Automated Network Application Classification: A Competitive Learning Approach

R. G. Goss
Department of Computer Science
University of Cape Town
Cape Town, South Africa
ryan@goss.co.za

G. S. Nitschke
Department of Computer Science
University of Cape Town
Cape Town, South Africa
gnitschke@cs.uct.ac.za

***Abstract*—The design of a sustainable application level classification system has, over the past few years, been the subject of much research by academics and industry alike. The methodologies proposed rely predominantly on predefined signatures for each protocol, applied to each passing flow in order to classify them. These signatures are often static, resulting in inaccuracies during the classification process. This problem is compounded by delays in signature update releases. This paper presents an approach toward automated signature generation, mitigating classification problems experienced with existing systems. A hierarchical system is proposed, where signatures are developed and deployed in real-time. The ideas set forth in this research are evaluated by experimentation in a live network environment. Discriminators of both encrypted and plain-text application protocol samples were recorded and automatically annotated by a *Hierarchical Self-Organizing Map* (HSOM). The clusters identified by the HSOM were used in a supervised training process that correctly identified protocols with an almost perfect (99% percent) success rate.**

***Index Terms*—Application Protocols, Network Flow Classification, Deep Packet Inspection, Self Organizing Maps**

I. Introduction

Networks have grown substantially in recent years, due in part to the increase in global reach of the Internet, network access speeds and available content. This growth spurred the introduction of unique application protocols, built on both client-server and peer-to-peer communication technologies. Although the former are easier to identify and manage, the latter present a great challenge to network administrators. Peer-to-peer application protocols operate in a decentralized manner, dynamically selecting the ports and protocols on which they communicate [1]. This dynamic nature complicates their classification and management, mitigating chances of accurate identification by standard protocol and port rules. Instead, vendors provide mechanisms for the identification of these protocols using *Deep Packet Inspection* (DPI) and statistical analysis. DPI has become an essential tool for network engineers, enabling them to search both packet header and payload (content) for predefined application protocol signatures [2]. These signature matches are often performed using regular expressions in software, or through the use of specialized hardware, such as a *Field Programmable Gate Array* (FPGA) [2]. DPI may yield excellent results in identifying plain-text flows, however encryption renders the content of packets opaque and thus the use of DPI inept.

Statistical analysis addresses the problem of opaque, encrypted flows, by inferring the application (see Gebski et al. [3]) or application class (see Auld *et al.* [1], Li *et al.* [4], Moore and Papagiannaki [5]) of network traffic flows through the examination of their flow information. The attributes considered by such methods are well researched in [1], [4], [6], [7]. The advantage of statistical analysis is that regardless of how applications attempt to disguise themselves, through encryption or port randomization, the statistical characteristics exhibited over their packet exchanges remain intact.

To identify an application protocol, a unique signature is required, provided by the vendor of the traffic management device in one or more signature packs. These signature packs need to be updated regularly to cater to advances in application protocol development. Each vendor is responsible for the creation of their own proprietary signature packs, compatible with their systems. The discriminators extracted and the mechanism employed by vendors is often a closely guarded secret. Hence, accuracy and performance variances between vendor equipment, such as the *Cisco Service Control Engine* (SCE) and the *Allot NetEnforcer*, can be substantial.

This process of creating and deploying signatures is sub-optimal, as the development of new application protocols far exceeds signature production by vendors. This results in a significant amount of network traffic remaining unclassified or inaccurately classified until an update is released.

A communication session established between two hosts on a network is known as a flow, described by a quintuple consisting of a source and destination address, source and destination ports and protocol [8], [9]. In order to successfully manage network flows, decisions need to be made as early as possible concerning their underlying application protocol. A significant amount of research has been conducted in academia and industry alike in the field of application protocol recognition [1], [3], [4], [5], [6], [7]. The majority of these studies concentrate on identifying discriminators and methods for use in distinguishing such protocols from one another. A vector of discriminators describing a flow is referred to by this research as a flow sample.

Whilst most other research proposes a method for manual signature creation, this paper introduces a methodology for

978-1-4673-5903-0/13/$31.00

automated signature creation using a select set of flow discriminators, unsupervised and supervised learning techniques. A *Hierarchical Self Organizing Map* (HSOM) is used for automatically grouping similar flow samples into annotated data sets. These data sets are then used to train classifiers in the identification of the underlying application protocol.

The use of a HSOM to accomplish automated data set clustering is not the only clustering method available. Other implementations include the use of a *Self Organizing Map* (SOM) for the first level of abstraction, followed by a second level using k-means to cluster the SOM data [10]. Vesanto and Alhoniemi [10] recommend the clustering of the SOM, rather than the initial data set, using the k-means algorithm. K-means is an example of crisp clustering, which relies on k clusters being known *a priori*. As the number of distinct application protocols within a recorded data set is not known *a priori*, the k-means clustering algorithm is not applicable.

A. Related Work

Statistical analysis and DPI are considered as suitable methodologies for distinguishing the underlying application protocol of a flow early in it's existence. For example, Bernaille *et al.* [11] describes a statistical method of grouping *Transport Control Protocol* (TCP) flows which exhibit similar behavior, using k-means, gaussian mixture model and spectral on hidden Markov model techniques. According to Bernaille *et al.* [11], the size information obtained from the first few packet exchanges of a flow serve as a good metric for identifying the underlying application protocol. Gargiulo [12] asserts that significant accuracy can be achieved in identifying a flow's application protocol by examining the direction of the first four packets along with the payload sizes of each. Moore and Papagiannaki [5] argue that in some cases, the amount of information read from the first payload-bearing packet is enough to identify the protocol, whilst in others up to 1 Kbyte of payload needs to be examined before a decision is made. Various statistics including the minimum, average and maximum packet lengths derived from a flow are described by Alshammari and Zincir-Heywood [13] as being capable of identifying the underlying application protocol of a flow. Finally, Goss and Botha [14] implement the ideas set forth by [5], [11], [12] and [13], building a single classifier incorporating both statistical information and deep packet inspection.

Goss and Botha [14] relied on a manually annotated training set being supplied by an expert for each protocol. This training set was then used to train a feed forward *Artificial Neural Network* (ANN) to derive a classifier.

The discriminator sets constructed by Goss and Botha [14] include the direction of flow for each of the first four payload-bearing packets, with directionality identified by observing the direction of the initial synchronize (SYN) packet. The reliance on observation of the SYN packet limits identification to protocols using the TCP protocol. The experiments conducted within this paper therefore include only TCP based flow sets. In addition to the directionality of flow, packet sizes of the first four payload-bearing packets were recorded and added to the discriminator set. The American Standard Code for Information Interchange (ASCII) integer values representing the first few characters of the first payload-bearing packet in each direction of the flow were also included in the discriminator set. The combination of these elements, according to Goss and Botha [14], provides enough granularity to distinguish one application protocol from another. Whilst the results obtained by Goss and Botha [14] indicate a high degree of accuracy in discriminating between various application protocols, the dependency on manually annotated data sets inhibits fully automated classification.

This research extends that of Goss and Botha [14] with a novel method to automate the annotation process.

B. Research Problem

The Internet has, since its inception as a public access communications enabler, become the universal communications infrastructure in business [8]. As such, the accurate identification of application protocols is essential in order for network traffic to be managed [14]. The identification and categorization of network traffic flows allows network administrators to quickly diagnose problems, plan network capacity and identify misuse of provisioned resources [1], [12], [15], [16]. A considerable amount of research was identified in the area of signature creation, most requiring a significant amount of manual discriminator annotation for each protocol. This research tests a novel approach for automating such tasks, using supervised and unsupervised learning techniques to address the following hypotheses:

- Distinct protocols are identifiable via unsupervised competitive learning using a shared, static discriminator set.
- Accurate classifiers can be constructed to identify future instances of identified protocols [14].

By addressing these hypotheses, this research proposes to elucidate whether an automated classification system, suitable for identifying application protocol flows, is feasible.

C. Research Objective

The primary objective of this work is to produce a method for automating the manual annotation and protocol data set grouping process described in Goss and Botha [14]. These automatically constructed protocol data sets are then used to produce a classifier designed to identify flows exhibiting similar properties on a live network environment.

The remainder of this paper is divided into 3 sections. The first section introduces the method for testing the first hypothesis posed by this research. The second section describes an experiment which tests the chosen technologies for suitability for automated data set annotation. An experiment is also setup to test the newly derived classifiers against real-world network traffic. The third section discusses the results of the experiments, comparing them to the those obtained by Goss and Botha [14] for the same protocols.

II. METHODS

A. Data Sets for Experimentation

A flow inspector application was developed by the authors to extract discriminators from flows observed on a network. For the purposes of this research, the authors connected a flow inspector to a 1 Gbit/s full-duplex interface on an enterprise network, setting it to record flow information over a period of 24 hours on a normal week day. The information recorded for each TCP flow, according to Goss and Botha [14], caters to a number of classification problems experienced when describing network traffic. These problems include applications which exhibit multiple behavior patterns over the same protocol and protocols which closely resemble one another [15].

A total of 1973 TCP flow samples were recorded during the observation window, consisting of a number of application protocols, including Hyper-Text Transfer Protocol (HTTP), Secure Shell (SSH), Post Office Protocol 3 (POP3), Simple Mail Transfer Protocol (SMTP) and Bittorrent. These recordings form the static test data set used throughout this research.

B. Automating Protocol Data Set Identification

Automatically distinguishing between a set of seemingly unrelated, unstructured data is a task appropriate for unsupervised learning algorithms. Competitive learning is a type of unsupervised learning used in ANNs where neurons compete for the right to respond to particular subsets of input data [17]. A winning neuron, or node, is one whose weight vector most closely matches the input. The winner subsequently updates its weights to more closely resemble the input, drawing it closer. This process results in cluster formation, describing the input vectors parsed through the learning algorithm over numerous iterations. An example of a competitive learning algorithm, adapted from Rojas [17], is: Let $X = \{x_1, x_2, ..., x_l\}$, a set of normalized input vectors in *n*-dimensional space, to be classified in *k* clusters. The number of neurons in the ANN, and the number of clusters *k* are equal. Each neuron has a weight vector with a magnitude of *n*. The algorithm uses the following process:

- *Start:* Randomly initialize normalized weight vectors $w_1, ..., w_k$
- *Test:* Select a random input vector, x, such that $x \in X$. Compute $x \cdot w_i$, for $i = 1, ..., k$. Select w_m such that $w_m \cdot x \geq w_i \cdot x$ for $i = 1, ..., k$
- *Update:* Substitute w_m with $w_m + x$ and normalize. Repeat Test until exit condition met

The exit condition for the algorithm is usually for it to run for a predetermined number of iterations. This is due to the difficulty experienced in providing a definite measure of convergence for certain data distributions [17].

There are two distinct methods available for clustering data, hierarchical and partitive approaches. Hierarchical approaches can be further divided into agglomerative and divisive algorithms, with the former following a bottom-up approach and the latter a top-down approach [10]. Agglomerative approaches start by considering each sample as a singleton cluster, which merge (or agglomerate) pairs of these clusters until a single cluster is formed. For divisive algorithms, a single cluster is split into various clusters over a number of iterations.

The data sets used for experimentation in this paper consist of a number of flow samples which need to be clustered, forming distinct groups describing various application protocols. Such method requires a bottom-up approach, where each sample is treated as a singleton cluster at the outset, merging them (agglomerating) with closely related samples to form clusters. Bottom-up hierarchical clustering is therefore referred to as *Hierarchical Agglomerative Clustering* (HAC) [18, p. 378]. Conversely, a top-down approach would require a method for splitting a cluster into multi clusters over a number of iterations until the end samples are derived [18, p. 378]. This research uses independent samples which require clustering and therefore calls for a HAC approach to clustering.

A number of works describe the incorporation of a SOM in order to fulfil their HAC requirements, such as that of Abdi *et al.* [19]. This research therefore tests the SOM as a suitable mechanism for grouping the recorded flow samples into their respective application clusters.

C. Self Organizing Map (SOM)

The SOM is a competitive learning algorithm [20], comprising a number of neurons, positioned on a lattice, each with an associated weight vector W such that $|W| = |i|$, where i is an element of the input data set. The weights of each neuron are initialized by assigning a random value, ensuring diversity within the neurons of the SOM.

An initialized SOM is trained, or tuned, by supplying it a set of input vectors to parse. Each input vector is checked against all nodes of the SOM, in search of the closest or *Best Matching Unit* (BMU). Although there are a number of algorithms available to calculate such distances, the authors opted to use the Euclidean distance algorithm for such determination. During each iteration, the distance between each input and each of the nodes is measured, thereafter the closest node for each named the BMU. This winning node is then adjusted to more closely resemble the input vector by adjusting the value of its weights. Neighboring nodes are also moved closer to the BMU by adjusting their weights to more closely correspond to it. This process is used for nodes within a given radius, where nodes positioned closer to the BMU having a higher rate of influence imposed upon them than those further out.

The SOM competitive learning algorithm [21] steps are:

1) Randomly initialize the weight vectors for each node.
2) Repeat Steps 3 and 4 for x iterations.
3) Select each input from the data set randomly and identify the BMU.
4) Adjust the weight vectors of the BMU and nodes in the neighborhood.

The SOM uses unsupervised learning, and as such has no target vector, making it difficult to realize convergence. Instead, the stopping criterion is generally the maximum

number of tuning iterations the user elects during the training process. Supervised neural networks rely on a cost function, calculating the delta between the target vector and current result in order to measure convergence. As a cost function is not possible to implement with a SOM, this research measures convergence by the shift in weight measurements, plotted linearly with respect to time. The SOM is said to be near convergence once the "knee" of the graph has been observed. The weight update rule is as follows.

For each node within the neighborhood of the BMU:

$$w_j = w_j + (l * exp(-(d)/(2 * n^2)))$$

Where, $j \leq |W|$, l is the learning rate, d is the distance between the node and the BMU squared and n is neighborhood radius. After each epoch, the learning rate is decreased:

$$l = r * exp(-i/m)$$

Where, r is a predefined constant learning rate, i is the current iteration (or epoch) and m the max training iterations permitted. The learning rate is directly proportional to the significance of the weight adjustments. As such, the higher the learning rate, the more significant the adjustments imposed on each weight. The result of grouping neighbors produces clusters which describe the input vectors. The SOM is thus able to learn and group inputs by their features, without aid of the correct answer or a predefined number of clusters.

A cluster, or feature group, is defined by Jain [22] as high density regions in feature space, separated by low density regions. The benefit of the SOM is thus the identification of feature clusters, describing distinct application protocols, based solely on discriminators inferred from network traffic.

Using a single layer SOM, the input data set forms one complex shape which follows the data distributions in the space, such that regions of the map can be interpreted as prototype clusters [21]. A second-layer SOM can take the outputs of the first SOM, causing them to divide and split into distinct cluster representations [21]. This multi-layer configuration is known to as a Hierarchical SOM (HSOM). In a HSOM, the BMUs from the first layer SOM form the input data set for the second. As the distance relations of the data samples are preserved on the map, the node indexes of these BMUs can be used as a measure of distance of the original data samples, instead of the node's weight vectors [21]. The HSOM thus allows each high dimensional input data vector to be mapped to a low dimensional, discrete value (the index of it's BMU), such that the comparison of these values implicitly allows for the comparison of the original distances [21]. The BMUs resulting from the tuning of the second layer represent the actual clusters present in the data and subsequently the distinct application protocols present within the original data set (figure 1).

Members of the initial data set can subsequently be mapped directly to a particular cluster in abstraction level 2, providing an automatically annotated data set describing each cluster and therefore application protocol. These data sets can then be used to train a supervised neural network classifier [14].

The approach uses a SOM to produce a large set of prototype clusters, much larger than the expected number of actual clusters present within the original data set. Each entry in the original data set belongs to the same final cluster as its nearest prototype [10]. Vesanto and Alhoniemi [10] cite the reduction of computational cost as the main motivation for a hierarchical approach.

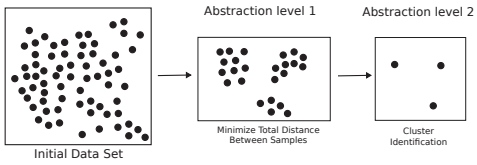

Fig. 1. Two level data abstraction approach to clustering

The advantage of the HSOM is the adaptive distance measure the SOM offers over classical clustering methods, such as k-means [21]. Furthermore, the Iso data type family of clustering, which includes k-means, can only make convex clusters due to the nearest-neighbor clustering rule [21]. The process of automatically annotating application flow sets within a mixed data set and classifier creation based on such annotations is tested in a set of experiments.

III. Experiments

Three experiments are used to test the HSOM approach of automating classifier generation. The first tests an automated mechanism for application protocol data set annotation. The second tests the creation of static classifiers, trained to identify each of the protocols discovered in experiment 1. The third experiment verifies the process where classifiers are tested using a data set derived from a real world network.

A. Automated Annotation of Application Protocol Data Sets

A 20 x 20 neuron SOM lattice was constructed as the initial abstraction layer, designed to parse the data set captured by the flow inspector application. The lattice size was selected in accordance with Kohonen [20], where the largest SOM deployed for practical applications was 1000 nodes. The maximum number of clusters present within the recorded data set was estimated to be less than 100, therefore a 20 x 20 SOM was deemed suitable.

Each node was initialized by assigning it a random value, in accordance with Kohonen [20]. The number of iterations was determined by tuning the SOM over numerous iterations and observing the location of the "knee" on the graph when plotting the average distance variance noted after each training iteration. This graph is shown in figure 2, indicating that the SOM is close to convergence after iteration 300.

A learning, or influence, rate of 0.0001 was used during the tuning process to adjust the neuron weights. This rate was determined by examining the average distance between a variety of samples within the initial data set. An influence rate of 0.0001 was decided upon to ensure gradual shifts in weights, rather than excessive adjustments after each iteration.

Once the tuning of the first level SOM had completed, the average distances between each node and it's immediate

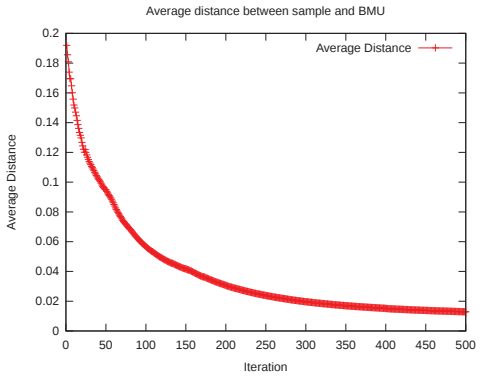

Fig. 2. Average Distance Between Sample and Associated BMU

neighbors was calculated and recorded. These values allow a graphical representation of the SOM to be displayed on a 2-dimensional image for easy visual evaluation as a U-Matrix. The U-Matrix representation of the first level SOM, highlighting the top scoring BMUs produced during the tuning process is displayed in figure 3.

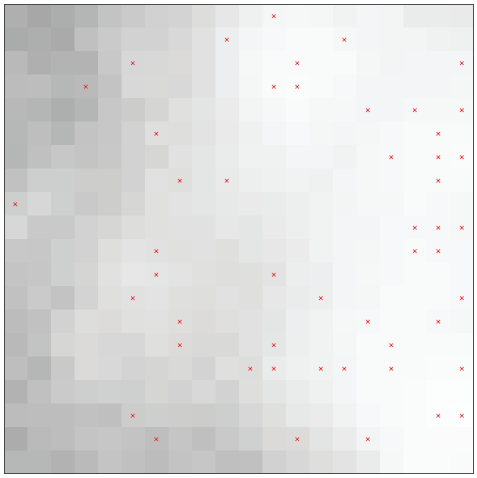

Fig. 3. U-Matrix Representation of Abstraction Level 1

Figure 3 demonstrates the separation of the data set into a number of prototype clusters. The level of illumination at each neuron indicates the average distance from its nearest neighbor, with the greater illumination indicating a closer proximity. The darker regions mark the edges of each cluster, defining the borders between one another. The x markers indicate the highest scoring BMUs after the final tuning iteration.

A second level SOM lattice was constructed using a 10 x 10 neuron configuration. The lattice size was reduced due to the inputs being index values of the level 1 SOM, a lower dimension than the original data set. The index values for each BMU node noted during the final phase of the level 1 SOM tuning were normalized and parsed through the second level SOM with a learning rate of 0.000001 over 100 iterations. The lower learning rate is attributed to the lower range of input values presented for clustering by the first layer. At this level of abstraction, only minor influence application is required in order to merge neighboring indexes, forming final clusters. The resulting BMUs observed after the tuning process completed are marked in figure 4 by an x. Each x on the image represents a unique cluster, or application protocol, to which members of the original data set are mapped.

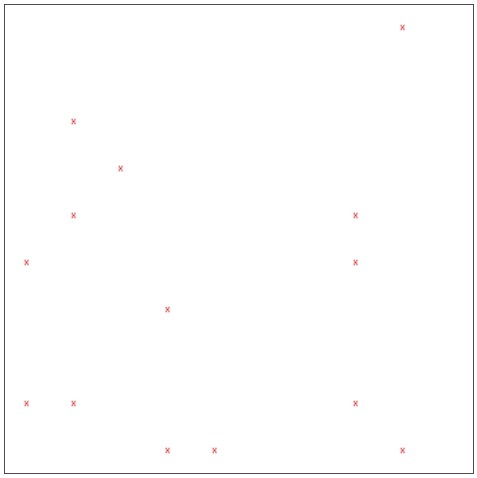

Fig. 4. Graphical Representation of Abstraction Level 2 Cluster Identification

The resulting data sets constructed by mapping each input to a particular cluster are shown in table I.

BMU Index (Cluster ID)	Samples
8	125
21	56
32	378
41	72
47	1
50	22
57	125
63	5
80	68
81	205
87	454
93	137
94	4
98	321

TABLE I
RESULTING ANNOTATED DATA SETS

The smaller protocol data sets were discarded due to the lack of sufficient samples. The lack of sufficient samples of a particular protocol is attributed to the relatively small observation window used to sample flows. Training an artificial neural network using limited samples could lead to an overly specific classifier (over-fitting), described by Cho and Cha [23]. For this reason, the authors decided to discard clusters

47, 63 and 94. The samples mapped to discarded clusters in a real-world application would remain in the training data set for future iterations, where additional samples of the same protocol would be captured over time.

The remaining 11 data sets were carried forward into the second part of the experiment, the training of classifiers to identify future versions of the protocol, as described by Goss and Botha [14].

B. Application Protocol Classifier Generation

A model for the training of supervised classifiers for the identification of application protocols was described by Goss and Botha [14]. The research by Goss and Botha [14] describes the requirements for achieving a high level of accuracy in the identification of application protocols at an early stage of their existence, using trained artificial neural network classifiers. These classifiers were trained using data sets manually annotated by experts. This research aims to extend the work of Goss and Botha in [14], by using the HSOM clustered data sets to train classifiers in the same manner. Removing the dependence on manually annotated data sets allows for new applications traversing a network to be identified and future occurrences classified automatically.

The original data set samples which were mapped to a single cluster collectively describe unique protocols and, as such, a new classifier is required in order to identify each. The structure of the classifier in each instance is in accordance with Goss and Botha [14], depicted in figure 5.

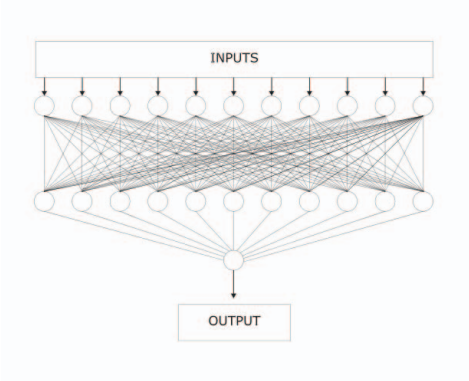

Fig. 5. Structure of each ANN classifier

Each classifier includes 3 layers. First, an input layer connects directly to the second, hidden layer. The hidden layer is directly connected to the final output layer. The output layer consists of a single neuron, the result of which representing the probability that a successful match was made for an input vector. The number of neurons in both the input and hidden layer are equal to the magnitude of the samples in the original data set. The flow inspector recorded 11 distinct discriminators per flow in accordance with the research of Goss and Botha [14], therefore the input layer comprises 11 neurons. The experiments of Goss and Botha [14] implemented a single hidden layer, with the number of neurons equal to that of the input layer. As such, the number of hidden layer neurons present the the classifier was configured as 11. The overall size of the network is purposefully kept small, both to conform with the design in Goss and Botha [14], as well as to avoid overfitting and improve generalization issues [23].

The training set for each classifier is constructed by marking all samples mapped to the specific cluster with a “1” and the others with a “0”. Each classifier was trained by passing its respective training set over a period of 1000 iterations. The number of iterations are the same as that used by Goss and Botha [14]. The graph representing the average delta between the expected output the actual output, with respect to each iteration is shown in figure 6.

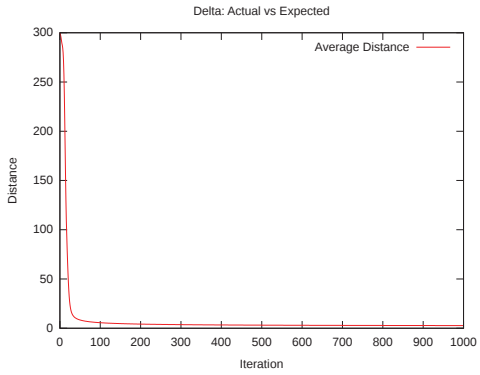

Fig. 6. Average Delta - Expected vs Actual Classifier Output

In figure 6, the “knee” of the graph is realized at approximately 50 iterations. Even so, 1000 iterations was maintained in order to more accurately compare this research with that of Goss and Botha [14]. Once the training process had completed for each classifier, the associated training set for each was passed once more for evaluation. The results for each cluster are displayed in table II.

Cluster ID	Accuracy
8	99.63%
21	99.68%
32	99.86%
41	99.74%
50	99.22%
57	99.87%
80	99.75%
81	99.86%
87	99.62%
93	99.58%
98	99.48%

TABLE II
TRAINED CLASSIFIER ACCURACY

This accuracy is based on the average certainty reported by the classifier for each sample tested. The results in table

II show that a high degree of accuracy was achieved when testing each classifier against its training set and evaluating the actual output against the expected output. The high degree of accuracy indicates the existence of a distinct pattern within each the training set. This indicates little or no overlap between training sets, validating the clustering process by the HSOM. The classifier is also shown to successfully converge during the training process, further attesting the distinction between the training data set and those samples external to the set.

Although the accuracy achieved by the classifiers parsing their original data set is important, they need to be able to identify these applications in real-world, live network traffic. The following section describes an experiment where these classifiers were tested against real-world network traffic.

The flow inspector software (FlowInspector v0.1)[1] used to generate the original data set was modified to not only extract discriminators for each flow observed, but also run these discriminators through each classifier trained in the previous section. The flow inspector was connected to the same network segment from which the original data set was derived.

The authors observed the actions of a number of users on the segment and determined a basic set of the most popular application protocols expected to be observed on the segment. These protocols are listed in table III.

Application Protocol	Model	Encrypted
POP3	Client-Server	No
SMTP	Client-Server	No
IMAP	Client-Server	No
HTTP	Client-Server	No
HTTPS	Client-Server	Yes
Soul-Seek	Peer-to-Peer	Yes
Bittorrent	Peer-to-Peer	No

TABLE III
EXPECTED APPLICATION PROTOCOLS

A number of flow samples were recorded, after which they were manually annotated by an expert by their respective application protocol. From these new data sets, five randomly selected samples for each were passed through the various classifiers and the average output scores recorded. The resulting scores for each identified protocol are shown alongside the results obtained by Goss and Botha [14] for the same protocol in table IV.

IV. RESULTS AND DISCUSSION

The results in table IV indicate a high level of accuracy in identifying the samples selected in each case. The POP3 samples scored the highest average accuracy on classifier 21, whilst SMTP by classifier 8. These results demonstrate that even though POP3 and SMTP exhibit similar characteristics relating to directional and packet size statistics, the SOM

[1]Available at: http://goo.gl/80BQE

Data Set	Best Match	Certainty	Goss and Botha [14]
POP3	21	99.88%	99.06%
SMTP	8	99.86%	96.92%
IMAP	57	99.77%	N/A
HTTP	80	69.60%	99.93%
HTTPS	98	99.35%	99.95%
Soul-Seek	81	99.83%	N/A
Bittorrent	93	99.70%	N/A

TABLE IV
PROTOCOL CLASSIFICATION ACCURACY VERSUS GOSS AND BOTHA [14]

mapping clearly distinguished them from one another. The IMAP protocol scored the highest on classifier 57, whilst the HTTP samples received the best score from classifier 80, a dismal 69.60%. HTTPS scored highest on classifier 98, with 99.83% average accuracy experienced.

The two peer-to-peer protocols, one encrypted and the other plain-text, were also easily identified by the system. The Soul-seek protocol scored highest on classifier 81, whilst Bittorrent on classifier 93.

The average score received for the HTTP protocol concerned the authors and warranted further investigation. The first step was to delve into the data set used to train classifier 80. These samples were compared to the samples annotated by the expert for the HTTP protocol and the problem was immediately apparent. The manually annotated data set contained samples which closely resembled those in the training set for classifier 80, however there were a substantial number of samples whose packet flow directionality discriminators were completely different. The remaining discriminators were, as expected, remarkably similar. The samples from the manually annotated data set which did not match those in classifier 80 were found to match samples in the data sets used to train classifiers 32, 41, 50 and 87. The HTTP protocol was, therefore, identifiable via classifiers 32, 41, 50, 80 and 87.

The reason for the directional discriminator variance amongst the HTTP protocol samples was due to the presence of HTTP pipelining. HTTP pipelining is a method whereby multiple HTTP requests can be sent through a single flow in succession. Whilst this improves the browsing experience for users on high latency links, it causes the directionality discriminators for HTTP flows to vary from a single client request packet to multiple, successive client packets. In the data set, the directional discriminators for each sample was represented as either a 1 for client to server packet, or 0 for server to client. The fact that 4 out of the 11 discriminators for each sample were the packet directionality indicators with such variance, led the SOM to identify multiple clusters. The remaining 7 discriminators were similar enough to have all samples join the same cluster, however the variance between the directional indicators was too extensive.

V. Conclusion

The research presented in this paper is an extension of the work of Goss and Botha [14]. Whilst Goss and Botha [14] made use of manually annotated training set data, this research showed it possible to remove the dependency on manually annotated data sets using competitive learning to perform such annotation automatically. The results obtained through the experiments in this paper show a high degree of accuracy is achievable using automatically generated classifiers. In the cases where the same protocol was trained in this research and by Goss and Botha [14], only one protocol scored significantly lower accuracy using methods described by this research. In the case of the POP3 and SMTP protocol, the results achieved in this research were slightly improved when compared with the results of Goss and Botha [14], whilst HTTPS scored marginally less. The HTTP protocol scored significantly less using the methods described in this research compared with using manually annotated data sets in [14].

An issue was subsequently discovered whereby the HTTP protocol exhibited multiple directionality properties and, even though the remaining discriminators were similar, multiple clusters were formed. The HTTP flow samples annotated by experts in this research therefore comprised of samples which spanned a number of clusters, causing low average certainty scores during the test. Goss and Botha [14] trained their HTTP classifier using flow samples which described the various directional properties HTTP may exhibit and, when exposed to the training set, achieved a high degree of average certainty. The problem of adapting the weights of the first layer SOM to mitigate this problem and the impact of such changes is still a topic of current research.

The results achieved in this research indicate a significant improvement in task performance, compared to the requirement for manual annotation described in [14]. Although a high accuracy was shown to be achievable, the authors advocate the use of the proposed method as a mechanism for hinting toward the underlying application protocol of a flow, early in it's existence, rather than a definitive assessment. Further analysis of the flow is required over a sustained duration in order to increase certainty of classification. This along with methods for weight adaption within the first layer SOM are still a work in progress.

References

[1] T. Auld, A. W. Moore, and S. F. Gull, "Bayesian neural networks for internet traffic classification," *IEEE Transactions on Neural Networks*, vol. 18, no. 1, January 2007.

[2] K. Huang and D. Zhang, "A byte-filtered string matching algorithm for fast deep packet inspection," in *The 9th International Conference for Young Computer Scientists*. The IEEE Computer Society, 2008, pp. 2073 – 2078.

[3] M. Gebski, A. Penev, and R. K. Wong, "Protocol identification of encrypted network traffic," in *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*. IEEE Computer Society, 2006, pp. 957–960.

[4] Z. Li, R. Yuan, and X. Guan, "Traffic classification - towards accurate real time network applications," in *Proceedings of the 12th international conference on Human-computer interaction: applications and services*, ser. HCI'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 67–76.

[5] A. W. Moore and K. Papagiannaki, "Toward the accurate identification of network applications," in *In PAM*, 2005, pp. 41–54.

[6] A. Moore, M. Crogan, A. W. Moore, Q. Mary, D. Zuev, D. Zuev, and M. L. Crogan, "Discriminators for use in flow-based classification," Tech. Rep., 2005.

[7] A. Mcgregor, M. Hall, P. Lorier, and J. Brunskill, "Flow clustering using machine learning techniques," in *In PAM*, 2004, pp. 205–214.

[8] Y. Zhang, Z. Li, S. Mei, and C. Fu, "Session-based tunnel scheduling model in multi-link aggregate IPSec VPN," in *Third International Conference on Multimedia and Ubiquitous Engineering*, 2009.

[9] R. Alshammari, A. N. Zincir-Heywood, and A. A. Farrag, "Performance comparison of four rule sets: An example for encrypted traffic classification," in *World Congress on Privacy, Security, Trust and the Management of e-Business*. The IEEE Computer Society, 2009, pp. 21–28.

[10] J. Vesanto and E. Alhoniemi, "Clustering of the self-organizing map," in *IEEE Transactions on Neural Networks*, vol. 11, no. 3, May 2000.

[11] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification," in *Proceedings of the 2006 ACM CoNEXT conference*, 2006.

[12] F. Gargiulo, L. Kuncheva, and C. Sansone, "Network protocol verification by a classifier selection ensemble," in *Proceedings of MCS*, 2009, pp. 314–323.

[13] R. Alshammari and A. N. Zincir-Heywood, "A flow based approach for ssh traffic detection," in *Proceedings of the IEEE International Conference on System, Man and Cybernetics*. The IEEE Computer Society, 2007, pp. 296–301.

[14] R. Goss and R. Botha, "Establishing discernible flow characteristics for accurate, real-time network protocol identification," in *Proceedings of the 2012 International Network Conference (INC2012)*, 2012.

[15] C. V. Wright, F. Monrose, and G. M. Masson, "On inferring application protocol behaviors in encrypted network traffic," *Journal of Machine Learning Research*, vol. 7, pp. 2745–2769, 2006.

[16] G. Szab, I. Szabo, and D. Orincsay, "Accurate traffic classification," in *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, June 2007.

[17] R. Rojas, *Neural Networks: A Systematic Introduction*. Springer-Verlag, Berlin, 1996.

[18] C. D. Manning, P. Raghavan, and H. Schutze, *An Introduction to Information Retrieval (Online Edition)*. Cambridge, England: Cambridge University Press, April 2009.

[19] A. Abdi and H. Szu, "Independent component analysis(ica) and self-organizing map(som) approach to multidetection system for network intruders," in *Proceedings of SPIE*, vol. 5102, 2003, pp. 348–353.

[20] T. Kohonen, "The self-organizing map," in *Proceedings of the IEEE*, vol. 78, 1990, pp. 1464–1480.

[21] J. Lampinen and E. Oja, "Clustering properties of hierachical self-organizing maps," *Mathematical Imaging and Vision*, vol. 2, pp. 261–272, 1992.

[22] A. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognition Lett.*, 2009.

[23] S. Cho and K. Cha, "Evolution of neural network training set through addition of virtual samples," in *in Proc. 1996 IEEE Int. Conf. Evolutionary Computation,ICEC96*. IEEE Press, 1996, pp. 685–688.