Class-of-Service Mapping for QoS: A Statistical Signature-based Approach to IP Traffic Classification

Matthew Roughan[§] Subhabrata Sen[⋆] Oliver Spatscheck[⋆] Nick Duffield[⋆]

School of Mathematical Sciences, University of Adelaide,SA 5005, Australia[§]
AT&T Labs – Research, Florham Park, NJ 07932-0971, USA[⋆]

matthew.roughan@adelaide.edu.au {sen,spatscheck,duffield}@research.att.com

ABSTRACT

The ability to provide different Quality of Service (QoS) guarantees to traffic from different applications is a highly desired feature for many IP network operators, particularly for enterprise networks. Although various mechanisms exist for providing QoS in the network, QoS is yet to be widely deployed. We believe that a key factor holding back widespread QoS adoption is the absence of suitable methodologies/processes for appropriately mapping the traffic from different applications to different QoS classes. This is a challenging task, because many enterprise network operators who are interested in QoS do not know all the applications running on their network, and furthermore, over recent years port-based application classification has become problematic. We argue that measurement based automated Class of Service (CoS) mapping is an important practical problem that needs to be studied.

In this paper we describe the requirements and associated challenges, and outline a solution framework for measurement based classification of traffic for QoS based on statistical application signatures. In our approach the signatures are chosen in such as way as to make them insensitive to the particular application layer protocol, but rather to determine the way in which an application is used – for instance is it used interactively, or for bulk-data transport. The resulting application signature can then be used to derive the network layer signatures required to determine the CoS class for individual IP datagrams. Our evaluations using traffic traces from a variety of network locations, demonstrate the feasibility and potential of the approach.

Categories and Subject Descriptors

C.2.3 [**Computer-Communications Networks**]: Network Operations—*Network Management, Network Monitoring*

General Terms

Algorithms, Management, Measurement

Keywords

Statistical Signatures, Traffic Classification, Class of Service (CoS), Quality of Service (QoS)

1. INTRODUCTION

The past few years have witnessed a dramatic increase in the number and variety of applications running over the Internet and over enterprise IP networks. The spectrum includes interactive (e.g., telnet, instant messaging, games etc.), bulk data transfer (e.g., `ftp`, P2P file downloads), corporate (e.g., Lotus Notes, database transactions), and real-time applications (voice, video streaming, etc.), to name just a few.

Network operators (particularly in enterprise networks) are actively seeking the ability to support different levels of Quality of Service (QoS) for different types of applications. The need is driven by (i) the inherently different QoS requirements of different types of applications (e.g., low end-end delay for interactive applications, high throughput for file transfer applications etc.); (ii) the different relative importance of different applications to the enterprise: e.g., Oracle database transactions may be considered critical and therefore high priority, while traffic associated with browsing external web sites is generally less important; and (iii) the desire to optimize the usage of their existing network infrastructures under finite capacity and cost constraints, while ensuring good performance for important applications. In spite of a clear perceived need, and the fact that various mechanisms (diffserv, traffic prioritization, etc. [8, 17, 7]) have been developed for providing different service quality guarantees in the network, their adoption has not been widespread. A pertinent question then is: *what ails QoS?*

Realization of service differentiation capabilities requires association of the traffic with the different applications, determination of the QoS to be provided to each, and finally, mechanisms in the underlying network for providing the QoS. Based on interactions with large enterprise network operators, we believe that a key issue behind the slow spread of QoS-use is not the lack of interest or need, but rather, the absence of suitable mapping techniques that can aid operators in classifying the network traffic mix among the different QoS classes. We refer to this as the Class of Service (CoS) mapping problem, and hypothesize that solving this would go a long way in making the use of QoS more accessible to operators.

In principle the division into CoS could be done by end-points — for instance by end-user applications. However, for reasons of trust, and scalability of administration and management it is typically more practical to do this within the network, for instance at the router that connects the Local Area Network (LAN) to the Wide Area Network (WAN). Alternatively there might be appliances which sit near the LAN to WAN transition point performing packet marking for QoS.

CoS mapping inside the network is a non-trivial task. Ideally, a network system administrator would possess precise information on the applications running inside their network, along with simple, unambiguous mappings from easily obtained traffic measurements to applications (e.g. by port numbers, or source and destination IP addresses). This information is vital not just for the implementation of CoS (e.g. via diffserv), but also in planning the capacity required for each class, and balancing tradeoffs between cost and performance that might occur in choosing class allocations. For instance, one might have an application whose inclusion in a higher priority class is desirable, but not cost effective (based on traffic volumes and pricing), and so some difficult choices must be made. Good data is required for these to be informed choices.

However, in general, the required information is rarely up-to-date or complete, if it is available at all. The traditional ad-hoc growth of IP networks, the continuing rapid proliferation of new applications, the merger of companies with different networks, and the relative ease with which almost any user can add a new application to the traffic mix with no centralized registration are some factors contributing to this "knowledge gap". Furthermore, over recent years it has become harder to identify network applications within IP traffic. Traditional techniques such as port-based classification of applications have become much less accurate (details in Section 2).

This paper presents a signature-based traffic classification framework as a candidate solution for the CoS mapping problem, and demonstrates the feasibility and potential of the approach using large traffic traces collected from multiple Internet locations. Our classification method is based on utilizing the statistics of particular applications in order to form signatures. By choosing signatures that are determined by the way in which an application is used (e.g. is it used interactively, or for bulk data transport), we should obtain signatures that are insensitive to the particular application layer protocol. We can therefore perform CoS categorization without detailed knowledge of the specific application protocol, or usage case (some applications may be used for multiple tasks with different QoS requirements). The method would be used off-line to form a set of port, or IP address based rules for CoS assignment that would then be applied on-line in the QoS implementation. Our evaluations indicate that the technique has relatively low error rates. For example, our cross-validation tests using 3-Nearest Neighbor (details in Section 5) yield classification error of 2.5% and 5.1% respectively when the traffic is categorized into three and four classes, using two features. We found that even better results (for instance 0.0% errors were possible) with three features. The evaluations even showed relatively low error rates even for fine grain traffic classes. The last suggests that such statistical classification techniques may be good candidates for identifying even individual applications. Such identification of the different applications and their associated network traffic has a number of important usages for network operations and management (outside of QoS implementation), including application-specific traffic engineering, capacity planning, provisioning, and security.

The remainder of this paper is organized as follows. Section 2 overviews existing techniques for identifying IP traffic and their limitations. Section 3 presents a three-phase framework for realizing CoS mapping. Section 4 presents our techniques for CoS classification from network traffic. Section 5 presents evaluations of our techniques using real traffic traces. Finally, Section 6 concludes the paper.

2. IP TRAFFIC CLASSIFICATION

One approach commonly used for identifying applications on an IP network is to associate the observed traffic (using flow level data, or a packet sniffer) with an application based on TCP or UDP port numbers. We argue here that this method (described below) is inadequate.

The TCP/UDP port numbers are divided into three ranges: the Well Known Ports (0-1023), the Registered Ports (1024-49,151), and the Dynamic and/or Private ports (49,152-65,535). A typical TCP connection starts with a SYN/SYN–ACK/ACK handshake from a client to a server. The client addresses its initial SYN packet to the well known server port of a particular application. The source port number of the packet is typically chosen dynamically by the client. UDP uses ports similarly to TCP, though without connection semantics. All future packets in either a TCP or UDP session use the same pair of ports to identify the client and server side of the session. Therefore, in principle the TCP or UDP server port number can be used to identify the higher layer application, by simply identifying which port is the server port and mapping this port to an application using the IANA (Internet Assigned Numbers Authority) list of registered ports [20]. However, port-based application classification has limitations. First, the mapping from ports to applications is not always well defined. For instance,

- Many implementations of TCP use client ports in the registered port range. This might mistakenly classify the connection as belonging to the application associated with this port. Similarly, some applications (e.g. old `bind` versions), use port numbers from the well-known ports to identify the client site of a session.
- Ports are not defined with IANA for all applications, e.g. P2P applications such as Napster and Kazaa.
- An application may use ports other than its well-known ports to circumvent operating system access control restrictions, e.g., non-privileged users often run WWW servers on ports other than port 80, which is restricted to privileged users on most operating systems.
- There are some ambiguities in the port registrations, e.g. port 888 is used for CDDBP (CD Database Protocol) and access-builder.
- In some cases server ports are dynamically allocated as needed. For example, FTP allows the dynamic negotiation of the server port used for the data transfer. This server port is negotiated on an initial TCP connection which is established using the well-known FTP control port.
- The use of traffic control techniques like firewalls to block unauthorized, and/or unknown applications from using a network has spawned many work-arounds which make port based application authentication harder. For example port 80 is being used

by a variety of non-web applications to circumvent firewalls which do not filter port-80 traffic. In fact available implementations of IP over HTTP allow the tunneling of all applications through TCP port 80.

- Trojans and other security (e.g. DoS) attacks generate a large volume of bogus traffic which should not be associated with the applications of the port numbers those attacks use.

A second limitation of port-number based classification is that a port can be used by a single application to transmit traffic with different QoS requirements. For example, (i) Lotus Notes transmits both email and database transaction traffic over the same ports, and (ii) `scp` (secure copy), a file transfer protocol, runs over `ssh` (secure shell), which is also used interactively on the same port (TCP port 22). This use of the same port for traffic requiring different QoS requirements is quite legitimate, and yet a good classification must separate different use cases for the same application. In practice, one use case may dominate on a particular VPN (Virtual Private Network), or use cases will have other discriminating factors such as the servers' IP addresses. Thus a clean QoS implementation is still possible through augmenting the classification rules to include IP address-based disambiguation. Server lists exist in some networks, but again, in practice these are often incomplete, or a single server could be used to support a variety of different types of traffic, so we must combine port and IP address rules.

A possible alternative to port based classification is to use a painstaking process involving installation of packet sniffers and parsing packets for application-level information to identify the application class of each individual TCP connection or UDP session. However, this approach cannot be used with more easily collected flow level data, and its collection is computationally expensive, limiting its application to lower bandwidth links. Also this approach requires precise prior knowledge of applications and their packet formats – something that may not always be possible. We use it here to identify Kazaa and Gnutella traffic from their application layer protocols, but this requires substantial effort for each application, and in some cases, each application version. Furthermore, the introduction of payload encryption is increasingly limiting our ability to see inside packets for this type of information.

In this paper we explore the potential of signatures derived from measured traffic for CoS categorization. In practice, this approach, in conjunction with the above techniques and the partial knowledge available for most corporate networks can be used to bear on the problem of application identification and traffic classification.

2.1 Related work

Previous related work has examined the variation of flow characteristics according to application. Claffy [10] investigated the joint distribution of flow duration and number of packets, and its variation with flow parameters such as inter-packet timeout. Differences were observed between the support of the distributions of some application protocols, although overlap was clearly present between some applications. Most notably, the support of the distribution of DNS transactions had almost no overlap with that of other applications considered. The use of such distributions as a discriminator between different application *types* was not considered. There exists a wealth of other research on characterizing and modeling workloads for particular applications, e.g., [22, 31, 4, 2, 9, 34]. An early work in this space, [29], examines the distributions of flow bytes and packets for a number of different applications. Interflow and intraflow statistics are another possible dimension along which application types may be distinguished. [30] found that user initiated events—such as `telnet` packets within flows or `ftp-data` connection arrivals—can be described well by a Poisson process, whereas other connection arrivals deviate considerably from Poisson.

All these studies assume that one can identify the application traffic unambiguously and then obtain statistics for that application. In contrast, we are considering the dual problem of inferring the application from the traffic statistics. This type of approach has been suggested in very limited contexts such as identifying chat traffic [12]. Our work extends this idea while providing a rigorous mathematical framework for performing classification based on signatures. Signature-based detection techniques have also been explored in the context of network security, attack and anomaly detection (e.g. [6, 5, 36, 26]) where one typically seeks to find a signature for an attack. However, we apply our classification techniques to identify everyday traffic.

3. REALIZING COS MAPPING

The constraints under which CoS mapping must operate are principally related to computational complexity. At high speeds, the rules that can be used for this task are rather limited. Typically, one can use simple criteria such as port numbers or IP source/destination addresses, but not details from the higher layer protocols. Our task is to use statistical signature based classification (trained on prior networks' traffic, and applied to traffic measurements from the current VPN) to form a set of local rules based on port, or IP addresses, which would then be applied on-line for CoS assignment.

We propose to realize this type of CoS mapping using a three stage process.

1. statistics collection,
2. classification,
3. rule creation.

The first stage — statistics collection — involves placing monitors in the network, and collecting appropriate statistics of the traffic from certain aggregates. In this case, the aggregates we consider are the server port P_i, and server IP address[1] I_i of a connection i. One could choose these aggregates in a more flexible manner, with the proviso that the aggregates used must be easily implementable as rules (see below). There are a number of techniques available for efficiently identifying the interesting aggregates, for instance, see [13]. We then form a vector of statistics $\mathbf{S}^C(i)$ for each connection i, and use this to update the statistics of each aggregate that connection is involved in, for instance statistics $\mathbf{S}^P(P_i)$ for port aggregates, and $\mathbf{S}^I(I_i)$ for server aggregates.

[1]Notice that server address are usually statically allocated, rather than allocated via DHCP, and so we do not need to worry about constantly shifting IP addresses, at least on the time scales of measurements considered here.

For statistics collected on TCP connections, the procedure would be:

```
 foreach packet
 if new TCP connection (give it index $i$++)
   determine the aggregates for $i$
     server port $P_i$ = dst port of SYN
     server IP address $I_i$ = dst IP of SYN
     ...
   initialize a set of statistics $\mathbf{S}^C(i)$
 else if part of existing TCP connection $i$
   update connection statistics $\mathbf{S}^C(i)$
 else if end TCP connection $i$
   update connection statistics $\mathbf{S}^C(i)$
   update statistics foreach aggregate
     by server port: $\mathbf{S}^P(P_i)$
     by server IP address: $\mathbf{S}^I(I_i)$
```

The update procedure for connections depends on the statistic in question. Ideally, we should choose statistics that can be updated on-line in a streaming fashion, i.e. recursively. This means that we do not need to store data per packet, but rather per connection, for instance, assuming an update algorithm like

$$S_k^C(i) \leftarrow f(X_j^i, S_k^C(i), \phi(i)),$$

where X_j^i is the measurement for the jth packet in connection i, and $S_k^C(i)$ is the kth statistic for connection i, and $\phi(i)$ is some (small) set of state information (e.g. the packet number j) for connection i, then the memory required to store the state depends on the number of connections, not number of packets. For example, for a series of real-valued data X_j, the following statistics may be easily computed recursively:

1. **average:**

$$\bar{X}_{j+1} = \frac{1}{j+1} X_{j+1} + \frac{j}{j+1} \bar{X}_j,$$

2. **variance:**

$$\text{var}(\mathbf{X}_{j+1}) = \frac{1}{j} X_{j+1} + \frac{j-1}{j} \text{var}(\mathbf{X}_j) + \frac{j}{j-1} \bar{X}_j^2 - \frac{j+1}{j} \bar{X}_{j+1}^2.$$

where $\bar{X}_j$ and $\text{var}(\mathbf{X}_j)$ are the mean and variance, respectively, of the first j samples of data. However, even for more difficult statistics, such as quantiles, there are a number of approximation algorithms [18] that can be used to approximate the statistic on-line. The variables X_j could represent packet size, or inter-arrival time, or other features, and so we can generate a moderately large number of statistics even with the limitation of on-line computation. Some statistics need only be computed at the start and end of the TCP connection — for instance, the duration, which we may compute by including the start time of connection i in the state variables $\phi(i)$.

Likewise, it is appealing to be able to update the statistics of each aggregate recursively, but this is not necessary, as it is much easier to store one set of statistics per connection than per packet. If the statistics for each connection are stored, then we could alternatively compute the statistics per aggregate off-line, after the data collection.

We may also finalize any extant TCP connections at the end of data collection in one of two ways: by including them in our statistics, or excluding them. Either approach biases the results — for instance, if we exclude the connections we naturally exclude any connections longer than our measurement interval, but if we include them we underestimate the duration of the connections. These edge effects will be minimized by having a longer data collection interval, so in this work we propose using one day worth of data, though it may be practical to use longer data sets. Ideally, the collection intervals for training data should be the same as those for test data, so that both data sets are subject to the same biases.

After statistics collection, unsurprisingly, one has a collection of statistics indexed by aggregate (in our case server port, and server IP address). The next step is to classify the traffic on each aggregate. We do so using the classification algorithms described in Section 4 (or alternative algorithms if these are shown to be more accurate), in conjunction with a pre-existing set of training data, carefully collected on well understood networks of the type under study.

Once we have completed classification we will have associated each aggregate with a class. Assuming the classes map directly to CoS we can immediately construct the rules required in the QoS implementation. That is, assume that aggregate A_i has been determined to belong to class C_j, which requires QoS Q_k. We now have a mapping

$$A_i \rightarrow C_j \rightarrow Q_k,$$

which can be instantiated as a rule. For instance, if $A_i \equiv P_i$ corresponds to a particular server port P_i, our rule is

```
place traffic to/from port $P_i$ in class $Q_k$.
```

One can obviously create a large set of such rules, and in general it might be non-trivial to reduce the size of this rule set to something manageable. However, in practice, differential pricing between classes means that only traffic specifically requiring high priority should go into high priority classes, and so the majority of application will most likely be placed in a lower priority class. Hence there are typically only a few classes, and the majority of aggregates will go into a default class, so few rules will be needed.

Once a set of rules has been created, these would then be implemented on (for example) the access router, which would use them to mark the packets appropriately, and place them in appropriate queues for forwarding. Thus the classification process described so far is an off-line process, to be applied before the fact to create a set of simple rules that would be used in the actual on-line QoS implementation. Note that the error rate of the classification algorithm is in forming these rules, not in classification of packets in the actual QoS implementation. This is an important distinction: the error rate does result in packets being placed in suboptimal classes, however, these packets only pay for the class in which they are placed. This is not the same, for example, as paying a business class airfare and being bumped to economy. It is simply a case of an administrator occasionally booking the wrong class of ticket because he has incorrectly categorized the importance of some of his administratees.

Furthermore, the above process (as described) is automated. However, it is desirable for a human to "double-check" the assignment rules. The method described here cannot be made 100% foolproof simply because a particular network operator may have specific requirements which deviate from the norm. A human would be able to map the classes/rules to known traffic and servers, and ensure that the known policy based rules were enforced in priority to those derived above. Think of this as our disgruntled business class passenger updating his administrator's list of important people, so that he receives the air transport he deserves in the future. In essence, the technique above is intended to fill in the unknown gaps in the best possible way.

A side benefit of the above approach is that the collected statistics can be used in other ways. For instance, the traffic volume per aggregate could be useful in planning the required capacity for each QoS class, or the network in general. In fact, given a set of prices per class, the measured volumes, and a set of utilities per application class, one could create the mapping from the application class C_j to QoS class Q_k through an optimization process using traffic data, rather than as a set of fixed rules.

Once monitors are installed, there is no reason one could not use them in an on-going manner. One could continue to make measurements (as above) of the statistics per aggregate, and if something changes significantly, then one could change the rules used. An example might be the introduction of a new application with different QoS characteristics, requiring a rule update. It is not, however, envisioned that these updates would occur often.

4. COS CLASSIFICATION

We next present the different components in our classification approach: identifying different application classes, selecting candidate classification features, and finally specific classification methods.

4.1 Class Definitions

In practice, service differentiation mechanisms like diffserv today only allow a relatively small number of application classes. For our initial study, we focus on the following 4 broad application classes, commonly found in corporate networks.

- *Interactive:* The interactive class contains traffic which is required by a user to perform multiple real-time interactions with a remote system. This class includes such applications as remote login sessions or an interactive Web interface.
- *Bulk data transfer:* The bulk data transfer class contains traffic which is required to transfer large data volumes over the network without any realtime constraints. This class includes applications such as FTP, software updates, and music or video downloads (say through an application such as Kazaa).
- *Streaming:* The streaming class contains multimedia traffic with realtime constraints. This class includes such applications as streaming and video conferencing.
- *Transactional:* The transactional class contains traffic which is used in a small number of request response pairs which can be combined to represent a transaction. DNS, and Oracle transactions belong to this class.

The choices were motivated by the need to select a small number of classes that would be simple, intuitive and still adequately represent the different QoS requirements of commonly used applications. We view these 4 classes as a starting point, with the actual choice of application classes being a topic for research.

To characterize each application class we need a reference data set for each class from which we can extract a set of *representative* features. Acquiring such a reference data set is made difficult by precisely the problems described in Section 2. Selecting the network traffic based on port numbers may not yield reliable statistics that are representative of any particular class, however, to classify them otherwise requires a reference data set. To break this circular dependency we selected some applications per class, which based on their typical use, have a low likelihood of being contaminated by traffic belonging to another application class. In particular we focus on applications which:

- are clearly within one class (to avoid mixing the statistics from two classes);
- are widely used, so as to assure we get a good data-set;
- have server ports in the well known port range to reduce the chance of mis-use of these ports.

These reference applications will then be used to estimate a number of statistics, from which we will select features for use in CoS categorization. Based on the criterion established in the previous section, the reference applications selected for each application class are:

- *Interactive:* Telnet,
- *Bulk data:* FTP-data, Kazaa,
- *Streaming:* RealMedia streaming,
- *Transactional:* DNS, HTTPS.

We include HTTPS in the transactional class, because a large proportion of HTTPS interactions involve users filling out a form, for instance to conduct secure credit card purchases over the WWW. In our current study, we do not use web traffic to train the classification. However we do include WWW traffic statistics (as captured using port 80) in some plots as an example application, as it provides some interesting intuition into the results. Also, although Kazaa does not have registered ports, in particular data sets we have reliable ways of identifying Kazaa traffic (as describe below).

4.2 Candidate Features

The list of possible features one could consider is very large. We can broadly classify these into five categories:

1. Simple packet-level features such as mean packet size and various moments such as variance, RMS (root mean square) size, etc., are simple to compute, and can be gleaned directly from packet-level information. They offer a characterization of the application that is independent of the notion of flows, connections or other higher level aggregations. Another advantage is that packet-level sampling, which is widely used in network data collection, has little impact on these statistics. Other statistics that can be derived from simple packet data are time series, from which we could derive a number of features, for instance relating to correlations over time (e.g. parameters of long-range dependence such as the Hurst parameter). An example of this type of classification can be seen in [23], where the authors use time-of-day traffic profiles to categorize web sites.

2. Flow-level statistics are summary statistics at the grain of network flows. A *flow* is defined to be an unidirectional sequence of packets that have some field values in common, typically, the 5-tuple (source IP, destination IP, source port, destination port, IP Protocol type). Example *flow-level* features include mean flow duration, mean data volume per flow, mean number of packets per flow, and variance of these metrics, etc. There are some more complex forms of information one can also glean from flows (or packet data) statistics, for instance, we may look at the proportion of internal versus external traffic within a category – external traffic (traffic to the Internet) may have a lower priority within a corporate setting. These statistics can be obtained using flow-level data collected at routers using, e.g. Cisco *NetFlow* [27]. They do not require the more resource intensive process of finer grain packet-level traces. A limitation is that flow-collection may sometimes aggregate packets that belong to multiple application-level connections into a single flow, which would distort the flow-level features.

3. Connection-level statistics are required to trace some interesting behavior associated with connection oriented transport-level connections such as TCP connections. A typical TCP connection starts and ends with well defined handshakes from a client to a server. The collection process needs to track the connection state in order to collect connection level statistics. In addition to the features mentioned for the flow-level, other features that are meaningful to compute at the TCP connection level are the symmetry of a connection, advertised window sizes and the throughput distribution. The connection-level data generally provides better quality data than the flow-level information, but requires additional overhead, and would also be impacted by sampling or asymmetric routing at the collection point.

4. Intra-flow/connection features There are features we might wish to collect which are based on the notion of a flow or TCP connection, but require statistics about the packets within each flow. A simple example is the statistics of the inter-arrival times between packets in flows. This requires data collected at a packet level, but then grouped into flows. We use the relative variance of these inter-arrival times as a measure of the burstiness of a traffic stream. Intra-flow/connection features include loss rates, latencies, etc.

5. Multi-flow: Sometimes interesting characteristics can be captured only by considering statistics across multiple flows/connections. For instance, many peer-to-peer applications achieve the download of a large file by bulk downloads of smaller chunks from multiple machines — the individual chunk downloads are typically performed close together in time. For some multimedia streaming protocols, the high volume data connection is accompanied by a concurrent, separate connection between the same set of end-systems, containing low volume, intermittent control data (e.g. RTSP [32]). These *multi-flow* features are more complex and computationally more expensive to capture than flow or connection data alone.

Table 1 gives a summary of the types of features, and which types of measurements are needed to collect them. At the conclusion of our ongoing work we would like to have identified a minimal set of statistical features that can be used to classify traffic into application classes reliably and independently of the actual application.

4.3 Classification methods

In this paper we test two simple, but common methods for classification: Nearest Neighbor (NN) and Linear Discriminant Analysis (LDA). The general problem of classification is: given K classes, M features, and N training data points, we wish to determine a set of general rules for classifying future data on the basis of a feature vector. Each training data point consists of a pair $X_j \in \mathbb{R}^M, G_j \in \{1,\dots,K\}$, where X_j is the feature vector, and G_j is the class of the jth data point, and our rule should provide a mapping $\hat{G} : \mathbb{R}^M \to \{1,\dots,K\}$.

One very simple method of classification is NN. In this classification method we assign a new data point to the class of its nearest (in this paper we shall use Euclidean distance as the metric for distance) neighbor from the training data. That is we take $\hat{G}(X)$ to be the class G_i of the data point X_i which minimizes the distance $||X_i - X||$. NN methods can be generalized to k-NN (where the k nearest neighbors essentially 'vote' on the class of the observation), to enhance its robustness. k-NN methods are generally very good on low-dimensional data (small M), but are less effective on high-dimensional data, and give little insight into the most useful features.

An alternative set of approaches can be drawn from statistical decision theory, in which we choose $\hat{G}(X) = G_i$ if

$$\Pr(G_i|x = X) = \max_{g\in\{1,\dots,K\}} \Pr(g|x = X).$$

That is, we choose the class with maximal conditional probability, given the feature vector X. This approach is known as the *Bayes classifier*. In general we can call $\delta_g = \Pr(g|x = X)$ a discriminant function, and we choose the class with the maximum discriminant. A k-NN approach can be seen as an approximation to the Bayes classifier above, where we are approximating the probabilities $\Pr(g|x = X)$ by the proportion of the k nearest neighbors of class g.

There are many alternative approaches for estimating the discriminant functions. LDA [19] is a simple method that can be derived via a number of approaches. Perhaps the most famous derivation is Fisher's — he posed the problem "Find the linear combination $Z = a^T x$ such that the between class variance is maximized relative to the within-class variance". When one considers this linear combination it forms a natural discriminant between the classes, that minimizes the overlap between them. Another simple method of derivation of LDA is to assume that each class g has a Gaussian distribution with mean μ_g and with the same intra-class covariance Σ for each class. In this case the distribution of features vectors within each class can be drawn from

$$f_k(x) = \frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\right),$$

where μ_k is the within class mean (in the feature space) for class k. From Bayes rule we may derive

$$P(G = k|X = x) = \frac{f_k(x)\pi_i}{\sum_i f_i(x)\pi_i},$$

where π_i is the prior probability of class i. In comparing two classes i and j, it is sufficient to compare the log ratio for which the normalizing constants cancel. The log ratio is given by

$$\log\left(P(G = i|X = x)/P(G = j|X = x)\right).$$

When simplified, this results in the *linear discriminant functions*

$$\delta_g(x) = x^T\Sigma^{-1}\mu_g - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_g + \log\pi_g,$$

	features				
data source	packet level	flow-level	connection-level	intra-flow	multi-flow
packet trace	yes	yes	yes	yes	yes
sampled packets	yes	biased	no	biased	biased
flow-data	some	yes	no	no	yes
SNMP	no	no	no	no	no
server logs	some	some	some	no	some

Table 1: The table illustrates the types of features available from different measurement tools. Notice that in some cases it may be possible to determine some features from a data, but not an arbitrary feature, e.g. flow level data can be used to measure average packet sizes, but not the variance of packet sizes. Server logs can be used to collect some data as well, but typically only for a particular application: though such data is limited for classification purposes, such data is helpful in training, because of the high reliability with which we know the application. Also, in some cases, in particular with sampled packet data, the feature measurements will be biased, the details of such biases can be quite complex.

where the prior probabilities π_g of each class g are estimated using $\pi_g = N_g/N$ (N_g being the number of training data points in class g), and the means μ_g and the covariance Σ are estimated using standard estimators. For any data point X we then choose the class with the largest $\delta_g(X)$. LDA is so named because the decision boundaries thus formed will be linear. This method may be generalized in a number of ways, for instance by assuming that the covariance of each class is different – an approach that generates Quadratic Discriminant Analysis [19, p.88] (QDA), so called because the decision boundaries are now quadratic curves. For QDA the discriminant functions are almost identical, except that Σ is replaced by Σ_k the covariance of the data in class k. Alternatively, one may perform LDA on a set of quadratic features – that is, rather than only using features (A, B), one uses the features (A, B, A^2, B^2, AB). This generates results similar to QDA [19, p.88].

Typically, one estimates π_k as above, but the data used here is from a variety of networks, including the public Internet, while the QoS questions of interest are typically important for enterprise networks. On such, one would expect somewhat different proportions of each application. Hence we shall use uninformative (uniform) priors here. Given more data from different types of networks, we should be able to improve these priors, and thus the results.

There are countless generalizations, and alternatives to the methods presented here, from the literature on classification, pattern recognition, and machine learning. However, the above approaches are representative of a wide variety of possibilities.

5. RESULTS

We first describe the different traffic data sets used in our evaluations and then present our results.

5.1 Data Description

4 sets of data from different sources were used for this study:

- *Waikato trace*: The Waikato Applied Network Dynamics (WAND) group at the University of Waikato(`http://wand.cs.waikato.ac.nz/wand`), are responsible for developing very precise high-speed packet monitoring hardware. The WAND group have used this hardware to collect an extensive archive of packet header traces from various parts of the Internet in New Zealand, which they archive with NLANR `http://pma.nlanr.net/PMA/`. For this study we used a seven day long packet header traces drawn from the Auckland-IV data set [25], collected from the University of Auckland uplink from the 23rd to the 29th of March 2001. The trace itself is much longer (over six weeks), but also very large (65 GB containing over 3 billion IP headers), and so we used only a segment here.
- The second data source was application level session logs from a commercial streaming service [34]. A streaming session is initiated when a new request for a streaming object is received at a streaming node. The session terminates either when the client sends a termination request, or due to some error situation. At termination, a single entry is created in the log summarizing a range of information for that session. The fields in each log entry include: requesting IP address, particulars of requested resource, whether the file is a Real or Windows Media object, transport protocol used for streaming (TCP or UDP), total data transmitted, total packets, session end-time, total session time, status/error codes, etc. Over 185,000 sessions for RealMedia streaming objects, over a seven days period from Dec 13-19, 2001 were analyzed.
- *Gigascope trace 1*: The third data set was collect at a choke point in an access network on a T3 line from the 7th of May to the 16th of July, 2003, using a Gigascope probe [11]. The Gigascope was configured to collect TCP connection records. Each such TCP connection record contains the information found in a typical flow records such as IP addresses and port numbers. However, in addition a TCP connection record combines the flow records in both directions and maintains additional statistics such as peak throughput and latency for each direction of the TCP connection. Another difference to typical flow records (such as Netflow) is that the Gigascope maintains a state machine for each TCP connection it monitors and, therefore, only one TCP connection record is generated per TCP connection. Cisco Netflow artificially truncates some 'flows' by flushing them from the cache whenever a timeout is exceeded, memory is full, and at arbitrary (typically 30 minute) intervals. A consequence of this difference is that the maximum TCP connection record duration is longer than that for the artificially truncated flows collected by Netflow. In addition, Gigascope can use keyword based filtering to select applications more accurately than port based algorithms, and we use this ability to explicitly separate Kazaa and Gnutella traffic, from the other applications (which are then classified using port numbers). There was only a small amount of Gnutella traffic present, so we do not consider this here. Table 2 presents

connection-level statistics per application for this trace. Note that the numbers corresponding to domain (i.e., DNS) only consider DNS communications over TCP.

- *Gigascope trace 2:* The next data set is also collected using Gigascope (see above). However, in this case, the Gigascope was placed on the span (monitoring) port of the major edge router of AT&T Research's local network. Thus we see all of the traffic to and from the public Internet over a T3 connection, and also from our local network to other company sites via another T3 connection, and also a significant amount of internal LAN to LAN traffic.
  The reasons for collecting this set of data are twofold: primarily we wanted to gain some additional insight into the issues relating to traffic on an enterprise network more like those which to which we might wish to apply the techniques described here. Secondly, in the previous sets of data we do not have packet header traces for a large volume of streaming traffic. Thus we cannot compute arbitrary statistics of the data, for instance, the statistics of the inter-arrival times. Hence, with this traffic stream, we do not perform TCP reconstruction, but rather maintain complete packet header traces, so that we can measure the inter-arrival times between packets in a flow (as well as other statistics).
  The network in question consists of several hundred users, with a similar number of PCs. It was observed from the 17th of October until the 3rd of November 2003. The most noticeable differences in this traffic, as compared to the above sources are (1) a lack of much traffic resembling P2P applications (at least at the port level), and (2) FTP transactions were typically within either the local network, or to nearby locations. Thus their throughput was very high due to the low Round-Trip Times (RTTs), and hence they differed significantly in this respect from the FTP connections in the previous data set. Such differences may be more typical in enterprise settings, where a large amount of data transfers may be local, but one might expect at least some of the traffic to be non-local. The HTTP traffic on this network was mostly external, and so we shall use this for comparison in this one data set to see the impact the RTT has on our measurements.

- *Gigascope trace 3:* Our last data set is similar to the previous Gigascope trace, in its monitor location and setup. However, the data was measured over 13 days from the 13th until the 26th of April 2004. It should be noted that even this packet trace alone was 60 GB of data, representing about 150 million flows. We note though that such large data sets are only required for the exploratory analysis of the data, such as we perform here. In actual practice, as noted above, on-line techniques can be used collect statistics both for training, and other data, and thereby reducing the volume of data requiring storage by orders of magnitude.

5.2 Simple Classification results

As previously stated, the first step in our classification process is to identify the important features necessary to classify the traffic. To demonstrate the potential of our approach we evaluated the following easily obtainable features in this regard: the average packet size, flow duration, bytes per flow, packets per flow, and Root Mean Square (RMS) packet size. Of these, the pair of most value were the average packet size and flow duration, and we shall use those characteristics to classify the reference applications introduced in Section 4. We then identify the main limitations of these features and introduce a new additional feature to provide more resolution.

We have tested a number of approaches including k-NN, LDA, and QDA, on a number of different features (listed above), but the approaches found to be most illustrative of the possibilities here were k-NN, and LDA with quadratic variables, applied to the flow duration and average packets size. LDA with quadratic variables, though similar to QDA, seemed to produce slightly better results given the small amounts of data available to estimate within class covariances. Figures 1 (a) and (b) show scatter plots of data from seven separate one day long WAND traces. The plots show average packet size versus average flow duration for four applications: Telnet, DNS, FTP-data, and Streaming video, the first three from WAND, and the last from the streaming application logs, because there is very little streaming traffic within the WAND data. In Figure 1 (a) we also show LDA boundaries (solid lines) when the traffic is broken into four classes, while in Figure 1 (b) 1-NN boundaries are shown.

Figure 2 presents similar plots from the first data set collected by the Gigascope (each data-point represents averages over one day's data). In this case the connection records used in the plots show the average TCP connection duration, rather than the flow lengths. As noted above, the durations in this data can be significantly larger than those seen in Figure 1. In these plots, we use six applications for training the classification (dns, ftp-data, https, kazaa, realmedia, and telnet). We can do so in this data set (as compared with the previous data) because we can use Gigascope's advanced features to make an accurate classification of applications such as Kazaa. WWW traffic is not used in training the classification, but is included in the plot because the results are interesting. WWW traffic falls between the bulk-data and transactional classes — exactly as one might suspect. It also has some occasional departures into the streaming class — also a possibility for web traffic. Finally, Figure 3 shows similar plots to Figure 2, but for a seven-class breakup.

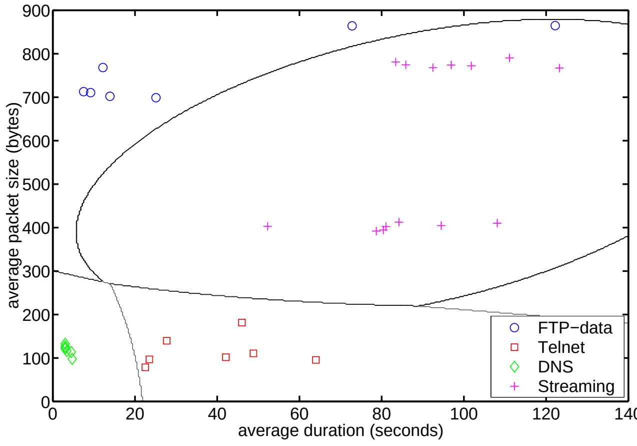
While the above plots suggest that the boundaries obtained are reasonable, they do not provide a quantitative measure of the quality of the classification. We next obtain such a measure using cross-validation [19]. To perform cross-validation we randomly divide the data into 10 blocks, and for each block use the other 9 blocks as training data, and then validate the classification on the block in question. We validate by comparing the true class of a measurement with that determined by the various algorithms above and report the results in Table 3. The table shows (in the 4 class column) that the errors are only 5.1% for the best method, 3-NN.

Interestingly in this data, analysis of the within-class variance of the measurements show that the variation decreases significantly as the number of connections increases. This is what one would expect if the measurements are statistically uncorrelated, and suggests that one could improve the accuracy of the results by simply increasing the sample sizes. Research is needed into how large a sample one requires for training and classification.

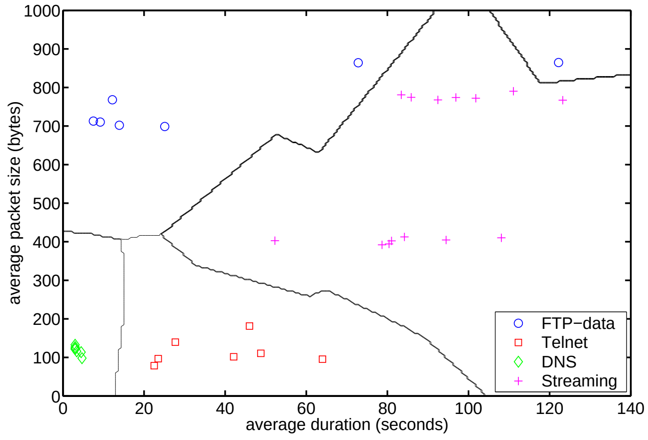
Although, we do not propose specific application classification as a general technique (it is not needed for QoS), it may be valuable in places, and so we also perform cross-validation for the problem of classifying applications. For the data in question, this leads to seven classes (domain, ftp-data, https, kazaa, realmedia, telnet, or www). One can also see from Table 3 that the errors are still not large, $<$ 10%, and once again 3-NN is the best method of those

Application	mean connections per day	duration (seconds)	packet size (bytes)
domain	3676	12.3 ± 25.6	224.0 ± 415.3
ftp-data	6105	89.8 ± 98.2	874.3 ± 219.0
https	155249	10.1 ± 3.1	404.0 ± 55.6
kazaa	1207491	65.1 ± 18.4	805.3 ± 42.8
realmedia	1212	175.1 ± 123.5	679.3 ± 131.6
telnet	8670	948.8 ± 716.4	148.7 ± 162.8
www	3905871	22.7 ± 38.6	651.1 ± 51.0

Table 2: Gigascope Trace 1 connection-level statistics: The duration and packet size is represented as mean value ± 1.96 times the corresponding standard deviation

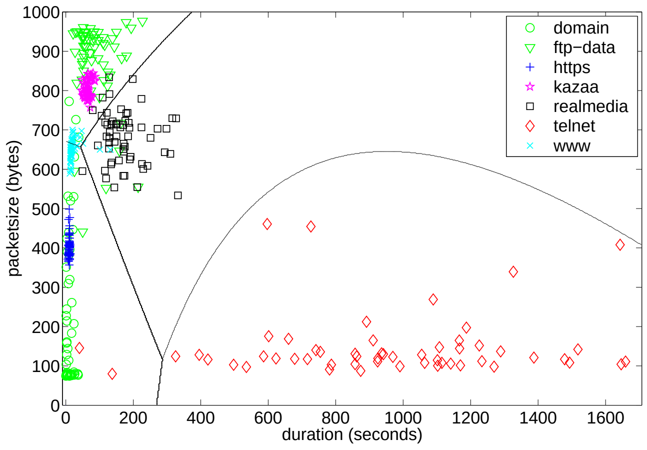


(a) LDA.

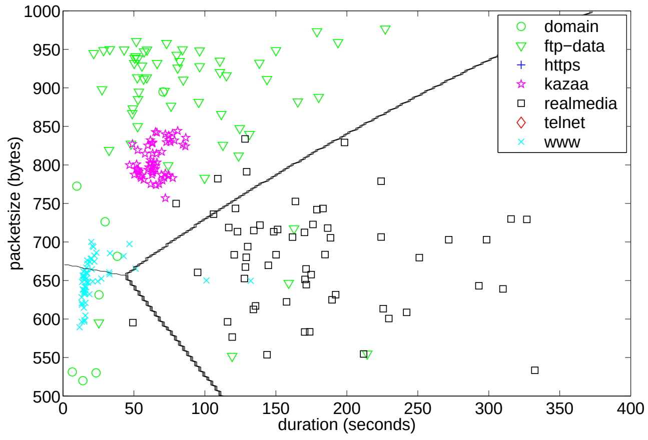


(b) Nearest Neighbor.

Figure 1: Four class breakup for the Waikato data.



(a) LDA.

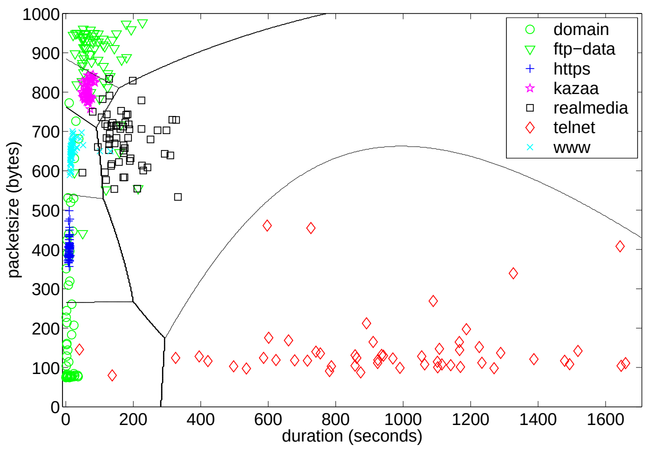


(b) Zoom into upper left region.

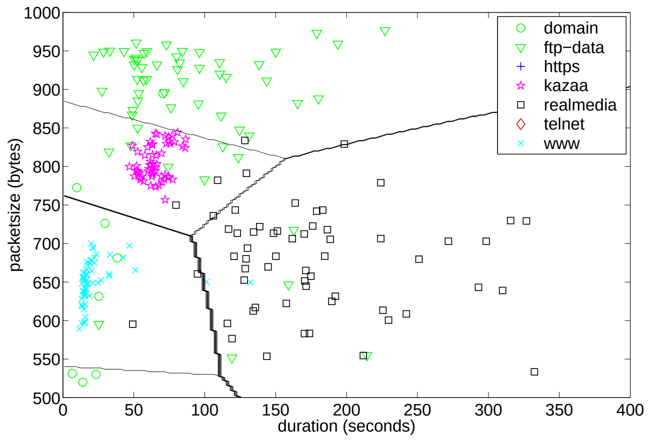
Figure 2: Four class breakup for the first Gigascope trace.

tried. Figure 3 shows the boundaries obtained corresponding to 7-class breakup using LDA.

While the above results show that we get fairly good classification, it may well be possible to do much better. For example, streaming video and bulk data transport have similar ranges of characteristics using the above features, and so are harder to discriminate. Table 3 shows results of a 3-class classification combining streaming and bulk-data transport classes, which has lower errors than the 4-class case. This may be because many streaming applications actually act much like bulk-data by filling their playout buffer as fast as possible. Therefore, simple statistics such as those used in our preliminary work are not ideal for discriminating the two. Perhaps, for many cases, streaming video does not warrant its own class, and could happily co-exist with bulk data. However, in



(a) LDA.



(b) Zoom into upper left region.

Figure 3: Seven class breakup for the first Gigascope trace.

algorithm	error rate		
	4 class	3 class	7 class
LDA	5.6 %	3.4 %	10.9 %
1-NN	7.9 %	3.4 %	12.6 %
3-NN	5.1 %	2.5 %	9.4 %
5-NN	5.6 %	2.5 %	9.9 %
7-NN	5.6 %	2.8 %	9.7 %
15-NN	6.2 %	3.4 %	11.4 %

Table 3: The cross-validation results.

practice, it is often desirable to prioritize one over the other, and so we need to do further investigation to see if other statistics make it possible to discriminate these classes more clearly. We shall explore this further in Section 5.3.

A pertinent question is: Can we discriminate classes without any flow level statistics, using only packet level statistics? The packet size alone is not sufficient to discriminate Telnet type applications from DNS type applications. Figure 4 shows the results of classification (using the WAND data) based on the packet size and RMS packet size. For a particular application there is a strong linear relationship between the two variables. However, the relationship appears to differ for the different applications, which leads to the ability to distinguish between DNS, and Telnet traffic in a way that would not be possible using one variable alone. Figure 4 shows an LDA classification of FTP-data, DNS, and Telnet traffic. Note however, that it would be impossible to distinguish FTP-data and WWW traffic on the basis of these features, so we can see that while packet level statistics can be useful, it is more practical to include features from higher level aggregations of the traffic as well.

5.3 Streaming vs data

As noted previously, the features so far considered were least effective in discriminating streaming traffic from bulk-data traffic. However, one might suspect that streaming traffic, which involves continuous transfer of data over some period, would have a more regular traffic pattern than data transfers. When transferring data, the typical goal is to move the data as quickly as possible, and the limiting factor is typically the TCP congestion control [21, 1]. The properties of this congestion control have been frequently studied; for a far from complete set of studies see [28, 24, 16, 14, 3]. It is well known (see, for instance, [15]) that TCP's congestion control, combined with ACK compression can induce additional burstiness in traffic. Hence, we might expect that in comparison streaming traffic would appear less bursty at the packet arrival level than bulk-data traffic (at the byte level Variable Bit Rate (VBR) video traffic may also show a large degree of burstiness because of the nature of the information content, and VBR video codecs, but this burstiness appears in the packet sizes, not the inter-arrival times).

Of course, one must note that much streaming traffic is carried over TCP to avoid firewalls that block UDP traffic (for instance on our LAN in particular), and so there may be some interactions between the video codec (which may use adaptive rates to try to get the best performance) and TCP, which may limit the transmission

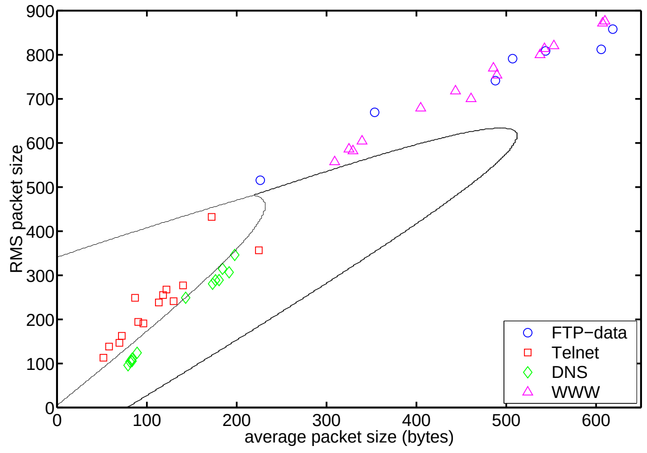


Figure 4: Classification of FTP-data, DNS, and Telnet traffic. WWW traffic is included here only to show that it would be hard to distinguish this traffic from FTP-data based purely on the packet level features.

rate during congestion. Hence, the relative burstiness may not be a cut-and-dried metric for classification, and so we test this conjecture using the second data set collected using Gigascope trace 2.

The data set is a packet header trace over multiple days. We form flows from the packets (using the standard five-tuple formed from protocol and source and destination IP addresses and ports), and within each flow compute both the mean and standard deviation of the inter-arrival times. We then take the ratio of these to give us a measure of the relative variation in inter-arrival times for the flow. The feature we consider here is the average of this ratio for a particular traffic aggregate. More precisely, take a set of flows $\mathcal{F} = \{f_i\}$, where flow f_i consists of n_i packets arriving at times t_{ij} for $j = 1, \ldots, n_i$. We compute the mean μ_i and standard deviation σ_i of a flow's inter-arrival times using the standard estimators

$$\mu_i = \frac{t_{in_i} - t_{i1}}{n_i - 1}, \tag{1}$$

$$\sigma_i^2 = \frac{1}{n_i - 2} \sum_{j=1}^{n_i - 1} \left[(t_{ij+1} - t_{ij}) - \mu_i \right]^2 . \tag{2}$$

Note in the above that we can only make computations of the variance of the inter-arrival times in a flow where that flow has at least three packets. The ratio r_i is given by $r_i = \sigma_i / \mu_i$, and the feature of interest is the average ratio, given by

$$E[r] = \frac{1}{N} \sum_{i=1}^{N} r_i, \tag{3}$$

where there are N flows during a given day. We shall refer to $E[r]$ as the *inter-arrival variability metric*.

In fact we needed to modify the above scheme slightly. We found, in examining traces of streaming traffic that while the average behavior was fairly regular (most of the time), in many cases the streaming traffic ended with a long (20-40 second) gap, followed by a few (2-7) packets. Figure 5 shows two examples of packet traces from streaming traffic (the data flow from server to client is shown in the upper graphs, and the ACK flows from client to server are shown in the lower graphs). The figures show *dot-strip plots*, first used in [33], and more recently used to good effect to show packet traces in [35]. The plot shows the time (in seconds) of a packet arrival along the x-axis, and to make the timings of individual packets more obvious, displays the milliseconds along the y-axis. One can observe the regularity of the streaming traffic in the two cases displayed as regular patterns in the plots. These are not completely regular, but vary at the individual packet level, and also in regions. However, the point to note here is the large gap at the end of the trace before the final few packets. This is a protocol related effect, and so not of such interest here, and it can be easily ignored by removing the final 10 packets from each flow. This has the impact of preventing us from considering flows without 12 packets, but this is not really an issue for the problem of distinguishing streaming from bulk-data, because such short flows would generate highly variable statistics in any case.

Figure 6 shows the results for Gigascope trace 3. The x-axis shows the inter-arrival variability metric described above, and the y-axis shows the average packet size. In Figure 6 (a) we make no attempt to distinguish flows of ACKs from data flows, whereas in Figure 6 (b) we separate these, and only compute statistics for the data flows (ACK packets will typically be very short and so reduce the average packet size, most notably for ftp-data). We plot three applications on this set of graphs: RealMedia, HTTP, and FTP-data. The reason for including WWW traffic is that the FTP traffic is almost all local traffic (either on our network, or to nearby networks) and so has very low RTTs, and concomitantly high throughput rates, along with very low average packet inter-arrival times (of the order of a few milliseconds). While this may be realistic in some scenarios, it may not be representative of the statistics seen in all networks. So we use WWW to see the impact of larger RTTs — the average inter-arrival times for HTTP packets were almost two orders of magnitude larger. Note also that we exclude days where there were a very small number (less than 10) of streaming flows because we cannot form accurate statistics with so few samples. Such days were all weekends, and weekdays had many more streaming flows.

The results demonstrate several features. Firstly, the inter-arrival variability metric appears to be a good method for distinguishing data transfer applications from streaming. In both plots we can see clear separation between the data applications (HTTP and FTP) and the streaming application (RealMedia). Furthermore, despite the very large difference in average inter-arrival time (and RTT) for the HTTP and FTP traffic, their inter-arrival variability metrics are very similar. Finally, notice that the values of the variability metric are not particularly sensitive to whether we include ACK streams or not. This is important because it reduces the accounting we need to do to determine whether a given uni-direction flow corresponds to an ACK stream or a data stream. It further allows us to make measurements in places where the traffic routing is asymmetric, and so we only see traffic from one direction of each connection.

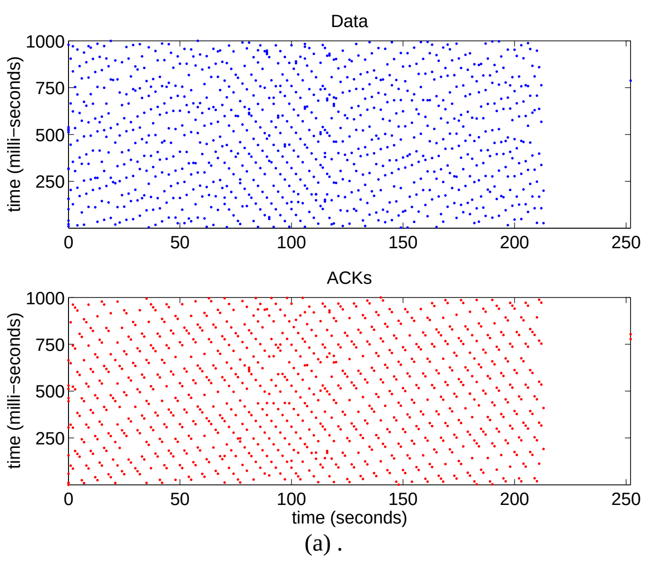
Notice also that the packet size is another distinguishing factor. However, as noted in the somewhat larger data sets, this feature alone does not provide adequate separation between the bulk-data and streaming traffic classes. Hence the need for an additional feature, although packet size is clearly a better discriminator when we can distinguish ACK flows and discard these. Hence the results of the previous sections should be seen as conservative bounds.

Figure 6 also shows LDA boundaries between the two classes. It should be clear that, at least for the data available, it is easy to form good boundaries between data sets, for both cases (including and excluding ACK streams), though we do not have enough data points in this data set to perform a good cross-validation.

Finally note, there are practical problems in collecting large, representative packet traces for use in this analysis. Full packet header traces take up considerable volumes of data, but are needed when exploring features that might be of use (we considered several other features in choosing the variability metric above). However, as noted previously, the statistics of interest, once determined, can be collected on-line, without the need to store the full packet header trace. At the least, a daily trace could be collected, and statistics for flows computed, and these stored. Hence, the limitations that apply while we are the research phase (testing and choice of features from a large possible set) do not apply when this method is applied in practice – one can collect very large, representative data sets, with minimal data storage requirements.

5.4 Temporal differences

We have shown a variety of classification results above, in order to illustrate various aspects of our approach. In this section, we



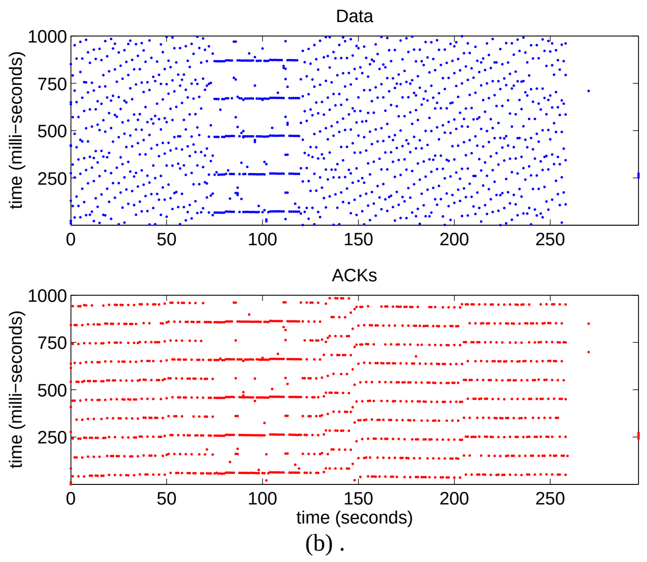
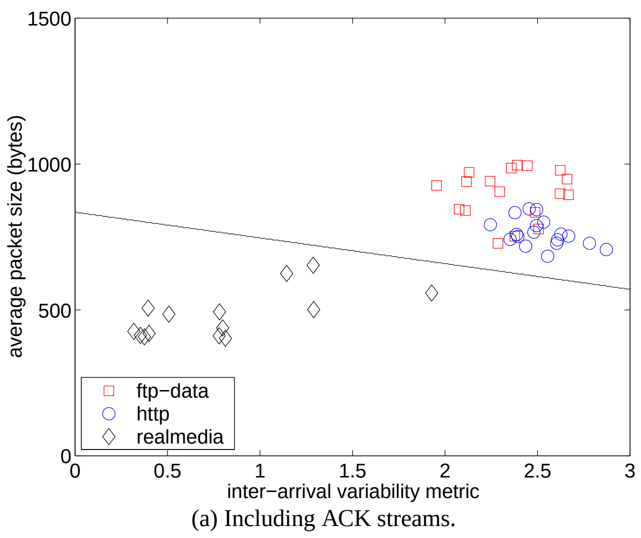


Figure 5: Two example dot-strip plots of the packet arrival times within two RealMedia flows. Note in particular the regular patterns over the majority of the interval, and the long gap before the final few packet transmissions.



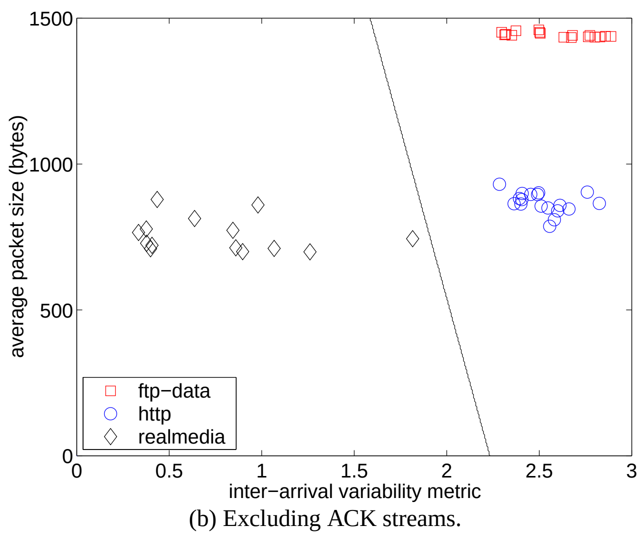


Figure 6: The inter-arrival variability metric for three applications: HTTP, FTP-data, and RealMedia.

present a test for which we have reserved the last set of data. The last set of data (Gigascope trace 3) has not been used in any of the training. In this test, we use data trained on Gigascope trace 2 to classify flow aggregates from Gigascope trace 3. While both traces were gathered from the same network, they were gathered at different times: trace 3 was gathered in late October, early November 2003, and trace 4 was gathered in April 2004, approximately six months later. As we will see below, the statistical behavior of the applications varies between the two time periods. Hence, this is a strong test of how well the method could perform in practice, where the training data may have been derived from an earlier period than the network where QoS is being implemented.

Again we censor the last 10 packets of each flow to avoid the problems documented above. As an obvious result, we exclude the majority of transactional flows, biasing results if we use those that remain. Hence, we will not try to classify transaction applications in this example. Note that transaction flows were easily classified earlier, because of the small packet size, and short duration. Thus we use the three classes, bulk data, interactive, and streaming in this test, as exemplified by ftp-data, telnet, and realmedia traffic, respectively. We do not separate ACKs from data in any of these tests.

Figure 7 shows scatter plots of the inter-arrival variability metric and average packet size for the two data sets. We can see that the three groups of applications are fairly distinct, but that the groups do not remain in exactly the same location. The class boundaries on both Figures 7 (a) and (b) were derived from the data in Figure 7 (a). We can see that these boundaries perfectly classify the data in Figure 7 (a) (not unexpected for boundaries trained on the same data). However, the interesting thing is that these boundaries also work perfectly on the data from Gigascope trace 3, shown in Figures 7 (b). Thus, the classification rules derived six months previously still work.

Figure 7 shows classification boundaries derived by LDA for only two features: the average packet size, and the variability metric. We can easily expand these results, to more than two features, though it is then hard to illustrate the results, except through percentage errors. We test the various classification methods described above using three features: in addition to the average packet size, and the variability metric, we use the average flow duration. Once again training on the earlier data set, and testing on the later data, we get an 0.0% error rate for simple LDA, and a 9.5%, 2.4%, 0.0%, 0.0% and 0.0% error rate for NN, using 1, 3, 5, 7 or 15 neighbors, respectively. A zero error rate given the differences in timing of the training and test data sets seems to be quite a strong validation of this approach.

5.5 Classification of a new application

One of our aims is to build a technique that can classify a new application, for which we do not have any specific training data, except from similar applications in the same class. In this section we extend the previous test of training and testing on different data sets to the classification of an unknown application. We test this here, in one particular case: classification of the application `rsync`, which is a protocol/software used to copy data, and therefore in theory in the bulk-transport class. However, `rsync` is clever enough to check files for changes, and only copy changed files. Hence, `rsync` will have aspects of bulk-data transfer, but will also, on many occasions not transfer large amounts of data, but only small packets checking the differences between files. This introduces some new characteristics into the application, that differentiate it from simple ftp-data transactions.

We perform the test by training the classifier rules on Gigascope dataset 2, using the applications ftp-data, telnet, and realmedia, and the features: average flow duration, average packet size, and the inter-arrival variability metric. We then perform the classification on each days `rsync` data from Gigascope dataset 3. The correct class is assumed to be bulk-data, and we found that the LDA classification was in error 57% of the time, and the errors for NN with 1, 3, 5, 7, and 15 neighbors were 57%, 14%, 14%, 21.5% and 28.6% respectively. The large error for LDA suggests that care must be taken in choice of algorithm used for classification, but we have consistently found good performance for NN-3 and N-5 throughout this paper, and likewise, here it results in a reasonable error. Certainly, we might wish to reduce this level of error, but given the demanding nature of this particular task (given both temporal difference in the training and test data, and classification of a new application), this is a very hopeful result.

6. CONCLUSIONS

QoS could provide substantial cost savings for network operators, while maintaining performance for critical applications, but it has been held back by operational difficulties in provisioning. The lack of scalable techniques for mapping traffic into classes acts to inhibit adoption of QoS by even the most enthusiastic network operators. Even the early adopters cannot reap the full benefits without solving this problem. In this paper we presented a framework and some results for classifying traffic into CoS based on measured traffic characteristics.

The method is based on statistics of the traffic which result from the way the application is used, and so can detect the typical use of an application, which may differ from the preconceived notions of how that application is used – for instance if HTTP were used for a large amount of the streaming traffic in an organization, rather than more traditional web browsing.

Evaluations using large traffic traces from different network locations indicate that the approach has relatively low error rates. The results are encouraging, but much more work remains: for instance, on alternative features, classes, and classification techniques. For example, it is plausible that one could combine the techniques used here with more ordinary port based classification to enhance the results. Further, data-sets from different enterprise settings are required to extend this work to the context of interest.

With this paper we hopefully convince the reader that this problem important and interesting, but not insoluble. We therefore believe that these results will stimulate research on a new set of problems within the domain of traffic analysis.

Acknowledgments

We would like to thank the WAND group for making their traces publicly available, and the anonymous reviewers for their valuable comments.

7. REFERENCES

[1] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. IETF Network Working Group RFC 2581, 1999.

[2] J. Almeida, J. Krueger, D. Eager, and M. Vernon. Analysis of educational media server workloads. In *Proc. Inter. Workshop on Network and Operating System Support for Digital Audio and Video*, June 2001.

[3] E. Altman, K. Avrachenkov, and C. Barakat. A stochastic model of TCP/IP with stationary random losses. In *SIGCOMM'2000*, 2000.

[4] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *Proceedings of ACM Sigmetrics*, June 1998.

[5] P. Barford, J. Kline, D. Plonka, and A. Ron. A Signal Analysis of Network Traffic Anomalies. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, Nov 2002.

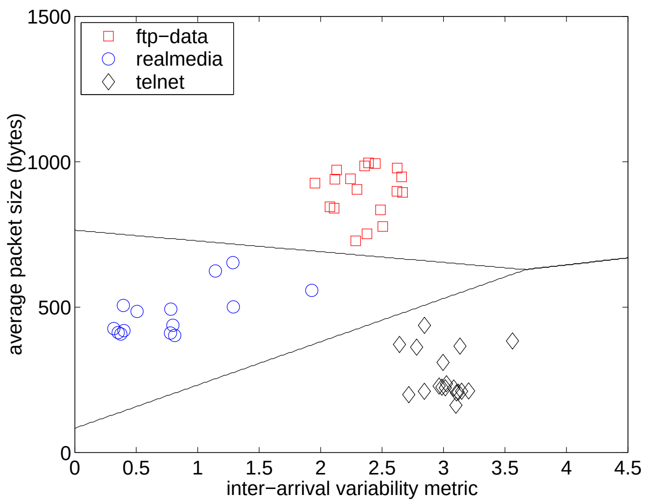
[6] P. Barford and D. Plonka. Characteristics of Network Traffic Flow Anomalies. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, Oct 2001.

[7] Y. Bernet, J. Binder, S. Blake, M. Carlson, B. Carpenter, S. Keshav, E. Davies, B. Ohlman, Z. Wang, and W. Weiss. A framework for differentiated services. Internet Draft, February 1999. `http://search.ietf.org/internet-drafts/draft-ietf-diffserv-framework-02.txt`.

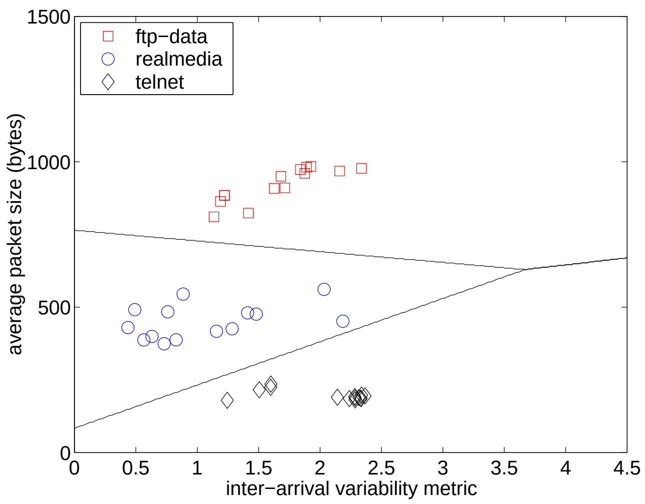
[8] S. Blake, D. Black, D. Black, H. Schulzrinne, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. Rfc 2475 - an architecture for differentiated service, December 1998. Available at `http://www.faqs.org/rfcs/rfc2475.html`.

[9] M. Chesire, A. Wolman, G. M. Voelker, and H. M. Levy. Measurement and analysis of a streaming media workload. In *Proc. USENIX Symposium on Internet Technologies and Systems*, March 2001.

[10] K. Claffy. *Internet traffic characterization.* PhD thesis, UC San Diego, 1994.



(a) Gigascope trace 2.



(b) Gigascope trace 3.

Figure 7: Scatter plots of the inter-arrival variability metric, and average packet size, along with classification boundaries derived from Gigascope trace 2 using LDA.

[11] C. Cranor, T. Johnson, and O. Spatscheck. Gigascope: a stream database for network applications. In *SIGMOD*, June 2003.

[12] C. Dewes, A. Wichmann, and A. Feldmann. An analysis of Internet chat systems. In *Proceedings of ACM SIGCOMM Internet Measurement Conference*, Oct 2003.

[13] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *ACM SIGCOMM*, Karlsruhe, Germany, 2003.

[14] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *Computer Communication Review*, 26(3):5–21, 1996. Available at `http://www.aciri.org/floyd/papers.html`.

[15] A. Feldmann, A. Gilbert, and W. Willinger. Data networks as cascades: Explaining the multifractal nature of internet WAN traffic. In *Proceedings of the ACM Sigcomm'98, Vancouver, Canada*, 1998.

[16] S. Floyd. Connections with multiple congested gateways in packet-switched networks, part I: One way traffic. *Computer Communications Review*, 21(5), 1991.

[17] C. Gbaguidi, H. Einsiedler, P. Hurley, W. Almesberger, and J. P. Hubaux. A survey of differentiated services architectures for the Internet, March 1998. `http://sscwww.epfl.ch/Pages/publications/ps_files/tr98_020.ps`.

[18] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *STOC*, 2002.

[19] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2001.

[20] IANA. Internet Assigned Numbers Authority (IANA). `http://www.iana.org/assignments/port-numbers`.

[21] V. Jacobson. Congestion avoidance and control. *Communication Review*, 18(4):314–329, 1988. Available at `ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z`.

[22] B. Krishnamurthy and J. Rexford. *Web Protocols and Practice*, chapter Chapter 10: Web Workload Characterization. Addison-Wesley, 2001.

[23] Z. Liu, M. S. Squillante, C. H. Xia, S. zheng Yu, and L. Zhang. Profile-based traffic chacterization of commercial web sites. In J.Charzinski, R.Lehnert, and P.Tan-Gia, editors, *Proceedings of the 18th International Teletraffic Congress (ITC-18)*, volume 5a, pages 231–240, Berlin, Germany, 2003.

[24] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communication Review*, 27(3):67–82, July 1997. Available at `http://www.psc.edu/networking/tcp_friendly.html#performance`.

[25] J. Micheel, I. Graham, and N. Brownlee. The Auckland data set: an access link observed. In *the 14th ITC Specialist Seminar on Access Networks and Systems*, Barcelona, Spain, April 25th-27th 2001.

[26] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial of Service Activity. In *Proc. of the USENIX Security Symposium*, Washington D.C., August 2001. Available at `http://www.cs.ucsd.edu/~savage/papers/UsenixSec01.pdf`.

[27] White paper-netflow services and applications. `http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm`.

[28] J. Padhye, V. Firoin, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *ACM SIGCOMM'98*, 1998. Available at `http://www.psc.edu/networking/tcp_friendly.html#performance`.

[29] V. Paxson. Empirically derived analytic models of wide-area TCP connections. *IEEE/ACM Transactions on Networking*, 2(4):316–336, 1994.

[30] V. Paxson and S. Floyd. Wide-area traffic: The failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[31] J. E. Pitkow. Summary of WWW characterizations. *W3J*, 2:3–13, 1999.

[32] H. Schulzrinne, A. Rao, and R. Lanphier. Real time streaming protocol (RTSP), request for comments 2326, April 1998. `ftp://ftp.isi.edu/in-notes/rfc2326.txt`.

[33] J. Tukey and P. Tukey. Strips displaying empirical distributions: I. textured dot strips. Technical report, Bellcore Technical Memorandum, 1990.

[34] J. van der Merwe, S. Sen, and C. Kalmanek. Streaming video traffic : Characterization and network impact. In 7^{th} *International Web Content Caching and Distribution workshop (WCW)*, Boulder, Colorado, August 14th-16th 2002.

[35] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. *Proceedings of the ACM SIGCOMM'95*, 1995. Available at `http://www.acm.org/sigcomm/sigcomm95/sigcpapers.html`.

[36] Y. Zhang and V. Paxson. Detecting backdoors. In *Proc. USENIX*, Denver, Colorado, USA, 2000.