Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis

Charles V. Wright[1]
MIT Lincoln Laboratory
Lexington, MA
cvwright@ll.mit.edu

Scott E. Coull
Johns Hopkins University
Baltimore, MD
coulls@cs.jhu.edu

Fabian Monrose
University of North Carolina
Chapel Hill, NC
fabian@cs.unc.edu

Abstract

Recent work has shown that properties of network traffic that remain observable after encryption, namely packet sizes and timing, can reveal surprising information about the traffic's contents (e.g., the language of a VoIP call [29], passwords in secure shell logins [20], or even web browsing habits [21, 14]). While there are some legitimate uses for encrypted traffic analysis, these techniques also raise important questions about the privacy of encrypted communications. A common tactic for mitigating such threats is to pad packets to uniform sizes or to send packets at fixed timing intervals; however, this approach is often inefficient. In this paper, we propose a novel method for thwarting statistical traffic analysis algorithms by optimally morphing *one class of traffic to look like another class. Through the use of convex optimization techniques, we show how to optimally modify packets in real-time to reduce the accuracy of a variety of traffic classifiers while incurring much less overhead than padding. Our evaluation of this technique against two published traffic classifiers for VoIP [29] and web traffic [14] shows that morphing works well on a wide range of network data—in some cases, simultaneously providing better privacy and lower overhead than naïve defenses.*

1 Introduction

Network traffic analysis is an increasingly common means of identifying security threats and providing efficient management of network resources, both within local networks and the Internet. Unfortunately, these same analysis techniques often lead to violations of user privacy. The typical answer to these privacy concerns is to simply encrypt the data traversing the network in order to limit the information available to traffic analysis. However, this alone is not enough since several features of the encrypted network data, such as packet sizes and timing, can still leak information about the traffic. By quantizing these features (e.g., padding packets to fixed sizes), the amount of information that is leaked can be minimized, but at the cost of degrading the efficiency and performance of the underlying network protocols. Certainly, one can pad all encrypted packets such that their sizes are always equal to that of the maximum transmission unit (MTU), but for many network protocols doing so would more than double the amount of data sent. For these protocols, such excessive padding is simply not a satisfactory solution to the problem.

Moreover, the performance of encrypted network protocols often takes precedence over privacy concerns in practical applications. While it may be possible to allow users to tune the tradeoff between efficiency and privacy to their liking, there is often no clear meaning in terms of the levels of privacy and performance associated with such actions. As a consequence of this bias towards efficiency, several security-oriented network protocols have been found to leak more information about the underlying data than originally thought. For instance, work by Sun et al. [21] and Liberatore and Levine [14] has shown that the identities of web pages can be inferred by examining the sizes of packets within an encrypted HTTP connection. More recently, Wright et al. [29] showed that packet lengths of encrypted Voice over IP (VoIP) calls can leak information about the languages being spoken.

In this paper, we propose a new method for optimally balancing privacy and efficiency through the use of mathematical programming (i.e., optimization) methods; specifically, convex optimization. At a high-level, our optimization problem is defined as follows. Given a source process, such as downloading a specific web page via HTTP, our goal is to *morph* the process such that it maximally resembles some target process (i.e., a different web page) with respect to a given set of features. Of course, we must ensure that the morphing we perform is also maximally efficient for some measure of efficiency, such as the number of bytes of overhead

[1]Work performed while at Johns Hopkins University.

added. For the remainder of this paper, we focus on the use of our morphing techniques in thwarting traffic classifiers that utilize features based on packet sizes.

As an example of the usage of our technique, consider a general web page classifier that uses packet sizes to determine the identity of web pages. If the user connects to *www.webmd.com* to search for medical information over an encrypted connection where packet sizes are not padded, the web page classifier would examine these sizes and determine that the user has indeed gone to *www.webmd.com*. The approach we take allows the user (with the cooperation of the web server or proxy) to morph her download to appear as a different web page (e.g., *www.espn.com*) to the classifier. Our morphing technique takes in each packet from the source web page as it is produced in *real-time*, and adds padding to the packet or splits the packet into multiple smaller packets. The change in packet size is performed in such a way that the download looks like the target web page with respect to the distribution of packet sizes over the entire HTTP session. As a result, the classifier will examine the morphed traffic and erroneously classify it as a download of *www.espn.com*, and only a minimal amount of additional overhead will be incurred in confusing the classifier.

To evaluate our method, we examine the efficacy of morphing against two known traffic classification techniques: the VoIP language classifier of Wright et al. [29] and the web page classifier of Liberatore and Levine [14], both of which examine packet sizes. Our results show that the morphing method is able to reduce the VoIP classifier's accuracy from 71% to 30% with only 15.4% overhead on average. Likewise, the accuracy of the web page classifier is reduced from 98.4% to 4.5% with 38.9% overhead. In addition, we evaluate the performance of our method against classifiers that are aware of the morphing technique and are allowed to adjust their training to compensate. Even in this more difficult scenario the results indicate that the VoIP classifier is unable to distinguish between the morphed and unaltered traffic, while the web page classifier performs only 26.8% better than random guessing. In contrast, padding the packets to the MTU of 1500 bytes would incur an overhead of 156% while still allowing the web classifier to distinguish web pages with an accuracy of 72.4% better than random guessing!

The remainder of the paper is organized as follows. We begin by discussing related work in Section 2. In Section 3, we present the convex optimization techniques used to determine the optimal way of morphing one process to another. We evaluate our morphing technique against the VoIP language classifier of Wright et al. [29] and the web page classifier of Liberatore and Levine [14] in Section 4, and finally, we conclude in Section 5.

2 Related Work

Several recent papers have shown that sensitive information can often be extracted from encrypted network traffic by examining patterns in the sizes of packets and their timing. Song et al. [20] showed that the interarrival times of packets in SSHv1 connections can be used to infer information about the user's keystrokes and thereby reduce the search space for discovering login passwords. Sun et al. [21] identified web pages within SSL–encrypted connections by examining the sizes of the HTML objects returned in the HTTP response. Moreover, they demonstrated that this technique was surprisingly robust against countermeasures such as padding the object sizes. Later work by Liberatore and Levine [14] showed that a similar attack is still possible, with some modifications, for HTTP traffic that uses persistent connections or that is tunneled through SSH port forwarding. Wright et al. [29] showed that the statistical distribution of packet sizes in encrypted Voice over IP (VoIP) connections can be used to identify the language spoken in a conversation when the voice stream is encoded using certain kinds of variable bit rate compression schemes. Work by Saponas et al. [17] showed that variable bit rate encoders for streaming video leak information in a similar fashion, allowing an observer in the network to identify movies within encrypted connections. More recent work by Wright et al. [28] showed how an eavesdropper could identify spoken phrases in encrypted VoIP.

Much of the work in security on misleading machine learning techniques has focused on defeating host-based intrusion detection systems. Wagner and Dean [26] proposed the idea of a *mimicry attack* on intrusion detection systems, whereby an adversary exploits a vulnerable program and then uses a legitimate sequence of system calls to perform some harmful action. Wagner and Soto [27] then presented a concrete instance of a mimicry attack on the host-based IDS `pH` [19]. They showed that by inserting system calls that do not perform any meaningful computation an attacker can make a malicious program mimic the sequences of system calls used by its victim program. Similar attacks were independently discovered and demonstrated against the `stide` IDS [11] by Tan et al. [22].

More closely related is the work of Fogla et al. [10], who developed techniques by which an adversary can automatically modify the payload of packets to deceive a network intrusion detection system. Their "polymorphic blending" technique enables an attacker to evade detection by altering the byte sequences in polymorphic shellcode to look like normal, non-malicious payloads. Unlike our work, their method deals with packet payloads rather than network-layer characteristics, and they have the advantage of knowing *a priori* exactly what ma-

licious payload they need to encode. In contrast, our traffic morphing technique must be able to handle arbitrary input traffic as it is generated; especially for real-time streaming traffic that encodes input from a user.

Newsome et al. [16] take a different approach to defeating statistical classifiers. Rather than modifying the features of the traffic to match a signature, they showed how an adversary can instead cause changes in the signatures to better match her traffic by maliciously injecting training data to change the detector's notions of benign and malicious traffic. Nelson et al. [15] show an instantiation of the attack in [16] for bypassing spam filters. Similarly, Chung and Mok [3, 4] showed that an adversary could abuse sensors that automatically generate signatures for zero-day worm outbreaks to cause denial of service for legitimate traffic. Venkataraman et al. [24] showed fundamental limits on the accuracy that can be achieved by pattern matching algorithms in such adversarial settings. Cretu et al. [7] then presented techniques for cleansing the training data to remove anomalous or adversarially-inserted data before applying learning techniques for signature generation.

3 Traffic Morphing

The goal of traffic morphing is to provide users who encrypt their network data with an efficient method of preventing information leakage that induces less overhead than deterministic padding (i.e., quantization). To do so, we must ensure that the distribution of packet sizes emitted by the source process maximally resembles the distribution of sizes for the intended target process, while simultaneously minimizing the overhead incurred by the alteration of sizes. In addition, the morphing algorithm should operate on the packets produced by the source process in an online fashion with a minimum of added latency. This ensures that the morphing algorithm can handle a variety of different types of traffic, including real-time streaming media, without requiring any changes to the program or a priori knowledge of how the packets will be generated aside from their distribution.

At a high level, traffic morphing operates as follows. First, the user chooses the source processes that she would like to protect using traffic morphing, as well as a (potentially different) set of target processes that she would like to make the source processes look like. These processes can be any of a variety of traffic classes, such as web pages, languages in VoIP calls, or even different applications. For each pair of source and target process, the user applies optimization techniques, which we describe throughout this section, to derive a *morphing matrix* that dictates how each packet size from the source process should be altered so that the resultant distribution maximally resembles the target with a minimum of overhead. Intuitively, after the matrices are generated offline, the morphing algorithm acts as a proxy between the applications and the network stack, thereby intercepting the data to be sent across the network before headers are calculated or encryption is applied. The morphing algorithm refers to the appropriate morphing matrix for the source process, and pads the data (or splits it into several smaller packets) according to the matrix. The altered data is then sent to the network stack, encrypted, and sent across the network as usual.

In the following sections we describe a set of techniques that can be selectively applied to perform morphing on a variety of network traffic. We begin by laying out the notation and general algorithm for morphing traffic in Section 3.1. In Section 3.2, we describe the basic convex optimization algorithm used to find a morphing matrix that minimizes a given cost function, such as the number of bytes of overhead. Section 3.3 describes how we can augment the basic optimization algorithm to constrain the way in which the packets can be morphed, like restricting the morphing to only increase the size of the packets. We describe a divide and conquer method for making our optimization method feasible even on arbitrarily large sample spaces in Section 3.4. Finally, in Section 3.5 we discuss several potential issues that may arise in practice, and methods to overcome them.

3.1 What is the Matrix?

Given a sample space of n possible packet sizes, we describe the source process as a column vector representing a probability mass function over the n sizes $X = [x_1, x_2, \ldots, x_n]^T$, where x_i is the probability of the i^{th} largest packet size under the source process. Similarly, we denote the target distribution as $Y = [y_1, y_2, \ldots, y_n]^T$. To morph X to Y, the user must find a positive $n \times n$ morphing matrix $A = [a_{ij}]$ such that $Y = AX$ and each column of A is a valid probability mass function over the n packet sizes. For each pair of packet sizes s_i and s_j, the matrix A gives the probability of morphing a packet of size s_j in the source distribution to size s_i in the target distribution as the cell at row i and column j, denoted a_{ij}. Thus, the probability distribution dictated by the j^{th} column of the matrix describes, probabilistically, how an input packet of size s_j should be altered to achieve an output distribution that resembles that of the target process. As we will see in the next section, convex optimization techniques can be used to find the matrix A that morphs the source process while minimizing a cost function, such as the number of bytes of overhead.

When the morphing algorithm receives a packet of size s_j from the source application, it looks at the j^{th} column in the appropriate morphing matrix and samples a target size s_i using the probability distribution from that column, then alters the packet's size to match s_i.

To sample the target size, the algorithm first sums the probabilities into a cumulative distribution function such that the cumulative probability of target size s_i is equal to the sum of the probabilities for all sizes $\leq s_i$. Then, the algorithm runs a pseudorandom number generator to get a random number $r \in [0, 1]$, and selects first target size with a cumulative probability $\geq r$. Once the target size s_i is sampled, the algorithm pads the data with zeros if $s_i > s_j$, or splits the data into multiple packets if $s_i < s_j$, and then sends the data to the network stack as usual. The process is repeated each time new data is received from the application, and the packet sizes that are output onto the network appear to be drawn from the target distribution Y.

As long as the source process generates a sufficiently large number of packets from a distribution that closely resembles the source distribution, X, the output of the morphing will converge in distribution to that of the target, Y. Moreover, the use of convex optimization techniques ensure that the matrix used to morph the traffic as described above induces the minimum overhead in altering the source packet sizes to match the target distribution. Additionally, the generation of the matrices can be performed offline before the morphing algorithm is used, and therefore the only operation that adds latency to the process is generating random numbers to sample from the distribution for each input packet size. In most cases, the pseudorandom number generation adds only negligible delay, and so the morphing process as a whole incurs a minimum of latency.

3.2 Morphing via Convex Optimization

Initially, the form of the equation $Y = AX$ might lead us to think of solving for the matrix A as solving a system of linear equations. Clearly, because A is an $n \times n$ matrix, we have n^2 unknowns (i.e., all a_{ij}). From the equation $Y = AX$, we can expand A, X, and Y into their components

$$\begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ & & \dots & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \quad (1)$$

and by performing the matrix multiplication, we get n equations of the form

$$y_i = a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \quad (2)$$

Furthermore, because the column vectors of A must each sum to 1 in order to describe valid probability mass functions, we also have n equations of the form

$$a_{1j} + a_{2j} + \cdots + a_{nj} = 1 \quad (3)$$

All together, we have $2n$ equations in n^2 unknowns. This defines an underspecified system of equations, and so there are an infinite number of solutions. Given this infinite variety of morphing matrices that solve the equation $Y = AX$, we frame the question of how to select the *best* solution as an optimization problem in n^2 variables.

In a mathematical optimization problem [2], the goal is to minimize some cost function $f_0(A)$, subject to m constraints given by functions $f_k(A) \leq b_k$, and constants b_k. The cost function $f_0 : R^{n^2} \rightarrow R$ describes whatever criterion the user decides on to pick the optimal solution, and the constraint functions $f_k : R^{n^2} \rightarrow R$ describe the requirements imposed on the eventual solution.

There are well-known algorithms in the optimization literature for solving any *convex* optimization problem in polynomial time when the cost function f_0 and all the constraint functions f_k are convex functions. In some cases, such as when f_0 and f_k are all *linear* functions, fast solutions based on linear programming exist. Other convex optimization problems can generally be solved using methods based on gradient descent. While the details of convex optimization algorithms are outside the scope of this work, we refer the interested reader to a full-length text on the subject, such as that of Boyd and Vandenberghe [2]. For our purposes, it is sufficient to point out that any local minimum of the cost function will also be the global minimum, and that for strictly convex functions there is at most one global minimum. Thus, the optimization algorithms for these problems are guaranteed to provide the global minimum in polynomial time. There are a number of widely available solvers for these kinds of optimization problems, and in our experiments we use the open source solver interface provided by Dahl and Vandenberghe [8].

An Example In the case of morphing network traffic with respect to packet sizes, we can define the cost function f_0 as the expected number of additional bytes that the user must transmit when using the morphing matrix A. We denote the overall cost of matrix A as

$$f_0(A) = \sum_{\forall i,j \in [1,n]} x_j a_{ij} (|s_i - s_j|) \quad (4)$$

where s_j and s_i are the sizes in bytes for the source j and target i, x_j is the probability of the source size s_j, and a_{ij} is the probability of morphing s_j to s_i. Additionally, we have $2n$ equality constraint functions of the form of Equations (2) and (3), which ensure that the matrix A is a valid morphing matrix. Another n^2 inequality constraints come from the fact that each entry in the A matrix must be a valid probability; that is, $0 \leq a_{ij}, \forall i, j \in [1, n]$.

This optimization problem can then be written in the standard form:

$$\begin{array}{lll} \text{minimize} & f_0(A) & \\ \text{subject to} & \sum_{j=1}^{n} a_{ij}x_j = y_i, & \forall i \in [1,n] \\ & \sum_{i=1}^{n} a_{ij} = 1, & \forall j \in [1,n] \\ & a_{ij} \geq 0, & \forall i,j \in [1,n] \end{array} \quad (5)$$

An important observation here is that, in this case, the constraints for the above optimization problem allows for morphing matrices that can *reduce* the size of packets. Certainly, the data itself cannot be reduced, but there are several methods by which we can either downgrade the quality of the data in streaming protocols or simply split the data among several packets. We discuss specific methods to deal with reduced packet sizes in Section 3.5.

3.3 Additional Morphing Constraints

As alluded to earlier, the user may choose to specify additional constraints to restrict the type of morphing performed to preserve the quality of the data or minimize the number of packets produced (e.g., only allowing packets to increase in size). To do so, she can add equality constraints to specify that the cells where $s_i < s_j$ are set to zero (i.e., there is no probability of performing a morphing from a large packet to a small one). One potential pitfall of adding so many constraints, however, is that the user runs the risk of creating an overspecified system of equations where there is *no* valid solution to Equation (1).

In these cases, the convex optimization must compromise the fidelity of the distribution produced by the morphing in order to satisfy the specified cost function and its constraints. That is, we must now determine a way to *optimally* balance the fidelity of the output distribution provided by the morphing algorithm to the actual target distribution with the efficiency of the morphing and compliance with the specified constraints. In this section, we show how she can overcome this problem by optimizing for multiple cost functions in an iterative manner, known as *multilevel programming* [25]. Specifically, the user first finds a matrix that creates a morphed output distribution that is as close as possible to that of the target distribution, yet satisfies all of her constraints. Then, the user takes that morphed distribution and finds another matrix that minimizes the amount of overhead produced in creating it. With our use of multilevel programming, we can optimize for any number of cost functions and derive a morphing matrix even when the system of equations used in the optimization problem is overspecified.

Our general strategy for determining a valid morphing matrix in these situations is to first derive an approximate target distribution that is as close as possible to Y with respect to some comparison function f_d while still meeting all of the constraints of the optimization. More formally, we use the convex optimization techniques described above to find a matrix A' such that $Z = A'X$, $f_d(Y,Z)$ is minimized, and all constraints on A' are met. The vector $Z = [z_1, z_2, \ldots, z_n]^T$ represents the closest distribution that X can be morphed to given the constraints on the optimization (i.e., only allowing padding). Once we have derived the approximation to the target distribution Z, we then run the convex optimization algorithm again to find a second matrix A such that $Z = AX$, the cost function f_0 described above is minimized, and all constraints are met.

An Example As a concrete example, we return to the case of performing morphing on packet sizes where we are constrained to only padding packets. As previously described, adding the constraints to ensure that our morphing matrix only pads packets leads us to an overspecified system, and so we must use our multilevel programming approach to generate a valid morphing matrix in these cases. We define our comparison function to be the χ^2 statistic:

$$f_d(Y,Z) = \sum_{i}^{n} \frac{(z_i - y_i)^2}{y_i^2} \quad (6)$$

In practice, any number of potential convex comparison functions could be used, including L1 distance, the χ^2 statistic, and Euclidean distance, among others. The first optimization problem in our multilevel optimization is written as follows.

$$\begin{array}{lll} \text{minimize} & f_d(Y,Z) & \\ \text{subject to} & \sum_{i=1}^{n} a'_{ij} = 1, & \forall j \in [1,n] \\ & a'_{ij} \geq 0, & \forall i,j \in [1,n] \\ & a'_{ij} = 0 & \forall i,j : s_i < s_j \end{array} \quad (7)$$

The final constraint in this optimization problem states that the cells of the matrix where a large packet size would be morphed to a smaller size must be zero to ensure that packet sizes can only be increased. Notice that, unlike in Equation (5), we do not require that $A'X$ be exactly equal to Y (i.e., that $\sum_{j=1}^{n} a_{ij}x_j = y_i$ for all $i \in [1,n]$). This allows the solver to search across many different values of $Z = A'X$ that meet all of our constraints, in order to find the one closest to Y.

The second optimization problem, in which we derive the morphing matrix A that most efficiently maps X to Z, is then written as:

$$\begin{array}{lll} \text{minimize} & f_0(A) & \\ \text{subject to} & \sum_{j=1}^{n} a_{ij}x_j = z_i, & \forall i \in [1,n] \\ & \sum_{i=1}^{n} a_{ij} = 1, & \forall j \in [1,n] \\ & a_{ij} \geq 0, & \forall i,j \in [1,n] \\ & a_{ij} = 0 & \forall i,j : s_i < s_j \end{array} \quad (8)$$

Here, the cost function $f_0(A)$ is the expected number of bytes of overhead as described in Equation (4) above.

3.4 Dealing with Large Sample Spaces

The astute reader would surely have notcied that one issue that we have seemingly ignored thus far is the impact of the size of the sample space on the practicality of finding optimal morphing matrices. Since the number of constraints in our optimization problem grows quadratically in the number of symbols we are morphing, the complexity of finding morphing matrices when n is large may become prohibitively high. For example, some network protocols produce any of the range of packet sizes found on an Ethernet network (i.e., 40 to 1500 bytes). In this case, our morphing matrix would be 1460×1460 in size, and would have more than two million constraints. By comparison, our test platform (a Pentium 4 2.8GHz machine with 8GB of RAM) requires just over one hour to solve an 80×80 optimization problem with 6,560 constraints. Clearly, the growth in the number of constraints would render the solutions presented thus far untenable for protocols where n is large, or when we are trying to morph higher-order ℓ-gram models (i.e., sequences of ℓ packet sizes) with n^ℓ possible symbols.

Intuitively, to tackle the problems associated with morphing arbitrarily large sample spaces, we apply a divide and conquer strategy. In doing so, we trade off the global optimality of our technique by finding the morphing matrices for subspaces of the large space independently of one another, rather than finding a single global morphing matrix. At a high level, we take the large space of n symbols and divide it into $m << n$ partitions of size n/m each. Then, we derive a coarse morphing matrix that morphs from the m input partitions to the m output partitions, as well as m^2 submatrices that dictate how we morph the n/m input symbols to the n/m output symbols in the respective subspaces. Notice that this procedure can be generalized to recursively create many levels of submatrices to handle arbitrarily large spaces.

Formally, we take the original source and target vectors X and Y with length n, and divide the symbols equally into m partitions. Thus, we create new source and target vectors X' and Y' with length m defining the marginal probabilities of the partitions. The symbols used to represent the partitions could be the averages of the values it contains, the median value, or any other coarse representation of the symbols contained within the partition. The symbols chosen to represent the partitions at this coarse level will be used to approximate the cost of morphing from one partition to another (e.g., by calculating the overhead). In addition, we also create the vectors $X_1, \ldots, X_m$ and $Y_1, \ldots, Y_m$, where the vector X_i (respectively, Y_i) dictates the probability distribution of symbols contained in the i^{th} partition. The vectors X_i and Y_i can be interpreted as the probability of the values conditioned on being in partition i. Using the vectors X' and Y', we find a morphing matrix A such that $Y' = AX'$. Similarly, for each pair of vectors X_j,Y_i we find the submatrix $A_{(ij)}$ such that $Y_i = A_{(ij)}X_j$. The multilevel programming methods shown in Section 3.3 can be applied to these optimization problems to add constraints on the morphing process.

An Example Let us examine how we might apply submatrix morphing to morph bigram distributions (i.e., ordered pairs). We start with source and target vectors X and Y of length n^2, which represent the probability distribution over all ordered pairs of packet sizes, (s_a, s_b). The original space is partitioned to create a coarse space that defines the marginal probabilities of the first size in the pair (i.e., $P(s_a)$), and subspaces that define the conditional probability of the second size conditioned on the first (i.e., $P(s_b|s_a)$). As such, we create the vectors X' and Y' of length n for the probabilities of the first packet sizes. To morph X' to Y', we find a morphing matrix A using the following optimization problem:

$$\begin{array}{lll} \text{minimize} & f_0(A) & \\ \text{subject to} & \sum_{j=1}^{n} a_{(ij)} x'_j = y'_i, & \forall i \in [1,n] \\ & \sum_{i=1}^{n} a_{(ij)} = 1, & \forall j \in [1,n] \\ & a_{(ij)} \geq 0, & \forall i,j \in [1,n] \end{array} \quad (9)$$

As with our previous optimization problems, we let the cost function f_0 be the expected number of bytes of overhead. The resultant morphing matrix A defines how we should morph single packets to match the marginal, or unigram, distribution.

In addition, we also create $2n$ vectors X_i and Y_i each of length n. The vector X_i (respectively, Y_i) represents the conditional probability of the n packet sizes conditioned on the first packet size in the bigram being s_i. For all pairs of vectors (X_j, Y_i), we find a morphing matrix $A_{(ij)}$ (with cells denoted as $a_{(ij),(hk)}$ for column k and row h) with the optimization problem:

$$\begin{array}{lll} \text{minimize} & f_0(A_{ij}) & \\ \text{subject to} & \sum_{k=1}^{n} a_{(ij),(hk)} x_{j,k} = y_{i,h}, & \forall h \in [1,n] \\ & \sum_{h=1}^{n} a_{(ij),(hk)} = 1, & \forall k \in [1,n] \\ & a_{(ij),(hk)} \geq 0, & \forall h \in [1,n], \\ & & \forall k \in [1,n] \end{array} \quad (10)$$

Again, the cost function is the expected number of bytes of overhead, and these submatrices tell us how to morph a packet *conditioned* on how the previous packet was morphed, thereby ensuring a consistent bigram distribution. Notice that when morphing a bigram distribution we have two distinct modes of operation. At the start of a new connection, there is no previous packet, so we must use the matrix A to morph the first packet to match the unigram distribution of the target process. For the following packets, when the previous packet was morphed from size s_j to size s_i, we can use the submatrix A_{ij} to

tell us how to morph the current packet. A similar procedure using submatrices can be used to morph traffic where the number of distinct packet sizes is large (e.g., general Ethernet traffic) by first clustering the sizes into a small number of equivalence classes, and then deriving submatrices for the sizes within those classes.

3.5 Practical Considerations

Beyond the technicalities discussed earlier, there are a number of additional considerations that must be addressed in order to apply our morphing technique to a broad class of network traffic. Here, we describe some potential pitfalls that one may encounter in practice, and provide guidance on how these may be addressed.

Short Network Sessions The underlying principle behind our morphing method is that as the number of packets produced by the source process approaches infinity, the output of our morphing approach converges in distribution to the target. For many network protocols, such as streaming video, Voice over IP, or file sharing, it is safe to assume that the number of packets generated will be quite large. VoIP, for instance, can generate hundreds of packets in only a few seconds. There are, however, a number of protocols that are not guaranteed to generate a sufficient number of packets. A prime example being HTTP where a preponderance of web page downloads have relatively few packets.

One way to address this is by keeping track of how similar the output distribution is to the target, and requiring that it reach some threshold level of similarity before the morphing process is terminated. Specifically, we can record the distribution of packet sizes produced by the morphing and simply calculate the L1 distance between the output and target distributions once the source process has stopped generating packets. If the distance is greater than some prescribed threshold, we then generate packets by sampling from the target distribution. This process continues until the distance between the distributions reaches the threshold. Obviously, there is a point at which the overhead from the generated packets negates the savings derived from the morphing process, and so in those cases it is often more efficient to use deterministic padding. That said, the results from our evaluation of morphing on web traffic show that our approach still achieves significant overhead savings even when applied to web page downloads with relatively few packets (i.e., < 100).

Variations in Source Distribution The output distributions produced by our morphing method are only guaranteed to converge to the correct target distribution if the distribution being produced by the source process closely resembles that which was used to create the morphing matrices. Naturally, if there is significant variability in the distribution, then the performance of our morphing method will degrade. This variation may include producing packets that were not observed in the distribution used to create the matrices, or substantial changes in the probabilities for the sizes.

The solution to this problem is to adapt the divide and conquer approach found in Section 3.4. Like the divide and conquer approach, we partition the space of sizes in the original distributions into m equivalence classes by clustering them. The assumption here is that the clusters represent canonical equivalence classes where the packet sizes within them are essentially interchangeable when they are produced by the source or target process. The partitions allow us to find a consistent, coarse morphing matrix that describes how these equivalence classes can be morphed. However, the variability in the sizes makes it impossible to create consistent submatrices for these classes as we would have done in Section 3.4. Instead, we keep track of the distribution of packet sizes in each of the target process classes and sample from those distributions to determine the specific packet size to output. Thus, when we morph a packet of size s_j, we find the nearest source class, use the coarse morphing matrix to determine the target class to morph to, and finally sample from the chosen target class' distribution to find the output size.

Reducing Packet Sizes Although it is possible to restrict the morphing process to only increase the size of packets, doing so necessarily reduces the performance of our technique. When this restriction is lifted, however, we face the task of redistributing or altering the data in the packet to allow the size to be reduced. For streaming data, such as video or audio data, the quality of the encoding can be reduced to shrink the packet size. For other network protocols, like HTTP, we are forced to redistribute the original data into several packets. Thus, if we morph a packet of size s_j to size s_i, with $s_j > s_i$, then we first send a packet of size s_i which leaves us with $s_j - s_i$ bytes remaining. We do not morph these leftovers because it is difficult to predict their sizes *a priori*, and so the morphing matrix A would not handle them optimally. Instead, to send the remaining data, we sample directly from the target distribution without performing any morphing to determine what size packet to send next, and place as much of the remaining data as possible into the newly generated packet. This process continues until all the data from the original packet is sent.

4 Evaluation

In what follows, we empirically evaluate the efficacy of our morphing technique against two existing traffic classifiers that are able to learn non-trivial amounts of information about encrypted traffic. First, we show how morphing can be used to efficiently thwart the technique of Wright et al. [29] for identifying spoken languages in VoIP, and the Bayesian classifier used by Liberatore and Levine [14] to identify web pages in encrypted traffic. Specifically, we show our morphing technique significantly reduces the accuracy of the classifiers by causing them to misidentify traffic as a class of our choosing. Next, we consider a far more difficult test wherein the classifiers are aware of the morphing technique and allowed to adjust their training to compensate. That is, the classifiers are trained on the original classes, as well as morphed versions of those classes, and are then asked to distinguish between the morphed and original traffic classes.

Our evaluation focuses on the ability of the morphing technique to deceive a *binary* classifier that distinguishes between two classes of traffic. Since binary classifiers are generally able to achieve greater accuracy than classifiers with $k > 2$ classes, we argue that if the morphing algorithm can thwart the binary classifiers then it would also be able to deceive a k-way classifier and would have many different ways to do so.

4.1 Encrypted Voice over IP

In our earlier work [29], we showed how a passive observer in the network could automatically identify the language spoken in Voice over IP connections that were compressed using a variable bit-rate (VBR) codec like Speex [23] and encrypted using the IETF standard Secure Real-time Transport Protocol [1]. Moreover, the results of our empirical evaluation showed that the most straightforward countermeasure against this attack—padding packets to a common length—incurred so much overhead that, from a pragmatic point of view, it completely nullifies the benefits of using the variable bit-rate compression. We now revisit the problem of protecting VoIP from eavesdroppers, and present a more elegant and efficient solution to disguise the language in use.

To evaluate our technique, we used the same CSLU "22 Language" [13] telephone speech corpus examined in [29]. As before, the entire corpus was encoded using Speex narrowband mode, which uses 9 distinct packet sizes. We observed the resulting sequences of packet sizes, and then attempted to morph the bigram (i.e., 2-gram) distribution of each of the 21 languages in the data set to each of the other languages. Specifically, for each pair of languages, source and target, we use two variations of the techniques presented in the previous section to morph the bigram distribution of the source to that of the target.

We investigate the performance of our morphing technique against increasingly powerful versions of the classifier, in order to determine what a classifier would need to do to be more robust against our new countermeasures. First, in Section 4.1.1, we show that traffic morphing is able to defeat the original χ^2 language classifier that is unaware that the traffic may be morphed. Then, in Section 4.1.2, we investigate the effectiveness of morphing in a much more difficult test where the classifier is aware of the morphing technique's existence and, by training on morphed traffic, seeks to distinguish it from normal traffic.

In each case, to account for the randomness inherent in the morphing technique, we perform the morphing process five times for each pair of source and target languages, and record the five resulting streams of packet sizes. Following the original evaluation in [29], we use leave-one-out cross-validation to estimate the accuracy of the binary classifier on this morphed data.

Whitebox vs Blackbox Morphing The Speex encoder is based on a technique called Code Excited Linear Prediction (CELP) [18], in which the sound in each packet is compressed by searching a codebook of excitation vectors for the codebook entry that best reproduces the input speech. The index of the best-matching codeword is then sent out over the network as part of the packet payload. In variable bit rate mode, some preliminary analysis is performed on the sound in each packet to select between several codebooks of varying size before the search is performed. Larger codebooks give higher-fidelity representation of the audio, but require larger packets (and hence higher bit rates). Where in this process we apply the morphing procedure is important for both the efficiency and the fidelity of the resulting morphed traffic. Therefore, we investigate the impact of two slightly different approaches for morphing VoIP traffic.

In the first variation, which we call *white box* morphing, all modifications to packet size occur in the codec after a bit rate has been selected for the packet but before the actual compression has been performed. Figure 1 shows a block diagram of the process for compressing, morphing, and transmitting a VoIP packet under this paradigm. Morphing components shown shaded grey. Note that, in this variation, we can either increase or decrease the size of the packet by modifying the encoder's bit rate. To find an optimal set of morphing matrices, we apply the technique presented in Section 3.4 for each pair of languages to create a coarse morphing matrix A and 81 submatrices $A_{(ij)}$.

In the second variation, shown in Figure 2, we assume only *black box* knowledge of the codec, so that morphing must be performed after compression has

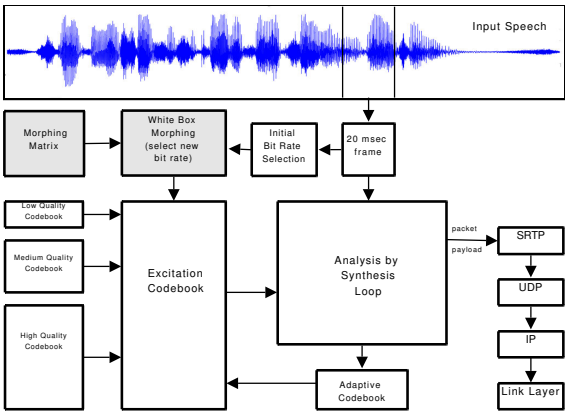

Figure 1. White box morphing for VoIP

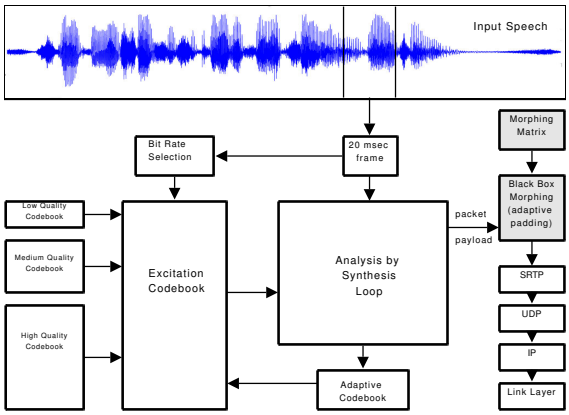

Figure 2. Black box morphing for VoIP

completed but before the packet is encrypted. In this second case, we can only increase the size of our packets by adding padding, and so we must always solve for morphing matrices with the constraint that $a_{ij} = 0$ for $s_i < s_j$. Consequently, in addition to using the divide and conquer method of Section 3.4, we also apply the multilevel programming methods described in Section 3.3 to add the padding constraints.

4.1.1 Defeating the Original Classifier

In this section, we investigate the reduction in accuracy caused by our morphing method first on the binary classifier examining bigrams, and then on the binary classifier examining trigram distributions. The complementary cumulative distribution functions (CCDFs) shown in Figures 3 and 4 show the results for the bigram and trigram classifiers, respectively.

With no countermeasures applied, the bigram classifier achieves an average accuracy of 71% for speakers of the source languages. However, its accuracy is significantly degraded on morphed traffic. With black box morphing, the classifier's average accuracy falls to 45%, with only 5% of the language pairs scoring better than 75% accuracy. Under the white box model, the classifier's average accuracy falls even further, to just 30%.

Overall, the trigram classifier's results are similar. With no countermeasures applied, its accuracy improves to 76%. Additionally, on morphed data, for some language pairs, it is significantly more effective than the bigram classifier; its maximum accuracy under white box morphing is 90%, while the bigram version was never over 65%. However, for most pairs of source and target languages, its accuracy is not significantly better than the bigram classifier. Specifically, its average accuracy falls to 47% under black box morphing, and to just 38% under white box morphing.

In this experiment, one expects our morphing technique to reduce the accuracy of the classifiers to a minimum of one minus the accuracy of the unmorphed data. The reason for this is that if the classifiers naturally confuse certain classes of traffic, then our morphing method will change those classes that are confused in the unmorphed data to actually be correctly classified with the morphed data. With that in mind, our white box morphing model achieves nearly the best possible reduction in accuracy against both the bigram and trigram classifier, while the black box method achieves only a slightly less notable reduction in accuracy.

4.1.2 Evaluating Indistinguishability

Having shown that traffic morphing can be used to defeat the original classifier from [29], we now consider a much stronger experiment where the classifier is aware of the morphing technique and how it works. The goal of the classifier is to distinguish traffic that has been morphed to look like a given target language from genuine instances of that language. To do so, the classifier is trained not only on normal traffic from the target language, but also on traffic from the source language which has been morphed to look like the target.

We note that, if the morphing is perfect, and the morphed traffic looks exactly like normal traffic, then the classifier can do no better than random guessing. In this case, the average accuracy would be 50%. Of course, the simplest way to make two languages indistinguishable is to make all packets the same size, by padding smaller packets up to the size of the largest. Therefore, for comparison, we also show the classifier's accuracy on traffic that has been padded to multiples of 512 bits, which makes all VoIP packets the same size. The results of these experiments are shown as CCDFs in Figures 5 and 6. Table 1 shows the overhead incurred by each countermeasure.

Figure 5 shows that the bigram classifier is not able to reliably distinguish morphed traffic from normal. In fact, white box morphing offers nearly the same privacy

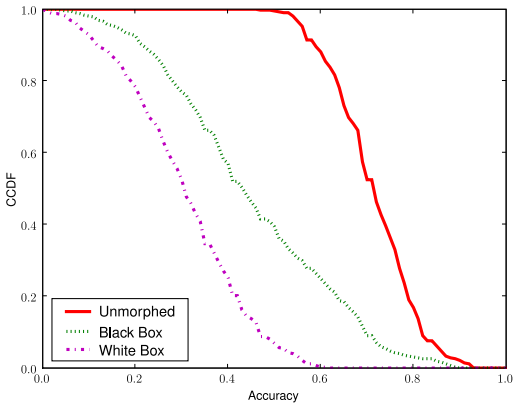

Figure 3. CCDF of bigram classifier results when morphing the bigram distrbution

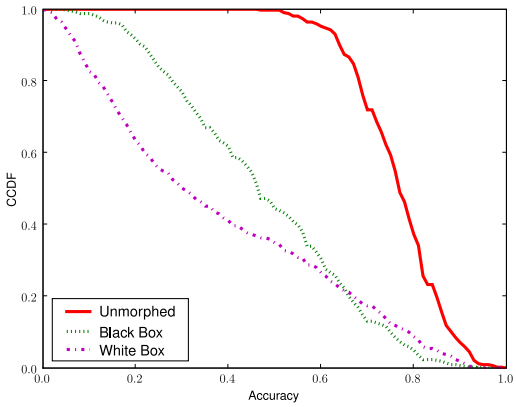

Figure 4. CCDF of trigram classifier results when morphing the bigram distribution

provided by padding all packets to the same size, but with less than half as much overhead. Black box morphing, on the other hand, is less effective at providing indistinguishability, though the overhead is substantially lower than padding.

The results change dramatically, however, for the trigram classifier, as seen in Figure 6. Through the use of knowledge of the morphing technique and a higher-order model than that used by the morpher, the trigram classifier is able to recognize languages in morphed traffic as accurately as on unmodified data. It is interesting to note that neither the higher-order model nor knowledge of the morphing technique alone is sufficient for resisting whitebox morphing; both appear to be necessary for constructing a truly robust classifier. We provide further discussion of the implications of these results in Section 4.3.

4.2 Web Page Identification

The web page classifier of Liberatore and Levine [14] showed that it is possible to accurately identify a web page that has been downloaded over an SSH tunnel using only the size of the packets and the direction in which they traveled (i.e., from or to the client). The mitigation strategies suggested, like those of Wright et al. , focused on the use of various padding techniques to limit the amount of information available to the classifier. The results of experiments with these padding techniques showed that significant information about web page identity was leaked even when padding all packets to have a size equal to the maximum transmission unit (MTU) of 1500 bytes. The reason for this surprising result is that even though all packets are of the same length, the proportion of packets sent by the server and client still identifies the web page. Through the use of our morphing method, however, we significantly improve the privacy of the web pages while simultaneously reducing overhead by more than half.

To classify web pages, Liberatore and Levine assume that the observer examines the sizes of packets sent over an encrypted SSH tunnel. From these packet sizes, the classifier derives *attributes* of the form $(direction, size)$ for each packet. For each unique $(direction, size)$ tuple, the classifier keeps a count for the number of packets with that attribute sent during a single download of a given web page, called an *instance*. Several instances of the web pages that the observer would like to detect are used to train a naïve Bayes classifier with Gaussian kernel estimation. Our evaluation makes use of the packet trace data set that was released by Liberatore and Levine. Additional information about the classifier and data set can be found in the original paper [14].

In order to facilitate a clean evaluation of our morphing technique, we modify the experimental setup used by Liberatore and Levine classifier to instead train and test on normalized distributions rather than raw counts of each packet type, or attribute. The reasons for this modification are two-fold. First, from a practical standpoint, several recent works [12, 6] have shown that accurately parsing web page downloads from interleaved web traffic is extremely difficult in practice, and so, in reality, it is unlikely that such a classifier would be able to generate accurate counts for the observed packet types. Second, our morphing method could be modified to morph distributions of raw counts by a number of means. Some naïve solutions include splitting single web page downloads into several smaller downloads so that the number of packets sent is correct, or generating extra packets from the target distribution to ensure a sufficient number of packets was sent. Consequently,

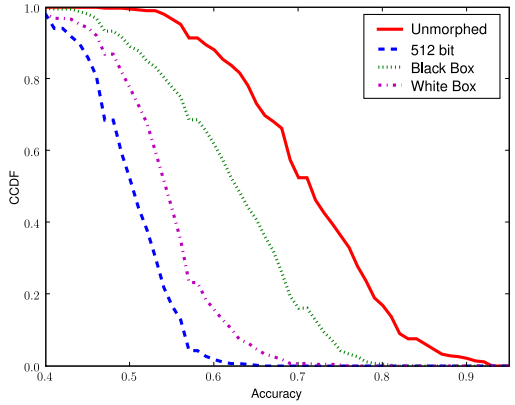

Figure 5. CCDF of indistinguishability results for the bigram classifier when morphing bigram distributions

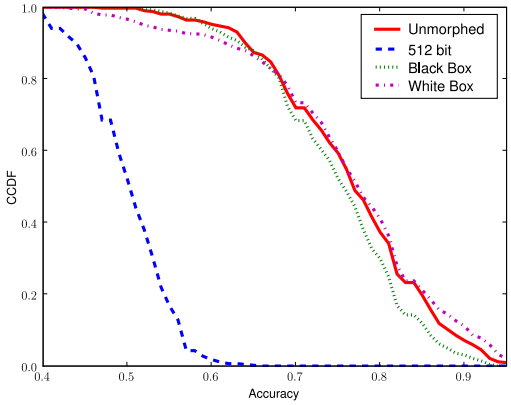

Figure 6. CCDF of indistinguishability results for the trigram classifier when morphing bigram distributions

Strategy	Overhead (%)	Bigram Accuracy (%)	Trigram Accuracy (%)
No Padding	0.0	71	76
Black Box	18.7	62	74
White Box	15.4	54	76
512 bit	42.2	50	50

Table 1. Results for morphing against the language classifier

in the case of raw distributions, our evaluation would be unduly influenced by our choice of method for accommodating for these raw distributions, which is not the primary focus of this paper. This change in setup allows us to focus our evaluation on the morphing technique at hand.

Our evaluation examines the naïve Bayes classifier of Liberatore and Levine trained on the first ten days of data for each of the top 50 sites in the data set. Both the classifier and our morphing technique are trained on odd-numbered days, and tested on even-numbered days. We create two morphing matrices, one for each direction of the connection (i.e., client and server), for each pair of sites in the data set using the techniques discussed in Section 3.5 to reduce source distribution variability. Thus, we morph the packets in real time on each side of the connection independently using the respective matrices. Recall from Section 3.5 that web pages may generate only a small number of packets, or the distribution of packet sizes generated may differ considerably from that of the data used to create the morphing matrices To accommodate these peculiarities in our evaluation, we implement the methods described in Section 3.5, including clustering packet sizes into equivalence classes and sampling from the target distribution to ensure that the output and target distributions are sufficiently similar. Specifically, we require that the output distribution's distance to the target be less than a threhold $t = 0.3$ according to the L1 distance measure[1]. In essence, we require that the two distributions differ by at most 30% of their respective probability mass functions. As in the VoIP test, we morph the data five times to reduce the impact of the random sampling on the results.

4.2.1 Defeating the Original Classifier

In our first test, we focus on the ability of the morphing technique to reduce the accuracy of the classifier trained solely on unaltered web pages. As shown by the complementary cumulative distribution function (CCDF) in Figure 7, our morphing technique greatly reduces the accuracy of the classifier. Specifically, the classifier achieves an average accuracy of 98.4% on the unaltered data, but its accuracy is reduced to just 4.5% by our morphing method. As a baseline, the best performance one can expect from our morphing technique in this case is approximately 1.6% $(1-.984)$ due to the classifier's natural tendency to confuse certain web pages.

[1]The threshold was empirically derived by evaluating the performance across a number of thresholds

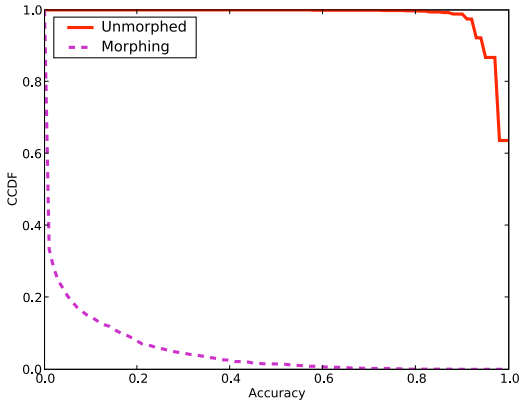

Figure 7. CCDF for morphing versus the original naïve Bayes classifier

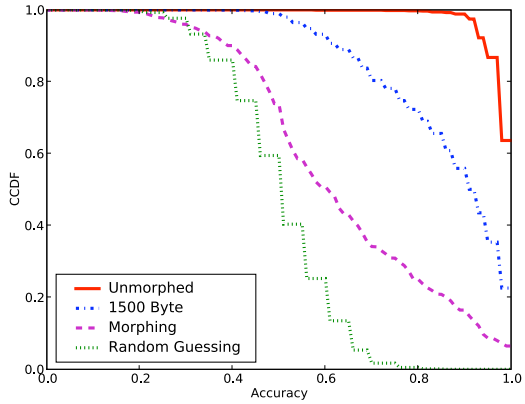

Figure 8. CCDF for indistinguishability against the naïve Bayes classifier

4.2.2 Evaluating Indistinguishability

Given the overwhelming success of our morphing method in deceiving the original classifier, we move on to the more difficult test wherein the classifier is allowed to train on both the morphed (or padded) and unaltered data, and asked to distinguish between the two. The results for our indistinguishability test, shown in Figure 8, clearly illustrate the superiority of our technique over deterministically padding all packets to 1500 bytes. Recall that in this test, the best result that one can hope for is that the classifier does no better than random guessing when distinguishing the pages (i.e., 50% accuracy on average). As shown in Table 2, our method achieves 63.4% accuracy with just 38.9% overhead, compared to 86.2% accuracy and 156.5% overhead for padding – a significant difference in both privacy *and* overhead!

Our use of sampling methods to ensure a sufficient level of similarity between the output and target distribution raises an important question: could the levels of efficiency and accuracy found in our evaluation be achieved with sampling alone? To illustrate the benefits of our morphing method, we alter the source traffic by sampling packet sizes according to the target distribution without employing our morphing technique. Table 2 shows that the accuracy for the two cases is comparable. However, the significant difference in overhead – 38.9% for our morphing technique versus 85.4% for sampling alone – shows that the overhead savings illustrated in our results is primarily due to our morphing method, and not the sampling techniques used to accommodate for short download sessions.

These results, while unintuitive at first glance, can be readily explained by revisiting the way in which the classifier and morphing method operate. The use of deterministic padding fails to preserve privacy because even though all packets are the same size, their direction in the connection still provides useful information. As a result, the distribution of packets sent from either side of the connection uniquely identifies the majority of web pages in the data set. Since our method morphs both sides of the connection independently, this is not a problem for the morphing algorithm. Also, while the performance of our morphing technique is very close to that of random guessing, it fails to be completely indistinguishable because a large proportion of the pages have few packets, and cannot produce a sufficiently close output distribution despite our distance threshold of 0.3. Of course, we can always reduce the threshold further, but this risks an increase in overhead from the additional packets that must be generated.

Strategy	% Accuracy	% Overhead
No Padding	98.4	0.0
1500 byte	86.2	156.5
Sampling only	64.5	85.4
Morphing	63.4	38.9

Table 2. Results for morphing against the naïve Bayes classifier

4.3 Discussion

Taken as a whole, our results on both VoIP and web traffic have shown that our morphing technique provides a significant savings over deterministic padding while still providing superior privacy. Practically speaking, our morphing technique offers an exciting alternative to deterministic padding for protecting privacy that can be used by a variety of services, including ToR [9] and Bit-

Torrent [5]. These results also bring to light a potential weakness in current traffic classification methodology. That is, current traffic classification methods operate under the implicit assumption that a user cannot maliciously alter their traffic characteristics so as to subvert the classification process. Clearly, our evaluation here indicates that traffic classifiers must be aware of the potential threats posed by such adversarial users, and choose analysis methods that are resistant to these techniques.

5 Conclusion

Until now, the only viable proposals for preventing a traffic classifier from inferring sensitive information from encrypted network traffic have involved quantizing the features of the packets, such as by padding their size and sending them at regular intervals. In this paper, we describe an alternative method to quantizing, which is able to optimally balance the privacy provided to the user with the amount of overhead incurred. The approach that we take, *traffic morphing*, chooses the best way to alter the feature(s) of a packet so that they match those of some target class that the user wants her traffic to look like. At the same time, our approach ensures that the changes made to the features also induce minimal overhead, in terms of latency or bytes of additional data sent. We accomplish this balance of privacy and efficiency through the use of a novel application of convex optimization techniques. Moreover, our method works in real-time and makes no assumptions on the traffic classifier being thwarted, other than knowing the feature(s) it uses.

Through our evaluations, we show that our method is applicable to a variety of traffic classifiers that use packet sizes as a feature. Specifically, our morphing method significantly reduces the accuracy of the VoIP classifier of Wright et al. [29] and the web page classifier of Liberatore and Levine [14]. When compared to traditional padding strategies, the morphing technique provides far better privacy to the user while simultaneously maintaining or *reducing* the amount of overhead incurred by padding. Aside from the potential benefit to privacy-preserving protocols, our techniques also push forward the state of the art in applying learning algorithms to traffic in adversarial environments. The challenge moving forward is to find classification methods that are robust to morphing attacks, either through the use of several orthogonal features (though this is unlikely in the case of encrypted traffic), or through the use of robust comparison functions that are difficult to thwart with morphing techniques.

Acknowledgements This work was supported in part by the U.S. Department of Homeland Security Science & Technology Directorate under Contract No. FA8750-08-2-0147 and by NSF grant CNS-0546350.

References

[1] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. The secure real-time transport protocol (SRTP). RFC 3711.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[3] S. P. Chung and A. K. Mok. Allergy attack against automatic signature generation. In *Proceedings of the* 9^{th} *International Symposium On Recent Advances In Intrusion Detection*, September 2006.

[4] S. P. Chung and A. K. Mok. Advanced allergy attacks: Does a corpus really help? In *Proceedings of the* 10^{th} *International Symposium On Recent Advances In Intrusion Detection*, September 2007.

[5] B. Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[6] S. Coull, M. Collins, C. V. Wright, F. Monrose, and M. K. Reiter. On Web Browsing Privacy in Anonymized NetFlows. In *Proceedings of the* 16^{th} *USENIX Security Symposium*, pages 339–352, August 2007.

[7] G. F. Cretu, A. Stavrou, M. E. Locasto, S. J. Stolfo, and A. D. Keromytis. Casting out demons: Sanitizing training data for anomaly sensors. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, May 2008.

[8] J. Dahl and L. Vandenberghe. CVXOPT: A python package for convex optimization. `http://abel.ee.ucla.edu/cvxopt`.

[9] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the* 13^{th} *USENIX Security Symposium*, pages 303–320, August 2004.

[10] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. Polymorphic blending attacks. In *Proceedings of the* $15^{t}h$ *USENIX Security Symposium*, pages 241–256, August 2006.

[11] S. Forrest, S. A. Hofmeyer, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy*, pages 120–128, May 1996.

[12] D. Koukis, S. Antonatos, and K. Anagnostakis. On the Privacy Risks of Publishing Anonymized IP Network Traces. In *Proceedings of Communications and Multimedia Security*, pages 22–32, October 2006.

[13] T. Lander, R. A. Cole, B. T. Oshika, and M. Noel. The OGI 22 language telephone speech corpus. In *EUROSPEECH*, pages 817–820, 1995.

[14] M. Liberatore and B. Levine. Inferring the Source of Encrypted HTTP Connections. In *Proceedings of the ACM conference on Computer and Communications Security*, pages 255–263, October 2006.

[15] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. Sutton, J. Tygar, and K. Xia. Exploiting machine learning to subvert your spam filter. In *Proceedings of the First USENIX Workshop on Large-Scale Exploits and Emergent Threats*, April 2008.

[16] J. Newsome, B. Karp, and D. Song. Paragraph: Thwarting signature learning by training maliciously. In *Proceedings of the* 9^{th} *International Symposium On Recent*

Advances In Intrusion Detection, pages 81–105, September 2006.

[17] T. S. Saponas, J. Lester, C. Hartung, S. Agarwal, and T. Kohno. Devices that tell on you: Privacy trends in consumer ubiquitous computing. In *Proceedings of the* 16^{th} *Annual USENIX Security Symposium*, pages 55–70, August 2007.

[18] M. R. Schroeder and B. S. Atal. Code-excited linear prediction(CELP): High-quality speech at very low bit rates. In *Proceedings of the 1985 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 10, pages 937–940, April 1985.

[19] A. Somayaji and S. Forrest. Automated response using system-call delays. In *Proceedings of the* 9^{th} *USENIX Security Symposium*, August 2000.

[20] D. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and SSH timing attacks. In *Proceedings of the* 10^{th} *USENIX Security Symposium*, August 2001.

[21] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical identification of encrypted web browsing traffic. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 19–30, May 2002.

[22] K. M. C. Tan, K. S. Killourhy, and R. A. Maxion. Undermining an anomaly-based intrusion detection system using common exploits. In *Proceedings of the Fifth International Symposium on Recent Advances in Intrusion Detection*, October 2002.

[23] J.-M. Valin and C. Montgomery. Improved noise weighting in CELP coding of speech - applying the Vorbis psychoacoustic model to Speex. In *Audio Engineering Society Convention*, May 2006. `http://www.speex.org`.

[24] S. Venkataraman, A. Blum, and D. Song. Limits of learning-based signature generation with adversaries. In *Proceedings of the* 15^{th} *Annual Network and Distributed System Security Symposium*, February 2008.

[25] L. N. Vicente and P. H. Calamai. Bilevel and multilevel programming: A bibliography review. *Journal of Global Optimization*, 5(3), October 1994.

[26] D. Wagner and D. Dean. Intrusion detection via static analysis. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2001.

[27] D. Wagner and P. Soto. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the* 9^{th} *ACM conference on Computer and communications security*, pages 255–264, November 2002.

[28] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson. Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, May 2008.

[29] C. V. Wright, L. Ballard, F. Monrose, and G. M. Masson. Language Identification of Encrypted VoIP Traffic: *Alejandra y Roberto or Alice and Bob?* In *Proceedings of the 16th Annual USENIX Security Symposium*, pages 43–54, Boston, MA, August 2007.