Enhancing Tor's Performance using Real-time Traffic Classification

Mashael AlSabah, Kevin Bauer, Ian Goldberg
Cheriton School of Computer Science, University of Waterloo
{malsabah,k4bauer,iang}@cs.uwaterloo.ca

ABSTRACT

Tor is a low-latency anonymity-preserving network that enables its users to protect their privacy online. It consists of volunteer-operated routers from all around the world that serve hundreds of thousands of users every day. Due to congestion and a low relay-to-client ratio, Tor suffers from performance issues that can potentially discourage its wider adoption, and result in an overall weaker anonymity to all users.

We seek to improve the performance of Tor by defining different classes of service for its traffic. We recognize that although the majority of Tor traffic is *interactive web browsing*, a relatively small amount of *bulk downloading* consumes an unfair amount of Tor's scarce bandwidth. Furthermore, these traffic classes have different time and bandwidth constraints; therefore, they should not be given the same Quality of Service (QoS), which Tor offers them today.

We propose and evaluate DiffTor, a machine-learning-based approach that classifies Tor's encrypted circuits by application *in real time* and subsequently assigns distinct classes of service to each application. Our experiments confirm that we are able to classify circuits we generated on the live Tor network with an extremely high accuracy that exceeds 95%. We show that our real-time classification in combination with QoS can considerably improve the experience of Tor clients, as our simple techniques result in a 75% improvement in responsiveness and an 86% reduction in download times at the median for interactive users.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General— *Data communications*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.4 [**Computer Systems Organization**]: Performance of Systems; K.4.1 [**Computers and Society**]: Public Policy Issues—*Privacy*

General Terms

Measurement, Performance, Security

Keywords

Tor, machine learning, traffic classification, quality of service

CCS'12, October 16–18, 2012, Raleigh, North Carolina, USA.

1. INTRODUCTION

Recent years have witnessed a dramatic increase in the use of privacy-enhancing technologies. One key example is Tor [11], the most widely used anonymity network with almost half a million users [37]. Not only does Tor enable its users to maintain their privacy online, but it also provides them with a means to resist and circumvent network surveillance. Tor today is an influential anti-censorship technology that allows people in oppressive regimes to access information without the fear of being blocked, tracked or monitored. The importance and success of Tor is evident from recent global uprisings where the usage of Tor spiked [10] as people used it as a revolutionary force to help them fight their social and political realities.

Although Tor succeeded in attracting a rapidly increasing number of users since it was launched in 2003, the number of its volunteer-operated relays has not been growing at the same rate. Today, there are approximately 2,500 relays that run from all around the world [37]. Because of the large client-to-relay ratio, users experience poor performance that manifests itself in the form of large and highly variable delays experienced in response and download times during web surfing activities.

Still worse, traffic congestion adds further delays and variability to the performance of the network. Previous studies have revealed that congestion has mainly two sources in Tor. First, the Tor network currently is not congestion controlled; instead it employs an end-to-end flow control approach which does not react to congestion in the network [4]. Second, although Tor was originally designed for interactive applications such as web browsing and instant messaging, a small number of Tor users use bandwidth-greedy applications such as BitTorrent that consume a large and unfair fraction of the available bandwidth in the network [29].

Not only does the poor performance hinder a wider adoption of the network, but it may potentially reduce its user base. Losing users has a significant impact on the anonymity provided to users, since reducing the user base results in decreasing the size of the network's anonymity set. Therefore, it is crucial to improve the performance and usability of Tor in order to enhance the anonymity it provides.

This problem has received great attention from the Tor research community. To reduce the high client-to-relay ratio, incentive-based schemes have been proposed [21,31,32] to encourage clients to run their own relays. However, the effects of such schemes are poorly understood. In fact, if such approaches are adopted, users in oppressive regimes unable to run their own relays will have to either give up using the network or continue accepting its below-mediocre service.

To reduce the congestion caused by greedy applications, global or static throttling techniques have been introduced [22, 31] that

limit the rate at which these applications can transfer data. There are a number of problems with global network throttling techniques. First, they are either based on fixed throttling rates that can also affect interactive or real-time application users, or their ability to detect greedy applications is based on simple metrics that are easy to game. Also, we believe that global network throttling prevents the network from optimizing its resources effectively.

In this work, we propose defining classes of service for Tor's traffic. Our solution is based on the key observation that different applications that use Tor have inherently different time and throughput requirements, and therefore should not need or get equivalent service, which Tor offers them today. By defining different classes of service, we can map each application class to its appropriate QoS requirement. This solution is significantly more flexible than previous approaches and it provides relays with more dynamic control over their limited resources. For example, it allows relays to prioritize or throttle one application class over another. When the prioritized class ceases communication, the throttled class can utilize the available bandwidth. Furthermore, even though the network has limited capacity, this approach allows for finer-granularity management of the existing network resources. Such management is needed to provide good performance to important applications.

Classification for Tor. Motivated by the need to improve the performance of the Tor network, we introduce DiffTor, a framework for classifying encrypted Tor circuits based on the applications they serve. The key novelty in our work is that we enable differentiated QoS using network traffic classification in anonymous communication systems. We explore two types of traffic classification: online and offline classification. Below we highlight the differences between the two classification types:

- **Online:** The main goal of online classification is for a router to classify circuits on the fly in order to provide differentiated services. This means that classification must take place while the circuit is alive, and before too much traffic has been forwarded along the circuit. Online classification is more challenging since the classifier has access to less information about the circuit. The classifier performs classification at the cell level and is mainly concerned with cell-level attributes and some limited circuit-level attributes. Also, since the classifier runs in real time, it must also be efficient.
- **Offline:** On the other hand, classification can be done offline by relay operators using circuit logs (assuming that logging is performed in a privacy-preserving manner). Because there are no time constraints in offline classification, we can use more global information about circuits as classification attributes.

The importance of offline classification is twofold. First, it is important to understand how much information an entry guard—the first router in a Tor circuit—can infer just from monitoring the traffic of its clients' circuits. This is especially important to investigate about entry guards since they receive direct connections from clients and continue to be used by the same clients for weeks. Indeed, we found that entry guards can identify the class of application of a circuit with an accuracy that exceeds 90%. Second, The Tor Project is researching techniques to understand more information about its users without compromising their privacy [27]. Classifying circuits offline is a novel approach to provide more insight into how the network is used today from the view of entry nodes unable to see plaintext traffic.[1]

[1]Previous work used Deep Packet Inspection (DPI) to study the usage of the Tor network from exit routers [9, 29]. It is important to avoid inspecting the clear traffic for legal and ethical reasons.

Next, we consider online classification. Classifying Tor circuits on the fly allows us to define Quality of Service (QoS) requirements for different types of traffic. Defining QoS parameters can be done either locally by the entry guard operator or globally by the directory authorities depending on the state of the network. For example, when the network is congested, entry guards can change their QoS parameters so that interactive real-time applications such as web browsing are prioritized, while bandwidth-consuming applications such as non-time-sensitive file sharing applications are throttled. This can improve the performance of the network and the experience of users, since different applications would get different QoS parameters based on their requirements and on the congestion state of the network.

In this work, we carry out the following approach. We first start by exploring how different applications use the Tor network. In particular, we seek to understand how the traffic of some popular applications looks and behaves when travelling through Tor circuits. The importance of this step is that it helps us derive useful attributes that we can use for our online and offline classification process.

Based on studying different applications, we first start by developing a simple threshold classifier that we can use to classify connections between clients and entry guards to two applications, BitTorrent and browsing. We recognize that this classifier cannot be used on middle and exit relays, and would not be useful if clients disable using entry guards. For these reasons, we explore different machine-learning classifiers and identify which algorithms are suitable for our offline and online classification, which unlike the threshold classifier, can also work at middle and exit routers in addition to entry guards. Finally, we implement an online classifier and we define simple QoS rules to show the performance benefits made possible using our online classifier.

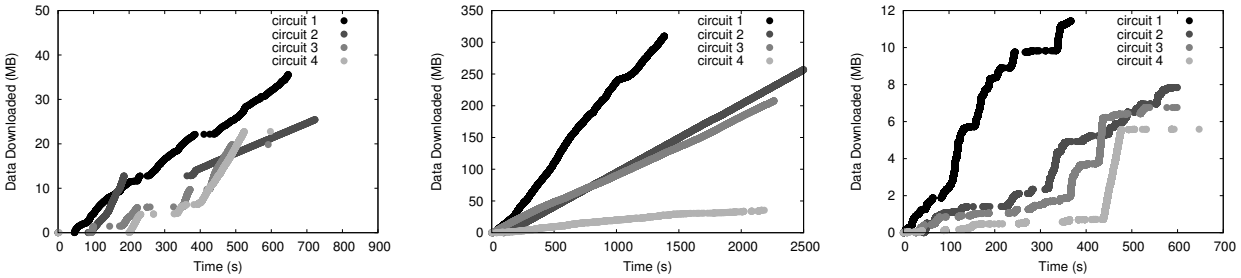
Contributions. This work contributes the following:

- We provide an important insight into how some popular applications behave and appear within Tor circuits.
- We propose a machine-learning approach to encrypted traffic classification for Tor. Our evaluation on the live Tor network indicates that our classifiers achieve over 95% accuracy. When combined with prioritization for interactive web-browsing circuits, we see an improvement in responsiveness of 75% and an improvement in download time of 86%.
- Our approach to prioritization is incrementally deployable, as our changes are local to any router and do not affect its operation with other routers.
- We discuss a variety of open issues related to the classification of encrypted Tor traffic.

Roadmap. The remainder of this paper is organized as follows: Section 2 presents a high-level description of Tor's design and how different applications use Tor. We present our threshold classifier and motivate for the need for machine-learning-based classifiers in Section 3. Section 4 presents our three broad classes, our candidate features and our machine learning classification methods. We evaluate our proposal in Section 5. We also discuss a variety of open issues related to the classification of encrypted Tor traffic in Section 6. Finally, we compare our contributions with related work in Section 7 and conclude in Section 8.

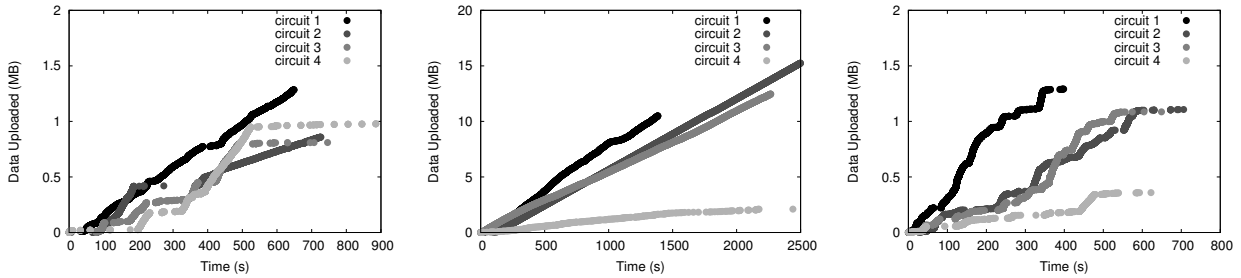
2. TOR BACKGROUND

Tor is a low-latency anonymity network which is based on a client-server architecture model. Clients, known as *Onion Proxies* (OPs), periodically connect to directory servers to download information about the currently available *Onion Routers* (ORs), and information on how to contact them such as the OR IP and public keys. Then, clients use ORs to form paths, known as *circuits*,



(a) Streaming (b) BitTorrent (c) Browsing

Figure 1: Downstream circuits



(a) Streaming (b) BitTorrent (c) Browsing

Figure 2: Upstream circuits

through the network to Internet destinations. By default, circuits are composed of three ORs, usually nicknamed the *entry guard*, *middle* and *exit* OR, depending on their position on the circuit. Of the three ORs, only the entry guard knows and communicates directly with the client, and only the exit knows the Internet destination that the client is communicating with, but no OR can link a client to a destination; this is how a client's privacy is maintained in Tor.

A SOCKS proxy interfaces between user applications and the OP. When the application sends data through Tor, the OP divides the data to 512-byte fixed-sized cells, and adds a layer of encryption for every node on the forward path. Then, cells are source-routed through the established circuits. Every hop, on receiving a relay cell, looks up the corresponding circuit, decrypts the relay header and payload with the session key for that circuit, replaces the circuit ID of the header, and forwards the decrypted cell to the next OR.

Circuit scheduling. Tor uses a label-switching design that multiplexes several circuits across the same Tor routers. In order to ensure that each circuit is given a fair share of the routers' bandwidth, Tor employs a round-robin queuing mechanism. Each circuit is serviced in a first-come, first-served manner, which ensures that each circuit is given a fair share of the available bandwidth.

However, it is been shown that the distribution of application traffic on Tor is not uniform across all circuits: a relatively small number of circuits (e.g., bulk file downloaders) consume a disproportional amount of the network's bandwidth [29]. To mitigate the unfairness, a circuit scheduling prioritization scheme has been proposed [36] so that interactive circuits tend to be serviced before bulk-downloader circuits. This prioritized circuit scheduling is currently deployed on the live Tor network.

Rate limiting. To give Tor router operators control over the amount of bandwidth that is consumed by the Tor process, a token bucket rate limiting feature on the granularity of a router is available. In addition, Tor also allows routers to rate limit individual connections (e.g., to certain clients or destinations). Such finely granular rate limiting has recently been proposed as another mechanism to quiet bulk downloaders in Tor. Tortoise [31] presents an argument for explicitly reducing the bandwidth rate of the bulk downloaders in order to conserve bandwidth and improve performance for the vast majority of interactive Tor users; Jansen *et al.* [22] present a similar argument and both proposals demonstrate an important performance benefit to the majority of Tor users. In our work, we extend this intuition and propose additional methods that offer differentiated services for different classes of applications that use Tor.

2.1 How Different Applications use Tor

We first seek to explore how different applications affect how the traffic looks on circuits. This is important for us to understand in order to be able to nominate some useful attributes for our classification process.

Figures 1 and 2 show data transferred upstream and downstream on some sample active Tor circuits that we logged on the live Tor network. The traffic was generated by our own browsing, streaming, and BitTorrent clients; for privacy reasons, we did not monitor other users' Tor traffic. We provide a detailed description on how we generate and log traffic on the live network in Section 5.1. The following differences can be observed from the figures.

Applications produce different circuit lifetimes. The circuit lifetime is different for different applications. Web browsing circuits almost never exceed 600 seconds, which is the default circuit lifetime, while actively downloading BitTorrent circuits can stay alive for as long as the client is actively downloading on the circuit. In fact, it can be seen in Figure 1(b) that BitTorrent downloads can cause the circuit to stay alive for more than 2500 seconds. Streaming circuits can exceed the 600-second mark, but once the streaming session is over, the circuit will be torn down.

BitTorrent traffic is bidirectional. In active BitTorrent circuits, data is continuously travelling upstream and downstream. Web browsing and streaming circuits on the other hand appear to have bursts of data that are followed by gaps of inactivity. Those idle periods may be a result of user think times. However, some web browsing circuits also seem to be continuously active due to the dynamic content of web pages.

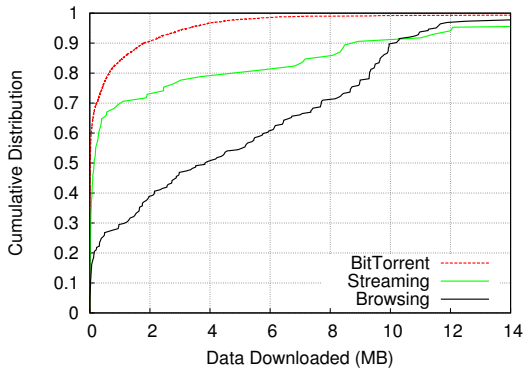


Figure 3: Comparison of the amounts of data downloaded by circuits of different applications

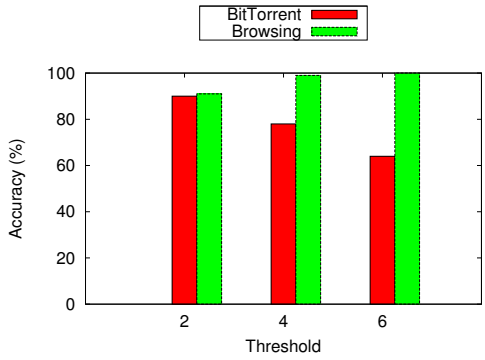


Figure 4: Per-class accuracy of the threshold classifier when $t = 100$

Interactive circuits transfer small amounts of data. More than 95% of circuits in each class download less than 14 MB, as shown in Figure 3. However, while the maximum observed download for any streaming circuit was less than 35 MB and for any browsing circuit was less than 23 MB, some BitTorrent circuits downloaded as much as 935 MB.

BitTorrent causes many idle circuits to be built. Finally, as shown in Figure 3, most circuits created for web browsing are indeed utilized, while only 30% of the BitTorrent circuits are actually used to download data. We observed that the BitTorrent client creates many more circuits than a browsing client. The reason is that the BitTorrent client attempts to contact several peers at the same time using different ports. Those circuits attempt to connect to peers that either are no longer participating in the file transfer, or do not respond to connection requests due to the presence of a firewall or network address translation (NAT). Interestingly, when a remote host fails to respond to a connection request (and does not respond with "connection refused"), Tor's default behaviour is to retry the same remote host again on a different circuit. This behaviour is rarely triggered by the browsing client since it usually connects to a few publicly accessible servers.

Creating several idle circuits has performance and anonymity implications. Clearly, circuit creation consumes OR memory and CPU cycles. As for anonymity, creating several circuits simultaneously can degrade the anonymity of the user as it can leak some information to the entry guard about the user. For example, we observed that in a 10-hour experiment, our web browser created around 100 circuits, while the BitTorrent client created more than 1000 circuits. If an entry guard observes that the user is using several simultaneous circuits, then very likely, the client is using BitTorrent over Tor.

3. STRAW MAN: CIRCUIT THRESHOLD CLASSIFIER

Recall that BitTorrent causes the Tor client to build significantly more circuits compared to other interactive applications, which usually do not require more than one circuit at a time. In this section, we show how an entry guard can use a simple threshold classifier to perform a connection-level classification on its clients' connections to two broad classes, BitTorrent and web browsing. The classifier's decisions are based only on the number of circuits created over a connection.

Our simple classifier monitors the clients' connections, and performs a classification periodically every t seconds based on the number of circuits T seen in the connection within the last t seconds. Figure 4 shows the per-class accuracy as a function of the threshold value when $t = 100$. If the threshold is $T > 2$, meaning that if we only observed more than two circuits in the connection within the 100-second interval, then the probability that the connection serves BitTorrent is 90%. Also, with 91% certainty, if the connection carries two circuits or fewer, then the connection is serving web browsing. As we increase the threshold value to $T > 6$, we get perfect 100% classification accuracy for web browsing connections at the expense of misclassifying 36% of BitTorrent connections as browsing connections.

Although this classifier is very simple and requires no prior collection of any training data, it has several limitations. First, this classifier works at the connection level, and therefore, we cannot use it at intermediate ORs in a circuit since the OR-to-OR connections multiplex circuits from several users. Second, a client can easily game the classifier by not using entry guards. That will significantly reduce the number of circuits observed by one guard for a specific user. In the next section, we show how we avoid these problems by using machine learning approaches to classify Tor traffic at the circuit and cell levels.

4. CLASSIFICATION FOR TOR

In this section, we start by defining the three traffic classes we are interested in. Then, we describe the attributes and the machine learning algorithms that we use for the online and offline classification.

4.1 Class Definition

A previous Tor traffic study [29] reported that web browsing traffic results in over 92% of the TCP connections in the Tor network, yet it accounts for only 60% of the aggregate traffic volume. The other 40% of the aggregate traffic volume is mainly accounted for by BitTorrent. Based on these findings, we define three broad classes of traffic:

Bulk transfer. This class contains applications that upload or download large volumes of data over the network. Such downloads have no time constraints, but are greedy and will try to utilize as much available bandwidth as possible. BitTorrent is the main application in this class[2] since it accounts for a great portion of the traffic in Tor.

Interactive. Interactive applications are real-time applications which require interaction between a client and a server (or another client in the case of instant messaging, for example). Those applications are time sensitive and require improved responsiveness. We focus on web browsing as the main application of this class.

Streaming. Streaming applications are non-interactive bulk transfers which require time and quality constraints. An example for

[2] Although BitTorrent is switching to UDP, recent studies in Tor suggest that there is still a significant amount of BitTorrent traffic served by Tor [8].

streaming applications is watching streaming videos online.[3] The reason we consider streaming applications is that recent Internet measurement studies reported a significant increase in the amount of streaming traffic online [28, 35].

4.2 Candidate Attributes

Attribute selection is a non-trivial problem in the area of network traffic classification. It gets particularly more difficult for an anonymity network like Tor where we do not have access to traditional useful attributes such as the connection ports and the IP packet sizes. We also take the challenge further as we try to realize real-time classification. In selecting candidate classification attributes for classifying Tor's traffic, we strived to identify high-level protocol features that are not influenced by human interaction variability. Below we present our attributes.

Circuit lifetime. By default, if a circuit has been used for 10 minutes and is currently idle, the Tor client tears down the circuit and starts attaching new streams to different circuits. Recall that we found in section 2.1 that bulk downloading applications such as BitTorrent cause circuits to remain alive well beyond the 10 minute mark.

Data transferred. The amount of data that a circuit transfers upstream and downstream can be very indicative of the type of application that is being used by a circuit. For example, circuits which serve bulk downloads are expected to transfer significantly more data than circuits that serve web browsing sessions. Moreover, the data uploaded to destinations outside the network can indicate that the circuit is serving a bulk transfer rather than an interactive application.

Cell inter-arrival times. The time between two consecutive cells in one circuit (and other related features such as the mean and variance of cell inter-arrival times (CIT)) can also indicate the type of application that uses the circuit. For example, bulk downloading applications such as BitTorrent almost never pause throughout their downloads. Interactive applications, on the other hand, consist of smaller bursts of data which are separated by gaps on inactivity. CIT statistics are useful in quantifying the *burstiness* of the traffic. Burstiness is a key attribute in separating different classes of traffic because it is a high-level attribute that is not influenced by the behaviour of individual users, but rather by how different application protocols work. Measuring the cell inter-arrival times can be done by subtracting the arrival times of two consecutive cells. To calculate some statistics of the inter-arrival times efficiently, we use the following recursive estimators [23]:

$$\mu_t = \frac{t-1}{t}\mu_{t-1} + \frac{1}{t}x_t \tag{1}$$

for the moving time average, where t starts at 1, x_t is the current measurement average, and μ_{t-1} is the previous average. Likewise, the variance can be computed as follows:

$$\sigma_t^2 = \frac{t-1}{t}\sigma_{t-1}^2 + \frac{1}{t-1}(x_t - \mu_t)^2 \tag{2}$$

where t starts at 2, x_t is the current measurement data, μ_t is the current average, and σ_{t-1}^2 is the previous variance. In addition to μ_t and σ_t^2, we also use the *coefficient of variation* as an attribute, which is given by $\frac{\sigma_t}{\mu_t}$.

The number of cells sent recently. Normally, it is expected with bulk downloads that the average number of cells seen recently would be consistently large, while interactive applications usually have smaller bursts of data followed by a period of inactivity. To quantify this feature, we use the Exponential Weighted Moving Average (EWMA) of the cells sent on a circuit in an algorithm that was proposed by Tang and Goldberg [36].

[3] It is possible to stream videos over Tor using HTML5.

4.3 Classification Algorithms

Machine learning techniques are categorized as *supervised* or *unsupervised*. Supervised machine learning techniques are based on two stages. The first stage is known as the *training* stage where a classification model is built using a classification algorithm and a training dataset that contains labelled instances. The second stage is known as the *testing* stage in which the classifier model obtained from training is used to classify unlabelled instances.

Unsupervised techniques, on the other hand, seek to group or cluster data together based on some similarities, or common characteristics, without knowing the classes beforehand.

In this section, we provide a brief description of the four supervised classification algorithms we use in this work and we focus on the Naïve Bayes classifier since we implemented it for our live experiments:

Naïve Bayes. Naïve Bayes is a probabilistic classification algorithm that is based on Bayes' theorem.

$$p(C|A_1, A_2, .., A_n) = \frac{p(C)P(A_1, A_2, .., A_n|C)}{p(A_1, A_2, .., A_n)} \tag{3}$$

Assume that a circuit X has attributes $A_1, A_2, .., A_n$. For example, A_i might denote the circuit lifetime, or the number of cells sent. We would like to find the probability that X belongs to a class C, which belongs to a finite set of classes. In our context, those classes are bulk, interactive and streaming. If the attributes are dependent, then computing the above conditional probability would be difficult. However, if we make the "naive" assumption that the attributes are independent, then, our conditional probability can be expressed as follows:

$$p(C|A_1, A_2, .., A_n) = \frac{p(C)\prod p(A_i|C)}{p(A_1, A_2, .., A_n)} \tag{4}$$

Note that $p(A_1, A_2, .., A_n)$ can be ignored as it is constant for all $p(C_i|A_1, A_2, .., A_n)$. The rest of the right-hand side of the above expression is determined from the training data. Since attributes in our case are numerical, a preprocessing step of discretization is needed to be able to compute $p(A_i|C)$. We have made the simplifying assumption that our data follows a normal distribution, so $p(A_i|C)$ can be computed using the normal density function. The last remaining step for the classifier is to make a decision rule based on the $p(C_i|A_1, A_2, .., A_n)$ values it computes for all classes. One common decision rule is to choose the most probable class.

Although the Naïve Bayes classifier is based on strong assumptions of feature independence, it is known to work well in practice, even if this assumption is violated. Furthermore, we chose to evaluate Naïve Bayes due to its strong performance in several related encrypted traffic classification tasks [19, 26].

Bayesian Networks. Bayesian Networks (or Bayes Nets) have been found to be very successful at classifying Internet traffic by application [5]. Bayes Nets are graphical representations that are used to model the dependency relationships between attributes and classes; therefore, unlike Naïve Bayes, they do not make the strong assumption that attributes are independent. They consist of a a directed acyclic graph (DAG) where the vertices represent attributes and classes. The edges between vertices represent the dependencies between the attributes and classes. Also, the Bayes Net contains conditional probability tables that allow for probabilistic inference. Friedman *et al.* present Bayes Nets in more detail [14].

Decision Trees. We also use two types of decision tree classifiers. We apply functional tree (FT) classification [15], which enable multivariate features in decision tree analysis. We also apply logistic model tree (LMT) classification [25]. We choose to evaluate a representative set of decision tree classifiers due to the success of similar classifiers in classifying multiple applications within IP packets traces [13].

5. EXPERIMENTS AND RESULTS

Our experiments were carried out as follows. First, from the live Tor network, we collected traffic logs, from which we created datasets that contained instances of the attributes we presented in Section 4.2, along with their respective classes. Then, we evaluated our online and offline classification algorithms on the datasets using the Waikato Environment for Knowledge Analysis (WEKA) software suite [17]. The WEKA software[4] supports a collection of machine learning algorithms, together with preprocessing functionalities that facilitate preparing our datasets for training and testing. Next, we fed the training model we obtained from WEKA into the simple Naïve Bayes classifier that we implemented in Tor in order to perform classification on the fly.

Experimental setup. To collect training data, we configured three client types: a BitTorrent client, a web browsing client and a streaming client. All our clients run from a single machine and are forced to choose an OR under our control as an entry node for all their circuits, as we performed our measurements at our entry node.[5] Next, we describe how we set up our clients and we explain how we carry out our measurements safely without endangering the privacy of other users.

Automating the web browsing client. In order to generate realistic browsing traffic, we use the iMacros Firefox plugin [2] to automate the process of web browsing. The automated browser works as follows: First, it picks a random URL from the list of the top 100 URLs reported by Alexa [1] and starts downloading the web page. Then, after the page loads, the browser waits for a random amount of time that is selected from the distribution of user think times [18]. If the URL chosen is a search engine, then the browser types a keyword, which is also randomly selected from the top 100 search terms reported by Alexa, in the search box. Finally, after the results are loaded, the browser follows a random link from the top 5 search link results. This process is repeated in a loop of several iterations.

BitTorrent client. In order to generate P2P file sharing traffic, we used the Vuze BitTorrent client [3], which allows us to configure the proxy port on which our Tor client is listening. Note that Vuze contains some explicit options to use the Tor network for tracker and file transfer downloads and uploads. We used these options to send all our Vuze client traffic through Tor. We capped the maximum upload and download rates to 2000 KB/s. During our experiments, we noticed that our BitTorrent client easily achieves around 1000 KB/s. We sampled our torrent files from popular legal torrent websites. Our downloads included music, movies, and some Linux distribution files.

Streaming client. Again, we use the iMacros Firefox plugin in order to automate a streaming client. The client searches for videos using a random keyword selected from the top 100 search keywords reported by Alexa. Then the client selects a random video link from the returned results and watches for a random time between 1 to 5 minutes.[6] This wait time captures scenarios where the client watches part of the video and then either browses away to another page, or chooses to finish watching the whole video before searching for another keyword. Note that the streaming activity includes some browsing activity as well.

MeasureMe cell. Because we collect data and traffic statistics on the live Tor network using our entry OR, it is important to ensure that we do not log statistics of other users' traffic as this might reveal their private information. To achieve this goal, we implemented a new relay command cell type, which we call the "MeasureMe" cell. This cell is sent from the client to any OR on the circuit to instruct it to start logging measurements on that particular circuit. To help distinguish which application uses which circuit, we configured each client with a different MeasureMe ID, which is sent to the measuring OR in the MeasureMe cell.

In our experiments, when each client starts building circuits, it will also send a MeasureMe cell to our entry node after a circuit is fully constructed. On receiving the MeasureMe cell, our node starts logging the classification attributes we described in Section 4.2 for the respective circuit.

5.1 Data Collection

Over a period of 6 weeks, from early March to mid-April 2012, our BitTorrent, automated browsing and streaming clients downloaded approximately 24 hours worth of traffic. Because we logged the MeasureMe id along with circuit id and its attributes, we were able to divide our traffic logs into three different application logs corresponding to each class we defined. We then extracted the circuits and their attributes of each class, and eliminated outlier circuits which downloaded only a few cells. From the logs, we created an offline data set and three online data sets and converted them to the Attribute-Relation File Format (ARFF) format for the WEKA processing. We next describe our offline and online datasets

Offline data set. We sampled approximately 200 circuits from the three application traces. Of the 200 circuits, 122 were browsing circuits,[7] 49 were BitTorrent circuits and the remaining 28 were streaming circuits. We then extracted the circuit-level attributes we used for offline classification: the circuit lifetime, the amount of data sent upstream and downstream, and the variance of the cell inter-arrival times of a circuit.

Online data set. From our application traces, we also extracted the cell attributes that were computed online for every cell that the entry node transferred for our clients. Each instance of our online data set is composed mainly of cell-level attributes such as the inter-arrival time of the current cell, its mean and variance. Also, an instance contains some circuit-level attributes such as the EWMA of the count of the cells sent recently on a circuit. We created three different ARFF files using the same data set, but for every file, we used different classes:

- TBS: This file contained sampled attributes from BitTorrent, browsing and streaming circuits. Each data instance was labelled with its respective class. Approximately 60% of the instances were extracted from browsing circuits, 20% were from BitTorrent, and the remaining 20% were from streaming circuits.
- TN: This file contained the same data as TBS, except that browsing and streaming instances were labelled in common as "NotBulk", while BitTorrent instances remained separate as "Bulk".

[4] We used version 3.6.6 for all our experiments.

[5] During the process of data collection, we capped our node's bandwidth to 500 KB/s.

[6] User think times are known to be larger for YouTube sessions than for traditional web workloads [16].

[7] Note that the Tor client uses on average 6 circuits per hour for browsing, so 122 circuits is approximately 20 hours' worth of browsing.

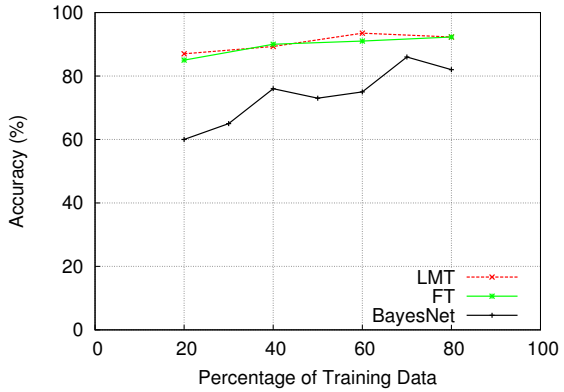


Figure 5: Accuracy of different classification algorithms used for the offline dataset when varying the percentage split of training data relative to the whole dataset

- BT: This file contained only the subset of browsing and BitTorrent instances from TBS.

5.2 Evaluation Metrics

To evaluate our classification algorithms, we utilized two widely used machine-learning metrics, the *accuracy* of the classifier, and the *F-measure*. We use the accuracy to evaluate the overall classification accuracy, whereas the F-measure is used in our evaluation as a per-class evaluation metric.

Accuracy reflects the percentage of the testing instances that have been classified correctly out of all instances:

$$Accuracy = \frac{TP + TN}{N} \tag{5}$$

where N is the total number of samples. The F-measure is defined as the harmonic mean of *precision* and *recall*, which are defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

where TP, FP and FN are the true positives, false positives and false negatives of a test, respectively. Therefore, the F-measure is given by:

$$F\text{-}measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{8}$$

Intuitively, a score of 1 for the accuracy or the F-measure indicates a perfect classifier.

5.3 Offline Classification

In this section, we present our results for the Bayes Nets, LMT and FT classifiers, which we overviewed in section 4.3, on our offline dataset. Figure 5 shows how the accuracy of the classifiers changes as we vary the percentage of the training data relative to the whole dataset. The figure shows that the accuracy of Bayes Nets seems to improve from 60% when training data is only 20% of the offline dataset to above 85% when the training data is around 70% of the offline dataset. The Bayes Net classifier starts to degrade to around 82% as the percentage of training data increases to 80%. This suggests that more data than 70% of the offline dataset can add noise to the classifier. FT and LMT provide better accuracy, and are very similar to each other. With around 60% training data, each classifiers provides us with greater than 90% accuracy.

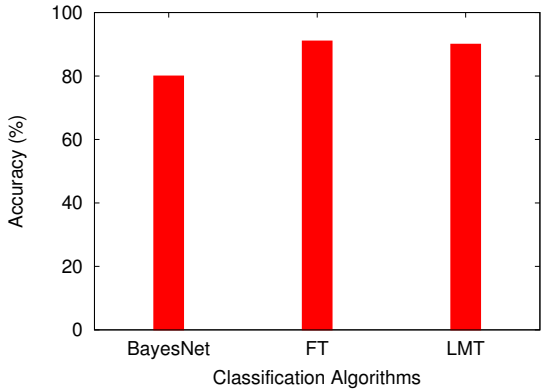


Figure 6: Accuracy of different classification algorithms used when 10-fold cross-validation is used on the offline datasets

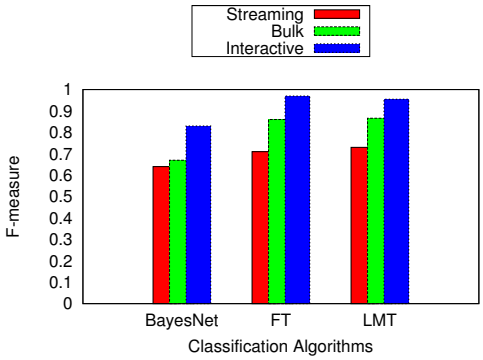


Figure 7: F-measure of the different traffic classes when using different classification algorithms with 10-fold cross-validation on the offline dataset

Figure 6 summarizes the accuracy of the classifiers when 10-fold cross-validation is used on our offline dataset. In the n-fold cross-validation, the training data is divided into n subsets. One subset is used for testing and the other $n - 1$ subsets are used for training. This process is repeated n times so that in every iteration, a new subset is used for testing. The advantage of this method is that the whole dataset is used for both training and testing, and the n results can be averaged. As seen in the figure, the FT classifier achieves 91% accuracy when 10-fold cross-validation is used.

Figure 7 depicts the F-measures of the different classes when different classifiers are used in combination with 10-fold cross-validation. Again, FT provides the highest F-measures for all of the classes. It can also be seen that all three classifiers perform well in identifying a browsing circuit, as their F-measure for the browsing class ranges from 0.83 for Bayes Nets to 0.97 for FT. FT and LMT classifiers also perform very well in identifying BitTorrent circuits. However, they both achieve around 0.71 for the streaming class.

5.4 Online Classification

We next present our classification results for our three online datasets. For the TBS dataset, we used both Bayes Nets and Naïve Bayes. Figure 8 shows the classifier accuracy as we vary the percentage of the training subset of TBS. With only 30% training percentage, Bayes Nets classifies most cells with an accuracy of 95%. Naïve Bayes on the other hand provides a very poor accuracy on the TBS dataset which does not exceed 31% regardless of the percentage of the training data. Figure 9 shows the accuracy of Bayes Nets and Naïve Bayes classifiers on the TBS dataset when 10-fold cross-validation is performed. Bayes Nets provide a 97.8% accuracy while Naïve Bayes provides a 31% accuracy.

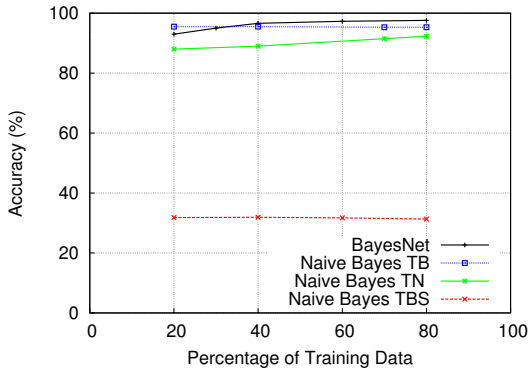


Figure 8: Accuracy of different classification algorithms on the online datasets when varying the percentage split of the training data relative to the whole dataset

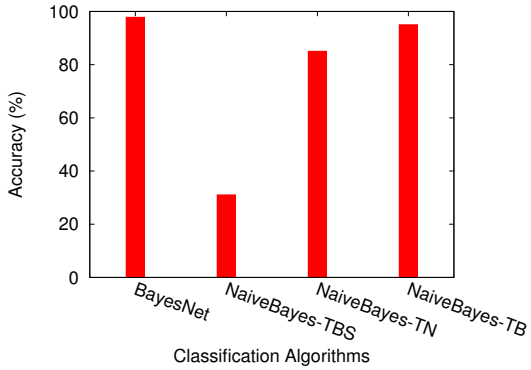


Figure 9: Accuracy of different classification algorithms used when 10-fold cross-validation is used on the online datasets

However, the accuracy of Naïve Bayes jumps to 85% if used to classify the TN dataset. This shows that the Naïve Bayes classifier is good in identifying the bulk class, and other non-bulk classes. The Naïve Bayes classifier further shows more accuracy on the TB dataset which reaches 95%. It can be seen from these results that the Naïve Bayes classifier performs best when we eliminate the streaming class training, as it is an excellent classifier in detecting the bulk and the interactive classes. This is evident from the F-measures we obtained for the bulk and interactive classes shown in Figure 10. Table 1 summarizes a high-level comparison between our threshold, offline, and online classifiers.

5.5 Live Tor Experiment

In order to show the performance benefits that are made possible using traffic classification, we implemented a Naïve Bayes classifier in the latest release of the Tor source code.[8] We used the training model we obtained from WEKA to train our classifier. In particular, we trained our classifier to recognize two traffic classes, the interactive and the bulk transfer classes. The reason we chose this classifier is twofold. First, as previously discussed, the largest application that is used on Tor is web browsing, and the second largest is BitTorrent. Therefore, our classifier should be able to identify the largest two classes of traffic that use the Tor network. Secondly, online classification requires classification at the cell level; therefore, it is important to ensure that our classifier is efficient and does not add any extra load on the operation of an OR. A simple Naïve Bayes classifier fits those two conditions, and its running costs are minimal.

[8] We used tor-0.2.2.35 for our experiments.

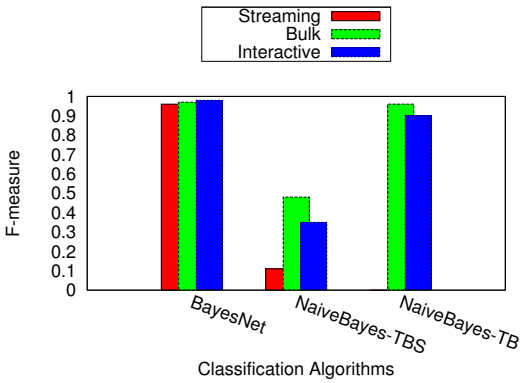


Figure 10: F-measure of the different traffic classes when using different classification algorithms with 10-fold cross-validation on the online dataset

We deployed a Tor entry node,[9] which runs our classification implementation, on the live Tor network in April 2012. We also implemented the following simple QoS rule.[10] If our OR classifies at least three circuits in one client connection as bulk, then the OR reacts by throttling the client connection to 50 KB/s. The OR decides the class of the circuit after classifying the first 3000 cells that travel downstream in the circuit towards the client. The OR decides that a circuit is bulk if the number of cells classified as bulk are at least two times the number of cells classified as interactive.

Recall from Section 5 that we *only* monitor our own traffic, so as to not accidentally deanonymize real Tor users.

We found that our classifier was able to classify a bulk and interactive circuits accurately after classifying approximately 2000 and 1000 cells, respectively; however, we chose the 3000 cell mark in order to have more confidence in our classification decision. Furthermore, although 3000 cells might sound like a large number of cells, we argue that 3000 cells are transferred by bulk applications in a matter of a few seconds.

We ran two Tor clients from the same machine. Our first client is the Vuze BitTorrent client, which acts as the bulk class traffic generator. Its setup remains similar the one described in Section 5. The second client, the interactive class traffic generator, runs a curl script in a loop to download a 300 KB file from an external server through Tor using our entry node as its first hop. The client pauses randomly for 3 to 30 seconds and then sends another download request to the external server to start downloading. Note that the behaviour of this interactive client is slightly different from the automated web browser we used to train our classifier because of the absence of the effects of dynamic contents, for example.

Figures 11 and 12 depict the performance of our interactive class user when the OR runs stock Tor and when it runs DiffTor. As can be seen in Figure 11, the download time — the time it takes the Tor client to complete downloading a 300 KB file — is significantly improved. While the download time that the client experiences is 16 seconds at the median without QoS, the median download time is substantially improved to 2.2 seconds when QoS is employed at the OR. Likewise, the time-to-first-byte, the time it takes the client to receive the first chunk of data[11] after issuing a request, is also significantly reduced from 3.5 seconds at the median for the non QoS case to 0.9 seconds when QoS is used. Table 2 shows the overall

[9] To induce congestion, we capped its bandwidth to 200 KB/s.

[10] Note that our live experiment is a proof-of-concept, rather than a precise throttling rule that should be used in practice. Because we showed in previous sections the possibility to classify Tor circuits accurately to three classes, one has the flexibility to define different QoS techniques to react to different classes.

[11] From a user perspective, this is an important metric because it shows how long the user is expected to wait before he sees changes to the browser.

Table 1: Comparison of threshold, online and offline classification methods

Classification type	*Classification level*	*Accuracy*	*Attribute(s)*
Threshold	Connection-level	90%	Number of circuits per connection
Offline	Circuit-level	91%	Data transferred, CIT variance, and circuit lifetime
Online	Cell-level	97%	CIT statistics and EWMA of cells sent

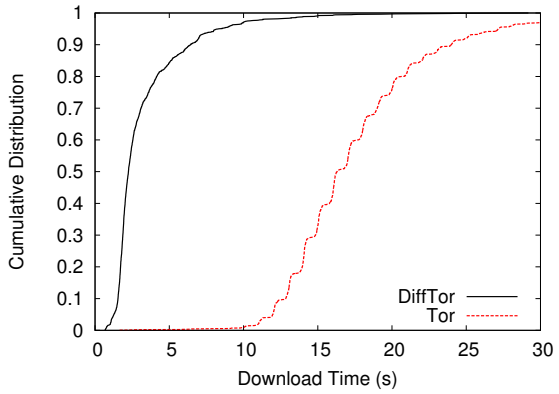


Figure 11: Comparison of the 300 KB download times that the web client experiences with and without QoS

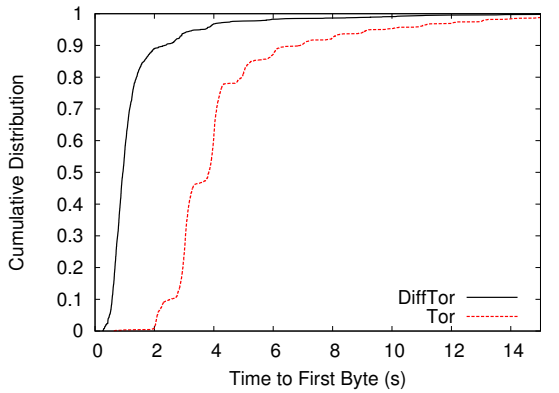


Figure 12: Comparison of the time-to-first-byte that the web client experiences with and without QoS

and per-class accuracy of the live experiments. To summarize our results, the download time is improved by 86% and the time-to-first-byte is improved by 75% at the median for the web browser. As for the BitTorrent client, the download rate was successfully throttled to 50 KB/s as intended by our example QoS rule.

6. DISCUSSION

Below we discuss a variety of issues such as the deployability of our scheme on the deployed Tor network and the effects of human variability on the performance of our techniques. We also express our thoughts on the effects of gaming our classification techniques and on the security and privacy implications of traffic classification in Tor. Finally, we talk about open issues that we plan to investigate as future work.

6.1 Incremental Deployment

Our work is currently applicable to entry guards and is incrementally deployable. Setting QoS parameters can be done locally by OR operators or globally by directory authorities. Although we have only demonstrated how we can implement a simple QoS rule that throttles a client connection when it detects a bulk transfer at an entry guard, our work extends naturally to middle and exit ORs.

Table 2: Overall and per-class accuracy of live experiments

Overall	*Bulk class*	*Interactive class*
77%	74%	95%

While it is infeasible for a middle or an exit OR to throttle connections for the sake of implementing QoS,[12] we can still implement QoS rules for middle or exit routers at the circuit scheduling level (as opposed to the connection level) to reduce the effects of congestion on interactive applications; this could be particularly useful to counter bulk downloaders who run their own (non-throttling) entry nodes. Furthermore, we suspect that QoS can provide even further improvements if combined with fair queuing on all circuits [38]. However, maintenance will be sometimes required: as the behaviour of applications change, and with continuous changes to the Internet, the classifier will need to be retrained to maintain its accuracy.

6.2 Human Variability

Because we used automated tools to generate web and streaming traffic in our data collection phase, one might wonder if our techniques would generalize to a broad range of users who use applications differently. In our work, we are confident that we addressed this issue in our realistic traffic generation and in our selection of solid attributes. We strived to generate as much realistic traffic as possible by exploiting previous studies that estimate user think times for both streaming and interactive applications, and by using the most popular URLs and keywords. Second, we also chose attributes that are strong enough to prevail even in face of user variability. Recall that our attributes identify burstiness, the variance of burstiness, and the amount of data sent recently. Those are very high-level application signatures that should not be different for different humans because they are influenced by how an application protocol behaves.

For example, even if different users employ different BitTorrent clients, our techniques will still identify the behaviour of the BitTorrent protocol. The BitTorrent protocol will always attempt to create several streams which would create many circuits and result in what looks like a continuous downloading circuit. While there is more variability in the interactive class since it involves human interaction, such variability will only affect the think times observed, but will not change the fact that interactive applications are bursty.

6.3 Gaming the Classifier

Because we perform classification at the circuit and cell levels, some users may change the behaviour of their applications to avoid certain QoS measures. For example, if an entry node throttles bulk transfers, then BitTorrent users might shape their traffic so that it is not classified correctly as bulk. This attempt to disguise one class of traffic's statistical features (e.g., packet sizes and timing) as another traffic class's features, called *traffic morphing* [39], has been proposed as a way to defeat statistical traffic classification techniques. We note that, even when armed with traffic morphing techniques that are informed with prior knowledge about the clas-

[12] Recall that an OR-to-OR connection multiplexes circuits from several users.

sification features and the training model, it is often the case that identifying features persist that reveal the true application [12].

With regard to our classification approaches, there are two scenarios to consider. First, to game the offline classification, in addition to shaping their traffic to look more bursty, BitTorrent users would have to send significantly less traffic upstream and downstream, and they would need to discontinue using the circuit within 10 minutes. We argue that shaping their traffic in this way would still be a win to the Tor network since that would significantly reduce congestion. Second, to game the online classification, if the classifier makes a classification decision periodically, and changes the QoS parameters accordingly, then BitTorrent clients would still need to continuously shape their traffic to look like interactive circuits, which again is a win to the network. However, BitTorrent clients can still use several circuits over the Tor network to achieve the desired aggregate bandwidth. Preventing such cheating users from obtaining an unfair bandwidth share is an area for future work.

6.4 Security and Privacy Implications

There are two security implications that traffic classification raises in Tor. First, an exit may be able to reduce the anonymity set of the entry guard using the circuit if it observes changes in the service. The extent of how much information is revealed depends on the QoS parameter defined. For example, if an entry guard throttles only bulk downloads, then the anonymity of bulk download users may be reduced. However, as more entry guards upgrade, it will be more difficult for an exit to guess who the entry guard is.

Second, since guards are directly connected to users, classifying their circuits can reveal private information about the activities of the users and how they use Tor. Since Tor mainly welcomes interactive real-time applications and since the majority of Tor circuits carry browsing traffic, then bulk download users actually deviate from the behaviour of the majority of circuits and thereby endanger their privacy. Actually, it is already well known that the anonymity of BitTorrent users on Tor is questionable, so classification does not really add further damage to the anonymity of this class of users [8]. Our work is another disincentive for greedy users to deviate from typical network usage.

6.5 Future Work

There are still some interesting areas for investigation in our work. First, as the network grows and starts to attract more users, we expect that more applications will emerge in the Tor network and we will therefore need to define more classes of service and investigate more attributes and algorithms. Second, it would be interesting to compare how the different machine learning algorithms trade off cost with accuracy. Another open research question that we would like to answer is how to use the classification techniques we presented in this paper to infer more information on how users currently use Tor, and the percentage of users of each traffic class. There are currently efforts and proposals to safely collect data on the live network to estimate the number of users [27]. It would be interesting to also log circuit information in a privacy-preserving manner to be able to perform offline classification and understand the current usage of the Tor network. Finally, another area of future work is to investigate different QoS techniques on different traffic classes and perform a whole-network performance evaluation on a Tor network testbed such as ExperimenTor [6] or Shadow [20].

7. RELATED WORK

Network traffic classification has been studied intensively in the networking literature [7,24,30,34,40]. Traditional port-based techniques of traffic classification are almost phased out because some applications allocate ports dynamically. Moreover, in addition to their legal and ethical concerns, techniques that are based on deep packet inspection are also ineffective with the use of encryption. Therefore, researchers propose different features extracted from the transport level, packet level and connection level and apply different machine learning techniques to be able to classify traffic to different categories.

Applying the previous solutions to classify traffic in the Tor network is more challenging for two reasons. First, most of the useful features that have been used previously are not available in the context of Tor. Examples include the client and server connection ports used and the packet sizes, which are fixed in Tor. Second, unlike IP network classifications, where packet inter-arrival times is a useful feature, we found that timing features are sometimes fragile in Tor because of the varying circuit and relay characteristics. The problem gets even more challenging as we try to realize real-time traffic classification for Tor.

From the Tor literature, Panchenko *et al.* [33] describe an attack where the adversary is a local eavesdropper that uses machine learning to infer information from the encrypted traffic sent by a Tor client regarding the web sites that the client visits. The authors first assumed a closed-world model where an attacker trains a classifier with some URLs that the client is known to visit. Later, the classifier is used to figure out what URLs the client is requesting. They further assume an open-world model where the attacker only wants to find out if the client is visiting a restricted website. For both models, the authors show a high true positives and low false positives.[13] In our work, we try to classify circuits to different classes depending on the application using the circuit mainly for performance improvement goals. Also, while the features used by Panchenko *et al.* are very useful, we cannot benefit from the packet sizes for online classification at the entry guards, for example, as Tor uses fixed-sized cells.

Other related work in the Tor literature was introduced by Moore *et al.* [31] and Jansen *et al.* [22]. From a high level, both proposals are similar in the sense that they both attempt to improve the performance of interactive applications in Tor by throttling bulk transfers. This is done by throttling all connections between the client and the entry guard using Tor's already-implemented token-bucket system. The approach of Moore *et al.* is to apply a certain limit that would slightly affect web browsing users, but would greatly slow down bulk transfers. The only way for clients to avoid the throttling is by donating bandwidth to the Tor network by running as relays. This incentives-based approach and other similar approaches such as BRAIDS [21], which also provides differentiated services, and the gold star scheme [32] are impractical as it is difficult to expect that Tor clients would run relays, and applying it may result in discouraging clients from using the network.

Jansen *et al.* propose and investigate three algorithms that adjust or throttle the connection between an entry guard and a client. Their most effective algorithm, nicknamed the threshold algorithm, maintains an EWMA value of the cells sent on a client connection. This value is used to sort circuits from loudest to quietest, with the goal of throttling a loudest threshold of all connections.

We note that the problem with this approach is that it would unnecessarily throttle time-sensitive streaming applications and because the threshold is high,[14] it may also throttle interactive circuits. Another problem with the threshold algorithm is that it is based only on the EWMA of the cells sent recently. Our experiments revealed that this value alone is sometimes unreliable as

[13] In response to this attack, the Tor browser bundle enables HTTP pipelining and randomizes the pipeline size and the size of requests.

[14] The authors report that a threshold of 90% yields the best results.

interactive circuits sometimes have large values of EWMA while they are downloading web pages.

The advantage of DiffTor over previous approaches is that applying QoS is more flexible since we define different classes of service. For instance, an entry guard can choose to give a defined small percentage of its bandwidth to BitTorrent, and assign more bandwidth to the other two classes. However, when the node is currently not serving streaming or web browsing, it can offer more bandwidth to the bulk transfer circuit, thereby not wasting bandwidth.

8. CONCLUSION

Motivated by the crucial need to improve the performance of the Tor network, we propose a machine-learning-based approach to provide differentiated services. We recognize that the current main traffic classes that use the Tor network have different constraints that can be addressed by defining appropriate QoS measures. Based on our study of traffic logs (of our own network usage) we obtained from the live Tor network, we derive useful attributes and use them in combination with well-known classification algorithms to classify our traffic logs with a very high accuracy. Furthermore, we implement our classifier in Tor and define a simple QoS rule to test our approach on the live network. Our results show high classification accuracy that results in significant improvements for the experience of interactive Tor users.

Acknowledgements. We thank the anonymous reviewers for their helpful comments and suggestions. We thank NSERC, ORF, Qatar University, and The Tor Project, Inc. for funding this research.

9. REFERENCES

[1] Alexa: The Web Information Company. `http://www.alexa.com/topsites`, 2012. Accessed February 2012.

[2] iMacros for Firefox. `https://addons.mozilla.org/en-US/firefox/addon/imacros-for-firefox/`, 2012. Accessed February 2012.

[3] Vuze: The Most Powerful BitTorrent App on Earth. `http://www.vuze.com`, 2012. Accessed February 2012.

[4] ALSABAH, M., BAUER, K., GOLDBERG, I., GRUNWALD, D., MCCOY, D., SAVAGE, S., AND VOELKER, G. M. DefenestraTor: Throwing out Windows in Tor. In *11th Privacy Enhancing Technologies Symposium* (July 2011), pp. 134–154.

[5] AULD, T., MOORE, A., AND GULL, S. Bayesian Neural Networks for Internet Traffic Classification. *IEEE Transactions on Neural Networks 18*, 1 (January 2007), 223–239.

[6] BAUER, K., SHERR, M., MCCOY, D., AND GRUNWALD, D. ExperimenTor: A Testbed for Safe and Realistic Tor Experimentation. In *Proceedings of the 4th USENIX Workshop on Cyber Security Experimentation and Test (CSET)* (August 2011), pp. 51–59.

[7] BERNAILLE, L., TEIXEIRA, R., AKODKENOU, I., SOULE, A., AND SALAMATIAN, K. Traffic Classification on the Fly. *SIGCOMM Comput. Commun. Rev. 36*, 2 (April 2006), 23–26.

[8] BLOND, S. L., MANILS, P., CHAABANE, A., KAAFAR, M. A., CASTELLUCCIA, C., LEGOUT, A., AND DABBOUS, W. One Bad Apple Spoils the Bunch: Exploiting P2P Applications to Trace and Profile Tor Users. In *Proceedings of the 4th USENIX Conference on Large-scale Exploits and Emergent Threats (LEET)* (March 2011).

[9] CHAABANE, A., MANILS, P., AND KAAFAR, M. A. Digging into Anonymous Traffic: A Deep Analysis of the Tor Anonymizing Network. In *Proceedings of the 4th International Conference on Network and System Security (NSS)* (September 2010), pp. 167–174.

[10] DINGLEDINE, R. Tor and Circumvention: Lessons Learned. In *Proceedings of the 31st Annual Conference on Advances in Cryptology (CRYPTO)* (August 2011), pp. 485–486.

[11] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium* (August 2004), pp. 303–320.

[12] DYER, K. P., COULL, S. E., RISTENPART, T., AND SHRIMPTON, T. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy* (May 2012), pp. 332–346.

[13] EARLY, J. P., BRODLEY, C. E., AND ROSENBERG, C. Behavioral Authentication of Server Flows. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC)* (December 2003), pp. 46–56.

[14] FRIEDMAN, N., GEIGER, D., AND GOLDSZMIDT, M. Bayesian Network Classifiers. *Mach. Learn. 29*, 2-3 (November/December 1997), 131–163.

[15] GAMA, J. Functional Trees for Classification. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)* (November 2001), pp. 147–154.

[16] GILL, P., ARLITT, M., LI, Z., AND MAHANTI, A. Characterizing User Sessions on YouTube. In *Proceedings of the 15th IEEE Multimedia Computing and Networking (MMCN)* (January 2008).

[17] HALL, M., FRANK, E., HOLMES, G., PFAHRINGER, B., REUTEMANN, P., AND WITTEN, I. H. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl. 11*, 1 (November 2009), 10–18.

[18] HERNÁNDEZ-CAMPOS, F., JEFFAY, K., AND SMITH, F. D. Tracking the Evolution of Web Traffic: 1995-2003. In *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunication Systems (MASCOTS)* (2003), pp. 16–25.

[19] HERRMANN, D., WENDOLSKY, R., AND FEDERRATH, H. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier. In *Proceedings of the ACM Cloud Computing Security Workshop (CCSW)* (November 2009), pp. 31–42.

[20] JANSEN, R., AND HOPPER, N. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *Proceedings of the 19th Network and Distributed Security Symposium* (February 2012).

[21] JANSEN, R., HOPPER, N., AND KIM, Y. Recruiting New Tor Relays with BRAIDS. In *Proceedings of the 17th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2010), CCS '10, ACM, pp. 319–328.

[22] JANSEN, R., SYVERSON, P., AND HOPPER, N. Throttling Tor Bandwidth Parasites. In *21st USENIX Security Symposium* (August 2012).

[23] KARDI, T. Recursive Average and Variance. `http://people.revoledu.com/kardi/tutorial/RecursiveStatistic/index.html`, 2006. Accessed January, 2012.

[24] KIM, H., CLAFFY, K., FOMENKOV, M., BARMAN, D., FALOUTSOS, M., AND LEE, K. Internet Traffic

Classification Demystified: Myths, Caveats, and the Best Practices. In *Proceedings of the ACM CoNEXT Conference* (December 2008), pp. 11:1–11:12.

[25] LANDWEHR, N., HALL, M., AND FRANK, E. Logistic Model Trees. *Mach. Learn. 59*, 1-2 (May 2005), 161–205.

[26] LIBERATORE, M., AND LEVINE, B. N. Inferring the Source of Encrypted HTTP Connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS)* (October 2006), pp. 255–263.

[27] LOESING, K. Safely collecting data to estimate the number of Tor users. `https://lists.torproject.org/pipermail/tor-dev/2010-August/000467.html`, 2012. Accessed April 2012.

[28] MAIER, G., FELDMANN, A., PAXSON, V., AND ALLMAN, M. On Dominant Characteristics of Residential Broadband Internet Traffic. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement* (November 2009), pp. 90–102.

[29] MCCOY, D., BAUER, K., GRUNWALD, D., KOHNO, T., AND SICKER, D. Shining Light in Dark Places: Understanding the Tor Network. In *Proceedings of the 8th Privacy Enhancing Technologies Symposium* (July 2008), pp. 63–76.

[30] MOORE, A. W., AND ZUEV, D. Internet Traffic Classification using Bayesian Analysis Techniques. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (June 2005), pp. 50–60.

[31] MOORE, W. B., WACEK, C., AND SHERR, M. Exploring the Potential Benefits of Expanded Rate Limiting in Tor: Slow and Steady Wins the Race with Tortoise. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC)* (December 2011), pp. 207–216.

[32] NGAN, T.-W. J., DINGLEDINE, R., AND WALLACH, D. S. Building Incentives into Tor. In *Proceedings of the 14th International Conference on Financial Cryptography and Data Security (FC)* (January 2010), pp. 238–256.

[33] PANCHENKO, A., NIESSEN, L., ZINNEN, A., AND ENGEL, T. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)* (October 2011), pp. 103–114.

[34] ROUGHAN, M., SEN, S., SPATSCHECK, O., AND DUFFIELD, N. Class-of-service Mapping for QoS: A Statistical Signature-based Approach to IP Traffic Classification. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement* (October 2004), pp. 135–148.

[35] SANDVINE. Sandvine Global Internet Phenomena Report — Fall 2011. `http://www.sandvine.com/downloads/documents/10-26-2011_phenomena/Sandvine%20Global%20Internet%20Phenomena%20Report%20-%20Fall%202011.pdf`, October 2011.

[36] TANG, C., AND GOLDBERG, I. An Improved Algorithm for Tor Circuit Scheduling. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS)* (October 2010), pp. 329–339.

[37] THE TOR PROJECT. Tor Metrics Portal: Data. `https://metrics.torproject.org/data.html#performance`. Accessed January 2012.

[38] TSCHORSCH, F., AND SCHEUERMANN, B. Tor is unfair — And what to do about it. In *Proceedings of the 36th IEEE Conference on Local Computer Networks (LCN)* (October 2011), pp. 432–440.

[39] WRIGHT, C., COULL, S., AND MONROSE, F. Traffic Morphing: An Efficient Defense against Statistical Traffic Analysis. In *Proceedings of the 16th Network and Distributed Security Symposium* (February 2009).

[40] ZUEV, D., AND MOORE, A. W. Traffic Classification using a Statistical Approach. In *Proceedings of the 6th International Conference on Passive and Active Network Measurement* (2005), PAM'05, pp. 321–324.