the high-activity clusters. That is, the void left by moving all high-activity input components into new cluster indexes can only filled by the remaining low-activity ones, which means improving the ψ values of some clusters reduces those of the rest. Since no bits and/or components are added or removed, the activity levels, and hence the clusters utilization, remain constant, which means that this approach is not capable of producing a new input set that mimics the high ψ values of a uniformly distributed input (Figure 4.2), which has been proven to produce high classification accuracy [28][29].

5.3 Flow ID Component Selection

In order to address the fact that ψ values are not improved by simply rearranging the input components around the available input clusters, a work-around solution can be to include only the high-activity input components into the final input set. Based on Figure 5.4, this approach entails including only the input components with wider red areas, thereby dropping some less active components such as the IP source and reducing the final input width, K. This solution is the basis of the bit activity thresholding proposed in Chapter 3.4. In fact, this is a coarse-grained version of the α-based bit selection approach presented in this work; only the input components with the most highly switching bits, i.e., components with wider red areas in Figure 5.4, are included into the input.

Examining Figure 5.4, however, shows that the only input component that is significantly less "active" than the rest is the IP source. This is also confirmed by α levels presented in Figure 3.3 The rest of the components have varying and/or almost equal levels of activity, which doesn't really put a clear criteria as to what should and should not be included into the final input. Moreover, generating an input set based on a coarse-grain measure of activity level doesn't optimize the selection process. That is, low-flipping bits, which does nothing to improve ψ values and therefore accuracy, are going to be carried over to the final input by selecting the whole flow ID component e.g., the low-flipping bits in the IP source or port source. In order to truly maximize ψ values, a finer granularity of input selection must be put in place. For this reason, this work has developed the bit activity thresholding approach presented in Chapters 3 and 4, which when coupled with the XOR-optimization solution that increases output cluster utilization by activating dummy neurons, achieves maximum accuracy at minimum cost.

Chapter 6

Related Literature

There are several ML-based approaches to IPC in literature. The state-of-the-art IPC solutions presented in [13], [14], [15] and [16] build upon existing ML algorithms in order to achieve high classification accuracy ($> 90\%$). The work in [13] introduced a partially supervised K-means clustering-based solution to IPC, coupled with a data preprocessing that involves packet attribute grouping and selection. This work, like most ML-based solutions, has established that using packet sizes as attributes produces the highest classification accuracy. During testing, the solution uses multiple parallel models, each with its own set of clusters that represent a predesignated set of n probabilities associating an input flow to n classes based on certain packet attributes. The selection of packet attributes vary per model. One model, and hence its corresponding clusters, may be assigned to process average and standard deviation of packet sizes, while another may process flow size and packet inter-arrival times. The resulting cluster from each model then becomes the basis for the selection of the specific ML algorithm to use for classification. Finally, the outputs from all the models (and ML algorithms) are then statistically consolidated to produce a single classification result. While this work is able to achieve a relatively high accuracy (low 90s %), it uses various complex ML models and clustering strategies to perform classification on just a single flow, which is not very efficient. The effect of clustering and increasing the number of models to classification time was not reported. Similarly, the work also failed to present the amount of memory necessary to build deploy the design to hardware.

The work in [14] used an improved SVM, where feature weights are used during learning in order to differentiate attributes importance and maximize classification accuracy.

In contrast, the work in [14] simply drops attributes that have little effect to accuracy. Fundamentally, SVM performs classification by determining a hyper-plane with the widest margins between classes of the training data. The feature weights, which scales the actual attributes used as features in training and testing the SVM model, are generated using kernel polarization. The kernel polarization function is maximized in order to maximize the separability of training flows. The main problem with this approach, however, lies in the fact that it uses an SVM variant that only performs *binary* classification. This means that in order to decide, for instance, which of the 10 applications a flow belongs to, a total of $\frac{10\times(10-1)}{2} = 45$ models *and* classifications have to be made. Each classification also its own set of calculated feature weights. This makes the entire classification process huge and inefficient. Similar to the work in [13], this work also only measured accuracy, and not the costs associated with creating and/or testing the model. The reported accuracy for this approach is $\sim 97\%$.

The work in [15] also uses weights to assign features with variable importance in performing classification, but replaces SVM with Random Forest algorithm. Random Forest classification involves the use of multiple unpruned decision trees, where each three is generated using randomly selected samples from the training data set. The branches of one tree ultimately ends to a leaf which represents one class. The final classification output of the model is the highly occurring class (mode) of the combined outputs from the individual trees. Changing the weights assigned to packet attributes changes the tree structure. This solution reported an accuracy of ~ 97 and a *training* time of ~ 1025 seconds. Like the rest of the recent ML-based solutions however, the costs associated with testing and real-time hardware implementation, *e.g.*, memory requirement, actual classification delay (testing), are not measured.

The work in [11] [17] targets an FPGA implementation of an SVM-based IPC that takes multiple packet sizes per flow as features (input). Since no information from the flow ID is included in the features, this solution has to dedicate a hardware called a flow builder, which tracks and stores flow IDs, accumulates five packet sizes per flow into an off-chip memory, records the sizes of the last three packets of a flow, and triggers the SVM classifier. In fact, the need for a flow builder applies to most, if not all, ML-based solutions, including the works in [13], [14], [15] and [16]. ML-based solutions are able to achieve high accuracy by using packet sizes solely, or in combination with other packet attributes, as features, which warrants the use of a flow builder. This dependence on flow packet sizes increases system

complexity and cost. This SVM-FPGA solution is able to achieve a very high accuracy ($\sim$ 100%) by using 8000 support vectors. This high number of support vectors imposes a penalty in classification speed; if the data rate exceeds $\geq$ 320 Kpackets/second, accuracy drops from 100% to 0%.

Unlike ML solutions, designing and training SD-SCN for IPC does not produce a *classifier* that "guesses" the class of a *new* input data. Instead, the process produces a memory system that decodes a match to a query that comes in the form of an unlabelled flow ID, with the assumption that its *complete* (labelled) version has been learned in the past. When an SD-SCN is unable to classify a partial input, it means that either: a) multiple possible matches have been decoded, or b) no match has been decoded at all. The occurrence of a) is minimized through the techniques detailed in Chapter 3. Specifically, accuracy can be maximized through bit activity thresholding, which maximizes the differentiability of stored flows by maximizing input clusters utilization, and XOR-optimization, which activates dummy neurons in the output cluster, which maximizes flow differentiability further by maximizing the output cluster utilization. Since SD-SCN-based IPC can only classify flows that have been learned in the past, it is important, therefore, to design and train a new SD-SCN IPC if the original training data, *i.e.*, input set used to design the existing SD-SCN IPC, is believed to be already obsolete. An input set can be considered obsolete if it no longer represents the unique flows running through the network. This can be true during instances where an existing class of application has started using new ranges of port numbers, an existing class is split into (new) separate classes, or new computers (with new IP addresses) are added into the network. In short, a new SD-SCN design must be generated if there are changes to the network that can affect flow IDs. This is true for ML-based solutions as well;since changes in the flow ID affects packet attributes new models are generated when changes are introduced into the network. Using an updated training set, a new SD-SCN IPC design can be generated using the approach presented in this work in order to maintain system reliability and accuracy. The main advantage of the SD-SCN-based IPC over the previously discussed ML-based solutions is that the design and training processes are more straightforward; the solution presented in this work is able to achieve high accuracy without using a flow builder, which minimizes system cost. The SD-SCN implementation of IPC is able to achieve 99.3% accuracy using only 44 Mbits of memory. Since SD-SCN is highly parallel, this solution is also able to classify a flow within 4.1 nanoseconds, which is almost 800$\times$ faster than the SVM-FPGA solution.

Chapter 7

Conclusion

This work has presented a new method of IPC using associative memory (AM) based on sparse-clustered network (SCN) with selective decoding (SD). This classification approach filters out low-switching bits from a network trace, and maximizes output cluster utilization by activating dummy neurons in order maximize accuracy and minimize total memory requirement, and classification delay. By taking flow ID bits as input, the solution removes the need for a flow builder. It classifies a flow within 4.1 nanoseconds, and achieves 99.3% accuracy while using only 44.0 Mbits of memory. This solution is able to achieve about the same accuracy as the FPGA-implemented state-of-the-art, but performs classification almost 800× faster, thereby providing a novel IPC solution suitable for high-speed networks.

References

[1] W. Zhao, D. Olshefski, and H. Schulzrinne, "Internet quality of service: an overview," *Columbia University Computer Science Technical Reports*, 2000.

[2] C. Hoene, S. Wiethölter, and A. Wolisz, *Quality of Service in the Emerging Networking Panorama: Fifth International Workshop on Quality of Future Internet Services, QofIS 2004 and First Workshop on Quality of Service Routing WQoSR 2004 and Fourth International Workshop on Internet Charging and QoS Technology, ICQT 2004, Barcelona, Catalonia, Spain, September 29 - October 1, 2004. Proceedings*, ch. Predicting the Perceptual Service Quality Using a Trace of VoIP Packets, pp. 21–30. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[3] Cisco, "Quality of service for voice over ip," *Cisco Technology White Paper*.

[4] P. A. Networks, "What is a service level agreement?," *Palo Alto Networks Learning Center*.

[5] J. Ding, J. Jin, P. Bouvry, Y. Hu, and H. Guan, "Behavior-based proactive detection of unknown malicious codes," *International Conference on International Monitoring and Protection (ICIMP '09)*, pp. 72–77, May 2009.

[6] T. Micro, "Worst viruses in history: A look back at malware through the ages," *Trend Micro Industry News*, July 2014.

[7] R. Simon, "'ransomware' a growing threat to small businesses," *The Wall Street Journal*, April 2015.

[8] C. Rottondi and G. Verticale, "Using packet interarrival times for internet traffic classification," in *Communications (LATINCOM), 2011 IEEE Latin-American Conference on*, pp. 1–6, Oct 2011.

[9] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," in *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '98, (New York, NY, USA), pp. 203–214, ACM, 1998.

[10] T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *Communications Surveys Tutorials, IEEE*, vol. 10, pp. 56–76, Fourth 2008.

[11] T. Groleat, M. Arzel, and S. Vaton, "Hardware acceleration of svm-based traffic classification on fpga," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International*, pp. 443–449, Aug 2012.

[12] K. Singh and S. Agrawal, "Comparative analysis of five machine learning algorithms for ip traffic classification," in *Emerging Trends in Networks and Computer Communications (ETNCC), 2011 International Conference on*, pp. 33–38, April 2011.

[13] A. Kumar *et al.*, "Incorporating multiple cluster models for network traffic classification," in *2015 IEEE 40th Conference on LCN*, pp. 185–188, Oct 2015.

[14] S. Hao *et al.*, "Improved SVM method for internet traffic classification based on feature weight learning," in *2015 International Conference on ICCAIS*, pp. 102–106, Oct 2015.

[15] C. Wang, T. Xu, and X. Qin, "Network traffic classification with improved random forest," in *2015 11th International Conference on CIS*, pp. 78–81, Dec 2015.

[16] S. Liu *et al.*, "Improved EM method for internet traffic classification," in *2016 8th International Conference on KST*, pp. 13–17, Feb 2015.

[17] T. Grolat, M. Arzel, and S. Vaton, "Stretching the edges of svm traffic classification with fpga acceleration," *IEEE Transactions on Network and Service Management*, vol. 11, pp. 278–291, Sept 2014.

[18] D. Rossi and S. Valenti, "Fine-grained traffic classification with netflow data," in *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, IWCMC '10, (New York, NY, USA), pp. 479–483, ACM, 2010.

[19] NetFPGA, "Netfpga: a line-rate, flexible, and open platform for research, and classroom experimentation,"

[20] IBM, "Bringing big data to the enterprise," *IBM InfoSphere Platform.*

[21] S. young Yu, N. Brownlee, and A. Mahanti, "Comparative performance analysis of high-speed transfer protocols for big data," in *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pp. 292–295, Oct 2013.

[22] H. Jarollahi, V. Gripon, N. Onizawa, and W. Gross, "Algorithm and architecture for a low-power content-addressable memory based on sparse clustered networks," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 23, pp. 642–653, April 2015.

[23] H. Jarollahi, N. Onizawa, V. Gripon, and W. Gross, "Architecture and implementation of an associative memory using sparse clustered networks," in *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, pp. 2901–2904, May 2012.

[24] H. Jarollahi, N. Onizawa, and W. Gross, "Selective decoding in associative memories based on sparse-clustered networks," in *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, pp. 1270–1273, Dec 2013.

[25] V. Gripon and C. Berrou, "Sparse neural networks with large learning diversity," *Neural Networks, IEEE Transactions on*, vol. 22, pp. 1087–1096, July 2011.

[26] M. F. Rutledge-Taylor, A. Vellino, and R. L. West, "A holographic associative memory recommender system," in *Digital Information Management, 2008. ICDIM 2008. Third International Conference on*, pp. 87–92, Nov 2008.

[27] V. O. Baez-Monroy and S. O'Keefe, "An associative memory for association rule mining," in *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, pp. 2227–2232, Aug 2007.

[28] B. Boguslawski, V. Gripon, F. Seguin, and F. Heitzmann, "Huffman coding for storing non-uniformly distributed messages in networks of neural cliques," in *AAAI Conference on Artificial Intelligence*, 2014.

[29] R. Danilo, P. Coussy, L. Conde-Canencia, V. Gripon, and W. J. Gross, "Restricted clustered neural network for storing real data," in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, GLSVLSI '15, (New York, NY, USA), pp. 205–210, ACM, 2015.

[30] H. Jarollahi, N. Onizawa, V. Gripon, and W. J. Gross, "Algorithm and architecture of fully-parallel associative memories based on sparse clustered networks," *J. Signal Process. Syst.*, vol. 76, pp. 235–247, Sept. 2014.

[31] N. Onizawa and W. Gross, "Low-power area-efficient large-scale ip lookup engine based on binary-weighted clustered networks," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pp. 1–6, May 2013.

[32] C. R. Meiners, A. X. Liu, and E. Torng, *Hardware based packet classification for high speed internet routers.* New York : Springer, 2010.

[33] F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, F. Risso, and k. c. claffy, "Gt: Picking up the truth from the ground for internet traffic," *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 12–18, Oct. 2009.

[34] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques," in *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '11, pp. 694–701, IEEE Press, 2011.

[35] T. Batu, L. Fortnow, R. Rubinfeld, W. D. Smith, and P. White, "Testing that distributions are close," in *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pp. 259–269, 2000.

[36] A. Glazer, M. Lindenbaum, and S. Markovitch, "Learning high-density regions for a generalized kolmogorov-smirnov test in high-dimensional data," in *Advances in Neural Information Processing Systems 25* (P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), pp. 737–745, 2012.