Bayesian Neural Networks for Internet Traffic Classification

Tom Auld, Andrew W. Moore, *Member, IEEE*, and Stephen F. Gull

***Abstract*—Internet traffic identification is an important tool for network management. It allows operators to better predict future traffic matrices and demands, security personnel to detect anomalous behavior, and researchers to develop more realistic traffic models. We present here a traffic classifier that can achieve a high accuracy across a range of application types without any source or destination host-address or port information. We use supervised machine learning based on a Bayesian trained neural network. Though our technique uses training data with categories derived from packet content, training and testing were done using features derived from packet streams consisting of one or more packet headers. By providing classification without access to the contents of packets, our technique offers wider application than methods that require full packet/payloads for classification. This is a powerful advantage, using samples of classified traffic to permit the categorization of traffic based only upon commonly available information.**

***Index Terms*—Internet traffic, network operations, neural network applications, pattern recognition, traffic identification.**

I. Introduction

ACCURATE identification of network applications is fundamental to numerous network activities, from security monitoring to accounting, and from quality of service measurements to forecasting for long-term provisioning. However, application classification schemes are inaccurate because the knowledge commonly available to the network, i.e., packet headers, increasingly does not contain sufficient information.

In common with [1], we describe a method with application to the networking community. We illustrate this method through examples applied to data which has been made available to us. We use supervised machine-learning to train a classifier, and then compare the predicted category with the actual category for each object. We attained a significantly higher degree of classification accuracy using less information than previous methods. Notably, for our neural network classifier, we do not use port or host information; this is the same situation as faced when attempting to classify anonymized traffic.

In the application of Bayesian technique, we plan to provide insight into the nature of network traffic itself. We demonstrate that information able to be derived from a traffic flow can allow the distinguishing of classes with an accuracy exceeding that of (standard) port-based mechanisms. We illustrate this classification process operating as an offline tool, as in the auditing of forensic work. Finally, we suggest ways to further improve the performance of our classifier, indicating that the most effective identifiers of an Internet application remain concentrated into a relatively small number of characteristics. The contributions of this paper include the following:

- illustration of the Bayesian framework using a neural network model that allows identification of traffic without using any port or host [Internet protocol (IP) address] information;
- a classification accuracy of over 99% when training and testing on homogeneous traffic from the same site on the same day;
- a classification accuracy of 95% in the (more realistic) situation of training on one day of traffic from one site, and testing on traffic from that site for a day eight months later; this is a figure significantly higher than for the approach adopted in [2];
- we document the features of flows, derived from streams of packet headers having of greatest contribution to classifiers based upon a naive Bayesian approach and upon a Bayesian neural network approach. We show that a small number of features carry high significance regardless of the classification scheme. We also show that there is some overlap in features of high importance to either method;
- finally, our work validates the premise that the activity of a network user (in terms of applications) is *reversible*. We define the process as *reversible* if it permits identification of a users network-based applications based upon the network traffic alone. Further, this process is performed without the benefit of IP (host) address or port information. The conclusion we draw, extending the work of others, is that traditional anonymization procedures that remove payload contents, IP address, and port numbers may not hide the network-based application observed.

II. Motivation and Related Work

The identification of Internet applications through traffic classification is a basic problem of broad interest. A clear interest exists among the operators of networks since, with knowledge of what applications are present in a network, network operators are better able to plan, budget, and bill. Network operators and users have a continuing interest in systems that can identify anomalies in network traffic to reduce the impact of malicious behavior [3]. Alongside these direct and practical applications of classification are others that set out to model the Internet: the

Manuscript received January 13, 2006; revised June 1, 2006.

T. Auld and S. F. Gull are with the Department of Physics, University of Cambridge, Cambridge CB3 0HE, U.K. (e-mail: tauld@mrao.cam.ac.uk).

A. W. Moore is with the Department of Computer Science, Queen Mary, University of London, London E1 4NS, U.K. (e-mail: andrew.moore@dcs.qmul.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2006.883010

modeling of traffic-mix, user composition, and so on. Our approach, relying upon packet observations but without access to site-specific information (such as the role of machines, access to the user and system-administrator community, or access to the content of packets) allows for a method with the widest possible application. We are independent of the original site that sourced the data and can operate in the traditional anonymization environments of packet traces without payload, IP address, or even port number. The ability to perform accurate identification without this information forms the motivation for this work.

The most common technique for the identification of network applications through traffic monitoring relies on the use of well-known ports: An analysis of the headers of packets is used to identify traffic associated with a particular port and thus of a particular application [4], [5]. Such a process is likely to give inaccurate estimates of the amount of traffic carried by different applications, since specific protocols, such as hypertext transfer protocol (HTTP), are frequently used to relay other types of traffic, e.g., a virtual local area network (VLAN) transported over HTTP. Also, emerging services avoid the use of well-known ports altogether as in some peer-to-peer applications. These traditional classification techniques face increasing inadequacy. A brief summary of previous classification methods now follows.

The work of Karagiannis *et al.* [6] about how accurate traffic classification can impact social and political decisions, cited differences in the perceived and actual growth of peer-to-peer networks. This approach requires accessible port (and IP address) information. Illustrating changes in accuracy derived from content-based classification, Moore and Papagiannaki [7] also noted an accuracy no better than 50%–70% for port-based classification using the official Internet Assigned Numbers Authority (IANA)[1] list.

In [8], Karagiannis *et al.* present a more generic classification scheme that, instead of examining individual flows, looks at patterns of flows among sets of ports and hosts. While this scheme is also shown to operate on payloadless flows, it supposes that access to full headers remain untested in situations where the IP address or port numbers are not available to the classification scheme.

Roughan *et al.* [9] classify traffic into a small number of categories suitable for quality of service applications. They use techniques such as clustering using nearest neighbor to provide the required classification. With a small set of features and an unknown (implicit assumption) of the accuracy of the testing and training data sets, the authors restrict themselves to broad properties common to relatively large sets of network-based applications.

In contrast, McGregor *et al.* [10] identify traffic with similar observable properties by applying an untrained classifier to the problem. The untrained classifier identifies classes of traffic having similar properties, but does not directly assist in understanding what or why applications have been grouped in this way—indeed, without content-derived information the authors are left to guess the precise type of each application. This work demonstrated that the properties of network traffic, using features on a per-flow basis, allow some differentiation to be inferred. As in [9], the authors restricted themselves to a small number of classes and a trivial set of features. Each class may have included a relatively large number of network-based applications, but the authors were never able to establish this method as anything more than a property-grouping technique. This work applied an unsupervised method to provide a clustering of traffic. Even in their final analysis, the clusters of traffic types were never identified beyond port-based methods: The technique itself introduces considerable error. McGregor *et al.* provide a useful contribution by illustrating that (unsupervised) machine learning is plausible, but the work is not comparable with our own as it is done without the benefit of an accurately preclassified training set.

The link between the applications causing network traffic and the features present in network traffic was further investigated by Zander *et al.* [11]. Following a motivation that intersects with our own work—inaccurate port-based classification and restricted access to payloads—they attempt maximum-likelihood (ML)-based classification of network traffic. In particular, they use sophisticated self-clustering to group similar-property traffic as a part of the classification process. However, like McGregor *et al.*, these authors used unsupervised learning but did not have access to a precise application membership. Such base-level uncertainty is in contrast with this paper.

Hernández-Campos *et al.* [12] aim to provide more accurate classification, based on simulation. They describe representative traffic patterns for input to the simulation but do not seek to identify the traffic itself. Soule *et al.* [13] perform flow classification aiming to identify membership among a small set of classes: Elephant flows (those of long-life and large data content), and nonelephant flows. Their method is used to estimate flow-membership probability.

Zuev and Moore [2] applied a naive Bayesian classifier to traffic classification. Significantly, the simple classifier assumed independence between each of the variables describing a particular object. The data, drawn from previous hand-classification work, exhibited considerable redundancy and significant interdependence among features describing each flow; this will significantly reduce the accuracy of classification. Clearly, a method that incorporates dependence, such as the Bayesian neural network, could provide a more robust and effective classifier.

Using an entropy maximization method, Gu *et al.* [14] illustrate the application of machine-learning techniques to the computation of baseline distributions representing traffic behavior. They are then able to identify deviations from this baseline as indications of anomalies. Papagiannaki *et al.* [15] demonstrate a similar method. Their technique performs a wavelet multiresolution analysis for the identification of overall long-term traffic trends. This trend prediction can then provide operational insight into the network. Mandjes *et al.* [16] also present work on anomaly detection in the specific application of voice over Internet traffic. These authors all report on the limits of measurement for effective anomaly detection.

In common with Papagiannaki *et al.*, Hajji [17] attempts to identify the baseline of normal network operation. Their method uses a stochastic approximation of the maximum-like-

[1]http://www.iana.org/assignments/port-numbers

lihood function and to again detect anomalies in the overall network behavior. For a further entropy-based mechanism, Xu *et al.* [18] documents a procedure for profiling the general makeup of traffic. However, these approaches are not intended to provide an insight at the level of network-based application, but to assist in an understanding of the higher level traffic mix.

A specific application of neural networks is demonstrated by Atiya *et al.* [19]. They describe techniques that are able to predict the utilization of video traffic using sparse-basis selection. They are able to both estimate the video traffic requirements and recursively update the estimation process.

The field of security also uses machine-learning techniques (see [20] for a recent survey). However, this work has largely been targeted at the detection of malicious behavior on a multiuser machine, or the filtering of false alarms [21].

For the recognition of a single flow of traffic, current intrusion detection systems for the network (e.g., [22] and [23]) commonly rely on signature matching and server ports as the primary identification technique. Our work is entirely compatible with these current systems and we foresee its implementation as part of a complete intrusion detection system (IDS) framework.

III. Data Set

We illustrate our method with preclassified data described originally in [7]. This data, also used in [2], consists of descriptions of Internet traffic that have been manually classified. Hand-classification of two distinct days of data for an active Internet facility provides the input for sets of the training and testing phases. The data was provided as a set of flows taken from two distinct days; each day consisted of ten sets of classified transport control protocol (TCP) traffic flows, with each object described by its membership class and a set of features.

Each of the ten data sets covered the same length of time (approximately 24 min); these nonoverlapping samples were spaced randomly throughout the 24-h period. The samples are intended to be representative of multiple times within the 24-h period. A description of the classes, definitions of the object of classification, and a description of the features of each object are given later. Further details of the original hand-classification are given in [7], and the data sets themselves are described at length in [24].

For the purposes of this study, we treated all data coming from a single day as a single *trace*—any subsampling of a trace meant we were subsampling from across the whole day, not selectively. We are then able to perform full cross correlation in the knowledge that we are not making unintended assumptions about the stability of either the traffic mix or features of the traffic over a 24-h period.

A. Data Collection

Throughout this paper, we have used data collected by the high-performance network monitor described in [25]. The data sets we used were based on traces captured using its loss-limited, full-payload capture to disk facility with time stamps having resolution better than 35 ns.

We examine data for several different periods in time from one site on the Internet. This site is a research-facility host to about 1000 users connected to the Internet via a full-duplex gigabit Ethernet link. Full-duplex traffic on this connection was monitored for each traffic set.

This site hosts several biology-related facilities, collectively referred to as a *genome campus*. There are three institutions on site that employ about 1000 researchers, administrators, and technical staff. The campus is connected to the Internet via a full-duplex gigabit Ethernet link. The monitor was located on this connection to the Internet. Each traffic set consists of a full 24-h weekday period in both link directions.

TABLE I
Examples of Network Traffic Allocated to Each Category [7]

Classification	Example Application
BULK	ftp
DATABASE	postgres, sqlnet oracle, ingres
INTERACTIVE	ssh, klogin, rlogin, telnet
MAIL	imap, pop2/3, smtp
SERVICES	X11, dns, ident, ldap, ntp
WWW	www
P2P	KaZaA, BitTorrent, GnuTella
ATTACK	Internet worm and virus attacks
GAMES	Microsoft Direct Play
MULTIMEDIA	Windows Media Player, Real

B. Traffic Categories

Fundamental to classification work is the idea of classes of traffic. In the present data set, traffic was classified into common groups of network-based applications. Other approaches to classification may have fewer definitions, e.g., malicious versus nonmalicious for an intrusion-detection system, or may opt for protocol-specific definitions, e.g., the identification of specific applications or specific TCP implementations [26].

Table I lists the classes we use alongside a nonexhaustive list of applications. Interestingly, each traffic class contains a variety of different types of data. For example, the BULK class consists of both the control and data flows of the file transfer protocol (FTP).

C. Classification Objects

Any classification scheme requires the parameterization of the objects to be classified. By using these parameters, the classifier assigns an object to a class.

The fundamental object classified in our approach is a TCP/IP traffic flow, which is represented as a flow of one or more packets between a given pair of hosts. It is defined by an n-tuple consisting of the IP address of the pair of hosts, and because we limit ourselves to TCP, the TCP port numbers used by the server and client [27].

In this paper, we are limited to the training and testing sets available. These consist of TCP traffic only, and are made up

TABLE II
EXAMPLES OF FEATURES DESCRIBING EACH OBJECT USED AS INPUT FOR CLASSIFICATION [31]. EACH OF THESE FEATURES IS PROVIDED FOR BOTH DIRECTIONS OF THE DUPLEX TRAFFIC, WHEREVER RELEVANT EACH FEATURE IS ALSO PROVIDED FOR THE TOTAL FLOW

Features
Flow metrics (duration, packet-count, total bytes)
Packet inter-arrival time (mean, variance, 1st & 3rd quartiles, median, minimum, maximum, ...)
Size of TCP/IP control fields (mean, variance, 1st & 3rd quartiles, median, minimum, maximum, ...)
Total packets (in each direction and total for flow)
Payload size (mean, variance, 1st & 3rd quartiles, median, minimum, maximum,...)
Effective Bandwidth based upon entropy [30]
Ranked list of top-ten fourier-transform components of packet inter-arrival times (for each direction)
Numerous TCP-specific values derived from `tcptrace` [31]
e.g., total payload bytes transmitted, total number of PUSHED packets, total number of packets,
total number of ACK packets seen carrying SACK information, minimum observed segment size, ...

of semantically complete TCP connections. Semantically complete flows are flows for which a complete connection setup and tear-down was observed. Although less well-defined flows may lead to a less accurate classification process, there is a degree of equivalence between any complete and incomplete flow definition [28]. The Bayesian methods we describe here are applicable regardless of the definition of a flow.

We have not evaluated our scheme against packet flows without an easily identified beginning or end. Along with incompletely observed TCP flows, such partially observed flows include those based on the user datagram packet (UDP) services and are routinely used for the Internet name services [domain name system (DNS)], streaming-media, and popular local-area protocols such as the Sun network file system (NFS). Claffy *et al.* [28] consider that employing an appropriate time-based mechanism—using inactivity timeouts to delimit flows—would be sufficient to delimit a flow for the purpose of work such as ours. We believe that an important interaction may take place between such timeout-based mechanisms and certain features we create; in itself, the topic of appropriate features for network traffic deserves further work in itself.

D. Manual Classification Process

The manual classification process consists of identifying classes, using content-based classification methods. In this process, a combination of manual and semiautomated processes was used to identify the membership class for each object (flow). An example of this process is that the total data-mix is reduced to many (millions of) flows for the 24-h period. The membership of each flow is then decided based on both the contents and knowledge about the systems that exchanged the flow. For example, a flow may be to or from a web server; the flow content is inspected to verify that the transfer contains web traffic, and the flow is then classified as being a member of world wide web (www). Another example is where inspecting the contents of a flow reveals that one previously identified as being to or from a web server, in-fact, contains peer-to-peer traffic. In this situation, the web server may be performing a dual-role: web-server and peer-to-peer participant, or the peer-to-peer traffic; it may be masquerading as web traffic.

This process of combining host knowledge, with verification-using content (also verification including contact with the users and site-administrators of the site) allowed the original data to be classified, by hand, with very high accuracy. The classification of this traffic is described in [7]. Related methods are discussed by others [6] and used in a number of intrusion-detection-systems, e.g., Bro and SNORT [22], [23].

As stated in the Introduction, the use of site-specific information, such as the role of machines, access to the user and system-administrator community, and the access to the content of packets, may not be assumed. We are motivated to use a supervised machine-learning process as it exploits the original manual-labor to *seed* classification for data quantities, such as that present in quantities that could not feasibly be classified using the same manual process as the original *seed* data.

E. Object Features

Each object is a flow of data described by its class and a set of features. The original flow data was not available to us, but each object has a number of particular properties, such as the client and server port of each flow, along with a number of characteristics parameterizing its behavior. These features allow for discrimination between the different traffic classes. Table II provides a summary of the 246 per-flow features that are available to us. A full description of these features is provided in [24].

The computational overhead for the total feature group has led us to use an offline technique at this time.

We make extensive use of the `tcptrace` tool [30] to provide features that are derived from the contents of the TCP and IP packet headers. Using a packet trace as input, this tool computes values based upon the packets it has processed. However, `tcptrace` processes a set of observations made by a monitoring system, so that measures such as round trip time (RTT) have components in both the (monitoring-point) $\rightarrow$ server and (monitoring-point) $\rightarrow$ client directions. This gives rise to two estimates of values such as RTT; in reality these values are partial-RTT as used and defined in [32]; conceptually, the two par-

tial-RTT values together construct the whole. The power of discrimination offered by certain features that are location-specific, like those derived from RTT, may change should the point of measurement be moved. The resulting change in the quality of discrimination is left for future work.

The IP address of the source or destination was not used as part of the data sets. This is to provide a degree of site-independence in the trained neural network; it also removes any dependence of the trained neural network upon address use within any network against which it is used. Early work indicated that the IP address incorporated an overwhelming amount of discrimination power. Though the particular information in an IP address has been used, for static classification, to good effect in the studies of others (e.g., [3], [6], [8], and [18]), we have been motivated by the desire to test features that are independent of a particular network configuration, especially IP addresses which, even on the same site, can change over time.

F. Note on the Construction of Features

All stages of the preparation of input data for this study—the construction of flows, the content-based classification of training sets, and the computation of features to describe flows—were performed as offline processes. Offline processing is consistent with a number of potential applications of accurate classification mechanisms, including, long-term forecasting and the auditing of network traffic. Each of these procedures may be performed online, in real-time, with appropriate tradeoffs.

As we noted previously, a method that relies upon the semantic definition of a TCP flow cannot provide flow objects until the end of the flow is observed. Thus, for real-time or continuous operation, semantically incorrect TCP flows must be constructed; and Section III-C discusses this approach for TCP and other datagram (e.g., UDP) data flows.

In view of this reliance on the complete flow observation, we do not compute object features until we have all flow observations for a period. However, the features themselves are relatively lightweight computations. Those with the greatest overheads rely upon the complete set of variables to compute a statistic, such as the median or first/third quartile.

Some of the features themselves can be computed only after a complete flow observation; a fertile area for further work is the establishing of flow features suitable across a wide range of applications. Such work would investigate the impact of lower *quality* features (such as prematurely computed median values) on the contribution of the feature to classification.

The computation of features for this work is object-independent, and linear in the number of features. Computation of a full set of features (246) for all traces took over 40 h on a dedicated system area network (SAN), but the removal of the computation of Fourier transfer values reduced the computation of the remaining features (226) across 553 188 objects to less than 40 s per flow, with a standard deviation of 22. Clearly, a better fast Fourier transform (FFT) implementation will contribute significantly to the computation period.

Another large contributor to the overall performance was the need to manipulate the files containing the trace data used in this work. In other implementations, these overheads will be replaced by the overheads associated with managing packet streams in memory and in the monitoring system. Of additional note is the large standard deviation in time taken per flow; the time taken to compute features is proportional to the size of the flow. Though there are some features that induce a fixed overhead, any computation based on a collection of packets of flows (e.g., the computation of interarrival time, packet size, and so on) incurred, for our features, an overhead proportional to the number of packets observed. With a modified definition of the flow object, the computation of any feature will be bound and finite; alongside the investigation of the adequacy of features, the definition of flows is a valuable area for further work.

G. Processing of the Features

As part of the procedure for using the Bayesian neural network, we process the features describing each traffic object. Feature preprocessing was not performed for the naive Bayesian work of Section IV.

The data comprise the class, the client and server port numbers, and 246 other flow features, some of which were described previously. Some are integers less than 65 536, others are floating point numbers, and some are boolean or have an unknown value. Later, we shall use a Bayesian neural network to predict the probabilities that a flow belongs to each of the ten categories.

We assume nothing about the inputs but believe the order for a given input may be important, and perform a "histogram equalization": the features were mapped between 0 and 1, and equalized by sorting and indexing. Thus, all numbers between 0 and 1 are represented approximately equally. All members having identical values are assigned the same output value. Boolean variables were labelled 0 or 1 in the obvious fashion, with unknown values being 0.5. Some further manipulation of the data is performed on an experiment by experiment basis, and this is described in Section V-A.

The data thus available to the classifier are 246 real numbers between 0 and 1. For each object (flow), or "row" of data, we also have the class (an integer between 1 and 10), which can be used for training and testing of the classifier. Following our note at the beginning of this section, we have two data sets each taken from two workweek days separated by eight months; there was a total of 337 000 flows in the first and 175 000 flows in the second.

Although we did not identify any remaining data issues, it may be desirable to identify the features with the most predictive power, and thus reduce the number of features in the categorization problem.

Preprocessing of the features for use by in the Bayesian neural network contributed a negligible overhead in the preparation of the features—the majority time remains in the overhead of their computation from the raw data.

We now proceed with a description of the methods used to classify our network data.

IV. Naive Bayesian Method

In this paper we compare two methods: The application of a Bayesian neural network described in Section V, and the naive Bayesian estimator used by Moore and Zuev [2].

TABLE III
COMPARATIVE RANKING OF VALUABLE FEATURES. *NB Rank* REFERS TO THE TEN MOST-VALUABLE FEATURES DERIVED USING THE FCBF AND THRESHOLD METHODS OF [2], SHOWN IN SECTION IV. *BNN Rank* REFERS TO THE FEATURE-INTERDEPENDENT RANKING DESCRIBED IN SECTION VII. PORT INFORMATION AND HOST IP ADDRESS WAS ASSUMED NOT TO BE AVAILABLE

NB Rank	BNN Rank	Description
1	9	The count of all the packets seen with the PUSH bit set in the TCP header. *server → client*
2	–	The total number of bytes sent in the initial window i.e., the number of bytes seen in the initial flight of data before receiving the first ack packet from the other endpoint. *client → server*
3	–	The total number of bytes sent in the initial window. *server → client*
4	–	The average segment size observed during the lifetime of the connection calculated as the value reported in the actual data bytes field divided by the actual data pkts reported. *server → client*
5	–	Median of total bytes in IP packet *client → server*
6	–	The count of all the packets with at least one byte of TCP data payload. *client → server*
7	–	Variance of bytes in (Ethernet) packet *server → client*
8	–	Minimum Segment Size *client → server*
9	–	The total number of Round-Trip Time (RTT) samples found.[2] *client → server*
10	7	The count of all the packets seen with the PUSH bit set in the TCP header. *client → server*
–	1	1st quartile of size of packet in bytes (all packets)
–	2	Post-loss ACKs: the total number of ACK packets received after we observed a (perceived) loss event and are recovering from it. *server→client*
–	3	The total number of ack packets seen carrying TCP SACK blocks (client→server)
–	4	Maximum idle time, calculated as the maximum time between consecutive packets seen in the direction. (client→server)
–	5	3rd quartile of size of packet in bytes (client→server)
–	6	3rd largest FFT component of packet IAT (server→client)
–	8	Total data transmit time, calculated as the difference between the times of capture of the first and last packets carrying non-zero TCP data payload. (server→client)
–	10	The time spent idle (where idle time is the accumulation of all periods of 2 seconds or greater when no packet was seen in either direction)
–	11	Minimum number of total bytes in IP packet (client→server)
–	12	The total number of ack packets seen (TCP segments seen with the ACK bit set) (server→client)
–	13	Third quartile of packet inter-arrival time (server→client)
–	14	Maximum of bytes in (Ethernet) packet (client→server)
–	15	The standard deviation of full-size RTT samples, where full-size segments are defined to be the segments of the largest size seen in the connection. (client→server)
–	16	The standard deviation of full-size RTT samples. (server→client)
–	17	The average throughput calculated as the unique bytes sent divided by the elapsed time i.e., the value reported in the unique bytes sent field divided by the elapsed time (the time difference between the capture of the first and last packets in the direction). (client→server)
–	18	The maximum window advertisement seen. (client→server)
–	19	The number of transitions between transaction mode and bulk transfer mode. Bulk transfer mode is the time when there are more than three successive packets in the same direction without any packets carrying data in the other direction
–	20	Median of bytes in (Ethernet) packet (both directions)

A. Naive Bayesian Approach

The naive Bayesian method associates an object with a particular class of membership based upon the sum of probabilities of membership for each feature. This approach assumes that all features are equally valuable and are independent. Further, in this simplest of approaches, the probability of membership is based upon the modeling of a feature by a Gaussian curve.

Clearly the assumptions of the model and the assumptions of the independence and importance of all features do not hold. In the work of Moore and Zuev [2], this approach is progressively refined to illustrate the impact that an improved model of the features and a refined list of features can give to the overall accuracy of the classification process.

Moore and Zuev [2] also include the use of naive Bayesian kernel estimation. This is similar to the naive Bayesian method algorithmically; the only difference arises in estimating the membership of an instance to a particular class: $f(\cdot|c_j)$, $j = 1, \ldots, k$. In contrast with naive Bayesian which estimated $f(\cdot|c_j), j = 1, \ldots, k$ by fitting a Gaussian distribution over the data, kernel estimation provides an estimate of the real density $f(\cdot|c_j)$ by

$$\hat{f}(t|c_j) = \frac{1}{n_{c_j} h} \sum_{x_i : C(x_i) = c_j} K\left(\frac{t - x_i}{h}\right) \tag{1}$$

where h is called the kernel bandwidth and $K(t)$ is any kernel, where a kernel is defined as any nonnegative function (although it is possible to generalize) normalized such that $\int_{-\infty}^{\infty} K(t)dt = 1$. Examples of such distributions include top hat $(K(t) = (1/2)I(|t| \leq 1))$, which generates a histogram to approximate $f(\cdot)$, Gaussian distribution $(K(t) = (1/\sqrt{2\pi})\exp(-t^2/2))$, and many others [33]. The naive Bayesian kernel estimation procedure used in [2] and replicated here uses a Gaussian kernel partly because it has desirable smoothness properties.

The accuracy of the naive Bayesian algorithm also suffers from irrelevant and redundant features. Following the procedure of [2], we used a fast correlation-based filter (FCBF), described in [34], as well as a variation of a wrapper method in determining the value of the threshold.

In FCBF, the value of a feature is measured by its correlation with the class-of-membership and with other valuable features. The *goodness* of a feature is improved if it is highly correlated with the class, yet not correlated with any other good features. Using the procedure described at length in [2, Sec. 4.4], the total number of features is reduced to a selection of ten described in Table III. Noteworthy is the feature *total number of RTT samples*, a curious feature that is part of the output of the `tcptrace` utility; this value counts the number of valid estimations of RTT that `tcptrace` can use [30] in its computation of the RTT estimate.

V. Classification Neural Networks

We shall use multilayer perceptron classification networks to assign classification probabilities to the flows. (See [35] and references therein for background; a Bayesian treatment is provided in [36].)

Fig. 1 shows a typical perceptron network with one hidden layer. The first layer contains the inputs, which in our problem are the 246 features described in Section III. The final layer contains the outputs, and in our problem these relate to the ten classes of membership to which a flow may belong. Intervening layers are described as *hidden*. There may be any number of hidden layers, comprising any number of nodes. The nodes in a hidden layer are connected to all nodes in adjacent layers. A particular network architecture is denoted by $N_{\text{in}} : N_1 : \ldots : N_{\text{out}}$ where N_{in} is the number of input nodes, N_i is the number of nodes in the ith hidden layer, and N_{out} is the number in the output layer. For example, Fig. 1 is described by $5:3:4$.

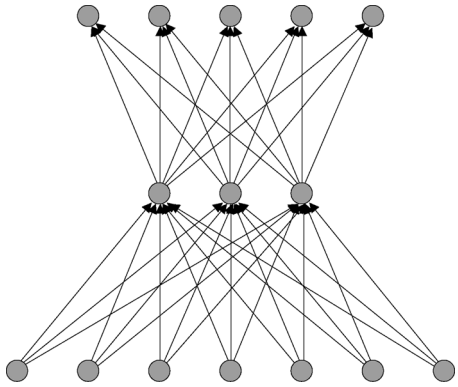

Fig. 1. Perceptron neural network with one hidden layer.

Each connection (represented by an arrow in Fig. 1) carries a weight w_{ij}, and we write $\mathbf{w} = \{w_{ij}\}$, the set of weights for the neural network. An activation function $g_j(u_j)$ is defined on the hidden layer, taking as its argument

$$u_j = \sum_i w_{ij} g_i(u_i)$$

where the sum is over all nodes i sending connections to node j in the hidden layer, as well as another node, the bias node, which is not included in the explicit architecture of the network. The bias node is taken to have constant output $= 1$ and is included to allow for additive constants in the activation function. The activation functions are chosen to be nonlinear, to enable the network to model any nonlinearities present in the problem. We will use the hyperbolic tangent function

$$g_j(u_j) = \tanh(u_j). \tag{2}$$

Other choices include a sigmoid function $g_j(u_j) = 1/(1 + \exp(-u_j))$ or a radial basis function network.[2] Derivatives of (2), which will be required in the Bayesian training algorithms described in Section V-A, are well defined.

In our problem, we interpret the value of the jth node in the output layer u_j, as being related to the probability that the class of the flow is j, given the weights and the inputs of a flow. The properties of a probability distribution require that $p_j \epsilon [0, 1] \forall j$ and $\sum_j p_j = 1$ for normalization. We apply the *softmax filter* to this layer to ensure that the activations $\{u_j\}$ in the output layer behave like a probability distribution

$$u_j = \frac{\exp(v_j)}{\sum_k \exp(v_k)} \quad v_j = \sum_i w_{ij} u_i.$$

[2]Here, the structure of the network is slightly different. The nodes in the hidden layer have activations $g_j(\mathbf{x}) = \exp -((\mathbf{x} - \mathbf{w}_j)^2/2\sigma^2)$ where $\mathbf{x}$ is the vector of inputs.

We now interpret $p_j = u_j$ and note that the filter ensures a positive, normalized distribution over the classes. Thus, given a network architecture $\mathcal{H} = 246 : N_1 : \ldots : 10$, a set of parameters $\mathbf{w}$, which are the weights of the network, and a set of 246 inputs for a flow $\mathcal{I} = \{i_j\}$, we now have defined a likelihood function for the classification problem

$$P(\mathcal{D}|\mathbf{w}, \mathcal{I}, \mathcal{H})$$

where

$$P(\text{Class} = j|\mathbf{w}, \mathcal{I}, \mathcal{H}) = \frac{\exp(u_j)}{\sum_i \exp(u_i)}. \tag{3}$$

For a particular input vector, the output vector of the network is determined by *feedforward* calculation. We progress sequentially through the network layers, from inputs to outputs, calculating the activation of each node, until we calculate the activation of the output nodes.

In this paper, we restrict ourselves to zero and single-layered networks. There are two reasons for this simplification. First, it is believed that single-layer networks contain enough structure to solve the classification problem to adequate accuracy. Second, the Bayesian training algorithms described in Section V-A are much more robust with zero and single-layer networks. Each network architecture we consider is thus of the form $246 : N_i : 10$ where i can be zero (no hidden layer) or positive (three layer network with i nodes in the single-hidden layer).

A. Bayesian Network Training

Network training consists of choosing the best weights of the network for a particular problem. In supervised learning, we use a training set to teach the network, and the problem is analogous to that of fitting a function to a set of data.

A common approach is to define a loss function c between the outputs or predictions $\{\rho_i\}$ of a network evaluated on some training inputs, and the targets $\{t_i\}$, which are the actual values those outputs take on the training set inputs. The predictions depend on the choice of the weights, and we write $\rho_i = \rho_i(\mathbf{w}) \forall\, i$. The total loss is defined as

$$L(\mathbf{w}) = \sum_i c\left(\rho_i(\mathbf{w}), t_i\right). \tag{4}$$

The aim is now to minimize the loss L in (4) with respect to $\mathbf{w}$ subject to some constraints of a regularizer $\Omega(\mathbf{w})$, which avoids overfitting to the training set. Using Lagrange multipliers, we thus aim to minimize

$$L(\mathbf{w}) + \alpha\Omega(\mathbf{w}). \tag{5}$$

We use a wholly Bayesian approach. We assign a prior to the weights and construct a posterior over these unknowns using Bayesian theorem. It can be shown that the two approaches are similar [36]. The log likelihood is equivalent to the loss function, and the log prior behaves like the regularizer. However, the Bayesian approach has certain advantages including the generation of error bars on the weights from the posterior. Other parameters in the prior (regularizer), such as α in (5), are also chosen consistently and the evidence $(P(\mathcal{D}|\mathcal{H}))$ may be computed and used for model selection. The experiments in this paper are conducted using the software package MemSys developed for this purpose by maximum entropy data consultants [37].

Some preprocessing of features is required prior to input as training data. We rescale features linearly, so that the mean of the training data is 0 and the standard deviation is 0.5. This is done because the neural network contains a nonlinear activation function [the $\tanh$ function of (2)] which is of order 1. This rescaling is performed so that the variance of the inputs is of a similar scale to that of the nonlinearity in (2). It has been shown from the performance and the evidence of similar networks that a standard deviation of 0.5 is optimal. In network training, this transformation is stored so that the inputs of the testing set can be rescaled for prediction.

In Section IV, we defined our likelihood function for the classification for a number of hypotheses $\mathcal{H}_i i \epsilon \mathcal{N}$, given the parameters $\Phi = \mathbf{w}$ (3). We now infer the maximum *a posteriori* probability (MAP) best estimates for the weights of the neural network $\mathbf{w}$, which will ultimately be used for class prediction.

The weights are the unknown parameters in the likelihood, and we require a prior distribution on them $P(\mathbf{w})$. A common choice of prior on the weights of such networks is the Gaussian prior. However, we have no bias for one particular neural network or another, and wish to consider as broad a choice of weights as possible. Gaussian distributions are of the form $\propto \exp(\alpha F(w))$ with $F(w) = -(1/2)w^2$. The maxent prior behaves like $F(w) = |w| \log(|w|)$ and converges more slowly to zero for extreme values of w than the Gaussian. The maxent prior is seen as a more appropriate choice than the Gaussian equivalent since it is less constraining on "large" values of the weights, which may be required to build an accurate classifier. Examination of the evidence also reveals that maxent priors provide better solutions in many problems.

We thus initially choose the maximum entropy prior

$$P(\mathbf{w}|\alpha) = \frac{1}{Z_S(\alpha)} \exp \alpha S(\mathbf{w})$$

where the entropy S is defined over both positive and negative values of the weights via

$$S(\mathbf{w}) = \sum_{w_{ij} \epsilon \mathbf{w}} \left(\psi_{ij} - 2m - w_{ij} \log\left((\psi_{ij} + w_{ij})/2m\right)\right.$$

$$\psi_{ij} = \left(w_{ij}^2 + 4m^2\right)^{\frac{1}{2}}.$$

The prior contains two parameters m and α. Here, m is called the default level and is chosen to be 0.15.[3] We will drop any dependence of the prior on this parameter, as it is found to make

[3]0.15 is chosen when the scale of the data is such that the standard deviation of the inputs is 0.5.

very little difference. The hyperparameter α has a much greater effect on training, and we keep it in the formulation at this stage.

Bayesian theorem can be written as

$$P(\Phi|\mathcal{H},\mathcal{D}) = \frac{P(\Phi|\mathcal{H})P(\mathcal{D}|\mathcal{H},\Phi)}{P(\mathcal{D}|\mathcal{H})} \tag{6}$$

which yields

$$P(\mathbf{w}|\mathcal{D},\alpha,\mathcal{H}) \propto Z_S(\alpha)^{-1}\exp\alpha S(\mathbf{w})P(\mathcal{D}|\mathbf{w},\mathcal{H}). \tag{7}$$

By writing the neural network classification likelihood in (3) as

$$P(\text{Class} = j|\Phi,\mathcal{I},\mathcal{H}) = u_j = f(j,\mathbf{w},\mathcal{I})$$

the likelihood in (7) becomes the product of the $f(j,\mathbf{w},\mathcal{I})$ for each flow in the training set $\mathcal{D}$[4]

$$P(\mathcal{D}|\mathbf{w},\mathcal{H}) = \prod_i f(C_i,\mathbf{w},\mathcal{I}_i) \tag{8}$$

where the product runs over each member of the training set $\mathcal{D}$. Here, C_i and $\mathcal{I}_i$ are the class and features of the ith flow in $\mathcal{D}$.

The prior contains a parameter α. This is known as a nuisance parameter. We could assign α its own prior $P(\alpha)$ and integrate it out of the posterior so that we have a distribution independent of α

$$P(\mathbf{w}|\mathcal{D},\mathcal{H}) = \int d\alpha P(\alpha)P(\mathbf{w}|\mathcal{D},\alpha,\mathcal{H}).$$

However, in view of the optimization algorithms described later, we prefer to retain the parameter α at this stage. The normalized posterior distribution $P(\mathbf{w}|\mathcal{D},\alpha,\mathcal{H})$ is thus

$$\frac{Z_S(\alpha)^{-1}\exp\alpha S(\mathbf{w})\left[\prod_i f(C_i,\mathbf{w},\mathcal{I}_i)\right]}{P(\mathcal{D}|\alpha,\mathcal{H})}. \tag{9}$$

Equation (9) provides a posterior distribution over the weights of the network for each value of α. We use the conjugate gradient algorithm in the MemSys environment to find the MAP values of the weights. We interpret the MAP weights as corresponding to the most likely neural network given the hypothesis (neural network architecture). The algorithm is quite involved and is described further in [37]. The search of the weight and α space is conducted first by holding α at a large constant value and evaluating the MAP in $\mathbf{w}$ space. The derivatives of α are evaluated, α is reduced, and the algorithm is repeated. A stopping value of α is found where the posterior is thus maximized.

We have thus inferred the most probable value of the weights of the neural network given some training data. This defines a classification scheme, which we now use to predict the classes of the Internet flows found in the data set.

[4]We interpret the training set $\mathcal{D}$ as comprising logically independent samples from the likelihood distributions. The probability of the whole sample thus factorizes into the product of individual likelihoods above.

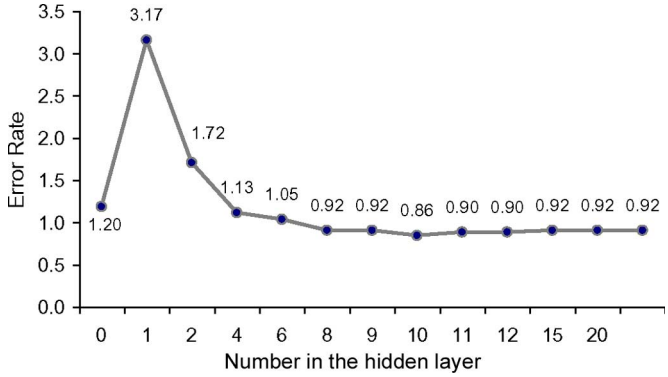

Fig. 2. Error rate versus number in the hidden layer.

B. Choosing the Architecture

We must decide what network architecture is most applicable in this problem. We are restricted to MLP networks with zero or one hidden layer, since the MemSys algorithm typically fails on networks with two or more hidden layers. It would be desirable to conduct experiments on a variety of architectures (or hypotheses $\{\mathcal{H}_i\}$) and calculate the corresponding evidences $(\{P(\mathcal{D}|\mathcal{H}_i)\})$. This would allow us to choose "the most probable" hypothesis. However, the integrals involved in $P(\mathcal{D}|\mathcal{H}_i)$ are over many dimensions and numerical computation proved unfeasible. Instead, we use cross validation: We train nets with different structures on a subset of the data, and calculate the classification error (the testing error) on another set of data. We consider networks with between 0 and 30 nodes in the hidden layer (0 being a network with no hidden layer).

Testing error is a function of the network and the training/testing sets, so in the cross-validation test, we use the same training and testing data for each architecture. We use a single set of training data consisting of classified flows from 10% of the data from the first trace. The test set is a further 10% of the data from this trace (distinct from the training data). The network training algorithm described in Section V-A was run to find the MAP values of the weights for each architecture, and the predictions calculated for the flows in the testing set. Fig. 2 shows the testing error. The errors for networks with more than seven nodes in the hidden layer are all similar, and lie between 0.86 and 0.92. Any of these architectures would be suitable, but it was decided to use the value at the minimum, which is 10. Some further justification can be provided by the presence of ten classes in the problem. The number in the hidden layer provides the dimension of the *hidden* space. This is the nonlinear projection of the input space via the map created by the weights and activation function in the first to second layer of the network. Having a dimension greater than or equal to the number of classes could provide the necessary complexity to accurately classify *all* classes of traffic.

We will now proceed with a variety of experiments to test the accuracy and properties of the classification scheme, including first a comparison with the naive Bayesian method.

TABLE IV
ERROR RATES WHEN TRAINING AND TESTING WITHIN TRACES 1 AND 2, AND WHEN USING TRACE 1 TO TRAIN AND TRACE 2 TO TEST

		Naive Bayes Accuracy Results			Neural Network Accuracy Results		
Training Set	Testing Set	Flow	Packet	Byte	Flow	Packet	Byte
50% Trace 1	Remaining 50% Trace 1	74.8 $\pm$ 1.7 %	86.8 $\pm$ 1.7	86.7 $\pm$ 2.0	99.26 $\pm$ 0.4 %	99.15 $\pm$ 0.3 %	99.11 $\pm$ 0.4 %
50% Trace 2	Remaining 50% Trace 2	95.0 $\pm$ 1.7 %	95.3 $\pm$ 1.1	93.9 $\pm$ 0.8	99.84 $\pm$ 0.2 %	99.8 $\pm$ 0.1%	99.78 $\pm$ 0.1 %
50% Trace 1	Trace 2	88.3 $\pm$ 2.3 %	90.7 $\pm$ 1.4	88.2 $\pm$ 1.8	95.3 $\pm$ 0.9 %	95.1 $\pm$ 0.9 %	94.8 $\pm$ 0.9 %

TABLE V
ACCURACY FOR DIFFERENT SIZES OF TRAINING SET

Training Set		Testing Sets	Accuracy	Training Time
0.1% Trace 1	n =377	Trace 2	88.3 $\pm$ 0.9%	<1min
1% Trace 1	n =3,775	Trace 2	91.9 $\pm$ 0.9%	12min
10% Trace 1	n =37,752	Trace 2	93.1 $\pm$ 1.3%	2h 54min
50% Trace 1	n =188,763	Trace 2	95.3 $\pm$ 0.9%	38h 48min

VI. NAIVE BAYESIAN AND NEURAL NETWORK COMPARISON

For both methods, naive Bayesian, and results gained using the Bayesian neural network, we do not use the server port number as a feature to describe flows of network traffic.

For the naive Bayesian experiments, we use the best approach of [2] naive Bayesian combined with a kernel-estimate method to provide an improved model of each feature and a reduced set of ten features, selected using the FCBF approach.

For the neural network, we use an MLP with ten nodes in a single-hidden layer, trained using the MemSys algorithm described in Section V-A.

We use three experiments to compare of the methods. The first experiment tests classification accuracy within trace 1. 50% of the data is chosen randomly as training data, and the remaining 50% used to test. The experiment is repeated five times, using different randomly chosen training sets of the same size and the standard deviation across the repeated experiments computed. The second experiment is identical to the first but Trace 2 is used in place of Trace 1. The third experiment uses training data created using (a sample of) Trace 1, and tests on the whole of Trace 2. It is not possible to train on the entire set from Trace 1 due to memory limitations. We select 50% of the flows from Trace 1 for use as training data. Again, we choose this data randomly, run the experiment five times, and indicate the mean accuracy along with the error margin representing standard deviation across the set of experiments.

We use three separate metrics to evaluate the success of the two classifiers. The first of these is the accuracy derived according to the classification of the objects (the TCP flows). This is the metric used to choose the neural network architecture. The second and third metrics are computed based upon the number of packets and the number of bytes of data carried by each flow. The results of the experiments evaluated using the three metrics are shown in Table IV.

In all cases, the neural network outperforms the naive Bayesian classifier. The accuracies of the first two experiments are higher than the third, for both methods. This shows that we are able to classify traffic that is homogeneous to a high (>99% in the case of the neural network) degree of accuracy. The composition of the traffic has changed during the eight months between Traces 1 and 2, but not so much as to make the classes unpredictable using the features, and we achieve accuracies around 95% for the neural network. Indeed, we consider this 95% success rate the headline result of the paper, since this accuracy was achieved in the near realistic situation of using a training set from eight months prior to the training set.

VII. NEURAL NETWORK INVESTIGATION

The neural network provides a likelihood which is not separable over the input features. In this section, we will further investigate the behavior of the neural network as the training set and other factors are varied. Here, we will concentrate on flow accuracy, but we believe we would obtain similar results and insights using packet, or byte metrics, as the figures obtained for different metrics in Section VI are very similar.

A. Makeup of the Training Set

In the experiments in Section VI, we used training sets of the order of hundreds of thousands of flows. For Bayesian neural networks, this required the use of hundreds of megabytes of memory and provided for a training time of the order of tens of hours. It would be desirable to use smaller training sets, if possible.

As the first investigation, the third experiment previously described was conducted using 0.1%, 1%, and 10% of the data in Trace 1 to train, and the whole of Trace 2 to test. The accuracies, along with the time taken for the training algorithm to converge[5] are shown in Table V for these and the corresponding experiment using 50% of the training data. Experiments were repeated five times. The mean is given along with the standard deviation (as error range).

It appears that we cannot reduce the size of training set without reducing the accuracy of the classifier, at least with the size of training sets used in this paper. The training time taken for the larger set is almost 39 h, but this is not larger than

[5]The algorithm is run on a single 2.2-GHz Intel Celeron-based host.

TABLE VI
CLASSIFICATION ACCURACY VERSUS COMPOSITION OF THE TRAINING SET (TRAINING AND TESTING WITHIN TRACE 1)

Class	N Train	Percentage	Accuracy
Atck	1793	0.47	68.6
Blk	11538	3.1	97.4
Db	2648	0.70	97.6
Gm	8	0.002	0
Int	110	0.03	0
Mail	28567	7.6	99.6
MM	576	0.15	67.0
P2P	2094	0.55	62.0
Srv	2099	0.56	96.0
Www	328106	86.9	99.8
Total	377539	–	99.3
Average Class Accuracy	–	–	68.8

TABLE VII
CLASS ACCURACIES ACHIEVED WITH 2441 FLOWS PER CLASS (TRAINING AND TESTING WITHIN BOTH TRACES)

Class	Atck	Blk	Db	Mail	P2P	Srv	Www
Accuracy	91.1	98.9	99.9	98.7	97.2	99.4	91

the time required to classify the training data, as discussed in Section III.

The classification accuracies produced so far are derived on a flow basis over the whole test set. However, the accuracies of the classes within the test set vary greatly. Table VI shows the accuracies for one of the neural networks in experiment 1 from Section VI, along with the make up of Trace 1.

Clearly some of the accuracies are very low (or even zero) and the average class accuracy is only 68.8%. The accuracies for the classes vary with the number of instances of that class in the training data. The "www" class makes up 87% of the flows in the data set.[6] and has the highest accuracy among the classes at 99.8%. Indeed, it is the high accuracy for this class which ensures a high overall accuracy of 99.3%.

If we are interested in other criteria for classification, for instance if we were only interested in a single class of flows, such as "mail" flows, or if we wanted to maximize the average class accuracy unweighted by the number of flows, then the above network is surely not optimal and we may wish to choose a different composition of training data.

To maximize the average class accuracy, we try a network with the same proportion of training data for each class. We use both Traces 1 and 2 here as training and testing data and, for practicalities sake, we omit the "Gm," "MM," and "Int" classes from our analysis since the number of instances is minimal. We also vary the total number of training data logarithmically and observe the effect on the accuracy. The results are shown in Fig. 3. The accuracies per class for the larger training set are also shown in Table VII.

[6]While the "www" class is well represented in flows, it represents less in terms of data packets (23%) and data bytes (22%).

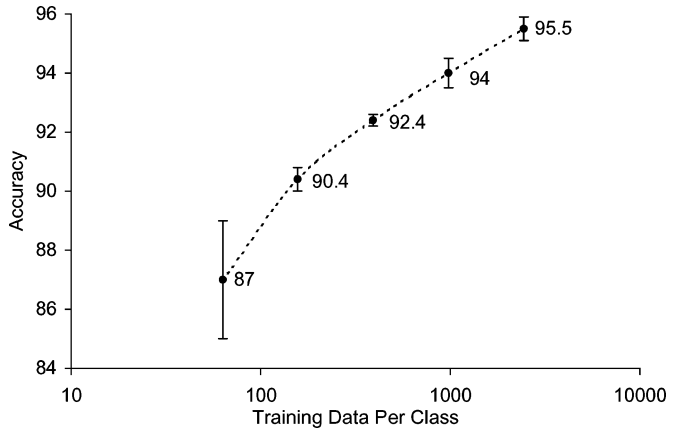

Fig. 3. Average class accuracy versus class training size (training and testing within both traces).

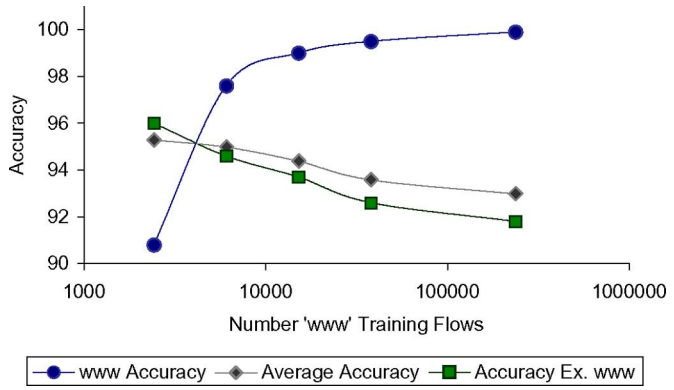

Fig. 4. "Www" class accuracy and average class accuracy as "www" flows are increased in the training set.

The average class accuracy has increased in these experiments, and the training set size (and hence training time) has reduced greatly. The accuracy increased roughly logarithmically with the training size, at least for the training sizes used in this experiment, except the very small set with ten flows per class. We were able to achieve an accuracy of 95.5% across all flows when both training and testing within Traces 1 and 2 combined. However, we are limited to around 2441 training data for each class in this set. Given the higher accuracies shown in Table VI, it is reasonable to expect that higher class accuracies would be achieved if more training data of the sparse classes were available to us. This would indeed be the case if we were to hand-classify more of the traces selectively, since only a proportion was hand-classified as described in Section III.

Next, we wish to investigate what happens when we hold the training sizes of all other classes constant at the 2441 level, and increase the proportion of "www" flows in the training set. We expect the "www" class accuracy to increase, but at what cost to the accuracies of the other classes? Fig. 4 illustrates how the "www" accuracy increases as the accuracy of the other classes decreases.

As expected, the "www" class accuracy does increase (up to 99.9% when using about 238 k flows), but at a cost to the average class accuracy. However, when using larger numbers of training data for the other classes ($\sim$2400), we are still able to retain respectable ($>$90%) class accuracies for the remaining classes, in contrast to the results in Table VI.

B. Interpretation of a Training Set as a Prior on the Composition of the Network Traffic

The makeup of the hand-classified traffic in Trace 1 is very skewed among certain classes (Table VI). In particular, 87% of the objects are "www" flows and over 97% of the flow objects are made up of three of the ten classes. Training on an unadjusted data set does give high accuracies over the whole set, but low accuracies on some classes. As we have shown, accuracies for a given class depend on the number of that class in the training data, and also the proportion of the total training data made up by that class.

If the aim of the classifier is to maximize the average class accuracy, it is optimal to use a training set made up of equal amounts of each class. Similarly, if accurate classification of a given class is more important than another, this should be represented in the makeup of the training data. In fact, it is not necessary to alter the makeup of the training set per se, since we can alter the terms in the likelihood in (8) correspondingly. It is possible to bias each member of the training set by an amount b_i ($b_i \geq 0$). If a flow i is present b_i times then the contribution to the likelihood from these flows is $f(C_i, \mathbf{w}, \mathcal{I}_i)^{b_i}$. Thus, if we wish to bias single flows of a particular type by an amount b_i each, we alter the likelihood to be

$$P(\mathcal{D}|\mathbf{w}, \mathcal{H}) = \prod_i f(C_i, \mathbf{w}, \mathcal{I}_i)^{b_i}. \tag{10}$$

$b_i = 1$ corresponds to no biasing, and if $b_i = 1 \; \forall \; i$ then (10) recovers the original likelihood as we would expect.

If we are aiming to maximize the accuracy over all flows, it is optimal to use a training set with classes in proportion to the expected frequency of the classes in the test set. In the example of using Trace 1 to train the network to classify Trace 2, which is the traffic from the same site eight months later, it could be reasonable to assume the composition of traffic will be similar. This may not be reasonable to assume, but in the absence of information to the contrary it is rational. If the user of the neural network believes there is a greater occurrence of a given class, then that user should be free to choose a training set [or bias the set as in (10)] that gives greater weight to that class. The situation is analogous to the use of a prior that allows the user to incorporate additional information in the inference problem posed in the problem.

Thus, in the process of deciding what training set to use, or how to bias each flow in the training set, the user must consider two things: What linear function of the class accuracies should be maximized, and second, if there is any available information about the composition of the test set.

C. Reducing the Number of Inputs

In this section, we aim to reduce the number of features in the classifier. Each of the current 246 features take a finite time to compute. Reducing the number of inputs to the network would reduce the training time and make real life applications faster and easier to implement. To identify the inputs having the greatest predictive power, we examine the weights of one of the trained neural networks. The feedforward calculation of the output vector given an input vector, described in Section V, defines how the input values influence the outputs. In the first (input) layer of a neural network, we see that the magnitude of the weights associated with each input node determines how much the input value of that node affects the activation of the nodes in the next layer. If all the weights are zero for a particular node, that input will have no effect on the activations in the next layer, or, hence, on all activations in the network. Thus, the output classification probabilities are not influenced by that node, and, hence, the input has no classification power. Conversely, if the weights for a given input node are much larger than all the other input nodes, the classification probabilities will depend solely on that node, and we need consider only that one input. Thus, the quantity of interest we evaluate is

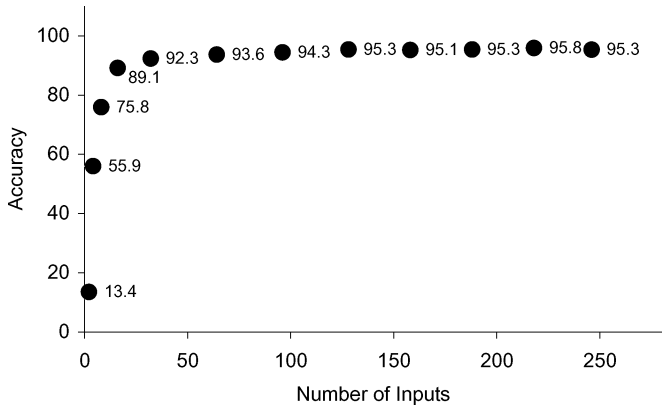

Fig. 5. Average class accuracy changes with number of inputs.

$$\text{Signal}(i) = \sum_{\text{j subject to } w_{ij} \epsilon \text{ input layer}} |w_{ij}|$$

where the sum is over all weights associated with the input node i, in the first layer of the network.

Inspection of this quantity revealed that a small number of inputs had a relatively large signal compared to the other inputs (this quantity had a skewed distribution). It was decided to observe how the classification accuracy changed when different numbers of inputs where used, using the signal quantity as a criterion for input exclusion. The training set used contained equal numbers (2441) of classes from Traces 1 and 2 (omitting the "Gm," "MM," and "Int" classes as in VII-A), and the test set is the remainder of the traces. The figure of interest is the average class accuracy over the classes. For each of the neural networks, the inputs omitted were the ones with the lower "signal." The experiments were started with very low numbers of inputs (4, 8, 16 ...) and the results are shown in Fig. 5.

For small numbers of inputs, the accuracy is reduced drastically from the previous experiments. The accuracy increases rapidly to the 95% region, and for all networks with inputs $\geq$128 it lies within error bars of the 95.1% figure obtained with 246 inputs. These results imply that a network with 128 inputs is sufficient for this classification problem.

These results have an important implication for the nature of network traffic: Aside from such obvious information as server port information, other flow features can provide an equally effective identification of the Internet application. Table III provides insight into what constitutes the 20 "important" inputs. The majority of these values are derived directly by observing one or more TCP/IP headers using a tool such as `tcptrace`

TABLE VIII
CONFUSION TABLE OF THE CLASSIFIER

Predicted class vs Actual Class	Atck	Blk	Db	Mail	P2P	Srv	WWW
Atck	89.3	0.2	0.2	0.7	51	–	6.8
Blk	–	97.9	–	1	–	0.3	0.5
Db	0.4	–	99.6	0.2	–	0.3	0.1
Mail	1.1	1	–	97.2	–	–	0.2
P2P	1.1	0.4	–	0.6	47.1	0.3	2.8
Srv	0.4	0.3	–	0.2	–	98.7	0.3
WWW	7.6	0.1	0.2	0.2	2	0.3	89.2

TABLE IX
ENTROPY MATRIX OF DISTRIBUTIONS FOR PREDICTION OF ONE CLASS, GIVEN ANOTHER

Predicted class vs Actual Class	Atck	Blk	Db	Mail	P2P	Srv	WWW
Atck	**0.11**	1.23	0.95	0.82	0.94	–	0.7
Blk	–	**0.09**	–	0.76	–	0.68	0.74
Db	0.15	–	**0.05**	0.96	–	0.79	0.82
Mail	0.54	0.55	–	**0.12**	–	–	1.35
P2P	0.56	0.76	–	0.71	**0.32**	0.8	0.64
Srv	1.12	1.3	–	1.45	–	**0.07**	0.8
WWW	0.59	1.31	1.62	1.08	0.29	0.58	**0.26**

[30] or performing simple time analysis of packet headers. We suggest that the ability to accurately derive such information about the causal application has important ramifications. An example is the provision of secure private network services where it is desirable to reduce the leakage of information. Clearly, anonymized IP address and port data may not be sufficient to disguise an Internet application.

From our two feature-reduction methods, we can compare features that provide the best discrimination power when used in a feature-independent setting (naive Bayesian) and when used in the correlated-feature setting (Bayesian neural network). Upon comparing each reduction, as shown in Table III, we note a limited overlap and a number of significant differences. A full study of the correlation of prediction of features and specific classes is left for future work, but values such as those dependent upon the RTT will be subject to change depending upon the site monitored; the precise impact of this upon their functionality as a feature deserves clarification.

D. Additional Information From the Likelihood Distributions

The probability distribution implied by the neural network likelihood, given some input data, has an interesting structure. It is expected that, when a false prediction is made, the probability distribution from the likelihood will yield some extra information about the quality of that prediction. As discussed in Section V, the network gives ten numbers $\{p_i\}$ which form a probability distribution over the classes. We study the *entropy* of this distribution, defined by

$$S(\{p_i\}) = -\sum_i p_i \log(p_i).$$

Shannon [38] showed that S behaves like the inverse of information; this quantity is maximized for a uniform distribution ($p_i = (1/10)\ \forall\ i$) and minimized (at zero) for a "delta" function corresponding to certainty ($p_i = 1$ some $i, p_j = 0 \forall\ j \neq i$). We now examine the entropy of the distributions when the net makes both correct and incorrect predictions.

Tables VIII and IX relate to a network trained on equal numbers of the classes from Traces 1 and 2, and tested on these traces. The three infrequent classes were omitted. Table VIII is a confusion table, showing the percentage occurrences of the flow classes amongst those with predictions for a given flow. For example, 89.2% of the instances where "www" was predicted were correct, and the remaining 10.8% of instances are spread out among other classes. Also, Table IX is a matrix of average entropies for the distributions for each class predicted, given the actual class of the flow. The diagonal elements of this entropy matrix are the average entropies of the distributions for which the predictions are correct. It is interesting to compare this number with others on the same row (where the same class was predicted, but incorrectly). The other numbers are higher except in the case where "www" is predicted, but the class is actually "Db." A higher entropy indicates that the distributions are closer to uniform and less sharply peaked. A distribution with a lower

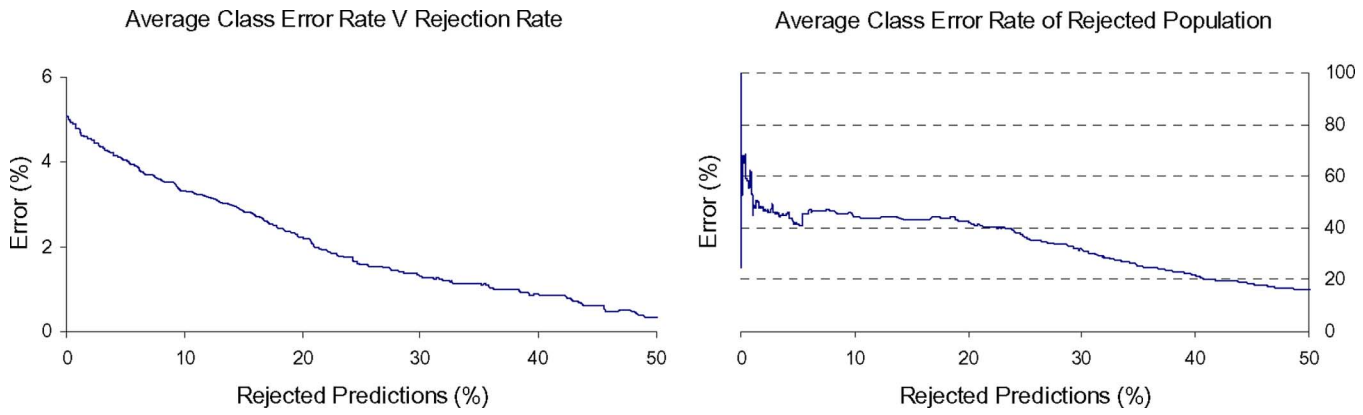

Fig. 6. Average class error rates versus rejected predictions, based on entropy.

entropy is one where the network is more "sure" of the prediction it is making. It is pleasing to observe that when the network makes a false classification it is less sure of that guess. Indeed, the average on diagonal elements is 0.15, much lower than the value of 0.85 for off diagonal entropies for misclassified flows.[7]

In a classification situation in which false predictions are to be minimized, the entropy of the distribution can be used to reject a prediction. Given a flow, we may have a maximum value of the distribution of the classes $\{p_i\}$ but given a high entropy of $\{p_i\}$, we can discard that prediction and infer that the classifier is unable to predict the class accurately for that flow. An experiment was conducted in which the percentage of prediction rejections (based on entropy) was varied, and the accuracy recorded. Again, we used the network trained on equal numbers of the classes from Traces 1 and 2 and tested on these traces, using average class error as our performance metric. Fig. 6 shows how the error changes with the percentage of rejections allowed, along with the error among the population of rejected predictions. The error rate falls as the rate of rejections increases. Also, the proportion of errors amongst those predictions rejected is much higher then the classifier error rate (typically 40% versus around 4%), demonstrating that, on average, we are discarding much worse quality predictions. These results confirm the extra information conveyed by the likelihood distribution, and show that this information has utility.[8]

E. Weakness

The posterior function of the weights (9) is very complicated. For a network with 100 inputs, ten nodes in the hidden layer, and ten outputs, there are 1120 weights. The conjugate gradient MemSys algorithm is, therefore, maximizing a function in over 1000 dimensions. It is a credit to the power of the software that this can be achieved, but for larger training sets (above 50 000 flows), where the posterior is very complicated (a term is included that arises from the product of each of the training data), the algorithm may not converge to the Bayesian stopping criteria.

[7]The entropy of these distributions must lie between 0 (delta function) and $\log(10) \simeq 2.3$ (uniform distribution).

[8]Curves were also plotted using the maximum value of the likelihood distribution as a criterion. The shapes of the curves were very similar, as expected since the entropy of these distributions will be closely related to the height of the peak.

It can be shown [39] that the maximum value of the posterior evidence for the parameter α in the prior is achieved when a quantity Ω is unity where Ω is defined as

$$\Omega = G/\left(-2\alpha S(\mathbf{w})\right)$$

where G is a quantity related to the "good degrees of freedom" of the likelihood L, evaluated on the training data. As discussed in Section V-A, the algorithm works by holding α initially at a high level, calculating the MAP of the marginalized posterior over the weights, and reducing α, until Ω increases to one.[9] However, for the larger training sets, Ω sometimes settles down to a value less than one.[10] When this occurs, we use the values of the weights which maximized the posterior for the value of this α, thus obtained. Maxent and Bayesian theorem tells us that this may not be the most probable value of α to use. However, the value of α is a parameter in the prior which is related to the variance of the distribution. When Ω is less than one, α is higher than the theoretical optimal value, suggesting that the weights will be constrained to values too close to zero.[11] Since the values of Ω obtained when the algorithm fails in this way are of the order of magnitude of the desired value (typically 0.5 compared with 1), this is not a catastrophic failure; we have indeed obtained good results. However, the mathematics is warning us that there is a better value for α and hence for the weights, if we had an algorithm powerful enough to find it.

We show how Ω varies with the number of iterations of the algorithm, along with the classification accuracy, in the case in which the net converges (Fig. 7) and when it does not (Fig. 8). High accuracies are achieved before Ω reaches unity in the case of convergence, and later iterations of the algorithm $(\Omega > 0.5)$ increase the accuracy only marginally. In the case where convergence does not occur, the accuracy ceases to increase before Ω reaches its maximum of just above 0.6. This suggests the accuracy would not increase significantly if a Bayesian Ω were to be achieved. Otherwise, there could be evidence in favor of using smaller sets of training data to ensure the algorithm did converge. However, the experiments conducted in Section VII-A

[9]In practice, the algorithm is halted when $\Omega > 0.9$.

[10]For all the experiments in the paper, this value was above 0.25 and most were above 0.5.

[11]The prior is uniform for zero α and sharply peaked around the origin for large α.

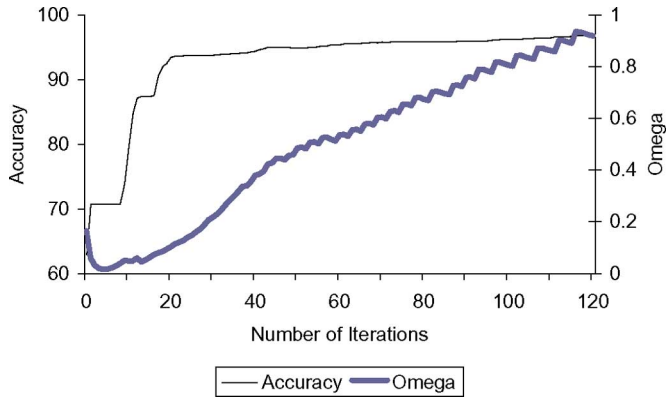

Fig. 7. Ω and the accuracy changing with successive iterations in the case where Ω achieves unity.

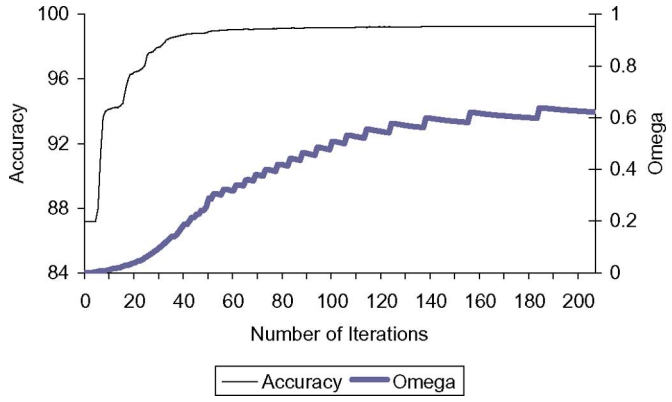

Fig. 8. Ω and the accuracy changing with successive iterations when the network does not converge to Bayesian Ω.

did show that accuracy increased with training set size, and this does not depend on whether Bayesian Ω is achieved.

VIII. Note on the Method and Results

In this paper, we have applied a technique, suitable for network traffic classification on real networks, in situations where the IP (host) address and application port number are not available. We have shown that the classification neural network, with Bayesian inferred weights, is an accurate classifier for this problem, and is superior to the naive Bayesian method of [2]. We draw attention to several points which will be of interest in applying the technique to a product which could classify network traffic in a wider range of environments.

First, the accuracy of the scheme was recorded as 99% when training and testing on data from the same day. This figure drops to 95% when testing on traffic eight months later. It is reasonable to suppose that the accuracy will increase as the dates approach each other, since the training data is then more similar to the testing data.

Second, the use of more training data could improve the accuracy. Table V shows that the accuracy increases with the size of the training data, with no leveling off for any of the sizes of set available to us. The benefits of higher accuracy with larger training data would have to be considered against the costs of longer training times. Training times of up to 39 h were recorded for our largest set using a single 2.2-GHz processor. If the set was increased in size by an order of magnitude, the neural network training time would take longer than the manual classification process and become the limiting factor, at least when using the 2.2-GHz processor. Also, larger sets would require several gigabytes of RAM; each flow with 246 features requires roughly 1 kB.

Third, the classification scheme has only been verified as successful on a single network, and when training and testing within that network. We make no claims about the stability of the composition of network traffic, or the difference of traffic between sites, but it is reasonable to suppose that new types of flows will arise over time. It would be interesting to test the networks trained in this paper on categorized traffic from other sites and from further time periods. Indeed, this technique could give useful insight into the evolution of network traffic composition, and differences between sites.

Finally, the reduction of the feature set requires comment. The step of choosing predictive inputs (Section VII-C) by inspection of the weights was somewhat ad-hoc. This process could be made more automatic by the use of a different prior on the weights. Automatic relevancy detection (ARD) priors have been used to infer weights of neural networks for prediction problems [40], [41]. These priors are Gaussian priors of the form $\exp(-\alpha_i \mathbf{w}_i^2)$ where the α's are allowed to vary for each weight (or subsets of weights). In training, weights which are relevant are identified via the corresponding α, and the others are switched off. As discussed, the reduction in inputs will speed up computation of the feature set, making a packaged product quicker and more efficient. Despite being of mathematical interest, we do not believe that using the ARD approach would increase classification accuracy. It would, however, provide a rigorous and efficient method for reducing the feature space which would cut down the time taken to classify and process the traffic and train the network.

IX. Conclusion

This paper has demonstrated the successful application of a Bayesian trained neural network to Internet classification on data from a single site for two days, eight months apart. Our main findings are as follows.

- A sophisticated Bayesian trained neural network is able to classify flows, based on header-derived statistics and no port or host (IP address) information, with up to 99% accuracy for data trained and tested on the same day, and 95% accuracy for data trained and tested eight months apart. Further, the neural network produces a probability distribution over the classes for a given flow. The entropy of this distribution is (negatively) correlated with prediction accuracy, and can be used as a rejection criteria for the predictions. Accuracy is further improved when a proportion of predictions may be rejected. The accuracy values significantly improve upon those from a naive Bayesian method [2] and compare favorably with the 50%–70% figure reported using the IANA port list [7]. By providing high accuracies without access to packet payloads or sophisticated traffic processing this technique offers good results as a low-overhead method with potential for real-time implementation.

- A wider ranging insight from our work is the comparison of the *best quality* features for either the naive Bayesian or the Bayesian neural network classification methods. A small number of certain features carry high significance regardless of the classification scheme. There is some overlap in features of high importance to either method, although the ordering of relative importance has changed between methods. A clear opportunity exists for further study of the traffic features.
- Finally, features derived from packet headers, when treated as interdependent can provide an effective method for the identification of network-based applications. This final conclusion cannot be overstated; the premise of our work was that the activity of a network user (in terms of applications) was reversible without the benefit of IP (host) address or port information. We consider that our work, alongside that of others (e.g., [9] and [24]), shows that, even with the removal of port and IP address, data-anonymization may not hide the application in use.

Future Work: There are a number of areas in which future work would further confirm the suitability of this technique and potentially.

An evaluation of our approach on further sources of classified data from this and other sites will give insight into the stability of the technique, and also the diversity and structure of Internet traffic itself. We anticipate that trained neural networks have a "half life" property, so that the classification accuracy declines over time as the composition of Internet traffic changes. Testing on data from later times will give an indication of the retraining interval, and of the robustness of the classifiers over very long periods.

We acknowledge that the *half life* property of any classification model implies that a suitable enhancement may be to combine attributes of supervised and unsupervised training. Such an approach would be particularly suitable when faced with a new form of traffic not available for hand classification. Such a method may combine the advantages of the present method with the possibilities of the approach raised by McGregor *et al.* [10]. The potential offered by the fusion of classification methods is a conspicuous area for future work.

We have highlighted the difference between feature importance for different classification methods. Further work will provide a valuable insight into the interaction of classification process with features. Also, as was noted in Section III, the definition of each object may significantly influence the classification process; the study of this interaction should provide a further valuable contribution.

This paper has used the Bayesian neural network approach for offline classification, but we believe this process can be refined through the selection of optimum features and appropriate algorithmic optimization. Specific issues of implementation and a study of the relevant optimization space is for further work in this topic.

Finally, we recognize a useful development in formalizing the identification of the predictive descriptors from the whole set of features. Training using ARD priors on the weights could be performed, where the inputs with little or no information are automatically switched off.

ACKNOWLEDGMENT

The authors would like to thank the anonymous referees and the Editor-in-Chief M. Polycarpou for their useful and illuminating comments, D. Zuev, N. Taft, J. Crowcroft, and D. Mackay for their instructive feedback and observations, G. Gibbs, T. Granger, I. Pratt, C. Kreibich, and S. Ostermann for their assistance in gathering the traces and providing tools for trace processing, J. Skilling for all his hard work in writing MemSys, and A. Garrett and R. Neill for proofreading and editing this paper.

REFERENCES

[1] Y. Zhang, M. Roughan, C. Lund, and D. Donoho, "An information-theoretic approach to traffic matrix estimation," in *Proc. ACM SIGCOMM*, Karlsruhe, Germany, Aug. 2003, pp. 301–312.

[2] A. W. Moore and D. Zuev, "Internet traffic classification using Bayesian analysis techniques," in *Proc. ACM Sigmetrics*, 2005, pp. 50–60.

[3] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," in *Proc. ACM SIGCOMM*, 2005, pp. 217–228.

[4] D. Moore, K. Keys, R. Koga, E. Lagache, and K. C. Claffy, "CoralReef software suite as a tool for system and network administrators," in *Proc. LISA 2001 15th Syst. Adm. Conf.*, Dec. 2001, pp. 133–144.

[5] C. Logg and L. Cottrell, Characterization of the Traffic Between SLAC and the Internet (July 2003) [Online]. Available: http://www.slac.stanford.edu/comp/net/slac-netflow/html/SLAC-netflow.html

[6] T. Karagiannis, A. Broido, M. Faloutsos, and K. Claffy, "Transport layer identification of P2P traffic," in *Proc. Internet Meas. Conf.*, Sicily, Italy, Oct. 2004, pp. 121–134.

[7] A. W. Moore and D. Papagiannaki, "Toward the accurate identification of network applications," in *Proc. 6th Passive Active Meas. Workshop (PAM)*, Mar. 2005, vol. 3431, pp. 41–54.

[8] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "Blinc: Multilevel traffic classification in the dark," in *Proc. ACM SIGCOMM*, 2005, pp. 229–240.

[9] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-service mapping for QoS: A statistical signature-based approach to IP traffic classification," in *ACM SIGCOMM Internet Meas. Conf.*, Sicily, Italy, 2004, pp. 135–148.

[10] A. McGregor, M. Hall, P. Lorier, and J. Brunskill, "Flow clustering using machine learning techniques," in *Proc. 5th Passive Active Meas. Workshop (PAM)*, Apr. 2004, vol. 3015, pp. 205–214.

[11] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *30th Annu. IEEE Conf. Local Comput. Netw. (LCN 30)*, Sydney, Australia, Nov. 2005, pp. 220–227.

[12] F. Hernández-Campos, A. B. Nobel, F. D. Smith, and K. Jeffay, "Statistical clustering of internet communication patterns," in *Proc. 35th Symp. Interface Comput. Sci. Stat., Comput. Sci. Stat.*, Jul. 2003, vol. 35, pp. 1–16.

[13] A. Soule, K. Salamatian, N. Taft, R. Emilion, and K. Papagiannaki, "Flow classification by histograms or how to go on Safari in the Internet," in *Proc. ACM Sigmetrics*, New York, NY, Jun. 2004, pp. 49–60.

[14] Y. Gu, A. McCallum, and D. Towsley, "Detecting anomalies in network traffic using maximum entropy estimation," in *Proc. Internet Meas. Conf.*, Berkeley, CA, Oct. 2005, pp. 345–350.

[15] K. Papagiannaki, N. Taft, Z. Zhang, and C. Diot, "Long-term forecasting of internet backbone traffic," *IEEE Trans. Neural Netw.*, vol. 16, no. 5, pp. 1110–1124, Sep. 2005.

[16] M. Mandjes, I. Saniee, and A. L. Stolyar, "Load characterization and anomaly detection for voice over IP traffic," *IEEE Trans. Neural Netw.*, vol. 16, no. 5, pp. 1019–1026, Sep. 2005.

[17] H. Hajji, "Statistical analysis of network traffic for adaptive faults detection," *IEEE Trans. Neural Netw.*, vol. 16, no. 5, pp. 1053–1063, Sep. 2005.

[18] K. Xu, Z.-L. Zhang, and S. Bhattacharyya, "Profiling internet backbone traffic: behavior models and applications," in *Proc. ACM SIGCOMM*, 2005, pp. 169–180.

[19] A. F. Atiya, M. A. Aly, and A. G. Parlos, "Sparse basis selection: new results and application to adaptive prediction of video source traffic," *IEEE Trans. Neural Netw.*, vol. 16, no. 5, pp. 1136–1146, Sep. 2005.

[20] L. Mé and C. Michel, Intrusion Detection: A Bibliography Supélec, Rennes, France, Tech. Rep. SSIR-2001-01, Sep. 2001.
[21] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur, "Bayesian event classification for intrusion detection," in *Proc. 19th Annu. Comput. Security Appl. Conf. (ACSAC)*, Las Vegas, NV, Dec. 2003, pp. 14–23.
[22] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Comput. Netw. (Amsterdam, The Netherlands)*, vol. 31, no. 23–24, pp. 2435–2463, 1999.
[23] M. Roesch, "Snort—lightweight intrusion detection for networks," in *USENIX 13th Syst. Adm. Conf. (LISA)*, Seattle, WA, 1999, pp. 229–238.
[24] A. W. Moore and D. Zuev, Discriminators for use in flow-based classification (2005), Intel Research Tech. Rep.
[25] A. Moore, J. Hall, C. Kreibich, E. Harris, and I. Pratt, "Architecture of a network monitor," in *Passive Active Meas. Workshop (PAM)*, La Jolla, CA, Apr. 2003, pp. 77–86.
[26] J. Padhye and S. Floyd, "Identifying the TCP behavior of web servers," in *Proc. SIGCOMM 2001*, San Diego, CA, Jun. 2001, pp. 287–298.
[27] J. Postel, Transmission Control Protocol (IETF), RFC 793, Sep. 1981.
[28] K. C. Claffy, H.-W. Braun, and G. C. Polyzos, "A parameterizable methodology for internet traffic flow profiling," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 8, pp. 1481–1494, Oct. 1995.
[29] N. G. Duffield, J. T. Lewis, N. O'Connell, R. Russell, and F. Toomey, "Entropy of ATM traffic streams," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 6, pp. 981–990, Aug. 1995.
[30] S. Ostermann, tcptrace (2003) [Online]. Available: http://www.tcptrace.org
[31] D. Zuev and A. W. Moore, "Traffic classification using a statistical approach," in *Proc. 6th Passive Active Meas. Workshop (PAM)*, Mar. 2005, vol. 3431, pp. 321–324.
[32] J. Hall, I. Pratt, and I. Leslie, "The effect of early packet loss on web page download times," in *Proc. Passive Active Meas. Workshop (PAM)*, Apr. 2003, pp. 159–170.
[33] M. P. Wand and M. C. Jones, *Kernel Smoothing*. London, U.K.: Chapman & Hall/CRC, 1994.
[34] L. Yu and H. Liu, "Feature selection for high-dimensional data: a fast correlation-based filter solution," in *Proc. 20th Int. Conf. Mach. Learn. (ICML)*, 2003, pp. 856–863.
[35] C. M. Bishop, *Neural Networks for Pattern Recognition*. London, U.K.: Oxford Univ. Press, 1995.
[36] D. J. C. MacKay, Bayesian methods for neural networks: Theory and applications (1997) [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/cpi_short.pdf
[37] S. F. Gull and J. Skilling, Quantified maximum entropy: Memsys5 users' manual (1999) [Online]. Available: http://www.maxent.co.uk/documents/MemSys5_manual.pdf
[38] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*. Chicago, IL: Univ. Illinois Press, 1949.
[39] S. F. Gull, "Developments in maximum entropy data analysis," in *Maximum Entropy and Bayesian Methods*, J. Skilling, Ed. Dordrecht, Germany: Kluwer, 1989, pp. 53–71.
[40] D. J. C. MacKay, "Bayesian non-linear modelling for the prediction competition," *ASHRAE Transactions*, vol. 100, pt. 2, pp. 1053–1062, 1994, Amer. Soc. Heating, Refrigeration, Air-conditioning Engineers.
[41] ——, "Probable networks and plausible predictions—a review of practical Bayesian methods for supervised neural networks," *Network: Comput. Neural Syst.*, vol. 6, pp. 469–505, 1995.

Tom Auld studied mathematics at Cambridge University, Cambridge, U.K., in 1994, and theoretical physics at Imperial College, London, U.K., in 1998. He returned to academia in 2004 to study for a doctoral degree at Cambridge University under Professor S. Gull's supervision. The title of his thesis is "Applications of Bayesian Neural Networks."

He most recently worked as a Derivatives Trader at an investment bank in London. He became interested in machine learning and neural networks due to their potential to model and predict market behavior.

Andrew W. Moore (M'86) received the B.S. and M.S. degrees from Monash University, Australia, in 1993 and 1995, respectively, and the Ph.D. degree from the University of Cambridge, Cambridge, U.K., in 2001.

He was recently appointed an Academic Fellow with the Department of Computer Science, Queen Mary, University of London, London, U.K. Prior to that, he was an Intel Research Fellow at the University of Cambridge Computer Laboratory. His interests include the identification of network-based applications and the related issues of network measurement-monitoring. He also works on the problems associated with data processing and storage and in bringing optical-switching to the system-area network.

Stephen F. Gull has been a member of the teaching staff at the Cavendish Laboratory, Cambridge University, Cambridge, U.K., since 1975. His research in radio astronomy brought him up against "the image processing problem": In 1978, he and G. Daniell published their joint paper on the maximum entropy method (MEM). During the next few years, he found himself applying MEM to a wide variety of physical problems, within both academia and also industry. In 1981, he and another radio astronomer J. Skilling, founded Maximum Entropy Data Consultants (MEDC) to exploit these opportunities. He is now a Professor of Physics at Cambridge University and a Fellow of St John's College, Cambridge, U.K., as well as Managing Director of MEDC.