A Novel Approach to Detecting Covert DNS Tunnels Using Throughput Estimation

Michael Himbeault

March 24, 2013

Abstract

In a world that runs on data, protection of that data and protection of the *motion* of that data is of the utmost importance. Covert communication channels attempt to circumvent established methods of control, such as firewalls and proxies, by utilizing non-standard means of getting messages between two endpoints. DNS (Domain Name System), the system that translates text-based resource names into resource records, is a very common and effective platform upon which covert channels can be built. This work proposes, and demonstrates the effectiveness of, a novel technique that estimates data transmission throughput over DNS in order to identify the existence of a DNS tunnel against the background noise of legitimate network traffic. The proposed technique is robust in the face of the obfuscation techniques that are able to hide tunnels from existing detection methods.

Contents

List of Tables

List of Figures

1 Introduction

Control of the data that moves between hosts on a computer network is vitally important in order to be able to enforce security policies and protect sensitive information. To this end systems such as firewalls, proxies, and content filters are put into place in order to monitor and control network traffic. Covert channels are utilized to circumvent these control mechanisms for purposes that range from benign to malicious. This work focuses specifically on DNS tunnels, however other types of covert channels do exist.

Detecting a DNS tunnel effectively on a busy network link becomes an exercise in discrimination. Since there is such a wide variety of network traffic that is generated on a busy link, there is generally no simple definition of *normal* for a particular class of traffic, including DNS. This tends to either rule out, or decrease the viability of, algorithms that depend on finding a definition of normal and alerting based on deviations from that norm.

A highly sensitive and specific detection of DNS tunnels on a busy network link is an important problem in the arena of network security as it enables administrators to block or otherwise control these potential sources of compromise. A non-exhaustive, but informative, list of things that covert channels (including DNS tunnels) can be used for is:

- Data exfiltration
- Communication through restrictive firewalls
- Receiving commands or information from a remote source
- Transport layer for complete VPN solutions

In enterprises that deal with sensitive information such as financial, health, personal or intellectual property information it is of the utmost importance to control the access to this information. Even if an enterprise does not have information that needs protecting, DNS tunnels should still be blocked in order to prevent malware from communicating. Because DNS tunnels can be used for arbitrary communication, they can be used as command-and-control channels for botnets or any other malicious system that relies on data transmission. Preventing botnets from operating is of the best interest for the Internet as a whole and should be a concern of every user of the Internet.

Existing methods of detection rely on one of two methods, character frequency analysis and signatures, to detect the presence of a DNS tunnel. The signature-based solutions attempt to identify portions of the tunnelling application data (as opposed to the user data that is sent using the tunnelling application) for which a signature can be built. These signature based solutions are subject to many of the same problems that plague signature based anti-virus solutions including their inability to detect zero-day situations. For example, if the application changes its communication schemes, the signatures may no longer be valid and may no longer trigger when they are intended and expected to. Additionally, signatures cannot be built for applications that are not available for study, or that aren't known of. For applications that use a custom communication scheme that has not had a signature built specifically for it, it is very likely that an existing signature will not be relevant and will not detect it.

Because signature based schemes are not effective in detecting DNS tunnels in a zero-day[1] situation, another method is required. Analysis based on character frequency is proposed

[1] *Zero-day* refers to a situation where nothing is known about the attack as it has never been seen or analyzed before.

in[**?**] by Kenton Born with similar approaches proposed by several other authors (see section 3 for more details) which uses character frequency analysis under the assumption that normal DNS traffic has a unique character frequency distribution that can be used as a discriminating fingerprint. Based on analysis of common domain names, he obtained a distribution that accurately describes normal DNS traffic while effectively detecting DNS tunnel traffic. Results obtained by Born indicate that DNS tunnels have an approximately uniform character frequency distribution whereas normal DNS queries do not.

A weakness of the approach proposed by Born is that it relies on the assumption that DNS tunnel traffic *necessarily* has a character frequency distribution that is different from that of normal DNS queries. This assumption is not necessarily true, and a proof-of-concept application was developed that demonstrates this fact. The tool performs a *probabilistic encoding* that takes an arbitrary data source and encodes it into a stream of characters that conforms to a given distribution. This tool can be used to modify the output of any DNS tunnel application so that their output conforms to the distribution that Born found for normal DNS traffic (or any other distribution, for that matter). By utilizing this transformation, Born's approach fails to detect the DNS traffic as it becomes, from the viewpoint of his algorithm, indistinguishable from normal DNS traffic. Details of this are given in section 6.1.

Due to the weaknesses listed above, it is clear that a new approach is necessary that detects DNS tunnels in a zero-day situation and without the known weakness involving character frequency and probabilistic encoding. This work proposes, and demonstrates the effectiveness of, a method of detecting DNS tunnels that meets these requirements.

The proposed method operates under the assumption that DNS tunnels move more data than a normal domain but don't necessarily do it by moving more bytes than a benign domain. This distinction is important since, on a busy network, a large content or service provider such as Google, Amazon, Facebook or Twitter may comprise orders of magnitude more DNS traffic by byte count than a DNS tunnel. By leveraging this fact, it is possible to detect any use of DNS to transmit arbitrary data by measuring the amount of data that is transmitted using a particular DNS domain or subdomain. The details of this measurement methodology are contained in 5 and requires estimating the amount of unique data that is transmitted by analyzing the queries themselves and not simply counting characters in the query string.

This proposed detection methodology is shown to detect DNS tunnels in as few as ten packets and continues to be robust in highly hostile detection environments such as those that contain a great deal of non-tunnel traffic as well as benign uses of DNS for transmitting arbitrary data[2]. Detector performance on commodity hardware is shown to scale to greater than two gigabits of UDP port 53 throughput per second, indicating that this methodology does not sacrifice performance for detection accuracy and remains practical for monitoring very large networks.

[2]Many security vendors utilize the fact that DNS and UDP port 53 are so loosely controlled in order to ensure that their deployed devices can communicate with the vendor intelligence unimpeded.

2 Background

2.1 Entropy

Entropy is intuitively speaking, a measure of how random a particular collection of items is. In the context of a random variable, entropy is a measure of the uncertainty in the output of the random variable. A random variable that has a distribution that outputs a particular symbol seventy five percent of the time has considerably less uncertainty, and thus less entropy, than one that outputs all symbols with equal frequency. In the context of a collection of symbols in a stream, or data source, entropy can be though of as a measure of the information content of that collection. If the collection comprises almost entirely a single symbol, then that collection can be thought of as containing less information, and thus having less entropy, than a collection where all symbols occur with equal frequency.

Entropy can be calculated for a collection of symbols $\mathcal{C} = \{c_1, \ldots, c_n\}$ with symbol c_i appearing with proportion $0 \leq p_i \leq 1$ where $\sum_{i=1}^{n} p_i = 1$ as

$$H(\mathcal{C}) = -\sum_{i=1}^{n} p_i \log p_i$$

The base of the logarithm determines the units of the resulting value. If a logarithm in base 2 is used, then the entropy has the units of *bits*, if the logarithm with the natural base e is used then the entropy has the units of *nats*, if the logarithm is in the common base 10, then the entropy has the units of *digits*.

2.2 Domain Name System (DNS)

DNS (Domain Name System) is the service through which names are mapped to resources. Typically, this maps a name (such as *www.google.ca*) to an IP address. The value of this service is that names are considerably more flexible, and considerably easier to remember, than the resource or record that they point to. For example, *google.com* is considerably easier to remember than one of the IP addresses that it points to, such as 74.125.226.34. *google.com* is also considerably more flexible, since it points to not just one but several addresses, and successive responses will receive different records in a round-robin, or random, fashion. This rotation of responses allows for a crude form of load balancing and automatic fail over, while retaining its ease of use.

The DNS protocol is assigned both UDP and TCP port 53 for communication with most communication operating over UDP as opposed to TCP. The use of TCP depends on the implementation of the resolver, however the specification indicates the TCP should be used if the response data exceeds 512 bytes or during a zone transfer[**?**]. DNSSEC (Domain Name System Security Extensions), due to the fact that it requires a signature of authenticity for all responses, will often cause the response to require TCP[**?**]. Because there is no *requirement* for when TCP be used, some resolvers may be implemented to use TCP for all responses as this does not violate the specifications for DNS.

The experiments related to this work do not consider the situation of DNS over TCP since the analysis techniques are identical since the formats of the UDP and TCP responses is identical once the TCP stream is reassembled. Modification of the tools developed for this

work would require the ability to perform TCP stream reassembly in order to extract the responses from the TCP response.

Because DNS is such an integral component of Internet communication it is not generally reasonable to simply block it while still expecting functional Internet connectivity. A common approach is called DNS *proxying* which forces all DNS queries to be made to a DNS proxy server that is controlled by the interested entity (ISP, company, etc...). This DNS proxy server is responsible for handling all DNS queries for the internal network, and any DNS queries that are destined for the Internet (as opposed to the proxy) are typically dropped by the firewall in this type of configuration. The DNS proxy server operates in *recursive mode*, which means that if a question is asked of it to which it does not know the answer, the proxy server will then query for the answer (by issuing its own query to the global DNS system) and then respond to the initial request using this response.

DNS is a heavily cached protocol due to how often data can be reused between queries. Consider how often a desktop Internet user causes a request for *google.com*. If a request had to traverse the entire DNS system every time, this would represent a very considerable amount of traffic being generated. To avoid this, every DNS record has extra information about it that includes, among other things, how long it can be cached for. Standard caching lengths put it around one hour which means that a DNS server will only recursively pass on a query for a record once an hour. This caching period is not constant and can be set differently, depending on the information the record contains. Some records require a considerably lower TTL (Time To Live), as low as one minute, while for others a considerably longer (months) may be appropriate.

This proxy architecture removes some naive operation modes for DNS tunnels that will be discussed in in more detail in 2.4.1, but does not offer any protection against the more sophisticated forms of DNS tunnels.

2.3 Covert Channels

Covert channels are methods of communication that use non-standard means of communication for the purpose of evading detection and/or blocking by the existing security infrastructure. Covert channels may utilize portions of an existing protocol or communication channel, or they may find ways of transporting information utilizing a completely new medium. An example of the latter is called a *timing channel*, which can utilize the timing between packets to convey information. A timing channel carefully controls the timing between packets sent to a remote server to encode information, thereby utilizing a method of communication that is not utilized by any *legitimate* protocol or communication method.

Covert channels come in may forms and not all types support the properties that are normally associated with a communication channel. Because they are built on unorthodox, or unreliable, transmission media and are subject to the effects of intermediate routing and networking devices they cannot always offer all of the same functionality as a legitimate channel.

Covert channels need not support bi-directionality due to either the constraints of the underlying medium, or the effect of intermediate devices. A covert channel that is only useful for reception is one that utilizes a third-party image hosting service. It is possible to embed arbitrary information into the header portion of an otherwise completely benign JPEG image file which could then be posted to Facebook, Flickr, or any other publicly accessible

image hosting service. This image file can then be checked by the remote hosts to pull the information however, due to the nature of the image services, the remote hosts may have no way of posting information back to the other endpoint, making the communication channel unidirectional.

Real time data transfer refers to the ability for a communication channel to send data immediately. UDP, by its very nature, supports this and TCP supports this via the PUSH flag which indicates that data is being sent before a full window has been accumulated. The TCP PUSH flag is used, for example, during an SSH connection in order to provide interactivity. Timing channels, or any channel that relies on modifying normal system traffic instead of generating their own traffic, by their nature, are unable to support real time data transfer. This is because they need to wait for a system packet in order to send their data, and if the system goes for a period of time without sending data then the covert channel must wait as well.

2.4 DNS Tunnels

DNS tunnelling is the method by which arbitrary data is made to be transferred over the same channels as DNS. DNS tunnels come in one of two types: raw, or conforming.

2.4.1 Raw DNS Tunnels

Raw DNS tunnels do not attempt to mimic or conform to the DNS specifications, and simply attempt to utilize the fact that UDP port 53 is often left open in firewalls. Raw tunnels attempt to exploit this by transmitting arbitrary traffic using UDP port 53 packets with arbitrary payload. This is the most efficient exploitation of the ubiquity of DNS as it incurs the lowest amount of overhead, both computationally and in terms of network throughput. The trade off for this efficiency is that it is the least conforming and the most likely to get stopped by either a firewall or a proxy. In the situation where all DNS queries are forced to be proxied through a dedicated DNS server, raw DNS tunnels will fail to operate as expected. This is because when the UDP port 53 traffic is redirected to the proxy, the DNS server will attempt to interpret the arbitrary payload as a DNS packet and will likely fail. When it fails, it will drop the packet thereby preventing all raw UDP port 53 communication. Because these types of tunnels are effectively blocked by standard firewall and proxy practises, detection of these tunnels is not considered in this work.

2.4.2 Conforming DNS Tunnels

Conforming DNS tunnels produce DNS packets that conform to all appropriate specification and RFC documents and, as far as any DNS server, is concerned the traffic generated is valid DNS traffic. These tunnels incur the highest computational and throughput overhead, but have the advantage that detecting and blocking them is a very difficult process. The detection of this type of DNS tunnels is the topic of this work. This type of tunnel is capable of operating, even in an environment with very strict firewall and proxy policies. Because this type of tunnel operates in very hostile (to the operation of the tunnel) environment, detection of this type of DNS tunnel is of interest to all levels of government and industry.

Conforming DNS tunnels operate by embedding the data for transmission into the query string and the response and require a modified, non-conforming, DNS server on one end of the

connection and a piece of software on the client end. Typically these types of DNS tunnels have one endpoint that is controlled by the tunnel user, with that controlled endpoint running dedicated server software that. The client and server software is responsible for transforming arbitrary information into DNS queries, and for translating DNS responses back into the arbitrary data that it represents. The precise details of how the translation is done between DNS and the raw data depends on the implementation.

2.4.3 DNS Tunnel Software

Some DNS tunneling software is OzymanDNS[3], Iodine[4], Dns2tcp[5], DNScat[6] and DeNiSe[7], and PSUDP[8]. Each of these have slightly different operational characteristics, but they all aim to do the same thing, which is transmission of arbitrary data over DNS.

Iodine supports raw tunnels, as well as moving information back from the server in A (IPv4 addresses), MX (mail server), TXT (arbitrary text) and may other supported DNS record types.TXT records are rarely used by consumers or end-user applications, and so a blanket block policy of TXT records for end-user devices would have very little impact on end-user applications. TXT records are as close to a raw tunnel that a conforming tunnel can get to in terms of throughput.

DNScat utilizes CNAME (alias to another record) records and a supplementary A record, when appropriate, however the A record is not actually used for throughput. OzymanDNS and DeNiSe utilize solely the TXT record, which is as close to the raw tunnel as possible, however can be easily neutered by simply blocking TXT records. Because these tools only uses TXT records it is possible that it is the least flexible and deployable out of those listed above given a hostile environment. Dns2tcp utilizes either TXT or KEY records, which makes it as flexible as OzymanDNS and DeNiSe. The KEY DNS record was designated for specific uses[**?**], but has been deprecated now[**?**] in favour of the DNSKEY record for use with DNSSEC[**?**] and the IPSECKEY for use with IPSEC[**?**]. Because of this deprecation, use of the KEY record is subject to strict filtering which greatly reduces the effectiveness of this solution.

All of the above tools utilize encoding and decoding mechanisms that would successfully propagate through a proxy server, at the cost of the fact that the tunnel applications have to generate their own traffic. PSUDP, proposed by Kenton Born, aims to remove the latter requirement by creating slack space within an existing DNS packets at the UDP transport layer. He proposes two ways of creating this slack space: naively placing it at the end of the packet, and rearranging the DNS query string to utilize pointers to create this slack space in the middle of the packet. Pointers allow for the re-use of DNS query strings within a packet to save on space. They are an optional component of the specification, but allow for considerable space savings. A pointer in a DNS packet is a special sequence of bytes that indicates where in the packet the processing of the query string should jump to. When processing a query string, only a single pointer can be followed, according to spec, which prevents multiple redirection and infinite loops (where a pointer points to itself). By having a pointer point forward in the

[3] http://dnstunnel.de/
[4] http://code.kryo.se/iodine/
[5] http://hsc.fr/ressources/outils/dns2tcp/index.html.en
[6] http://tadek.pietraszek.org/projects/DNScat/
[7] http://c0re.23.nu/c0de/snap/DeNiSe-current.tar.bz2
[8] http://www.kentonborn.com/psudp

packet, it is possible to cause the parsing of a query string to skip a number of bytes, creating slack space.

This method relies on this slack space, which is not parsed by normal servers or clients, but can be contain arbitrary data that is extracted by special clients. However, because this slack space is not processed by DNS servers, in an environment where all DNS queries must go through a proxy, this method is incapable of producing a covert channel that is able to penetrate a strictly proxied DNS environment.

3 Review of the State of the Art

The solutions that exist to date generally make very little use of complex, static, signatures but rather attempt to exploit a characteristic trait or property that the DNS tunnel will exhibit. If a tunnel can be crafted to not exhibit that feature, then those detection strategies will normally fail in their detection. This section summarizes the known detection methods, their commonalities, advantages, and disadvantages. It will then take the body of detection methods as a whole and establish the existence of any gaps, or weaknesses, that could be exploited by an application to circumvent an IDS that made use of *every one of the mentioned detection methods*. This essentially identifies gaps in the current state of the art that could be filled by a new technique, or an adaptation of an existing technique.

3.1 General Covert Channel or Anomaly Detection Research

(Browne, 1994)[**?**] establishes an entropy conservation based approach for testing the completeness of general (that is, not specific to DNS) covert channel analysis and detection methodologies. (Shaffer, 2008)[**?**] Proposes a Security Domain model for assessing the surface of a piece of software for exploitable covert channels.

(Ray, 2008)[**?**] proposes a protocol for use in a covert channel that incorporates stealth, low overhead, data integrity, data confidentiality, and data reliability. The protocol can be used on top of any other covert channel transport method (ICMP, IP, HTTP, DNS, etc. . .)

(Horenbeeck, 2006)[**?**] discusses, briefly, DNS tunnels and their implications. A short mention of proxying DNS requests is given as a potential solution but without examining the multitude of ways that a DNS tunnel could still operate in such an environment. The rest of the paper discusses the risk management and policy based mitigations that can be applied to covert channels in general.

(Moskowitz, 2003)[**?**] investigates the link between anonymity and covert channels. It identifies several linking factors such as covert channel capacity and the properties of anonymizing networks, and investigates how these affect anonymity of participants in a communication.

(Newman, 2007)[**?**] discusses covert channels in a broad sense, examining the various types of covert channels along with the relationship between covert communication, cryptography, steganography and secrets.

(Okamura, 2010)[**?**] discusses a fascinating type of covert channel for communication between virtual machines that share a physical host. The paper proposes that virtual machines can manipulate their CPU core load, which is peripherally visible to other virtual machines on the host, in order to send and receive information.

3.2 Non-DNS Related Research

Tunnel Hunter[**?**] is an application that aims for general covert channel detection over a variety of tunnelling communication channels.

(Bauer, 2003)[**?**] discusses a new type of HTTP-based covert channel that adds the unwitting web browser application to the anonymity set.

(Borders, 2004)[**?**] discusses a method of detecting data egress using HTTP-based covert channels.

(Cabuk, 2004)[?] and (Cabuk, 2009)[?] discuss the design and detection of IP (Internet Protocol) based covert timing channels. (Gianvecchio, 2007)[?] discusses an entropy-based approach to detecting covert timing channels on the Internet based on their effect on the original process' entropy properties.

3.3 DNS Covert Channel Research

The SANS Institutes's InfoSec Reading Room published a report on the design and detection of DNS tunnels[?]. The report covers a very wide variety of topics including background information, tunnel-specific information, technical information, existing applications, detection techniques, detection implementations, and a sample detection scenario. This report is exceptionally good reading as a primer on the topic. The sample detection scenario employs an analysis technique very similar to the technique that will be outlined in section 5.

(Karasaridis, 2006)[?] proposes and evaluates mechanisms that use network flow data[9] to detect DNS anomalies including cache poisoning and tunnels. Their detection of DNS tunnels involves estimating average packet size distributions over a given time interval (in the paper hourly distributions were produced), and then comparing the actual distributions with a baseline distribution using cross-distribution entropy computation. The authors are able to observe considerable changes in their cross-distribution entropy measurement during the onset of the Sinit virus in their real-world data. This approach is discussed in additional detail in (Roolvink, 2008)[?].

(Born, 2010)[?] discusses a way of using javascript in a web browser to exfiltrate data from a network, while [?] discusses a novel way of crafting a DNS tunnel that exploits the nature of a DNS packet and the ability to create unused space in the packet in which arbitrary data can be stored. [?] discusses a method of detecting DNS tunnels by examining character and n-gram frequencies in the names that are being queried for. [?] demonstrates the effectives of data visualization when attempting to detect a DNS tunnel using a custom visualization engine using the character frequency analysis proposed in one of the Born's previous publications. If a DNS tunnel can be crafted such that its character frequencies are distributed sufficiently close to those of legitimate DNS names, then it is possible to hide a DNS tunnel from this type of analysis.

(Butler, 2011)[?] demonstrates a way of quantitatively analyzing covert communication channels with particular focus on DNS covert channels. It proposes a *codeword mode* of communication over DNS where a specific lexicon is chosen that allows the two endpoints to communicate with. Each word in the dictionary has a particular meaning[10] that is understood by both endpoints. This lexicon must be chosen *a priori* and must be common to all endpoints wishing to communicate using this method. Butler also discusses the concept of perfect stealth of a covert channel based on DNS, and proposes a deep packet inspection based countermeasure that utilizes the *Jensen-Shannon divergence* measure. The method relies on an assumption very similar to that in Born's character frequency analysis where the tunnelled traffic uses DNS names with a measurably different character distribution than that

[9]Flow data is a way of digesting network packet data into information per communication, stream, or (in the case of UDP since there is no inherent concept of a stream of interrelation of packets) temporally contiguous collection of packets.

[10]The words can represent binary information, or can represent higher level constructs such as commands in the context of a botnet or piece of malware.

of legitimate DNS traffic. Because of this similarity of assumption this approach suffers from the same vulnerability as Born's.

(Romana, 2007)[**?**] discusses their analysis of DNS data on a large campus network. They use the output of a DNS resolver's query logging as their input, but the process works just as well on other inputs even though they aren't discussed. Digestion of the large query log file is done with standard Unix utilities and logic available on almost all Unix-based systems. The authors estimate the entropy of the source IP address (of the DNS query) and teh queries themselves, and perform analysis based on that output. The scalability of this approach is not discussed in detail, nor is an adaptation of it to consider more refined sets of queries since this approach only takes all parts of all queries together for analysis. Adaptation of this approach for real-time analysis is not discussed either. Because this approach is does not discriminate with respect to finely grained slices of time, nor to domain or subdomain information, it necessitates that all of the query data be kept around for postmortem analysis. The approach is able to alert to the fact that something was detected, but it cannot indicate precisely when, indicate in a timely fashion, nor can it give any indication as to what caused the alert. These details must be ascertained from the raw query data after the proposed approach throws an alert which increases response time and the manpower required to investigate an alert.

(Thomas, 2011)[**?**] proposes and evaluates the efficacy of an FPGA (Field Programmable Gate Array) based solution for detecting malicious DNS packets on a high throughput network link. The work extends prior work to including more flexible detection and to support more current-generation network infrastructure (the previous work was limited to 100Mbit network connections, with the new work operating on 1000Mbit network connections). The fact that this approach makes use of specialized hardware makes it prohibitively complex for smaller companies to implement and use. The analysis performed on the DNS packets in order to determine their validity is done via a signature-based system where the DNS query is hashed, and the hash is compared to a blacklist of domains that are disallowed due based on the network policies. Because this is signature and blacklist based, this approach suffers from the standard problems such as weakness against zero-day situations and the inability to be agile in the face of an adaptive attacker[11]. The tests of this system use highly synthetic testing methods that do not involve real-world data or synthetic data that attempts to resemble real-world data.

(Dietrich, 2011)[**?**] examines the use of DNS for command and control of botnets based on the reverse engineering of the *Feederbot* botnet application. Based on the lessons learnt from Feederbot, the authors applied their methods to other real-world traffic and detected other botnets that also use DNS as their command and control medium. The authors make use of two different approaches for classifying malicious DNS traffic from benign and legitimate traffic. The first approach makes use of entropy calculated over the responses DNS queries, very similar in theory to the character frequency analysis proposed by Born[**?**] and is vulnerable to the same methods of circumvention[12]. This portion of their approach operates a single packet at a time, and does not consider aggregate information. The authors also propose the use

[11]In this case, if the attacker chooses to use a new domain, the tunnel will succeed since the system does not have the new domain on its blacklist yet. Similarly, if an attacker is using a publicly available domain for their tunnel, the blacklist can affect other legitimate traffic on that domain.

[12]That is, if the botnet architected the data in the examined fields to conform to the author's model of benign traffic, then the botnet could effectively masquerade as benign traffic and become invisible to this method of detection.

of behavioural analysis on data and statistics gathered from the aggregate of several packets to estimate the persistence of DNS queries as well as the amount of data moved over DNS by each host on the network. The persistence of the connection is estimated by considering the maximum time between DNS packets whereas the throughput over DNS is measured by counting the bytes in all of the data segments of response packets and summing over time. The persistence analysis can countered by a botnet that models its communication with the command-and-control server based on a Poisson distribution with inter-packet times imitating that of legitimate traffic. Similarly, the throughput analysis can be countered by employing codewords and rate limiting to utilize a high-level protocol compression to reduce the amount of data that needs to be sent, and to reduce the amount of data actually transmitted in a given time period to reduce the footprint of the bot on the network.

(Paxson, 2011)[**?**] is a slide deck that discusses the author's searches through large campus networks for DNS tunnels in the wild. The author proposes an approach for detecting DNS tunnels that is very similar to the method proposed in this work in that it examines the approximate amount of data transferred per domain and/or subdomain. The author, instead of utilizing entropy measures, makes use of the utility *gzip*[13] to estimate the amount of data moved under a domain in a given collection of queries. The author's assumption is that gzip already is optimized for compressing data, which can be thought of as measuring the amount of unique data that is contained in the uncompressed stream. The author also mentions the codebook/codeword method of embedding information into DNS queries, similarly as to what was discussed in [**?**]. The author successfully applies this approach to real-world data and identifies DNS tunnels that were previously unknown. The author, however, defines absolute thresholds for use in detecting which domains classify as being a tunnel as opposed to comparing the data moved from each domain to its peers and performing a more relative and context-aware analysis. The utilization of the gzip algorithm reduces the scalability of the approach due to the computational overhead incurred, limiting the applicability to smaller networks or postmortem (as opposed to real-time) analysis.

jhind[**?**] gave a presentation at DefCon 17 that discusses the use of artificial neural networks to identify DNS tunnel traffic. The author proposed that the neural network operate on the euclidean distance between the various queries to a particular subdomain, treating the queries as vectors in higher dimensional Euclidean space. The author successfully detected DNS tunnels as produced by several software packages (Iodine, Ozymandns and Dns2tcp) using the described approach. The approach outlined by the author suffers from the normal training problems associated with neural networks, such as over or under fitting to the training data, which may or may not prove to be problematic in the real world. The author does not go into detail about the accuracy and precision of the neural network which is very important for real-world applications where false alarms and false negatives are costly errors. Additionally, it is theoretically possible for a DNS tunnel to encode its outputs using a codebook where every word has a distance from every other code word that is within a desired range. Such a codebook could constructed by including all words that, when treated as vectors in n dimensional Euclidean space, lie within a ball of radius r where r is the maximum desired distance. By doing this, it is possible for a tunnel to fit within the neural network's definition of normal and to pass by undetected.

Jeffrey Guy blogged in 2009 about visualization as an aid for detecting DNS tunnels

[13]gzip is a compression library that is used to compress input streams such as archives or other files.

by looking at frequency and request/DNS name length plotted together. In the concluding portions of the article the author mentions briefly, and in passing, that the count of the number of different host names per domain could be of value. The provides no further discussion of this topic, however, and leaves it as an anecdote to the article.

Static signatures exist for at least three common network anomaly detection engines (Snort[**?**], Proventia[**?**], and TippingPoint[14]) engines, with others likely offering similar functionality. It is important to note that these filters and rules do not trigger on all DNS tunnelling applications and may not be robust in the face of a zero-day situation.

3.4 DNS Tunnel Detection Landscape

Taken together as a collective body of work, the detection approaches for DNS tunnels can be summarized as follows, with the weaknesses and strengths of each general approach outlined.

- Signature based approaches exist for several popular detection platforms.

  Strength: The fact that the platforms are common and already deployed makes it very easy to deploy these signatures to a large number of existing networks.

  Weakness: The static nature of the signatures means that they are not flexible enough to effectively identify more than a small portion of the available tunnelling tools.

- A detection method based on flow data, which offers a more scalable approach due to the reduced amount of information that needs to be processed, is proposed which examines average packet length and statistical deviations thereof compared to a normal baseline.

  Strength: This approach is flexible in that it is not limited to looking at characteristics of particular applications, but rather at patterns of behaviour that may be exhibited by any DNS tunnel.

  Weakness: This approach assumes that DNS tunnel software will exhibit longer packet and query lengths than normal traffic which is not necessarily true. DNS tunnels can use carefully constructed encodings to ensure that their queries stay small enough so as not to stand out against benign and legitimate traffic. Simply limiting the size of their queries will not suffice, since the proposed detection algorithm relies on comparing the distribution to a known normal distribution, however carefully choosing the length of the queries such that they satisfy the normal distribution will allow the tunnel to remain undetected. Further, since this approach relies on identifying a baseline, it is not necessarily suitable for links with a high variability of traffic patterns (perhaps due to time-of-day variability, or where it is not feasible to determine if the chosen normal baseline contains malicious traffic or not) where false alarms and false negatives may become common.

- The use of artificially created slack space in a packet is a novel approach with a great deal of flexibility for creating a DNS tunnel.

  Strength: The slack space requires application aware inspection that performs deep packet inspection to determine the existence of, and then the contents of, the slack space.

[14]TippingPoint does not make information about its filters available as public information, however correspondence with a TippingPoint user revealed that filters 9932 and 9938 trigger on the application data contained in DNS packets generated by Ozymandns.

Weakness: This type of DNS tunnel has a crucial weakness in that this slack space is not processed by recursive resolving DNS servers, and such will not persist past the first resolver in a chain in such an environment. If these packets are not sent directly to the DNS tunnel server endpoint, the payload will not survive and the tunnel will not operate. Because of this, no special detection or analysis mechanisms are required, and a simple DNS proxy will suffice in preventing these types of tunnels.

- A form of character frequency analysis is used in several approaches to detect the existence of DNS tunnels.

  Strength: This approach makes use of the assumption that DNS tunnels produce queries and/or responses with a measurably different character distribution than that of benign traffic. Since this assumption is quite general, it applies to any DNS tunnelling application.

  Weakness: Because this approach relies on the assumption that the distributions are measurably different, if a DNS tunnel were able to construct its queries such that its character distribution matched the expected distribution, then it would be able to evade this type of detection. A proof-of-concept approach and software application are presented in section 6.1 that is able to perform a loss-less two-way coding from a high entropy source (such as compressed or encrypted data) to a stream whose character frequency matches any[15] given distribution.

- Hashes and blacklists are used along with an FPGA based implementation of the algorithm for analyzing DNS traffic and blocking packets deemed to be malicious.

  Strength: This approach, due to its fast hashing algorithm and FPGA based implementation, scales to very high throughput.

  Weakness: Due to the blacklist nature of this approach, it suffers from the same vulnerabilities as other signature based methods; inability to react intelligently to a zero-day situation or clever adversary. Further, since it is built on highly custom hardware requirements, it is not always practical for smaller network operators to deploy.

- A few approaches examine the behaviour of DNS tunnels and their effects on the statistical properties of the queries themselves over time. These approaches consider very similar approaches to the one given in this paper, explained in detail in section 5.

  Strength: These approaches are considering the most fundamental source of information for a DNS tunnel; the queries themselves. Because DNS tunnels use the queries as their communication, it makes the most sense to attempt to examine these queries for the keys to detecting the tunnels.

  Weakness: The weaknesses of the techniques proposed in the existing literature include cleverly constructed queries (such that they sit within a ball of a desired radius in n dimensional Euclidean space), they are not suitable for real-time analysis (such as the use of higher overhead measuring mechanisms like gzip) or they do not discriminate between different domains or subdomains.

[15] There are some small caveats that are explained in detail along with the rest of the algorithm.

4 Problem Statement

4.1 Brief Statement

The purpose of this work is to investigate the feasibility of real-time DNS tunnel detection that does not suffer the common weaknesses of existing techniques.

4.2 Detailed Problem Description

DNS tunnel detection is a complicated task made worse by the fact that DNS tunnel traffic can appear to be completely legitimate network traffic that conforms to all standards and restrictions. It need not violate any established standards or conventions, which makes it difficult to detect against the background of normal DNS traffic based on testing for violations.

This property of DNS tunnels makes them a particularly effective transport mechanism when data exfiltration or network control circumvention is the end goal. For this reason an efficient method of detecting DNS tunnels is required that can effectively detect a DNS tunnel against normal DNS traffic with a low false-positive rate and that must not be susceptible to existing methods of circumvention.

From section 3 it is evident that there are currently several approaches to detecting DNS tunnels that are not signature based as well as signature based approaches of varying flexibility. There is only one mention of performing real-time analysis at the domain/subdomin level, and it is anecdotal in nature with no clear analysis of its merits or validity. The only other similar approach involves aggregating all domains together and taking their queries together for analysis which obliterates any per-domain statistics that could have been gathered.

Looking at this landscape, it becomes evident that there is a highly advanced theoretical DNS tunnel that could evade all of the proposed real-time detection techniques. Any detection that may occur postmortem would not be able to alert to the threat in an adequate time frame to stop the attack in progress. This tunnel would have the following traits:

1. All of its DNS packets would conform to all appropriate DNS RFCs.
2. Its queries would be chosen such that the character frequency distribution matches benign DNS queries (to evade [**?**] and similar approaches).
3. Its queries would be chosen such that they have a distribution of lengths that matches benign DNS queries (to evade [**?**] and [**?**]).
4. Its queries are chosen such that they do not span too great a space when taken as vectors in higher dimensional Euclidean space (to evade [**?**]).

Item one is already demonstrated in practise by most of the tunnelling applications available, and item two is shown to be possible in section 6.1. Item three is easily accomplished by simply splitting queries based on a statistical model of the desired lengths, and item four will be shown in section 6.2 to approximately follow from item two based on empirical measurements.

4.3 Solution Evaluation Criteria

The objectives that must be met for an approach to have successfully solved the problem posed in section 4.1 are as follows:

- Successfully discern both synthetically generated tunnels and known tunnels in real-world data from normal, benign, DNS traffic.
- Be resistant to known obfuscation methods compared to existing detection methods.
- Be able to operate at high speed on general purpose, easily obtainable hardware.

The proposed approach will be evaluated against these criteria to determine whether or not it can be considered an improvement on the state of the art for this type of detection.

5 Proposed Detection Method

The method proposed in this work examines the information theoretical properties of the DNS queries to each subdomain, thus retaining the flexibility to filter and alert per domain as opposed to more generally on the set of all DNS queries. The tools developed to test this approach utilize full packet data for its analysis, but can be modified to use name server query logs (as were used in [**?**]) or other sources of query information. The prototype software is easily capable of running at greater than gigabit speed on inexpensive, off the shelf, hardware making this approach inexpensive to deploy on smaller networks or in resource constrained situations.

6 Demonstration of Existing Weaknesses

This section will demonstrate that the weaknesses listed in section 3.4 are in fact exploitable by software and are not simply theoretical in nature.

6.1 Probabilistic Encoding

Several detection methods in the literature involve examining the character distribution in the queries, or responses, of DNS packets. These approaches compare the distributions obtained from the traffic being analyzed, and compare them to a known distribution that can be considered normal. If there is a sufficiently significant measurable difference, then the packet is flagged as anomalous.

If it were possible for a DNS tunnel to encode its output in order to match the distribution that these detection methods consider normal, then it would be possible for it to evade detection by masquerading as benign DNS traffic. These detection methods assume that the output of a DNS tunnel will have high entropy (due to compression and/or encryption) and/or span a wide range of character values, whereas normal traffic does not have these features. The solution to this problem becomes one of encoding a high entropy source into a lower entropy encoding in a way that the high entropy stream can be recovered from the low entropy encoding, where the low entropy encoding has a specific distribution of characters.

A proof-of-concept tool was written in C that performs precisely this task. The tool takes, as input, what it assumes to be a high entropy source[16] and a configuration file that describes the desired output distribution.

[16]Since any source can be used to produce a high entropy source through encryption or compression, this is considered a safe assumption and limitation.

Character	Count	Character	Count
<*space*>	0.11965	u	0.0242803
e	0.111823	m	0.0211814
t	0.0797253	w	0.0207765
a	0.0718989	f	0.0196144
o	0.0660886	g	0.0177392
i	0.0613258	y	0.0173783
n	0.0594154	p	0.0169821
s	0.0557003	b	0.013135
h	0.0536491	v	0.00860991
r	0.0527071	k	0.00679637
d	0.0374417	j	0.00134695
l	0.0354345	x	0.00132054
c	0.0244916	q	0.000836341
		z	0.000651466

Table 1: Probability distribution used for english character frequency in the sample encoding and decoding.

```
HEzQcP9uxPOzeOB1SfRP+DQ7x3X5dTA1WF3vpcnO5kgLQQEP7xnUrnUy7U8ISNdbNQd+8da64+Ci
nNBRvu3TRlGOKJvLzb6QDBWqxVfa4VeAU+cSvj+uzoajp2F0jd5/csHxJQlguoeqT76pq8oStoLG
3k48PZWuebgweZx5KdxAgUfN7/kvzwBzG7O+9B7J/O8y6BfGz6InOH+LuDCOsFqMljljX3SkgJq/
yPka52SDxa3D4GXecj9d
```

Table 2: The base64 encoding of the binary data used for the sample encoding.

6.1.1 Sample Encoding

A sample encoding is given using the english character frequency distribution with the character distribution table used given in Table 1. In order to ensure that interested parties can verify this output, the source high-entropy data is contained in Table 2. A data matrix image of the same base64 encoding is also given in Figure 1. This can be converted back to the original source via the standard Unix command line tool *base64*. The output of the encoder using english character frequencies and the given input data is shown in Table 3 which is clearly not english to any human observer or any analysis tool with knowledge of English di- and tri- gram frequency information.

Analyzing the frequencies of the output shows that it closely approximates english character frequencies. A larger sample of random data (one hundred thousand bytes) was also encoded, and its output analyzed as well. The small sample, the large sample, and actual english character frequencies are displayed together in figures 2 and 3. As is evident from the combined plots of the various distributions, there does not exist a clear distinction between them and thus it becomes very difficult to perform automated classification when information is encoded using this tool.

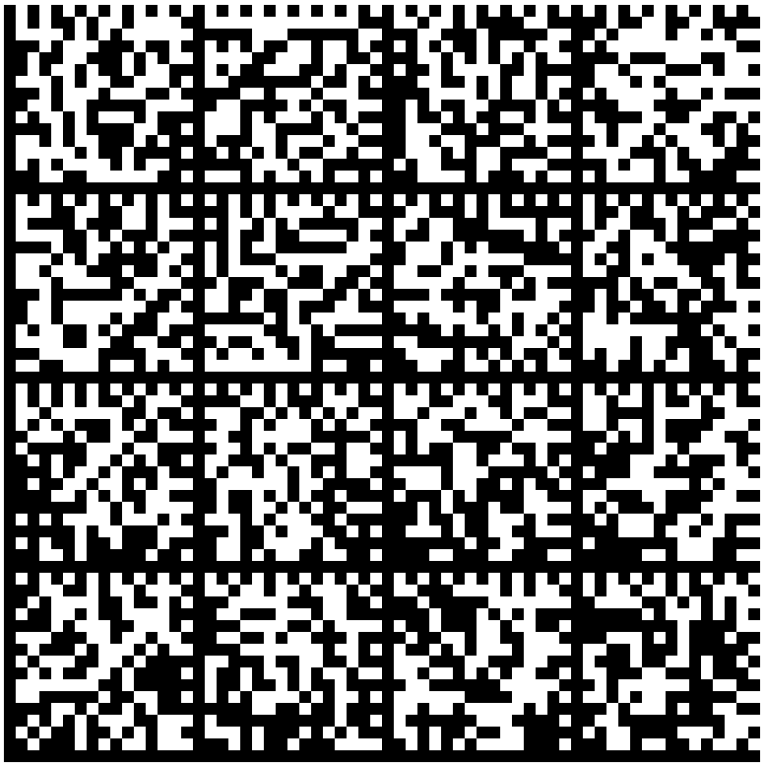

Figure 1: The base64 encoding of the binary data used in the sample encoding, represented in a data matrix image.

```
ea oircqktushn ptodelmj n gvitdza wenxoecft  toairngbse dsorwatle n
hepusihaoeyhftl nathredec o tmsinyrboiauel ieeisdyth  goeuhahwsftrmra
anpleiceeiarsvnod lh moeeistwut od bhta csoeytaekg  rirsoefmnah anch edrlio
erslihea seatleptronwuaig eyfiraeotlmnfeabnustdvieirtealpshreyn s tdmo ascw
ueniehtof ir eknabeltdnt parwiu dmtheaelaieeiygsoncrfas tres d olhtaeheoc
ogussae iiltemswhdaynrenofarsv  ittuepoiblencs r mdoae irta eukoeirnyltesnh datn
wmti ceeirg hhoebsfnhraptdelai osnt w heaoultnheycs tgo ifpaeursn thvremooia
erdicebhslea iaslaoenmy sthu etori aeeierfgntheks rno acs molbduahiaw
fderoeythidsw latenplt eo
```

Table 3: The encoded form of the binary data according to English character frequencies.

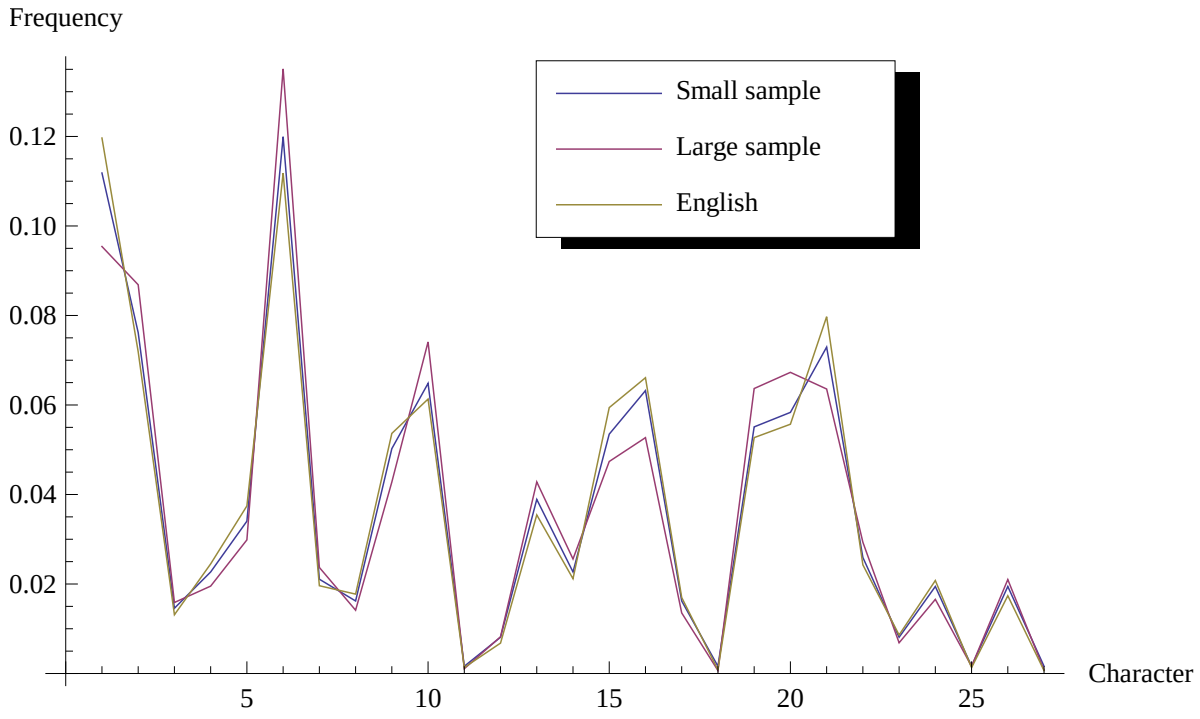

Figure 2: The character frequencies of the small and large encoded samples, as well as the English target distribution sorted by character.

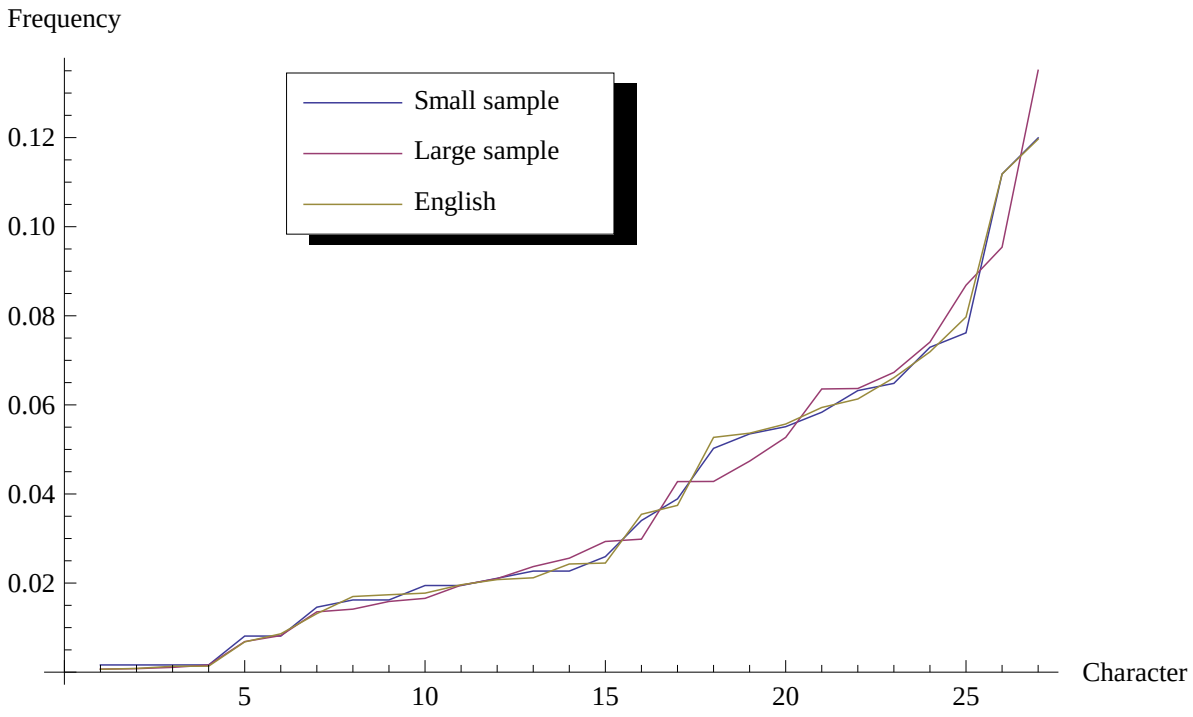

Figure 3: The character frequencies of the small and large encoded samples, as well as the English target distribution sorted by frequency.

6.1.2 Technical Details

The probabilistic encoder is open-source and freely available, licensed under the Mozilla Public License version 2 which is GPL compatible. The source code is available at https://www.riebart.ca/hg/probcode for download and use under the MPL2 license terms. The source code compiles on Unix and other POSIX based systems sych as Cygwin on Windows.

The encoding and decoding processes can be broken up into two steps; probability distribution analysis and then the actual conversion. The analysis of the distribution is the same for both the encoding and decoding operations, however the encoding operation is somewhat more involved than the decoding operation from an algorithmic point of view.

The probability distribution is analyzed in order to map each element of the distribution (character, word, etc...) to a string of bits. An assumption is made that all bit strings of a given length are equally likely to appear in the high-entropy source, and this places a constraint in the output PDF; the most frequent symbol cannot be more frequent than the most frequent symbol in the high-entropy source. That is, no symbol in the output distribution can have a frequency above 50%. This is considered a fair assumption since in English, the most common letter has a frequency of approximately 11%.

The mapping from symbol to bit string is done by sorting the distribution from most to least frequent, and then selecting a shortest available bit string of a length that has an expected occurrence rate approximately equal to, but not less than, the symbol's occurrence rate in the output distribution. For example, the bit string to symbol mapping for the English unigram encoding sample is given in Table 4 along with the approximate frequency of the letters in English text. Observe that the bit-string's length is chosen as the longest possible such that its occurrence rate, $\frac{1}{2^{length}}$, is greater than the symbol's frequency.

Once the mapping has been built, and the various lookup tables built, the encoding or decoding process can begin. For encoding, a table is maintained that keeps track of how many times each symbol in the desired distribution has been output. The purpose of this table is to facilitate the greedy selection of symbols in order to ensure that one symbol is not chosen more often than is necessary and that those symbols that ar farthest from meeting their quota in the output stream get preferential selection when the input bits match their mapping.

6.2 Euclidean Distance Between Queries

Character	Bit-String	Count
<*space*>	1	0.11965
e	0	0.111823
t	11	0.0797253
a	01	0.0718989
o	10	0.0660886
i	00	0.0613258
n	111	0.0594154
s	011	0.0557003
h	101	0.0536491
r	001	0.0527071
d	110	0.0374417
l	010	0.0354345
c	100	0.0244916
u	000	0.0242803
m	0011	0.0211814
w	1101	0.0207765
f	0101	0.0196144
g	1001	0.0177392
y	0001	0.0173783
p	1110	0.0169821
b	0110	0.013135
v	1010	0.00860991
k	0010	0.00679637
j	1100	0.00134695
x	0100	0.00132054
q	1000	0.000836341
z	0000	0.000651466

Table 4: Bit-string mapping for English characters and their frequency for comparison.