Leaving Timing Channel Fingerprints in Hidden Service Log Files

Bilal Shebaro[†], Fernando Perez-Gonzalez[‡], and Jedidiah R. Crandall[†]
[†]Univ. of New Mexico Dept. of Computer Science
[‡]University of Vigo, Spain, Signal Theory and Communications Dept.
[‡]Univ. of New Mexico Dept. of Electrical and Computer Engineering
{bshebaro,crandall}@cs.unm.edu, fperez@gts.tsc.uvigo.es

Abstract

Hidden services are anonymously hosted services that can be accessed over an anonymity network, such as Tor. While most hidden services are legitimate, some host illegal content. There has been a fair amount of research on locating hidden services, but an open problem is to prove that a physical machine, once confiscated, was in fact the machine that had been hosting the illegal content. In this paper we assume that the hidden service logs requests with some timestamp, and give experimental results for leaving an identifiable fingerprint in this log file as a timing channel that can be recovered from the timestamps. In 60 minutes, we are able to leave a 36-bit fingerprint that can be reliably recovered.

The main challenges are the packet delays caused by the anonymity network that requests are sent over and the existing traffic in the log from the actual clients accessing the service. We give data to characterize these noise sources and then describe an implementation of timing-channel fingerprinting for an Apache web-server based hidden service on the Tor network, where the fingerprint is an additive channel that is superencoded with a Reed-Solomon code for reliable recovery. Finally, we discuss the inherent tradeoffs and possible approaches to making the fingerprint more stealthy.

1 Introduction

In this paper, we consider the problem of leaving fingerprints in the log files of hidden services so that if the machine hosting the service is recovered by law enforcement the fingerprint can be recovered as proof that that particular machine was hosting the service. Our threat model is the following. Illegal content is being hosted on a hidden service of an anonymity network such as Tor [5]. Hidden services allow clients on the anonymity network to access the service while preserving the anonymity of both the client and server. The server's IP address is not revealed to clients, instead clients request the service using a pseudodomain, *e.g.*, `http://gaddbiwdftapglkq.onion/`. There are many ways that the hidden service can be identified, both technical (*e.g.*, the methods [12, 18, 1, 13, 16] that we describe in Section 6 where we discuss related works) and non-technical (*e.g.*, the crime is reported). In this paper we consider the following problem: given a hidden service that is believed to be hosted by a machine that will be confiscated by law enforcement, how can we leave a fingerprint on the machine through the hidden service that can be recovered and identified on the physical machine at a later time[1]?

The threat model we assume in this paper is a passive observer that does not suspect this form of fingerprinting but does observe bursts in the log file. A stronger threat model where the hidden service host suspects that fingerprinting will be employed is left for future work. The approach we take in this paper is to assume that the underlying service that is being offered as a hidden service has a log file of client requests that contains a timestamp. For our implementation, we use an Apache web server. By making requests to the service from a client on the anonymity network that will be logged, we can create an additive timing channel to leave the fingerprint in the log. The main challenge for this type of channel is the tradeoff between stealth and the amount of time it takes to leave the fingerprint. Because IP addresses are hidden by anonymity technologies, we assume that during

[1] In practice, more than one fingerprint will typically be left to ensure sufficient evidence for conviction.

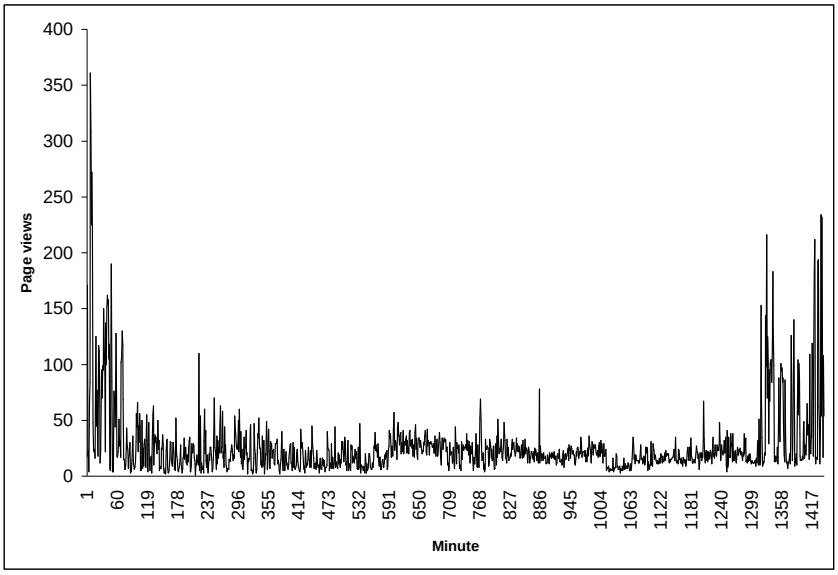

Figure 1: **Web server traffic for a 24-hour period.**

the process of recovering the fingerprint no distinction can be made between the added requests and requests from real clients. The two main sources of noise that must be accounted for through redundancy in the fingerprint, then, are the delays of requests that are added by the anonymity network and bursts in the actual traffic from real clients for that particular service. We present results that characterize both of these sources of noise, and describe an implementation that can leave an easily recoverable 36-bit fingerprint in an Apache log file over the Tor network in 60 minutes.

There are three main reasons why, among the many information channels various log files afford, we focus on only timing channels using the timestamps:

- For legal reasons, standardized methods are preferable to ad-hoc methods, because precedents can be established for well-analyzed algorithms for recovering a footprint. This requires that a single method be used for many services, and, while various services log different data that is application-specific, most contain some sort of timestamp.
- Anonymization technologies sometimes hide IP addresses, URLs, and other objects in the log file. For example, when Apache is set up as a Tor hidden service using privoxy [19], the IP address for all logged GET requests is `127.0.0.1` due to local proxying. Timing information, on the other hand, is typically preserved.
- By using exclusively timing and timestamps for leaving the fingerprint, the other channels of information (*e.g.*, the URL of the document being requested) can be reserved for other information that the fingerprinter may want to preserve in the log (*e.g.*, proof of the existence of a file on the server at a given time).

The rest of this paper is structured as follows. First, we describe our measurement methodology for characterizing the two main sources of noise in Section 2, followed by the results from these measurements in Section 3. Then, we describe our implementation of hidden service fingerprinting in Section 4. A discussion of stealth techniques and possibilities for future work is in Section 5. Related works are discussed in Section 6, followed by the conclusion.

2 Measurement methodology

In this section we describe our methodology for two sets of measurements: delays of HTTP GET requests in the Tor network, and the traffic characteristics of existing GET requests for a web service. Because our fingerprinting method uses an additive code where GET requests are added to existing traffic in the log file, Tor network delays and existing GET requests are the two main sources of noise that must be accounted for to reliably recover the fingerprint.

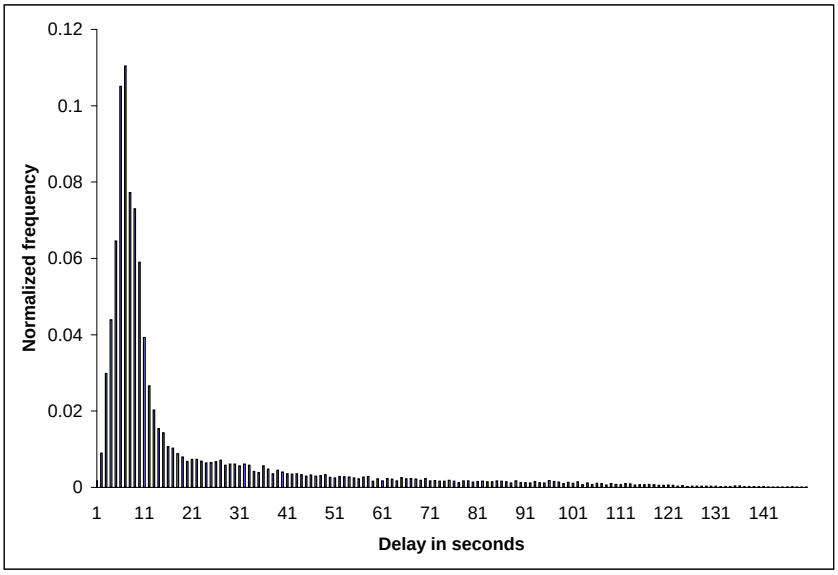

Figure 2: **Histogram for Tor delays in seconds.**

2.1 Tor network delays

To measure the delays of (and potentially also dropping of) GET requests that the fingerprinter as a client will send to the hidden web service, we set up two Linux machines. One hosts the Apache web server version 2.2.12 as a Tor hidden service and is configured using privoxy [19], which acts as a local web proxy interface to the Tor network. Thus, all GET requests appear to come from the IP address `127.0.0.1`, which is the loopback interface. The Apache log file logs GET requests when they are received, with a granularity of 1 second. For these experiments, the other machine acts as a web client, also configured with privoxy, with wget as the web client which makes requests for the hidden service. The server and client were located in the same place geographically, but Tor onion routing sets up a new circuit between server and client every ten minutes that each use different entry, exit, and intermediate nodes which are geographically distributed across the globe. *For measurement purposes only*, each GET request was tagged with a unique integer identifier so that we could easily detect dropping and reordering of packets. For our fingerprinting implementation in Section 4, we assume that no such identifier can be put in the GET request and use only the timing channel.

We sent a GET request from the client to the server every 5 seconds for 55 hours, to ascertain the distribution of delays introduced by onion routing and the connection loss rate. Because different circuits being built on a global scale every 10 minutes accounts for most of the variance in Tor delays, diurnal patterns were not evident in this data so we consider it as a single distribution independent of the time of day.

2.2 Existing HTTP GET requests

The other major source of noise in our fingerprint timing channel is existing traffic in the log file. For our implementation, we assume that the fingerprinter, when recovering the fingerprint, cannot distinguish their requests from existing requests (from real clients that are accessing the service). The additive channel is thus prone to errors due to bursts of traffic in the web server, so we sought to characterize existing traffic for a real web server. We obtained the log file for a university department that runs Apache 2.2.3 and receives approximately $35K$ to $40K$ requests per day. The log file we obtained covers a 36-hour time period.

We calculated the distribution of GET requests per minute for a normal web service using this log file. We assert that this distribution shape is representative of typical web services, but the mean of the distribution depends on how popular the web service is. Our fingerprinting technique assumes that the fingerprinter has some estimate of this mean for a given web service that they want to fingerprint[2].

[2]This estimate can be a conservative estimate, at the cost of finger-

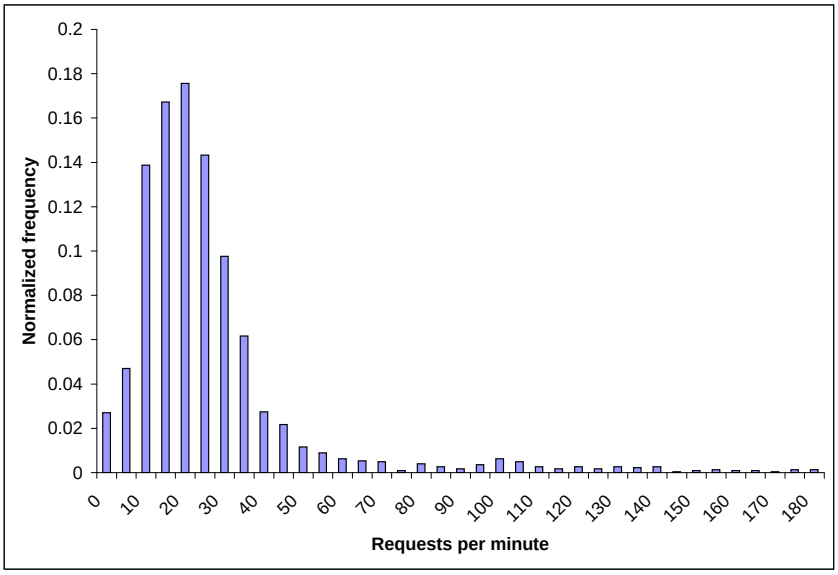

Figure 3: **Histogram for existing HTTP GET requests per minute.**

Figure 1 shows the number of requests over time for this department web log file. There is a spike in traffic at about midnight GMT, which corresponds to late afternoon local time. We designed our fingerprinting algorithm to be robust for any hour of the day, but since the fingerprinter can choose when to leave the fingerprint they can also choose a time of day when the traffic for the web service is known to be lower.

3 Measurement results

The purpose of our measurements was to characterize the two main sources of noise for our finerprinting technique: delays or connection drops by the Tor network and existing traffic in the web service log file. The main tradeoff we consider in this paper is the amount of time it takes to leave the fingerprint *vs.* the number of requests we need to make per minute. The faster fingerprinting is performed, the more requests per minute will be necessary to reliably recover the fingerprint. Our main consideration in this paper is this tradeoff, we do not make any claims regarding the stealthiness of our current implemention. Other stealth techniques besides reducing the rate of requests are discussed in Section 5.

The main questions regarding the speed *vs.* request rate tradeoff that we sought to answer are:

printing taking a longer time if the mean is overestimated, due to the redundant bits necessary for the error correction code.

1. When the fingerprinter sends a request, what is the loss rate and distribution of delays for when the server actually sees and logs the request? This is one of the two major sources of noise in our timing channel fingerprinting. Note that anonymity networks such as Tor deliberately route packets in unpredictable ways, meaning that the delays will also be much more unpredictable than normal Internet traffic.

2. What is the distribution of existing traffic in a web service log file? In particular, the relationship between the number of requests we add per minute and bit error rate when recovering the fingerprint is determined by this distribution.

3.1 Tor delays

Figure 2 shows the histogram for the delays added to requests by the Tor network from our measurements. The mean delay is 21.1. Not pictured are the 2975 requests that were dropped. Based on the measured probability density, 83.6% of sent requests will be logged by the server within a minute of being sent, 8.9% will arrive after more than a minute, and 7.5% will be dropped. Based on these results, for our additive timing channel we chose to send all requests for a 60-second period in a burst at the beginning of the 60-second period. They will arrive with roughly the same distribution shown in

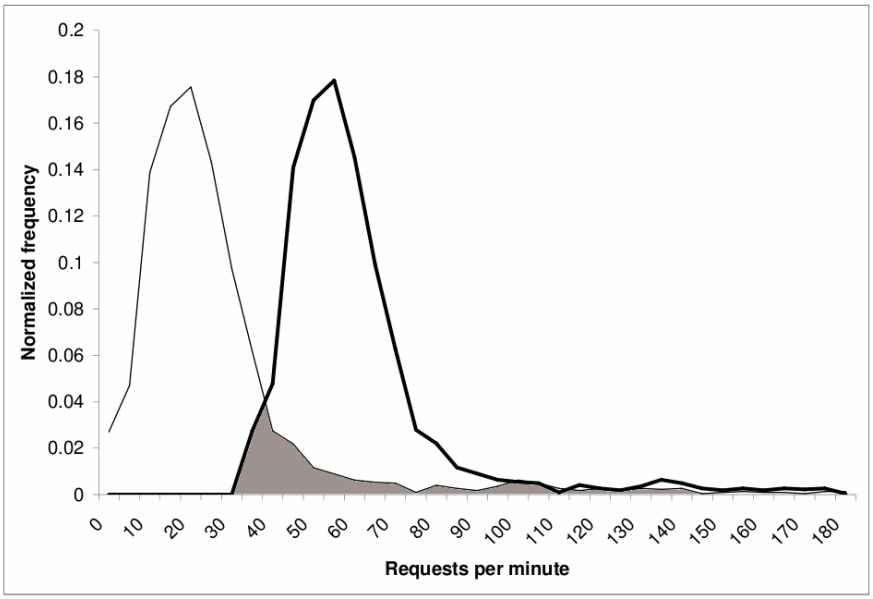

Figure 4: **Histogram of existing HTTP GET requests per minute before (thin line) and after adding additional requests for fingerprinting purposes (bold line).**

Figure 2. Because of these delays, the server will not see the packets in a burst as the client had sent them. This is important so that conspicuous bursts of traffic do not appear in the log file.

3.2 Existing HTTP GET requests

Figure 3 shows the distribution of requests per minute for the university department log file. This distribution helps us to predict the rate of errors due to existing requests that we can expect for different numbers of requests per minute added. This is illustrated in Figure 4, which shows this error rate for the parameters we chose for our implementation as the shaded area under the intersection between the two curves (the bold line is the same distribution with 35 added to each value). For this chosen parameter of adding 35 requests in a minute for our additive channel, the error rate for errors that are due to existing requests is approximately 11.5% and the optimal cutoff to determine whether requests were added to a given minute is 40. Lowering the parameter of 35 to make the fingerprint less conspicuous will move the bold curve to the left and increase the error rate, meaning that longer error correction codewords are needed and therefore more time is required to leave a fingerprint that can be reliably recovered. Increasing this parameter moves the bold curve to the right which decreases the error rate and makes fingerprinting quicker, but creates a more conspicuous burstiness in the log file.

Figure 3 shows the same shape for the histogram as previous work on measuring Tor network delays [10], but the mean of our histogram is different. One possible reason for this is that the Tor network has grown considerably in the past several years. Another possible reason is that in that work changes were made to the Tor client to choose the first introduction point from the hidden service descriptor, whereas our measurements are end-to-end application-layer measurements from when the request was made by the client to when the server logged the incoming GET request. In general, we found the Tor network delays to have a large amount of variance between different circuits. In our additive channel, we can account for this variance, but in future work to improve the stealthiness of fingerprinting a more detailed and up-to-date model of Tor network delays will be needed.

4 Implementation

Based on the results in Section 3, we developed a prototype implementation of timing channel log file fingerprinting that is based on an additive channel that is superencoded with a Reed-Solomon code [21]. In this section we describe the implementation and present results to demonstrate that it is robust to Tor delays and exist-

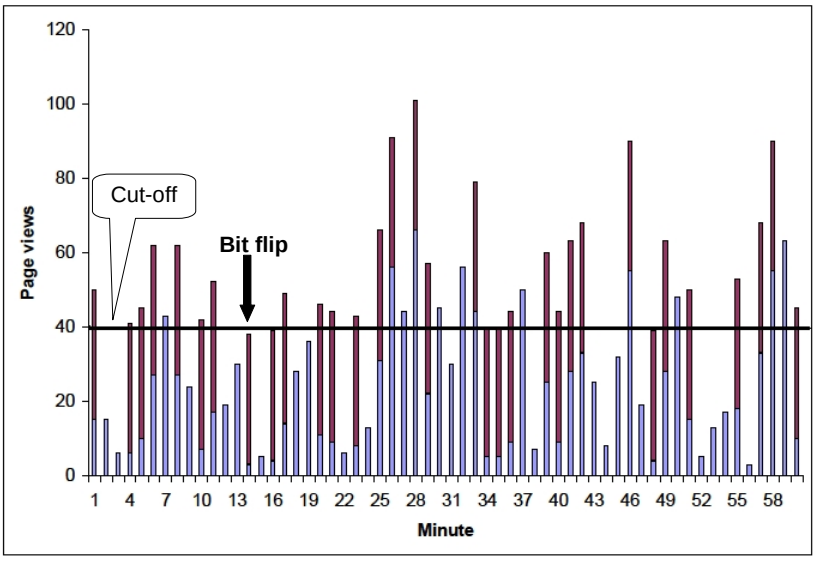

Figure 5: **How a 60-bit codeword appears in the log file.**

ing requests in terms of the ability to reliably recover the fingerprint.

Our implementation starts with a random 36-bit bitstring that will serve as the fingerprint. In this way, the probability of recovering a matching fingerprint by chance is approximately $\frac{1}{2^{36}} \approx 1.46 \times 10^{-11}$. Before being sent over the Tor network to appear in the server log file, this fingerprint is superencoded as a Reed-Solomon code word, which is 60 bits in length. Reed-Solomon coding is an error correction code that uses redundancy of the original word in a codeword that is based on an oversampled polynomial to recover from errors automatically (in contrast to an error detection code, in which errors could be detected, but not corrected).

We then transmit this 60-bit codeword, at a rate of 1 bit per minute, to the server's log file as follows. To transmit a 0 in a given minute, we do nothing. To transmit a 1 we make 35 requests for the hidden service over the Tor network at the beginning of the minute. These requests will arrive at the server and be logged with the distribution shown in Figure 2. The overall distribution of requests that will be seen in the log is not conspicuous, as is shown in Figure 8 where lights bars show the histogram before fingerprinting and dark bars show the histogram after fingerprinting.

To recover the fingerprint once the log file is obtained, we scan the log file near the time when the fingerprint was transmitted and attempt to recover the code word as follows. Within a minute, we read a 0 if less than 40 requests appear in the log file within that minute, and a 1 for 40 or more requests. We then apply Reed-Solomon error correction and compare the word that is recovered from the measured codeword to the fingerprint. When a match is found, then we are highly confident that the log file provided is the log file for the hidden webservice that was fingerprinted. Bit errors can occur in two different ways, which are discussed in Section 4.1.

We tested our fingerprinting implementation for different hours of the day for the department log file as follows. First, we do the fingerprinting for a Tor hidden service and record the requests as they are received by our hidden service. Then we superimpose this onto an hour of the log file. This is equivalent to doing the fingerprint live, since we are using an additive channel, but allows for repeatability of our results and does not require access to a hidden service that receives a lot of traffic. We tested the fingerprinting 24 times (once for each hour of the day) and were able to recover the fingerprint 22 times. This includes three tests that were performed for the three highest-traffic hours of Figure 1. We did this to test the robustness of the fingerprinting implementation in the limit. The fingerprints recovered in the two cases that failed had a low Hamming distance with the original fingerprint, but our results suggest that leaving multiple fingerprints at different times of day is important in practice.

Note that Reed-Solomon codes perform well for correlated bit errors, which is why they are used in scenarios

Input Codeword	Page View / Min	Page View / Min after Fingerprinting	Output Codeword
:	:	:	:
1	50	83	1
0	7	9	0
0	25	25	0
1	7	38	0
1	28	66	1
1	33	68	1
0	25	25	0
1	8	41	1
:	:	:	:

Figure 6: **Details on a bit error from the example.**

such as compact disc encoding where bit errors can be due to scratches. Thus, the small correlations between bit errors in our scheme (such as delayed requests showing up in the next minute) are easily handled by the superencoding.

4.1 Example

Here we give an example of one iteration of fingerprinting. The process is shown in Figure 7. The process can be repeated to leave multiple fingerprints at different hours of the day for added robustness, but here we describe only one iteration, which takes 60 minutes. The first step is to choose a random 36-bit word as the fingerprint. In our example we choose "1101 1000 1111 0011 1100 0101 1010 0010 1101". The second step is to apply Reed-Solomon encoding to produce a 60-bit codeword: "1001 1101 0110 0101 1001 1010 1101 1000 1111 0011 1100 0101 1010 0010 1101". For each minute in the 60-minute process, if the corresponding sequential bit in the codeword is a 0, we do nothing, if it is a 1 we make 35 requests from a Tor client to the hidden service at the beginning of the minute.

After the fingerprinting process is complete, we assume that the machine that hosted the hidden service has been physically recovered (*e.g.*, by law enforcement) and then the second half of the process is to recover the 36-bit fingerprint. To account for inaccuracies in the hidden server's system time, the process of attempting to recover the fingerprint can be repeated for some number of seconds into the past or future. For a given start time alignment, the 60-bit codeword that is received in the log file is generated as follows. For each sequential minute, we record a 0 if less than 40 total requests appear in the log during that minute, and a 1 if 40 or more appear. If there are 12 bit errors or less then applying Reed-Solomon decoding to this 60-bit codeword will produce the original 36-bit fingerprint.

There are two types of bit flips. One is that the number of requests from actual clients during that minute is very low and thus the number of requests we add to the log file for a 1 (which can be less than 35 if Tor drops some connections or delays them for more than 60 seconds) is not sufficient to put the total above the threshold of 40. This will flip a bit in the codeword from 1 to 0. The more common type of bit flip is from 0 to 1. This happens when either the number of existing requests from actual clients already exceeds the threshold, or it is near the threshold and a few delayed requests from a previous bit in the codeword show up during that minute instead of the earlier minute they were intended for.

Figure 5 shows how the codeword "1001 1101 0110 0101 1001 1010 1101 1000 1111 0011 1100 0101 1010 0010 1101" is added to a log file. The lighter bars are the existing requests from actual clients and the darker bars are the requests added by fingerprinting. This figure has 8 bit flips, one of which is highlighted and shown in more detail in Figure 6. Note that there are 7 bit flips from 0 to 1 and only 1 from 1 to 0. The received codeword in the log file will be: "1001 1111 0110 0001 1001 1010 1111 1101 1111 1011 1100 0101 1110 0010 1111".

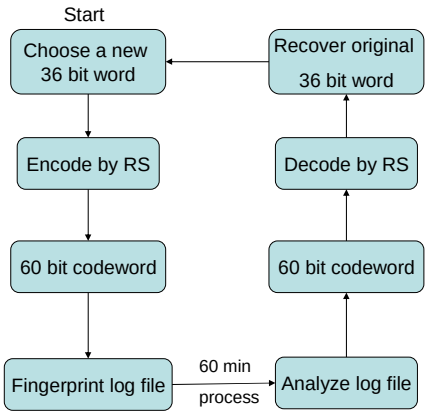

Figure 7: **Our fingerprinting algorithm cycle.**

By applying Reed-Solomon decoding, we then recover the original fingerprint of "`1101 1000 1111 0011 1100 0101 1010 0010 1101`".

5 Discussion and future work

In this paper we have explored the tradeoff in terms of how long it takes to leave a fingerprint in a hidden service log file *vs.* how much traffic must be added per minute. The threat model we assumed was a passive observer that does not suspect this form of fingerprinting but does observe bursts in the log file. For future work, we plan to explore the tradeoffs in a stronger threat model where the hidden web service host suspects that fingerprinting will be employed so that an extra degree of stealth in leaving the fingerprint is required.

In our work under progress we are modelling the probability distribution of the network delays, which is required for a closed-form expression of the uncoded bit error probability. This expression will in turn constitute the basis for predicting the probability of correct detection when superencoding is used. Furthermore, a time domain analysis of the gathered data will provide useful elements for the design of channel coding mechanisms, as they will depend among other factors on the coherence time. For instance, preliminary results show that the autocorrelation of the observed data can be reasonably modelled by an autoregressive process. The fact that the delays corresponding to consecutive requests are strongly correlated suggests that a differential encoding scheme would be beneficial.

A good model for both the distribution and the second order statistics of the request delay is also crucial for an information-theoretic approach to the problem, which will reveal what the fundamental limits of this delay-based communication scheme are. In addition, this approach will give insights for the design of better channel codes.

An underlying assumption of our current fingerprinting technique is that the fingerprinter has a good estimate of the mean of traffic requests per minute for the hidden service. This estimate can be an overestimate, which will cause the fingerprinter to use more redundant bits than necessary for superencoding and take longer than necessary to do the fingerprint. Overestimation also makes the fingerprinting more conspicuous. For future work, we plan to explore methods for estimating the traffic rate of a hidden service indirectly and accurately. For example, infrequent observations of the last access/modification time have been shown to be useful in estimating the access/modification rate [11].

6 Related work

The work most related to ours is efforts to *locate* hidden servers. Overlier and Syverson [18] describe several different methods for locating hidden services in Tor and provide recommendations to prevent them, which were adopted by the Tor project. Murdoch and Danezis [13] consider the problem of traffic analysis,

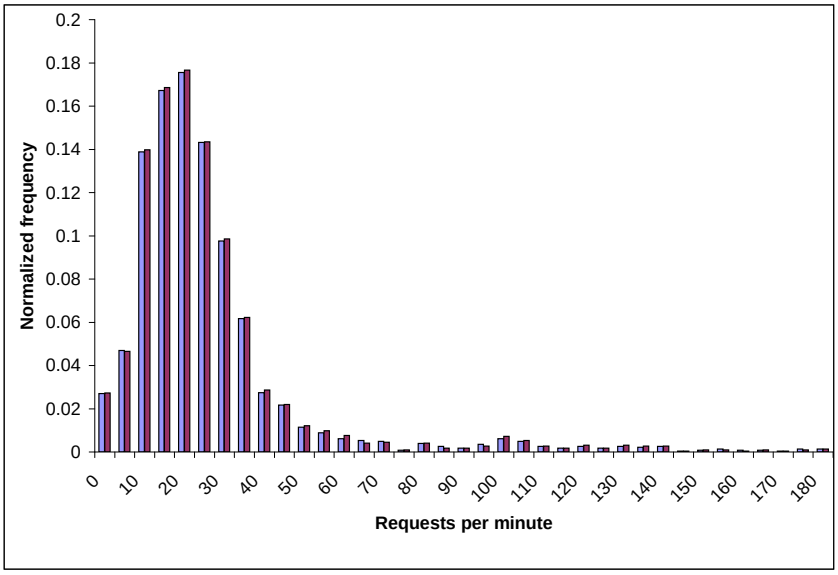

Figure 8: **Histograms with and without fingerprinting.**

and Murdoch [16] considers Internet Exchange-level adversaries. Bauer *et al.* [1] describe attacks on Tor anonymity based on nodes misrepresenting the amount of resources they have. Murdoch [12] demonstrates that it is possible to determine whether a given IP address is hosting a particular hidden service or not based on clock skew. The basic idea is to send some amount of traffic to the hidden service, and query for TCP timestamps from the IP address suspected of hosting the service. As the server becomes busier, it will heat up causing a timing skew in the quartz crystal that maintains the system time, which will be seen in the timestamps. Murdoch also shows some results suggesting that geolocation is possible based on diurnal patterns associated with heat. In contrast to these works, our work assumes that the hidden server has been located and we describe a way to prove that a physical machine (once confiscated) was the one that had been hosting the service.

Our work falls in the general domain of covert timing channels [22, 8, 9, 7, 6]. There has been a considerable amount of work on creating and detecting covert timing channels based on TCP/IP [2, 17, 14]. To the best of our knowledge, ours is the first work to consider timing channels as a method for leaving fingerprints in hidden service log files.

The relationship of our measurement results for Tor delays to the results of Loesing *et al.* [10] was described in Section 3. Other works have considered the timing characteristics of Tor and other anonymity networks in the context of timing attacks on anonymity [3, 4, 20, 15].

7 Conclusion

We demonsrated a technique for leaving timing channel fingerprints in hidden service log files. The technique presented in this paper is robust, even for the random delays introduced by the Tor network and realistic, bursty traffic from existing clients for the web service. We were able to reliably recover a 36-bit fingerprint with a 60-minute fingerprinting process.

References

[1] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. Low-resource routing attacks against tor. In *WPES '07: Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 11–20, New York, NY, USA, 2007. ACM.

[2] S. Cabuk, C. E. Brodley, and C. Shields. Ip covert timing channels: design and detection. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 178–187, New York, NY, USA, 2004. ACM.

[3] G. Danezis. The traffic analysis of continuous-time mixes. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*, volume 3424 of *LNCS*, pages 35–50, May 2004.

[4] C. Diaz, L. Sassaman, and E. Dewitte. Comparison between two practical mix designs. In *Proceedings of ESORICS 2004*, LNCS, France, September 2004.

[5] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router, 2004.

[6] S. Gianvecchio and H. Wang. Detecting covert timing channels: an entropy-based approach. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 307–316, New York, NY, USA, 2007. ACM.

[7] M. H. Kang and I. S. Moskowitz. A pump for rapid, reliable, secure communication. In *CCS '93: Proceedings of the 1st ACM conference on Computer and Communications Security*, pages 119–129, New York, NY, USA, 1993. ACM Press.

[8] B. W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, 1973.

[9] S. B. Lipner. A comment on the confinement problem. In *SOSP '75: Proceedings of the fifth ACM Symposium on Operating Systems Principles*, pages 192–196, New York, NY, USA, 1975. ACM Press.

[10] K. Loesing, W. Sandmann, C. Wilms, and G. Wirtz. Performance measurements and statistics of tor hidden services. In *SAINT '08: Proceedings of the 2008 International Symposium on Applications and the Internet*, pages 1–7, Washington, DC, USA, 2008. IEEE Computer Society.

[11] N. Matloff. Estimation of internet file-access/modification rates from indirect data. *ACM Trans. Model. Comput. Simul.*, 15(3):233–253, 2005.

[12] S. J. Murdoch. Hot or not: revealing hidden services by their clock skew. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 27–36, New York, NY, USA, 2006. ACM.

[13] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of tor. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 183–195, Washington, DC, USA, 2005. IEEE Computer Society.

[14] S. J. Murdoch and S. Lewis. Embedding covert channels into tcp/ip. In *Information Hiding: 7th International Workshop, volume 3727 of LNCS*, pages 247–261. Springer, 2005.

[15] S. J. Murdoch and R. N. M. Watson. Metrics for security and performance in low-latency anonymity networks. In N. Borisov and I. Goldberg, editors, *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)*, pages 115–132, Leuven, Belgium, July 2008. Springer.

[16] S. J. Murdoch and P. Zieliński. Sampled traffic analysis by Internet-exchange-level adversaries. In N. Borosov and P. Golle, editors, *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)*, Ottawa, Canada, June 2007. Springer.

[17] J. G. N, R. Greenstadt, P. Litwack, and R. Tibbetts. Covert messaging through tcp timestamps. In *in Workshop on Privacy Enhancing Technologies*, pages 194–208, 2002.

[18] L. Overlier and P. Syverson. Locating hidden servers. In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy*, pages 100–114, Washington, DC, USA, 2006. IEEE Computer Society.

[19] privoxy. `http://www.privoxy.org/`.

[20] V. Shmatikov and M.-H. Wang. Timing analysis in low-latency mix networks: Attacks and defenses. In *Proceedings of ESORICS 2006*, September 2006.

[21] S. B. Wicker. *Reed-Solomon Codes and Their Applications*. IEEE Press, Piscataway, NJ, USA, 1994.

[22] J. C. Wray. An analysis of covert timing channels. In *IEEE Symposium on Security and Privacy*, pages 2–7, 1991.