Quantitatively Analyzing Stealthy Communication Channels

Patrick Butler, Kui Xu, and Danfeng (Daphne) Yao

Department of Computer Science
Virginia Tech
Blacksburg, VA 24060

Abstract. Attackers in particular botnet controllers use stealthy messaging systems to set up large-scale command and control. Understanding the capacity of such communication channels is important in detecting organized cyber crimes. We analyze the use of domain name service (DNS) as a stealthy botnet command-and-control channel, which allows multiple entities to pass messages stored in DNS records to each other. We describe and quantitatively analyze new techniques that can be used to hide malicious DNS activities both at the host and network levels.
We also present and experimentally evaluate statistical content-analysis techniques as a countermeasure, which require deep packet inspection. Our techniques are beyond the specific DNS security problem studied. We give a formal definition for the *perfect stealth* of a communication channel; point out the fundamental limits in achieving it, as well as the practical issues in the detection. We perform comprehensive statistical analysis that makes use of a two-month-long 4.6GB campus network dataset and 1 million domain names obtained from `alexa.com`.
Keywords: DNS tunneling, command and control, information theory.

1 Introduction

Botnet command and control (C&C) channel refers to the protocol used by bots and botmaster (i.e., botnet controller) to communicate to each other, e.g., for bots to receive new attack commands and updates from botmaster, or to submit stolen data. A C&C channel for a botnet needs to be reliable, redundant, non-centralized, and easily disguised as legitimate traffic. Many botnet operators used the Internet Relay Chat protocol (IRC) or HTTP servers to pass information. Botnet operators constantly explore new stealthy communication mechanisms to evade detection. HTTP-based command and control is difficult to distinguish from legitimate Web traffic. For example, detecting frequent and periodic HTTP requests was proposed for identifying botnet traffic [7]. However, this method may give high false positives as legitimate websites also automatically refresh pages. The feasibility of email as a stealthy botnet command and control protocol was studied by researchers in [11]. In this paper, we systematically investigate the sole use of DNS queries for botnet command and control.

The decentralized nature of domain name systems (DNS) with a series of redundant servers potentially provides an effective channel for covert communication of a large distributed system, including botnets. The focus of this paper is on analyzing the feasibility of a pure DNS-based command-and-control. We analyze a stealthy communication channel based on DNS updates, queries, and responses with existing infrastructure, without enlisting any Web or special-purpose servers. The DNS channel is aided by being a high traffic channel such that data can be easily hidden. As virtually anyone can create their own domain name and DNS servers, it is a system that can easily be infiltrated by hackers and botnet operators.
DNS tunneling is a technique known for transmitting arbitrary data via DNS protocol [**?**,**?**,4]. One application is to bypass firewalls, as both inbound and outbound DNS connections are usually allowed by organizational firewall rules. **(1) Because DNS is often overlooked in current security measures, it offers a command-and-control channel that is unimpeded. Because nearly all traffic requires DNS to translate domain names to IP addresses and back, simple firewall rules can not easily be created less they harm legitimate traffic. Whereas, other channels such as HTTP might be limited to well known sites.** Compared to the existing studies on DNS security and botnet C&C, the novelty of our work is two-fold: *i)* presenting and quantitatively evaluating new DNS-based techniques for distributed and stealthy communication, and *ii)* more importantly, analyzing the practical limitations of information-theoretic based detection techniques. These limitations largely contribute to the current arm race between attackers and defenders.
Our analysis is useful beyond the specific DNS tunneling problem studied. There has not been a systematic study on its robustness against statistical detection methods. Understanding the capacity of botnets communication power helps identify and eliminate nefarious attacks launched from them. Therefore, our work is not yet another botnet command-and-control solution. Our techniques – including the countermeasure based on analyzing content distributions and a model for perfect stealth in content-based communication channel – are useful beyond the specific DNS problem studied.
Our contributions While the technology of DNS tunneling for command and control has been observed [6], it was still unclear how effective and feasible to use this technique to sustain large botnets. We provide the first systematic analysis on the constraints associated with DNS-based botnet communication channels. Our technical contributions are summarized as follows.

– We give a formal description of a botnet command-and-control protocol through DNS queries and responses associated with domains under botmaster's control. We describe a new *codeword mode* of communication, which employs a shared vocabulary between bots and botmaster for stealthy dissemination of attack commands or code updates.
– We describe techniques for hiding query activities, including *i) piggybacking query strategy* – a bot blends its (outbound) DNS queries

with legitimate DNS queries and *ii) exponentially distributed query strategy* – a bot probabilistically distributes DNS queries so that inter-arrival times follow an exponential distribution.
– We give the first formal definition for the perfect covert channel in terms of the distinguishability between normal traffic and attacker's traffic. We discuss how information theoretic analysis can be used to detect malicious traffic. We evaluate statistical methods for detecting anomalies in the content of DNS packets, through comparing the probability distributions of normal DNS traffic and tunneling traffic. More importantly, we point out the practical limitations (namely efficiency and scalability) associated with these information theoretic methods.
We perform comprehensive experiments to evaluate the behaviors of proposed query strategies in terms of how quickly new commands are disseminated to a large number of bots. Our analysis utilizes a 4.6GB two-month-long wireless network trace obtained from an organization.
– We also raise an open question on how to efficiently and automatically generate short-lived domain names that resemble legitimate domain names and evade anomaly detection. We give evidences on the difficulty of the problem.

Organization We describe the basic DNS tunneling mechanisms in Section 2. We present new strategies for improving the stealthiness of DNS-based command and control in Section 3. Our experimental evaluation results are described in Section 3.2. We describe a countermeasure that requires examining the content of DNS packets and performing statistical analysis in Section 4. We raise an open question regarding how to automatically generate practical domain names in Section 5. Related work is given in Section 6. Conclusions are given in Section 7.

2 Communication Modes

In this section, we describe protocols that pass messages over the DNS between distributed entities, and illustrate the ease of setting up large-scale command-and-control via DNS. We describe two forms of communication modes: *codeword* mode and *tunneled* mode. *Codeword communication* allows one-way communication from botmaster to a bot client, which is suitable for issuing attack commands. *Tunneled communication* allows for the transmitting of arbitrary data in both directions between bot and botmaster, which may be used for both issuing commands and collecting stolen data. **(2), The former only requires the ability to set a particular domain name response, this could be done via any free DNS service, while the latter requires setting up an authoritative domain server.**
The controller of the botnet first needs to create a domain or subdomain, which is administered from a special DNS server. This DNS server waits for special name lookups, which it then translates into incoming data. The DNS server then responds with the appropriate data using the agreed-upon semantics. We assume that the botnet controller (i.e.,

botmaster) has access to the authoritative domain name server for some domains or sub-domains. Bots across the Internet frequently receive commands and updates from a botmaster and launch attacks accordingly, as well as submit stolen data to the botmaster. We give brief background information on DNS records.

DNS Resources Records The DNS system allows a name server administrator to associate different types of data with either a fully qualified domain name or an IP address. To send a message to a bot, an adversary can store data in any one of these types of records.

– `A` record specifies an IP address for a given host name.
– `CNAME` and `MX` records can point to textual data representing the alias or mailing host of a particular host name.
– `TXT` records are designed to store arbitrary textual data up to 255 characters.
– `EDNS0` record allows storing up to a 1280 byte payload [4].

2.1 Codeword Mode

The *codeword mode* is a new stealthy communication mechanism. It requires a botnet operator to decide upon a set of agreed upon codewords *a priori*. Each codeword represents a specific type of commands or attacks. The codeword appears in the DNS query as an innocent hostname, for example `codeword.domain.com`. **(3)This hostname may be stored as any type of record (e.g. A, MX, CNAME). A request for an A or CNAME record tends to be the most common and therefore a preference should be given to these records types so that queries would appear most like legitimate traffic** The client queries `codeword.domain.com`, and waits for a particular value in the server's response. Upon receiving the query, the DNS server (controlled by the botnet operator) returns the pre-set response that contains command information. If the codeword corresponds to denial-of-service (DoS) attacks, then the response may represent a target of DoS attacks. If the codeword corresponds to update, the client may contact the IP address returned for updated code or other instructions.

It is important to note that the codeword can be chosen *arbitrarily* and does not need to correspond to a specific host or service. The codeword method allows a stealthy *one-way* commanding system. It can effectively evade detection approaches based on non-conforming packet sizes [6], i.e., DNS packets whose sizes are outside the range of [28, 300] bytes. Codewords may be arbitrarily generated, or may be common service names such as `www`, `mail`, or `ftp`. In the latter case, packet statistics cannot be performed to find anomalies.

2.2 Tunneled Mode

The purpose of tunneled mode is to allow the *two-way* transfer of arbitrary binary data between a server and a client. This mode is referred to as tunneled mode, as one can tunnel streaming data over this DNS communication method. **(3) I think this table explains the query types for tunneling well enough**

- *Upstream communication* is for a client to submit data to a (malicious) domain server. The client submits the data as a `CNAME` query by *i)* encoding the data using a base32 encoding, *ii)* using the encoded string to construct a host name, and *iii)* send a `CNAME` DNS query. An example is shown in Figure 1.
- *Downstream communication* is for the server to issue commands to clients. Upon receiving the above query from the client on a hostname h, the server *i)* encodes the response as base32 data, and *ii)* constructs and returns a `CNAME` record for h. An example is shown in Figure 1.

- Upstream: Ask CNAME for:
  `NBSWY3DPFQQHO33SNRSA000.domain.com`
- Downstream: CNAME points to:
  ```
  NBUSYIDCN5ZXG000.domain.com
  3600
  CNAME
  NBSWY3DPFQQHO33SNRSA000.domain.com
  ```

Fig. 1. Example data packets sent to and from a server in tunneled mode: To server: "hello, world". From server: "hi, boss". In this example, the domain server for domain.com is the malicious server and the response has one hour TTL.

To prevent DNS caching from disrupting the communications, the server may set a short *time-to-live* (TTL). This tunneling method gives an operator the most options after implementation as the data stream can be arbitrary. Because of the arbitrary payload, the distribution of packet bytes may differ significantly from conventionally DNS payload. We perform more analysis in Section 4.
There are two main characteristics of DNS-based communication. First, because the DNS protocol does not allow the server to initiate a connection with the client, the client needs to continually *pull* updates from the server. Second, DNS is based on UDP, and thus does not guarantee reliable data transfer or message order. To mitigate the problem, sequence numbers have to be appended to messages for bookkeeping purposes.
Both the tunneled mode and codeword mode require clients to frequently pull updates from name servers by querying the corresponding botnet's domain. Straightforward querying patterns are easy to detect (e.g., periodically sending DNS queries) and susceptible to simple aggregate analysis, such as counting DNS queries for each unique domains and identifying domains with abnormally large query volume at the host, local area network, or internet service provider levels. We analyze several simple-yet-effective methods for bots to hide their DNS traffic in the next section.

3 Query Strategies and Quantitative Evaluation

In this section, we play the devil's advocate and describe and experimentally evaluate new techniques for hiding DNS query activities on a host, in order to defeat anomaly detection that targets abnormal temporal patterns. The proposed strategies are useful for both the tunneling and codeword modes. We quantitatively analyze the detection countermeasures in Section 4.

3.1 Exponentially Distributed Query and Piggybacking Query

We describe an exponentially distributed query strategy and a piggybacking query strategy, both can be used to hide bot activities while communicating with a botmaster in a timely fashion. In our experiments in Section 3.2, we provide an experimental evaluation on both query methods.
Exponentially distributed query strategy The Poisson process is previously believed to be a suitable model for representing stochastic processes where arrivals are independent on each other, i.e., memoryless. In [9], client-side DNS request arrivals are modeled by Poisson processes with exponential random variables with different rates λ (e.g., 2.63 queries/hour for `www.google.com` and 0.78 queries/hour for `www.cnn.com`). In our exponentially distributed query strategy, a bot probabilistically distributes DNS queries so that their intervals follow an exponential distribution with a parameterized arrival rate λ_b. Because of the memoryless feature of the model, the bot does not need to store the previous communication history. One simple way to implement this query strategy is as follows.

1. The bot sends a DNS query;
2. It computes an interval t by drawing from an exponential distribution with parameter λ_b (hardcoded or dynamically generated);
3. The bot sleeps for t, and repeats from Step 1.

There is a trade-off between being stealthy and communication efficiency. We study a bot's strategy in finding an optimal λ_b in Section 3.2, given the host-wide DNS query rates.
Piggybacking query strategy Many (legitimate) websites contain content from multiple independent domains due to third-party content delivery, advertisements, or content mashup. Therefore, multiple DNS queries are usually issued by a host with temporal proximity. The composition of domains is usually dynamic. The piggybacking query strategy leverages this fact. A bot passively listens on the host's DNS traffic or name-translation related function calls and sends DNS queries when legitimate DNS queries are being made. Thus, the bot's query is blended among a group of legitimate DNS queries.
In the piggybacking query strategy, a bot's communication with the controller is constrained by the host's activities. Therefore, we focus on analyzing its timeliness, in terms of the dissemination efficiency of new command and data. We define *time-to-communicate* (TTC) and *minimum TTC* as follows. Minimum TTC is a threshold aiming to prevent a bot from sending queries too frequently.

Definition 1. *Time-to-communicate (TTC) is defined as the time interval between two network connections (DNS queries in our setting) of* *a bot for retrieving information from or submitting data to the botmaster server.*

Definition 2. *Minimum TTC is the lower bound of time-to-communicate. A bot does not send any DNS query if the bot's last DNS query was sent within the minimum TTC.*

In this piggybacking mode, bots need to know when a legitimate query is made. Since the DNS service in a server listens on port 53 for incoming requests, an outgoing packet from host to a destination IP on port 53 is an indication of a DNS request. Therefore, the bot program may monitor the host's network traffic through functions in a packet capture library, (e.g., `pcap`), and launch its own communication DNS query upon successful detection of legitimate queries.

An alternative method for learning the host's (legitimate) DNS traffic is to watch the calls for DNS-related APIs such as `gethostbyname()` function in Linux `libbind` library. `gethostbyname` looks up all IP addresses associated with a host name and is implemented in the resolver library. One way of hooking into the API function is for the bot to register a `.so` file (shared library) to the `LD_PRELOAD` environmental variable. In the registered `.so` file, the target API function is replaced by the attacker's version which can notify the bot whenever this function is called. Similarly, in Windows a bot may replace the corresponding Winsock DLL file in order to implant a DNS-notifier function.

3.2 Experimental Evaluation

The goal of this evaluation is to understand how effective the aforementioned stealthy query strategies are. Specifically, how soon botmaster disseminates commands to all or most bots; and how soon stolen data is harvested bots by botmaster? We do not allow bots to submit DNS queries at will, in order to avoid detection. We only allow bots to either piggyback their queries with legitimate DNS queries from the victim host, or follow a special inter-query distribution.

Our implementation uses the Python Modular DNS Server (`pymds`) and a specially designed plugin to respond to DNS requests. PyMDS implements the full DNS protocol while allowing the user to implement a programmatic and dynamic backend to generate the DNS records returned. Instead of returning records from a static file, PyMDS allowed for the decoding of codewords and the creation of appropriate responses.

To evaluate the *piggyback query strategy*, our dataset is a two-month-long network trace obtained from a university and collected with the IPAudit tool. The trace covered users from three departments and several research and education centers. (All machines were connected to the Internet wirelessly, i.e., there was no wired connection.) The raw dataset is 4.6GB. We identify and analyze the DNS traffic on port 53 of remote destinations. For data preprocessing, we select the most active 200 users from the our dataset by partitioning users by their (static) MAC address and sorting users by their traffic volume. We simulate the piggyback DNS-query strategy by having a bot send outbound communication whenever a host issues a UDP datagram on remote host port

53. Figure 2 shows the percentage of packets whose TTC is above the given a minimum TTC in a 10-hour-span. Three minimum TTC values are analyzed: 1, 30, and 60 minutes.

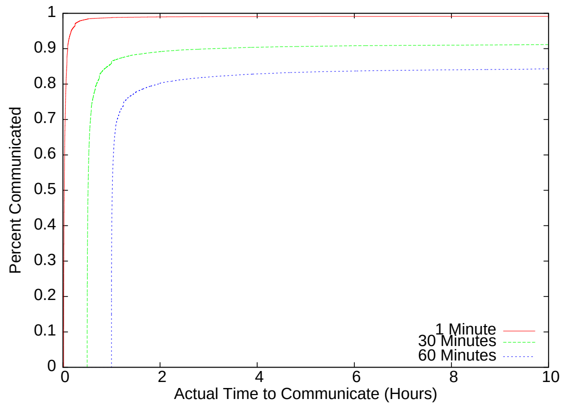

Fig. 2. Cumulative density function on the percentage of bots that have successfully sent at least one DNS query by piggybacking after a time period (X-axis). Each line corresponds to a different minimum TTC (1, 30, and 60 minutes). The figure shows a 10-hour span.

Results in Figure 2 show that the piggybacking query strategy is quite effective – at least 80% of bots are able to communicate with the botmaster within 2 hours. Clearly, there is a trade-off between minimum TTC and how soon bots communicate with the headquarter. For an active botnet where commands may change every day, minimum TTC may be set to 60 minutes.

Piggybacking case studies We select four hosts from our dataset to simulate the piggybacking behaviors on them and evaluate the mean time-to-communicate. The four hosts are the first, 50-th, 100-th, and 200-th most active hosts according to their total traffic volume during the 2-month period. Figure 3 plots how the mean TTC changes with the minimum TTC in a piggybacking query strategy. The results show that bot's communication efficiency is higher on more active hosts. Mean time-to-communicate grows with minimum TTC and is almost always greater than minimum TTC. Their relationships for the 200 hosts studied are shown in Figure 4.

For the *exponentially distributed query strategy*, our goal is to identify an optimal range for λ_b – bot's query arrival rate on a host. We analyze the difference between two distributions: *i)* host-wide inter-arrival time for regular DNS queries with arrival rate λ, and *ii)* inter-arrival time for the bot-mixed DNS queries, i.e., new arrival rate $\lambda + \lambda_b$, where λ_b is the bot's query rate.

We use Kolmogorov-Smirnov (KS) test, which is suitable for comparing unbinned distributions that are functions of a *single* independent variable

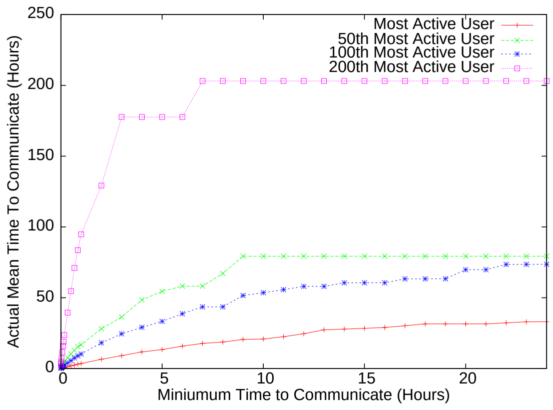

Fig. 3. Case studies on the time-to-communicate for four hosts with varying active traffic volume, given minimum TTC values shown on X-axis.

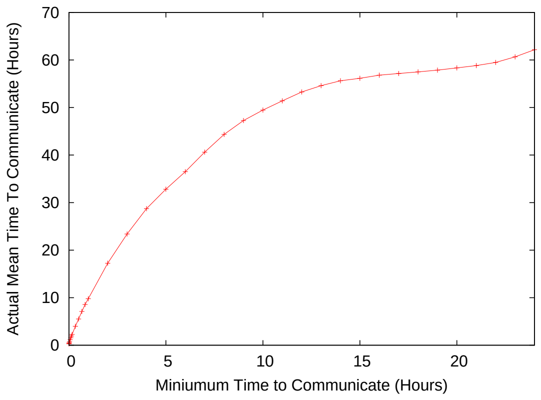

Fig. 4. Mean TTC vs. minimum TTC for 200 hosts.

as in our case [8]. In our KS test, a higher p value ([0, 1]) represents a higher resemblance between the normal and the bot-mixed distributions. To simulate the Poisson process, we use two estimated λ values – high arrival rate of 131.5 queries/hour and low arrival rate of 39 queries/hour – based on results from [9] and [14].

Intuitively, a higher legitimate DNS query rate makes it easier for a bot to blend in its traffic. Our results in Figure 5 and Figure 6 confirm the intuition. High rate $\lambda = 131.5$ is shown in Figure 5, and low rate $\lambda = 39$ in Figure 6, where each line represents a different amount of data collected: 10, 24, 48, and 100 hours. X-axis is the varying λ_b value. The horizontal line represents a 5% cut-off threshold that may be used for detecting anomalies.

Our results show that longer traces make it easier to discern data. Higher λ tolerates higher λ_b, allowing bots to communicate more often. Given a p value threshold, the KS test can be used to find a suitable λ_b. The experiments show that even when data is collected for long periods of time, such as 100 hours, it can be difficult to detect bots using a small λ_b. In the case of less active hosts, λ_b can be come undetectable at 4 requests per hour and with more active hosts, λ_b can be as high as 10 requests per hour.

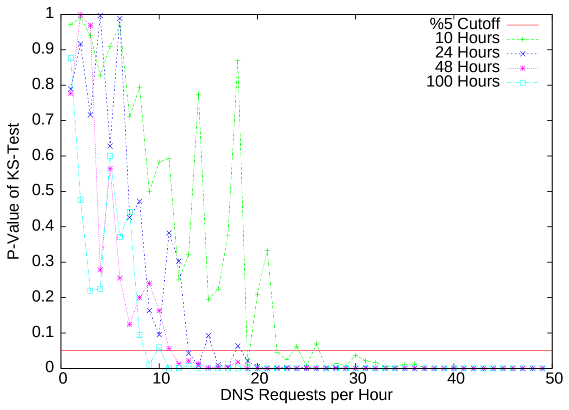

Fig. 5. KS test results between queries with the arrival rate of $\lambda = 131.5$ queries/hour and bot-mixed queries of $\lambda + \lambda_b$ (X-axis). Four runs of simulation lasting for 10, 24, 48, and 100 hours are shown.

Summary The experiments suggest that both the piggybacking and exponentially-distributed query strategies can be effective in allowing the majority of bots to communicate in a reasonable time frame without being detected. The exponentially-distributed query strategy gives the bot slightly more control over when to query. On the other hand, the optimal query rate λ_b depends on the host-wide query rate, which may change.

4 Perfect Stealth and Countermeasures

In this section, we describe and experimentally evaluate a countermeasure against DNS-based stealthy messaging systems that requires deep packet inspection and statistical analysis. Deep packet inspection examines packet payload beyond the packet header. Specifically, we quantitatively analyze the probability distributions of (bot's) DNS-packet content. We also give a formal definition for the perfect stealth in content-based communication channels in terms of the distinguishability of probability distributions between legitimate and attacker's traffic.

4.1 Perfect Stealth in Content-Based Covert Channel

Despite the existing work on constructing, measuring, and detecting general covert channels [**?**] and specific covert timing channels [**?**], there

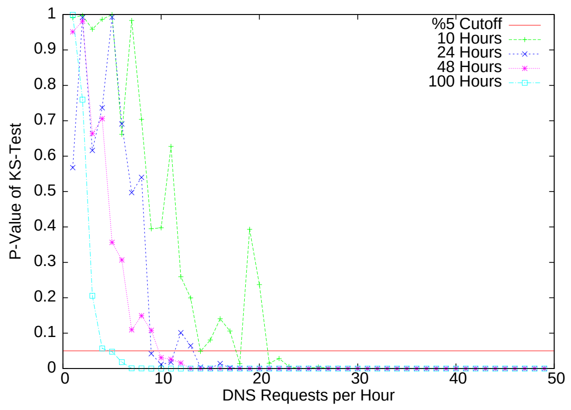

Fig. 6. KS test results between queries with the arrival rate of $\lambda = 39$ queries/hour and bot-mixed queries with $\lambda + \lambda_b$ (X-axis). Four runs of simulation lasting for 10, 24, 48, and 100 hours are shown.

still lacks any formal definition for specifying *perfect* stealth in covert channels. We refer to content-based stealthy communication channel as where an adversary aims to hide her communication among normal payload. Such a channel differs from covert timing channel [**?**] where the adversary cannot modify the packet payload, but can tamper with the packet-sending schedule to reveal sensitive data. Next, we present a formal definition aiming to capture the ultimate goal of the adversary in realizing stealthy communication. We further discuss the fundamental limits and practical issues in encapsulating and detecting stealthy communication channels over the Internet.

Our definition for the perfect stealth in content-based covert channel is formulated following the indistinguishability in Equation 1, and is specified as a game between a challenger and a defender as follows. Specifically, our definition is given in terms of the indistinguishability between attacker's communication distribution and that of normal communication (not random distribution). Given the legitimate message space $\mathcal{M}_0$ and malicious message space $\mathcal{M}_1$ of equal size, our definition is modeled as a game between a challenger $\mathcal{C}$ – who chooses a challenge message M_b^* randomly from $\mathcal{M}_0 \cup \mathcal{M}_1$, and a defender $\mathcal{D}$ – who aims to break the perfect covert channel and guess the bit b. During the game, the defender $\mathcal{D}$ can learn some classified messages (with labels) (as the preparation) from the challenger, denoted by $\{\hat{M}\}$. The channel is perfectly covert if and only if defender's guess b' equals b with $\frac{1}{2} + \epsilon$ probability, where ϵ is negligible.

$$Pr[b' = b \mid \mathcal{C} \xrightarrow{M_b^* \notin \{\hat{M}\}} \mathcal{D}; \mathcal{D} \xrightarrow{outputs} b'] = \frac{1}{2} + \epsilon \text{ (negligible)} \qquad (1)$$

We note that this definition applies to all types of communication protocols, not limited to DNS. Similar definitions can be given to capture

the indistinguishability of the temporal property in legitimate traffic and adversary's traffic, e.g., probability distributions of query intervals.
The perfect stealth imposes a very strong requirement for attackers. Because botnet is designed to carry out special information and data, known botnet communication indeed has characteristics patterns different from legitimate traffic in practice – making it possible to detect. This observation implies the limitation of attackers/malware in hiding their content over the Internet.
On the other hand, as long as the malware communication has a different probability distribution from the normal traffic, then given sufficient storage and computation power, defenders can deploy deep packet inspection to detect suspicious sets of packets. We demonstrate the use of information theoretic measures, namely Jensen-Shannon (JS) divergence, for the analysis in the next section.

4.2 A Countermeasure Based on Deep Packet Inspection

We describe and evaluate a concrete countermeasure against stealthy DNS channels through statistically analyzing traffic content. To compute the byte distribution in normal and tunneling traces, we use the Jensen-Shannon (JS) Divergence D_{JS}, which is a common metric for quantifying the difference between two probability distributions P and Q, and is a commutative version of Kullback-Leibler divergence of Q from P. A lower D_{JS} value means a higher similarity in two probability distributions. The JS Divergence is particularly suited in situations where the random variable is discretized. It is computed as follows.

$$M = \frac{1}{2}(P + Q) \tag{2}$$

$$D_{KL}(P, Q) = \sum_{i=0}^{n} p_i \log \frac{p_i}{q_i} \tag{3}$$

$$D_{JS} = \frac{1}{2}(D_{KL}(P, M) + D_{KL}(Q, M)) \tag{4}$$

We experimentally compare DNS packet traces recorded on a host, specifically, on how different tunneling packets are from legitimate ones in terms of the probability distribution of content, assuming that content is not encrypted.
(8) Such probability measures may be taken on a per-host or per subnet basis, however since a filter based on these methods must only keep an probability distribution of the bytes in a packet, no identifying information can be inferred. In this way privacy concerns can be kept at a minimum.
Note: More formally we might say identifying information is contained in high order dependencies i.e to identify the word HACKER we must know that R occurs only after, E which occurs only after K and so on. This would be require a knowledge of 5th order dependencies. Informally to determine if a particular word were in a packet we would need to know

$P(R|E|K|C|A|H)$. **In fact we store probability without any dependence i.e.**$P(H)$. **Anyways that is a mouthful and I am not sure we want to put all that in.**

In the following tests, three normal DNS traces were recorded and one tunneling DNS trace via tunneled mode was recorded. Each trace corresponds to an hour-long network activities on a host. Sizes of our traces are as follows: 862KB for the tunneling trace, 823KB for normal trace 1, 699KB for normal trace 2, and 153KB for normal trace 3. **(4)In addition, the tunnel trace contained 191 A queries and 1433 TXT queries, while the normal trace 1 contained 1750 A queries and no TXT queries, and normal trace 2 contained 2417 A queries and no TXT queries.** Tunneling trace contains encrypted Secure Shell (SSH) activities, i.e., SSH traffic through DNS tunneling.

When the entire packet including header is analyzed, we find that the divergence of normal traces (normal 1 and normal 3) is large (not shown). To get a more stable comparison, we drop the UDP headers and only observe the DNS payload. Figure 7 shows how the Jensen-Shannon divergence changes as more tunneling message carrying packets are mixed in. The X-axis is the ratio of tunnel trace to normal trace 1. Our results show that a divergence threshold of 0.015 can sufficiently distinguish normal traces from mixed traces containing more than 30% bot queries. These results indicate that analyzing DNS payload gives a better result than the entire DNS datagram; and the JS divergence can be used for determining anomalies in a stream of DNS packets.

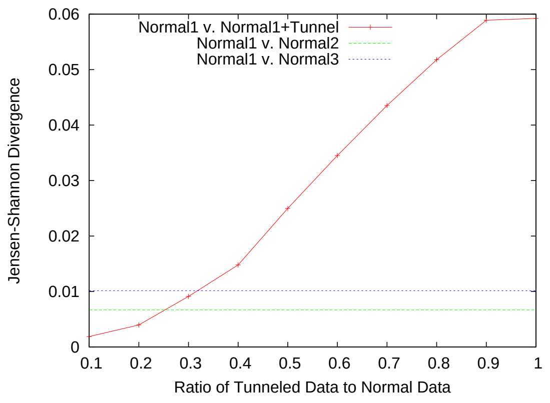

Fig. 7. Divergence computed from the payload of UDP datagrams. Horizontal lines represent the divergence of normal streams. The red line is the divergence of mixed traces.

Practical limitations: This countermeasure requires the access of DNS packet content and thus may not be scalable for real-time analysis, especially for high-bandwidth routers. **(6)Each host would require a 256 item floating point array to hold the probability distribution**

of the DNS packets, over thousands of hosts, this could create issues with storage. I am not sure how to quantify processing time required rigorously. In addition, many legitimate applications use DNS for storing non-IP data, such as public keys in the DomainKeys protocols [2, 3]. The above content-distribution based analysis may result in false positives.
There are also several practical issues and constraints when executing the detection in reality, besides the computational and storage costs. *i)* Because of traffic diversity such as in HTTP and SMTP, it is difficult to generate the standard probability distributions representing legitimate traffic in general. *ii)* The use of end-to-end encryption may prevent defenders from analyzing byte distribution of payload. *iii)* For stealthy communication as in our codeword mode described in Section 2.1, the attacker's extra payload is small and subtle without significantly affecting the overall byte distributions. The abnormal traffic is mixed with and diluted by normal traffic, making it harder to detect. Conventional signature-based detection relies on known patterns, and is not effective against new malware activities.

5 An Open Question on Domain Names

Understanding the capability of adversaries in setting up stealthy DNS-based communication channel is important. In this section, we describe an open question about how to automatically generate a large number of realistic-looking domain names for command and control purposes.
Long-lived domain names are easy to manage and cheaper to maintain, however, they are susceptible to aggregate analysis. Domain flux refers to using short-lived domain names in botnet C&C [**?**,12]. Domain flux typically requires bots and botnet controller to independently derive new domain names periodically. To have short-lived domains, a static approach is to have a botmaster generate an ordered list of domain names and pack the list in malware code for bot to look up. However, there are two disadvantages for this method: large storage and high code-homogeneity – long lists of domain names shared by all bot code making the code susceptible to signature-based malware detection. An alternative is for the botmaster to send to all bots the new domain name during the current epoch. However, a communication failure may prevent the bots from learning the correct name for the next epoch.
One simple approach is for bots and their controller to independently compute the hash value of an incremental counter and a shared secret at each epoch, i.e., $H(counter \| secret)$, where H is a one-way collision-resistant hash function. However, automatically generating realistic-looking domain names by distributed parties is still an open question, which is further explained next.
Popular (and legitimate) domain names usually have semantics, i.e., meanings, whereas automatically generated domains do not. This difference (among other features) was recently used to identify anomalous domains [**?**]. For hash-generated domains and legitimate domains, their entropy may differ. For example, we obtained the top one million popular

domains from `alexa.com` on May 25, 2010, which we used to represent legitimate domain names. For hash-generated domains, we use the first 32 bits of a hash value to generate 8 characters. The average entropy for hash-generated domains is 2.97, which is close to the maximum entropy for an 8-character string: $-\sum_{i=1}^{8}\frac{1}{8}\log_2\frac{1}{8} = 3$. In comparison, the averaged entropy for legitimate domains is 2.17. We also evaluate the Mahalanobis distance for comparing byte distributions in `alexa.com` domain names and hash-generated domains in Figure 8. hash-generated domains give slightly higher Mahalanobis distances than legitimate domains. An interesting finding is that outlier domains from `alexa.com` that give high Mahalanobis distance tend to contain digits, e.g., `8555.com` and `c8048.com`.

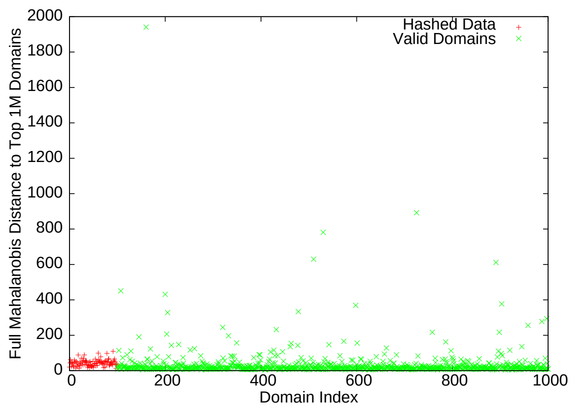

Fig. 8. Mahalanobis distances of 100 hash-based domains and 1,000 legitimate domains. X-axis is domain index with hash-based domains in 1, 100].

These evidences prompt us to raise an interesting open question as to how to design algorithms and protocols that enable multiple parties to automatically generate synchronized domain names that resemble legitimate domains and evade statistics-based detection.

6 Related Work

Despite the fact that DNS tunneling is known for bypassing firewalls and encapsulating arbitrary data such as SSL traffic [4, **?**], Exploring DNS protocol as a practical command-and-control channel and identifying its limitations have not been scientifically studied. Various proof-of-concept botnet command and control systems via unconventional media exist, such as via bluetooth [**?**] and social networks [**?**]. In comparison, our work is useful beyond the specific DNS-based communication channel studied in two aspects.

– We present new quantitative techniques and evaluation regarding the detection and construction of general-purpose distributed stealthy communication systems, including temporal strategies for making stealthy communication and statistical content analysis.
– We give the first attempt to formalize the perfect stealth in content-based covert channel and point out its practical implications. We also describe an open question with a cryptographic and algorithmic flavor regarding how to automatically generate useful domain names.

For DNS-based anomaly detection, Karasaridis *et al* described the use of the Kullback-Leibler distance mentioned in Section 4 to measure byte distribution in DNS datagrams [6]. Dagon [1] proposed to quantify how anomalous the number of queries for each domain name during an hour in a day with Chebyshev's inequality and distance measures previously used for examining anomalous payloads. DNS-based anomaly detection approaches are presented in [13] for detecting botnet C&C activities. One method is to detect dynamic domain names whose query rates are abnormally high or temporally concentrated using outlier detection metrics such as Chebyshev's inequality. Our work describes stealthy DNS behaviors whose querying patterns are hard to distinguish with legitimate domains, which make the counting based detection less effective. We note that DNSSec protocol does not prevent stealthy communication via DNS.

Stone-Gross *et al* observed the use of domain flux in Torpig botnet [12], where new communication domains are generated periodically and registered by the C&C server. Torpig bots communicated with the server over HTTP, after resolving the domain name. In comparison, we investigate the feasibility of solely DNS-based command and control, without requiring any additional Web servers.

Our piggybacking DNS queries should not be confused with previously reported piggybacking methods for reducing DNS traffic. Those techniques usually take advantage of empty payload space in UDP datagrams. For example, renewal using piggyback method was proposed to piggyback cached DNS records to DNS queries to refresh expired cached records [5]. Related domains may also be piggybacked in DNS queries [10], e.g., to include `i.cnn.net` in the DNS packet for `www.cnn.com` as they are likely to be requested together by the browser.

Millen did pioneering work on covert-channel analysis [**?**,**?**], in particular in a system (host) environment. Covert channel has been heavily analyzed in the context of traffic-analysis prevention [**?**] and routing anonymity [**?**]. Our perfect covert channel definition (for one-to-one communication) can be applied to traffic matrix (for n-to-n communication) defined in [**?**]. Our work differs from them in that we focus on experimentally evaluating and detecting practical covert channels across the Internet.

7 Conclusions

We conducted a systematic study on the use of pure DNS queries for massive-scale stealthy communications among entities on the Internet.

Our work shows that DNS – in particular the codeword mode combined with advanced querying strategies – can be used as a stealthy command-and-control channel. Because almost all computers need domain-name resolution, it is impossible to block DNS traffic. For the tunneling mode, we presented a payload-inspection based countermeasure for detecting anomalies in DNS traffic through analyzing the probability distributions of content. However, the payload inspection techniques do not apply to codeword systems.
The focus of this paper is *not* on presenting offensive techniques for attackers. Rather, we used information theoretic analysis and experiments to illustrate the need and importance of understanding the potential capabilities of adversaries. We further pointed out that although it may be difficult for attackers to achieve a perfect stealth for C&C, practical constraints may prevent detection methods such as the JS divergence test from being effective. We leave an open question on how to algorithmically generate short-lived and realistic-looking domain names.
For future work, we plan to formally give the definitions for the indistinguishability of temporal property in legitimate traffic and adversary's traffic, in particular, in terms of the distributions of query intervals. We will prove that our query strategies give the perfect stealth in terms of these distinguishability definitions.

References

1. D. Dagon. Botnet detection and response, the network is the infection, 2005. Domain Name System Operations Analysis and Research Center Workshop.
2. Yahoo! Anti-Spam Resource Center - DomainKeys. http://antispam.yahoo.com/domainkeys, "2008". This is an electronic document. Date retrieved: February 1, 2007.
3. M. T. Goodrich, R. Tamassia, and D. Yao. Accredited DomainKeys: a service architecture for improved email validation. In *Proceedings of the Conference on Email and Anti-Spam (CEAS '05)*, July 2005.
4. M. V. Horenbeeck. Dns tunneling. `http://www.daemon.be/maarten/dnstunnel.html`.
5. B. Jang, D. Lee, K. Chon, and H. chul Kim. Dns resolution with renewal using piggyback. *Journal of Communications and Networks*, 11(4), August 2009.
6. A. Karasaridis, K. S. Meier-Hellstern, and D. A. Hoeflin. Detection of dns anomalies using flow data analysis. In *GLOBECOM*. IEEE, 2006.
7. A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 7–7, Berkeley, CA, USA, 2007. USENIX Association.
8. D. A. W. Myles Hollander, editor. *Nonparametric Statistical Methods.* Wiley-Interscience, second edition, 1999.
9. M. A. Rajab, F. Monrose, A. Terzis, and N. Provos. Peeking through the cloud: Dns-based estimation and its applications. In S. M.

Bellovin, R. Gennaro, A. D. Keromytis, and M. Yung, editors, *ACNS*, volume 5037 of *Lecture Notes in Computer Science*, pages 21–38, 2008.

10. H. Shang and C. E. Wills. Piggybacking related domain names to improve dns performance. *Comput. Netw.*, 50(11):1733–1748, 2006.
11. K. Singh, A. Srivastava, J. T. Giffin, and W. Lee. Evaluating email's feasibility for botnet command and control. In *DSN*, pages 376–385. IEEE Computer Society, 2008.
12. B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: Analysis of a botnet takeover. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, November 2009.
13. R. Villamarín-Salomón and J. C. Brustoloni. Identifying botnets using anomaly detection techniques applied to dns traffic. In *Proceedings of the 5th IEEE Consumer Communications and Networking Conference (CCNC)*, 2008.
14. H. Xiong, P. Malhotra, D. Stefan, C. Wu, and D. Yao. User-assisted host-based detection of outbound malware traffic. In *Proceedings of International Conference on Information and Communications Security (ICICS)*, December 2009.