Enhancing Censorship Resistance in the Tor Anonymity Network

Philipp Winter

Department of Mathematics and Computer Science
Karlstad University

Abstract

The Tor network was originally designed as low-latency anonymity network. However, over time Tor earned a reputation as also being a useful tool to circumvent Internet censorship—at times, the network counted 30,000 users only from China. Censors reacted by tightening their grip on the national communication infrastructure. In particular, they developed different techniques to prevent people from being able to access the Tor network. This arms race now counts several iterations and no end is in sight.

This thesis contributes to a censorship-resistant Tor network in two ways. First, it analyses how existing censorship systems work. In particular, the Great Firewall of China is probed in order to obtain a detailed understanding of its capabilities as well as unexplored circumvention opportunities. Second, this thesis proposes practical countermeasures to circumvent Internet censorship. It discusses a novel network protocol which is resistant to the Great Firewall's active probing attacks.

Some of the concepts and results of this thesis led to the creation of software prototypes. All the code is available under a free license. By developing and deploying software, we are not just limited to a theoretical understanding of censorship systems. Rather, we can gain valuable practical experience in the rapidly progressing arms race that is Internet censorship.

Keywords: Tor, censorship, circumvention, anonymity, network measurement

Acknowledgements

I want to thank my family and in particular my parents for their continuous support over the years. Also, I would like to thank both my supervisors, Stefan Lindskog and Simone Fischer-Hübner, as well as the entire PriSec research group for fruitful discussions.

Finally, I want to thank Internetfonden whose research grant made this work possible.

Karlstad, Sweden, January 8, 2014 Philipp Winter

Contents

List of Appended Papers

I. Philipp Winter and Stefan Lindskog. How the Great Firewall of China is Blocking Tor. In *USENIX FOCI*, 2012.

II. Philipp Winter. Towards a Censorship Analyser for Tor. In *USENIX FOCI*, 2013.

III. Philipp Winter and Tobias Pulls and Juergen Fuss. ScrambleSuit: A Polymorphic Network Protocol to Circumvent Censorship. In *ACM WPES*, 2013.

Comments on my Participation

Paper I This paper was joint work with Stefan Lindskog. The bulk part of the work was done by me. Stefan and I met regularly to discuss results, experimental design, and future steps. Stefan also reviewed several draft versions of this paper. Helpful feedback was given by the people listed in the acknowledgements.

Paper II I am the sole author of this paper. Helpful feedback was given by the people listed in the acknowledgements.

Paper III This paper was joint work with Tobias Pulls and Jürgen Fuß. Parts of the cryptographic scheme was designed together with Tobias and Jürgen provided helpful feedback on mathematical aspects and earlier versions of this paper. The experimental part done by me.

Selection of Other Publications

- Philipp Winter and Tobias Pulls and Juergen Fuss. ScrambleSuit: A Polymorph Network Protocol to Circumvent Censorship. *Technical Report*, Karlstad University (May 2013).
- Philipp Winter and Jedidiah R. Crandall. The Great Firewall of China: How it Blocks Tor and Why it is Hard to Pinpoint. In *USENIX ;login: (6)*, 2012, pp. 42–50.
- Philipp Winter and Stefan Lindskog. How China Is Blocking Tor. *Technical Report*, Karlstad University (March 2012).

Introductory Summary

1 Introduction

In 2012, Reporters Without Borders published a report which identifies twelve "enemies of the Internet" [1]. These twelve enemies are in fact countries; namely Burma, China, Cuba, Iran, North Korea, Saudi Arabia, Syria, Turkmenistan, Uzbekistan, and Vietnam. What these countries have in common is their tight grip on national communication infrastructure. The report mentions Bahrain's arrest of bloggers, Iran's launch of its "national Internet" and Uzbekistan's Internet monitoring, just to name a few examples. Furthermore, the report identifies 14 countries under surveillance. This list also contains Australia and France which shows that surveillance is also part of the Western world.

The difference between Internet *censorship* and *surveillance* appears significant from a policy point of view. Censorship, which is frequently conducted by repressive regimes and dictatorships, is typically associated with web sites blocks, arrests and harassment of bloggers, and the deletion of regime-critical content. Surveillance, on the other hand, is often seen as a necessary part of democratic countries. It is typically conducted by intelligence agencies with the goal to thwart criminals and terrorists. From a technical point of view, however, Internet surveillance can quickly turn into censorship: Both rely on special equipment which is deployed in national communication infrastructure. Often, the only difference is merely a set of configuration options. Surveillance equipment can be turned into censorship equipment within a matter of minutes. Furthermore, Reporters Without Borders' report identified two countries which made the unfortunate step from a surveillance to a censorship country: Bahrain and Belarus.

Censorship comes in many shapes. It can range from the widespread arrest of political opponents to subtle self-censorship typically done by individuals out of fear. Ultimately, Internet censorship is merely a symptom of severe social and political problems and technology alone is unlikely to solve them. Technology can, however, be a useful tool towards solving these problems. The technological aspect of censorship is the content of this thesis. In particular, this thesis discusses censorship of the *Tor network*.

The Tor anonymity network was designed to thwart many forms of Internet surveillance and censorship [2]. It was originally designed as low-latency anonymity network and as of November 2013, it consists of almost 5,000 volunteer-run Tor relays. In a nutshell, Tor clients first download the *network consensus* from *directory authorities* which is a signed list of all relays which together form the network. After clients obtained the consensus, they can now create *circuits* which are essentially virtual tunnels—consisting of three relays—through the Tor network. An example of a circuit is given in Figure 1.

Tor provides protection against a local adversary such as a user's ISP. A malicious ISP is unable to read a user's transmitted data and cannot tell with whom the user is communicating with. The network further provides anonymity towards communication destinations. A malicious web site should be unable to determine where a Tor user is located on the Inter-

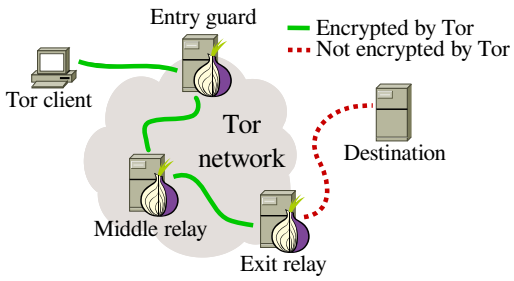


Figure 1: The schematic structure of the Tor network. A client established a three-hop circuit and connects to a destination. All traffic up to the exit relay is encrypted by the Tor protocol. After the exit relay, the user is responsible for encrypted traffic, e.g., by using HTTPS.

net. Given its nature as low-latency anonymity network, Tor cannot protect against a global adversary engaging in traffic analysis [3].

While Tor was originally designed as a pure anonymity tool, an ever increasing set of people began using it to circumvent censorship systems. Censors reacted by blacklisting the small and static set of directory authorities which comprise an ideal choke point for censors. This development led to the creation of *bridges* which are essentially non-public Tor relays. The idea is to give bridges to censored users while trying to keep them secret from censors. In practice, this difficult balancing problem is tackled by making it easy for an individual to obtain a small set of bridges but hard to obtain a large set. Paper I and III discuss censorship-resistance of bridges whereas Paper II proposes a censorship analyser for the Tor network.

This thesis is organised as follows. Section 2 begins by giving an overview of related work. The research questions of this thesis are then asked in Section 3. The research methods which were employed by the appended papers are discussed in Section 4. The contributions are listed in Section 5 and a summary of all appended papers is provided in Section 6. Finally, this thesis is concluded in Section 7.

2 Related Work

It is convenient to divide related work into censorship *analysis* and *circumvention*. A large variety of censorship-resistant schemes were proposed over the past year. We only discuss low-latency circumvention schemes. Several schemes were proposed to disguise network traffic, e.g., as VoIP [4, 5], email [6, 7], or HTTP [8]. Other systems rely on ordinary web users as proxies [9] or can disguise packet payload as dictated by a well-chosen set of regular expressions [10]. While it is not hard to design systems which can evade a censor's filter at one point in time, it is difficult for systems to be a major obstacle to censors. Accordingly, recent research demonstrated that most traffic

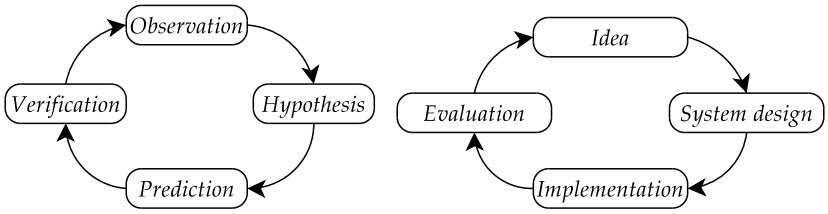
obfuscation systems fail in various ways. For example, one class of circumvention systems *mimics* widespread innocuous protocols such as VoIP. This approach was shown to be problematic as it is very difficult to perfectly mimic a given target protocol [11]. Furthermore, a censor is sometimes able to prevent protocol tunneling by breaking the tunneled protocol while leaving the cover protocol mostly intact [12]. Paper III proposes a censorship-resistant protocol which differs from previous work in its ability to change its "protocol shape". Furthermore, it is optimised for throughput rather than for obfuscation.

Another design category requires cooperating backbone routers [13, 14, 15]. The basic idea is that censored users embed a steganographic tag in their network traffic which signals to cooperating backbone routers that they should hijack the client's TCP stream and reroute it to the actual and hidden destination. The rerouting happens after the packets left the censor's network which should make the design undetectable. Later, it was shown that censors are often able to "route around" these decoy routers as they can control their own routing decisions [16].

Compared to circumvention, the field of censorship analysis has received less attention. This is mostly due to the difficulty of probing censorship systems while not being inside the censoring regime. Some systems such as the Great Firewall of China (GFW) operate symmetrically, i.e., censoring ingress as well as egress traffic. This convenient property was exploited in one of the earliest contributions to censorship analysis [17]. This paper was followed by numerous others which investigated how the GFW conducts DNS poisoning [18, 19], how it is structured [20, 21, 22] and how it can be monitored [23]. Paper I expands our knowledge about the GFW by determining how it blocks the Tor network.

Recently, the research community began to focus on other countries. This can be surprisingly difficult as other country-wide censorship systems do not always operate symmetrically which means that access to machines within the country must be obtained first. Aside from China, Iran was subject to recent research efforts which provided an overview [24, 25], analysed traffic throttling as a means of censorship [26], and discussed Iran's infamous Hidden Internet [27]. Further research investigated Internet censorship in Pakistan [28]. Independent of the censoring country, other work investigates side channels in TCP implementations which can be used to detect intentional drops on the Internet [29].

While most censorship analysis work focuses on the address and transport layer, recent work began investigating network services, e.g., how censors interfere with services such as Weibo [30], or Twitter [31]. Finally, concepts to continuously monitor censorship over time were proposed [32, 33]. In Paper II, we propose a similar concept adapted to the Tor network.



(a) The scientific method. (b) System design experimentation.

Figure 2: A comparison between the scientific method in Figure 2(a) and experimentation in system design in Figure 2(b) (cf. [34]).

3 Research Questions

This thesis provides answers for the following three research questions.

1. *How do real-world censorship systems work?*

   An answer to this question is given in Paper I. It provides an analysis of a nation-scale censorship system, namely the Great Firewall of China. While the GFW is just one system among many, it is commonly regarded as one of the most sophisticated censorship systems and knowledge about one system can frequently be transferred to other systems as well.

2. *How can censorship analysis be "crowdsourced"?*

   Paper II proposes a framework which can assist in analysing censorship systems. It differs from previously proposed systems in that it relies on volunteering users inside the censoring regime. This approach creates new technical challenges which are also addressed in the paper.

3. *How can network protocols be polymorphic and active probing-resistant?*

   A novel censorship-resistant network protocol is proposed in Paper III. It was motivated by the results obtained in Paper I and discusses the design, implementation, and evaluation of a protocol which seeks to disguise Tor traffic from censors.

4 Research Method

Computer science is often divided into the sub-fields *systems* and *theory*. This thesis is part of the systems branch. In particular, the main research methods which were used in this thesis are the *scientific method* and *system design experimentation*.

The scientific method is illustrated in Figure 2(a). It has its origins in the natural sciences and consists of four basic steps even though the details frequently vary. First, *1)* observations about the real world are gathered. Based

on these observations, *2)* a hypothesis is formed which should explain the observed phenomena. In the next step, *3)* the hypothesis is used to predict new observations. Finally, *4)* these predictions are verified. If they turn out to be accurate, step 3 is repeated. If not, step 2 is repeated. The scientific method was used in Paper I as it discusses real-world measurements.

In contrast to the natural sciences, the systems branch of computer science also seeks to propose novel systems rather than just measuring and reasoning about them. A popular research method for that is system design experimentation which is illustrated in Figure 2(b). Conceptually, it is similar to the scientific method. First, *1)* a novel system typically starts with an idea. Based on this idea, *2)* a system is designed. In the next step, *3)* the theoretical design leads to a concrete implementation. Finally, *4)* the implementation is evaluated. System design experimentation was used in Paper II and III as these papers propose novel systems.

In the scientific method, an important criteria of the hypothesis is *falsifiability*. It must be possible for a hypothesis to be proven wrong, e.g., by observing real-world examples which are in contrast to the hypothesis' predictions. Accordingly, the experimentation method's system design must be falsifiable as well. Often, a system design aims to supersede a previously proposed system, e.g., by performing better or by providing stronger security properties. As a result, the evaluation has to determine whether the new system design can really provide what it claims. Consequently, falsifiability in the experimentation method means that it must be possible to design experiments to reject a system design.

5 Contributions

This thesis provides the following five contributions.

1. *An understanding of how the Great Firewall of China is blocking the Tor network.*

   Paper I discusses a variety of networking experiments which were designed to shed light on how the Great Firewall of China operates. The experiments made heavy use of decoy Tor connections which originated from a machine in China which was under our control. By doing so, we could attract the GFW's scanners and gather data which was then analysed in detail.

2. *A lightweight circumvention tool to evade the Great Firewall of China.*

   Paper I proposes a lightweight tool which enables server-side evasion of the GFW's fingerprinting. The tool rewrites a server's TCP window size and can be run by bridge operators to prevent active probing attacks.

3. *A design for a lightweight censorship analyser for Tor.*

   Paper II proposes a design for a lightweight censorship analyser for the Tor network. The analyser is meant to assist the Tor developers in debugging censorship incidents. The main contribution is that the analyser is "crowdsourced", meaning that it is supposed to be run by ordinary computer users.

4. *The design and implementation of a blocking-resistant transport protocol.*

   Paper III discusses the design and implementation of a blocking-resistant transport protocol. The protocol proposes two active probing-resistant authentication mechanisms and techniques to change the transported protocol's flow signature. Finally, the paper contains an initial evaluation of the protocol.

5. *Software prototypes.*

   Paper I and III discuss software prototypes for server-side circumvention as well as a prototype of ScrambleSuit, the blocking-resistant transport protocol. All the code is available under a free license at: http://www.cs.kau.se/philwint/.

6 Summary of Appended Papers

This section summarises the three papers which are appended to this thesis.

Paper I – How the Great Firewall of China is Blocking Tor

This paper investigates how the Great Firewall of China is blocking the Tor anonymity network. In particular, experimental data of Chinese network scanners were gathered over a period of several weeks. The data was analysed and additional network experiments were designed and conducted to hypothesise how the block functions.

Paper II – Towards a Censorship Analyser for Tor

This paper discusses the design of a lightweight censorship analyser for the Tor anonymity network. The paper considers both, usability as well as technical requirements as the analyser is meant to be run by non-technical users.

Paper III – ScrambleSuit: A Polymorphic Network Protocol to Circumvent Censorship

This paper introduces a network protocol—ScrambleSuit—which should provide censorship resistance for protocols such as Tor. In particular, ScrambleSuit should provide protection against active probing attacks as well as simple attempts of traffic analysis. Finally, the proposed protocol is evaluated using a prototype.

7 Conclusions and Future Work

There is a strong need for technology which enables the free retrieval of information and the Tor network is one of these tools. This thesis discussed two aspects of Tor's resistance to censorship. First of all, it investigated and proposed censorship *analysis* techniques. Sound analysis techniques are crucial since circumvention technology relies on it. Secondly, this thesis investigated censorship *circumvention* technology which was motivated by our own censorship analysis findings. This thesis proposes several theoretical techniques as well as practical tools that give a better understanding of how real censors block Tor and how these very blocks can be circumvented.

There is lots of space for future work. Several censorship-resistant protocols were proposed in recent years [35]. These protocols are composed of components which are quite useful for other protocols as well. While the Tor

project's pluggable transport system clearly decouples anonymity from censorship resistance, there is no decoupling *within* censorship resistance components. As a result, future work could further modularise these protocols which should enable faster prototyping and composition. Furthermore, there is no structured approach to evaluating the *quality* of censorship-resistant protocols. To date, evaluations are based on vague assumptions about censor's capabilities. Future work should investigate frameworks to evaluate and quantify the censorship-resistance of protocols.

References

[1] Reporters Without Borders. *Internet Enemies*. 2012. URL: http://march12.rsf.org/en/.

[2] Roger Dingledine, Nick Mathewson, and Paul Syverson. "Tor: The Second-Generation Onion Router". In: *USENIX Security*. USENIX Association, 2004. URL: http://static.usenix.org/event/sec04/tech/full_papers/dingledine/dingledine.pdf.

[3] Steven J. Murdoch and George Danezis. "Low-Cost Traffic Analysis of Tor". In: *Security & Privacy*. IEEE, 2005. URL: http://www.cl.cam.ac.uk/users/sjm217/papers/oakland05torta.pdf.

[4] Amir Houmansadr et al. "I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention". In: *NDSS*. The Internet Society, 2013. URL: http://www.cs.utexas.edu/~amir/papers/FreeWave.pdf.

[5] Hooman Mohajeri Moghaddam et al. "SkypeMorph: Protocol Obfuscation for Tor Bridges". In: *CCS*. ACM, 2012. URL: http://www.cypherpunks.ca/~iang/pubs/skypemorph-ccs.pdf.

[6] Wenxuan Zhou et al. "SWEET: Serving the Web by Exploiting Email Tunnels". In: *HotPETS*. Springer, 2013. URL: http://petsymposium.org/2013/papers/zhou-censorship.pdf.

[7] Qiyan Wang et al. "CensorSpoofer: Asymmetric Communication using IP Spoofing for Censorship-Resistant Web Browsing". In: *CCS*. ACM, 2012. URL: http://hatswitch.org/~nikita/papers/censorspoofer.pdf.

[8] Zachary Weinberg et al. "StegoTorus: A Camouflage Proxy for the Tor Anonymity System". In: *CCS*. ACM, 2012. URL: http://www.owlfolio.org/media/2010/05/stegotorus.pdf.

[9] David Fifield et al. "Evading Censorship with Browser-Based Proxies". In: *PETS*. Springer, 2012, pp. 239–258. URL: http://crypto.stanford.edu/flashproxy/flashproxy.pdf.

[10] Kevin P. Dyer et al. "Protocol Misidentification Made Easy with Format-Transforming Encryption". In: *CCS*. ACM, 2013. URL: http://eprint.iacr.org/2012/494.pdf.

[11] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. "The Parrot is Dead: Observing Unobservable Network Communications". In: *Security & Privacy*. IEEE, 2013. URL: http://www.cs.utexas.edu/~amir/papers/parrot.pdf.

[12] John Geddes, Max Schuchard, and Nicholas Hopper. "Cover Your ACKs: Pitfalls of Covert Channel Censorship Circumvention". In: *CCS*. ACM, 2013. URL: http://www-users.cs.umn.edu/~hopper/ccs13-cya.pdf.

[13] Amir Houmansadr et al. "Cirripede: Circumvention Infrastructure using Router Redirection with Plausible Deniability". In: *CCS*. ACM, 2011, pp. 187–200. URL: http://hatswitch.org/~nikita/papers/cirripede-ccs11.pdf.

[14] Eric Wustrow et al. "Telex: Anticensorship in the Network Infrastructure". In: *USENIX Security*. USENIX Association, 2011. URL: http://www.usenix.org/event/sec11/tech/full_papers/Wustrow.pdf.

[15] Josh Karlin et al. "Decoy Routing: Toward Unblockable Internet Communication". In: *FOCI*. USENIX Association, 2011. URL: http://static.usenix.org/event/foci11/tech/final_files/Karlin.pdf.

[16] Christopher Thompson Max Schuchard John Geddes and Nicholas Hopper. "Routing Around Decoys". In: *CCS*. ACM, 2012. URL: http://www-users.cs.umn.edu/~hopper/decoy-ccs12.pdf.

[17] Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson. "Ignoring the Great Firewall of China". In: *PETS*. Springer, 2006. URL: http://www.cl.cam.ac.uk/~rnc1/ignoring.pdf.

[18] Sparks et al. "The Collateral Damage of Internet Censorship by DNS Injection". In: *SIGCOMM Computer Communication Review* 42.3 (), pp. 21–27. URL: http://conferences.sigcomm.org/sigcomm/2012/paper/ccr-paper266.pdf.

[19] Graham Lowe, Patrick Winters, and Michael L. Marcus. *The Great DNS Wall of China*. Tech. rep. New York University, 2007. URL: http://cs.nyu.edu/~pcw216/work/nds/final.pdf.

[20] Xueyang Xu, Z. Morley Mao, and J. Alex Halderman. "Internet Censorship in China: Where Does the Filtering Occur?" In: *PAM*. Springer, 2011, pp. 133–142. URL: http://www.eecs.umich.edu/~zmao/Papers/china-censorship-pam11.pdf.

[21] Jong Chun Park and Jedidiah R. Crandall. "Empirical Study of a National-Scale Distributed Intrusion Detection System: Backbone-Level Filtering of HTML Responses in China". In: *Distributed Computing Systems*. IEEE, 2010, pp. 315–326. URL: http://www.cs.unm.edu/~crandall/icdcs2010.pdf.

[22] Sheharbano Khattak et al. “Towards Illuminating a Censorship Monitor’s Model to Facilitate Evasion”. In: *FOCI*. USENIX Association, 2013. URL: https://www.usenix.org/system/files/tech-schedule/foci13-papers-archive.zip.

[23] Jedidiah R. Crandall et al. “ConceptDoppler: A Weather Tracker for Internet Censorship”. In: *CCS*. ACM, 2007, pp. 352–365. URL: http://www.csd.uoc.gr/~hy558/papers/conceptdoppler.pdf.

[24] Simurgh Aryan, Homa Aryan, and J. Alex Halderman. “Internet Censorship in Iran: A First Look”. In: *FOCI*. USENIX Association, 2013. URL: https://www.usenix.org/system/files/tech-schedule/foci13-papers-archive.zip.

[25] John-Paul Verkamp and Minaxi Gupta. “Inferring Mechanics of Web Censorship Around the World”. In: *FOCI*. USENIX Association, 2012. URL: https://www.usenix.org/system/files/conference/foci12/foci12-final1.pdf.

[26] Collin Anderson. *Dimming the Internet: Detecting Throttling as a Mechanism of Censorship in Iran*. Tech. rep. 2013. URL: http://arxiv.org/abs/1306.4361.

[27] Collin Anderson. *The Hidden Internet of Iran: Private Address Allocations on a National Network*. Tech. rep. 2012. URL: http://arxiv.org/pdf/1209.6398v1.

[28] Zubair Nabi. “The Anatomy of Web Censorship in Pakistan”. In: *FOCI*. USENIX Association, 2013. URL: https://www.usenix.org/system/files/tech-schedule/foci13-papers-archive.zip.

[29] Roya Ensafi et al. “Detecting Intentional Packet Drops on the Internet via TCP/IP Side Channels”. In: *PAM*. Springer, 2014.

[30] Tao Zhu et al. “The Velocity of Censorship: High-Fidelity Detection of Microblog Post Deletions”. In: *USENIX Security*. USENIX Association, 2013. URL: https://www.usenix.org/system/files/conference/usenixsecurity13/sec13-paper_zhu.pdf.

[31] John-Paul Verkamp and Minaxi Gupta. “Five Incidents, One Theme: Twitter Spam as a Weapon to Drown Voices of Protest”. In: *FOCI*. USENIX Association, 2013. URL: https://www.usenix.org/system/files/tech-schedule/foci13-papers-archive.zip.

[32] Arturo Filastò and Jacob Appelbaum. “OONI: Open Observatory of Network Interference”. In: *FOCI*. USENIX Association, 2012. URL: https://www.usenix.org/system/files/conference/foci12/foci12-final12.pdf.

[33] Andreas Sfakianakis, Elias Athanasopoulos, and Sotiris Ioannidis. "CensMon: A Web Censorship Monitor". In: *FOCI*. USENIX Association, 2011. URL: http://static.usenix.org/event/foci11/tech/final_files/Sfakianakis.pdf.

[34] Dror G. Feitelson. *Experimental Computer Science: The Need for a Cultural Change*. Tech. rep. The Hebrew University of Jerusalem, 2006. URL: http://www.cs.huji.ac.il/~feit/papers/exp05.pdf.

[35] Philipp Winter. *Selected Papers in Censorship*. URL: http://veri.nymity.ch/censorbib/.

Paper I

How the Great Firewall of China is Blocking Tor

Reprinted from

USENIX FOCI, 2012.

How the Great Firewall of China is Blocking Tor

Philipp Winter and Stefan Lindskog
philwint@kau.se

Abstract

Internet censorship in China is not just limited to the web: the Great Firewall of China prevents thousands of potential Tor users from accessing the network. In this paper, we investigate how the blocking mechanism is implemented, we conjecture how China's Tor blocking infrastructure is designed and we propose circumvention techniques. Our work bolsters the understanding of China's censorship capabilities and thus paves the way towards more effective circumvention techniques.

1 Introduction

On October 4, 2011 a user reported to the Tor bug tracker that unpublished bridges stop working after only a few minutes when used from within China [1]. Bridges are unpublished Tor relays and their very purpose is to help censored users to access the Tor network if the "main entrance" is blocked [2]. The bug report indicated that the Great Firewall of China (GFC) has been enhanced with the potentiality of dynamically blocking Tor.

This censorship attempt is by no means China's first attempt to block Tor. In the past, there have been efforts to block the website [3], the public Tor network [4, 5] and parts of the bridges [6]. According to a report [3], these blocks were realised by simple IP blacklisting and HTTP header filtering. All these blocking attempts had in common that they were straightforward and inflexible.

In contrast to the above mentioned censorship attempts, the currently observable block appears to be much more flexible and sophisticated. The GFC blocks bridges dynamically without simply enumerating their IP addresses and blacklisting them (cf. [7]).

In this paper, we try to deepen the understanding of the infrastructure used by the GFC to block the Tor anonymity network. Our contributions are threefold: (1) we reveal how users within China are hindered from accessing the Tor network, (2) we conjecture how China's Tor blocking infrastructure is designed and (3) we discuss and propose circumvention techniques.

We also point out that censorship is a fast moving target. Our results are only valid at the time of writing[1] and might—and probably will—be subject to change. Nevertheless, we believe that a detailed understanding of the GFC's current capabilities, a "censorship snapshot", is important for future circumvention work.

2 Related Work

In [8], Wilde revealed first crucial insights about the block of Tor traffic. Over a period of one week in December 2011, Wilde analysed how *unpublished* Tor bridges are getting scanned and, as a result, blocked by the GFC.

Wilde's results showed that when a Tor user in China establishes a connection to a bridge or relay, deep packet inspection (DPI) boxes *identify* the Tor protocol. Shortly after a Tor connection is detected, *active scanning* is initiated. The scanning is done by seemingly random Chinese IP addresses. The scanners connect to the respective bridge and try to *establish a Tor connection*. If it succeeds, the bridge is *blocked*.

Wilde was able to narrow down the suspected cause for active scanning to the cipher list sent by the Tor client inside the TLS client hello[2]. This cipher

[1] The data was mostly gathered in March 2012 and the paper written in April 2012.

[2] The TLS client hello is sent by the client after a TCP connection has been established. Details can be found in the Tor design paper [9].

list appears to be *unique and only used by Tor* although for a long time it was identical to the cipher list advertised by Firefox 3. That gives the GFC the opportunity to easily identify Tor connections. Furthermore, Wilde noticed that active scanning is started at the beginning of multiples of 15 minutes. An analysis of the Tor debug logs yielded that Chinese scanners initiate a TLS connection, conduct a renegotiation and start building a Tor circuit, once the TLS connection was set up. After the scan succeeded, the IP address together with the associated port (we hereafter refer to this as "IP:port tuple") of the freshly scanned bridge is blocked resulting in users in China not being able to use the bridge anymore.

With respect to Wilde's contribution, we (1) revisit certain experiments in greater detail and with significantly more data, we (2) rectify observations which changed since Wilde's analysis, and we (3) answer yet open questions.

3 Experimental Setup

During the process of preparing and running our experiments we took special care to not violate any ethical standards and laws. In addition, all our experiments were in accordance with the terms of service of our service providers. In order to ensure reproducibility and encourage further research, we publish our gathered data and developed code[3]. The data includes Chinese IP addresses which were found to conduct active scanning of our bridge. We carefully configured our Tor bridges to remain unpublished and we always picked randomly chosen high ports to listen to so that we can be sure that the data is free from legitimate Tor users.

3.1 Vantage Points

In order to ensure a high degree of confidence in our results, we used different vantage points. We had a relay in Russia, bridges in Singapore and Sweden and clients in China. There is, however, no technical reason why we chose Russia, Singapore and Sweden.

Bridge in Singapore: A large part of our experiments was conducted with our Tor bridge located in Singapore. The bridge was running inside the Amazon EC2 cloud [10, 11]. The OpenNet Initiative reports Singapore as a country conducting minimal Internet filtering involving only pornography [12]. Hence, we assume that our experimental results were not interfered with by Internet filtering in Singapore.

Bridge in Sweden: To reproduce certain experiments, we set up Tor bridges located at our institution in Sweden. Internet filtering for these bridges was limited to well-known malware ports, so we can rule out filtering mechanisms interfering with our results.

Relay in Russia: A public Tor relay located in a Russian data center was used to investigate the type of block, public Tor relays are undergoing. The

[3] URL: http://veri.nymity.ch/gfw

relay served as a middle relay, meaning that it is not picked as the first hop in a Tor circuit and it does not see the exit traffic of users.

Clients in China: To avoid biased results, we used two types of vantage points inside China: open SOCKS proxies and a VPS. We compiled a list of public Chinese SOCKS proxies by searching Google. We were able to find a total of 32 SOCKS proxies which were distributed amongst 12 distinct autonomous systems. We connected to these SOCKS proxies from computers outside of China and used them to rerun certain experiments on a smaller scale to rule out phenomena limited to our VPS.

Our second vantage point and primary experimental machine is a VPS we rented. The VPS ran Linux and resided in the autonomous system number (ASN) 4808. We had full root access to the VPS which made it possible for us to sniff traffic and conduct experiments below the application layer. Most of our experiments were conducted from our VPS, whereas the SOCKS proxies' primary use was to verify the results.

3.2 Shortcomings

Active analysis of a censorship system can easily attract the censor's attention if no special care is taken to "stay under the radar". Due to the fact that China is a sophisticated censor with the potential power to actively falsify measurement results, we have to point out potential shortcomings in our experimental setup.

We have no reliable information about the owners of our public SOCKS proxies. Whois lookups did not yield anything suspicious but the information in the records can be spoofed. It might even be possible that the SOCKS proxies are operated by Chinese authorities. Second, our VPS was located in a data center where Tor connections typically do not originate. We also had no information about whether our service provider conducts Internet filtering and the type or extent thereof.

4 Analysis

4.1 How Are Bridges and Relays Blocked?

The first step in bootstrapping a Tor connection requires connecting to the directory authorities to download the consensus which contains all public relays. We noticed that seven of all eight directory authorities are blocked *on the IP layer*. These machines responded neither to TCP, nor to ICMP packets. One authority turned out to be reachable and it was possible for us to download the consensus. We have no explanation why this particular machine was unblocked.

After the consensus has been downloaded, clients can start creating a circuit. Using our Russian relay, we found out that when a client in China connects to a relay, the GFC lets the TCP SYN pass through but drops the SYN/ACK sent by the bridge to the client. The same happens when a client

tries to connect to a blocked bridge. However, clients are still able to connect to different TCP ports as well as ping the bridge. We believe that the reason for the GFC blocking relays and bridges by IP:port tuples rather than by IPs is to minimise collateral damage.

4.2 How Long Do Bridges Remain Blocked?

To answer this question, we started two Tor bridges on our machine in Singapore. Both Tor processes were private bridges and listening on TCP port 27418 and 23941, respectively. Both ports were chosen randomly.

In the next step, we made the GFC block both IP:port tuples by initiating Tor connections to them from our VPS in China. After both tuples were blocked, we set up iptables [13] rules on our machine in Singapore to whitelist our VPS in China to port 23941 and drop all other connections to the same port. That way, the tuple appeared unreachable to the GFC but not to our Chinese VPS. Port 27418 remained unchanged and hence reachable to the GFC. We then started monitoring the reachability of both Tor processes by continuously trying to connect to them using telnet from our VPS.

After approximately 12 hours, the Tor process behind port 23941 (which appeared to be unreachable to the GFC) became reachable again whereas connections to port 27418 still timed out and continued to do so. In our iptables logs we could find numerous connection attempts originating from Chinese scanners. This observation shows that once a Tor bridge has been blocked, it only remains blocked if Chinese scanners are able to *continuously connect* to the bridge. If they cannot, the block is *removed*.

4.3 Is the Public Network Reachable?

To verify how many public relays are reachable from within China, we downloaded the consensus published at February 23 at 08:00 (UTC). At the time, the consensus contained descriptors for a total of 2819 relays. Then, from our Chinese VPS we tried to establish a TCP connection to the Tor port of every single relay. If we were able to successfully establish a TCP connection we classified the relay as reachable, otherwise unreachable.

We found that our VPS could successfully establish TCP connections to 47 out of all 2819 (1.6%) public relays. We manually inspected the descriptors of the 47 relays, but could not find any common property which could have been responsible for the relays being unblocked. We checked the availability of the reachable relays again after a period of three days. Only one out of the original 47 unblocked relays was still reachable.

4.4 Where Does the Fingerprinting Happen?

We want to gain a better understanding of where the Chinese DPI boxes are looking for the Tor fingerprint. In detail, we tried to investigate whether the DPI boxes also analyse domestic and ingress traffic.

We used six open Chinese SOCKS proxies (in ASN 4134, 4837, 9808 and 17968) as well as six PlanetLab nodes (in ASN 4538 and 23910) to investigate domestic fingerprinting. We simulated the initiation of a Tor connection multiple times to randomly chosen TCP ports on our VPS, but could not attract any active scans.

Previous research confirmed that HTTP keyword filtering done by the GFC is bidirectional [14], i.e., keywords are scanned in ingress as well as in egress traffic. We wanted to find out whether this holds true for the Tor DPI infrastructure too. To verify that, we tried to initiate Tor connections to our Chinese VPS from our vantage points in Sweden, Russia and Singapore. Despite multiple attempts we were not able to attract a single scan.

The above mentioned results indicate that Tor fingerprinting is probably *not done in domestic traffic* and only with traffic going from *inside China to the outside world*. We believe that there are two reasons for that. First, fingerprinting domestic traffic in addition to international traffic would dramatically increase the amount of data to analyse since domestic traffic is believed to be the largest fraction of Chinese traffic [15]. Second, at the time of this writing there are no relays in China so there is no need to fingerprint domestic or ingress traffic. Tor usage in China means being able to connect to the outside world.

4.5 Where Are the Scanners Coming From?

To get extensive data for answering this question, we continuously attracted scanners over a period of 17 days ranging from March 6 to March 23. We attracted scanners by simulating Tor connections from within China to our bridge in Singapore[4]. To simulate a Tor connection, we developed a small tool whose sole purpose was to send the Tor TLS client hello to the bridge and terminate after receiving the response. We could also have used the original Tor client to do so, but our tool was much more lightweight which was helpful, given our resource-constrained VPS. After every Tor connection simulation, our program remained inactive for a randomly chosen value between 9 and 14 minutes. The experiment yielded *3295 scans of our bridge*. Our findings described below are based on this data.

4.5.1 Scanner IP Address Distribution

We are interested in the scanner's IP address distribution, i.e., how often can we find a particular IP address in our logs? Our data exhibits two surprising characteristics:

1. More than half of all connections—1680 of 3295 (51%)—were initiated by *a single* IP address: 202.108.181.70.

[4]We reproduced this experiment with a bridge in Sweden and with open Chinese SOCKS proxies. Our findings were the same.

2. The second half of all addresses is almost *uniformly distributed*. Among all 1615 remaining addresses, 1584 (98%) were unique.

The IP address 202.108.181.70 clearly stands out from all observed scanners. Aside from its heavy activity we could not observe any other peculiarities in its scanning behaviour. The whois record of the address states a company named "Beijing Guanda Technology Co.Ltd" as owner. We could only find a company named "Guanda Technology Amusement Equipment Co., Ltd" on the Internet. It is not clear whether this is the same company. However, as explained below, we have reason to believe that the scanner's IP addresses are spoofed by the GFC so the owner of the IP address, assuming that it even exists, might not be aware of the scanning activity.

Whois and reverse DNS lookups of all the seemingly random IP addresses suggested that the IP addresses were coming from ISP pools. For example, all valid reverse DNS lookups contained either the strings *adsl* or *dynamic*.

4.5.2 Autonomous System Origin

We used the IP to ASN mapping service provided by Team Cymru [16] to get the autonomous system number for every observed scanner. The result reveals that all scanners come from *one of three* ASes[5] with the respective percentage in parantheses:

- *AS4837*: CHINA169-BACKBONE CNCGROUP China169 Backbone (65.7%)
- *AS4134*: CHINANET-BACKBONE No.31,Jin-rong Street (30.5%)
- *AS17622*: CNCGROUP-GZ China Unicom Guangzhou network (3.8%)

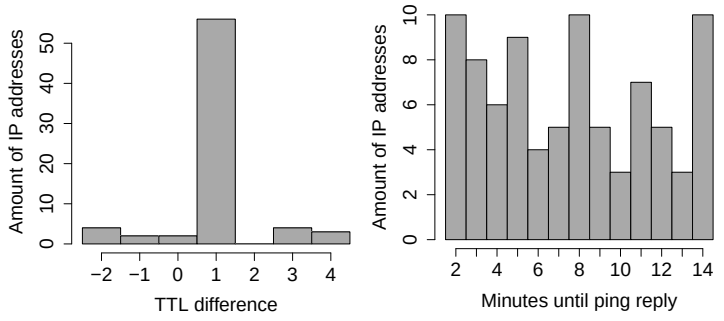
AS4134 is owned by China Telecom while AS4837 and AS17622 is owned by China Unicom. AS4134 and AS4837 are the two largest ASes in China [17] and play a crucial role in the country-wide censorship as pointed out by Xu, Mao and Halderman [18]. Furthermore, Roberts et al. [17] showed that China's AS level structure is *far from uniform* with a significant fraction of the countries traffic being routed through AS4134 or AS4837.

4.5.3 IP Address Spoofing

During manual tests we noticed that sometimes shortly after a scan, the respective IP address starts replying to pings[6], but with a *different IP Time-to-Live* (TTL) than during the scan. In order to have more data for our analysis, we wrote a script to automatically collect additional data as soon as a scanner connects and again some minutes afterwards. In particular, the script (1) runs

[5] Recent research efforts by Roberts et al. showed that China operates 177 autonomous systems [17].

[6] Note that this is never the case during or immediately after scans. All ICMP packets are being dropped.



(a) The IP TTL difference between after and during the scan.

(b) The amount of minutes until the hosts started replying to pings.

Figure 1: IP TTL difference (a) and duration until ping replies (b).

TCP, UDP and ICMP traceroutes immediately after a scan and again 15 minutes later, (2) continuously pings the scanning IP address for 15 minutes and (3) captures all network traffic during these 15 minutes using tcpdump.

Between March 21 and 26 we started an independent experiment to attract scanners and let our script gather the above mentioned data. We caused a total of 429 scans coming from 427 unique IP addresses. From all 429 scans we then extracted all connections where the continuous 15 minutes ping resulted in at least one ping reply. This process yielded a subset of 85 connections which corresponds to approximately 20% of all observed connections. We analysed the 85 connections by computing the amount of minutes until the respective IP address started replying to our ping requests and the IP TTL difference (new TTL – old TTL) between packets during the scan and ping replies.

The results are shown in Figure 1(a) and 1(b). Figure 1(b) illustrates how long it took for the hosts to start replying to the ping requests. No clear pattern is visible. Figure 1(a) depicts all IP TTL differences after the scan (when the host starts replying to ping packets) and during the scan. We had 14 outliers with a TTL difference of 65 and 192 but did not list them in the histogram. It is clearly noticeable that the difference was mostly one, meaning that after the scan, the TTL was *by one more than during the scan*.

One explanation for the changing TTL, but definitely not the only one, is that the GFC could be *spoofing IP addresses*. The firewall could be abusing several IP address pools intended for Internet users to allocate short-lived IP addresses for the purpose of scanning.

4.6 When Do the Scanners Connect?

Figure 2 visualises when the scanners in our data set connected. The y-axis depicts the minutes of the respective hour. Contrary to December 2011, when Wilde ran his experiments, the scanners now seem to use a broader time interval to launch the scans. In addition, the data contains two time intervals

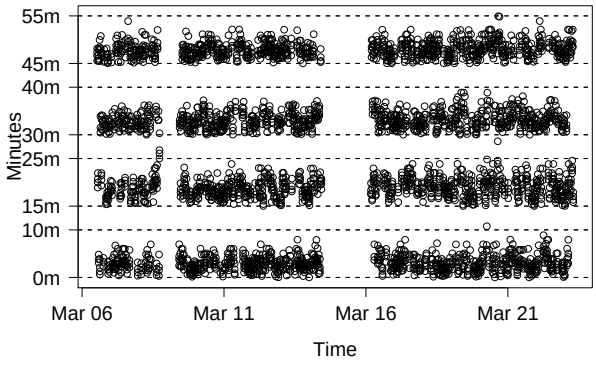


Figure 2: Time points when scanners were found connecting to our bridge.

which are free from scanning. These intervals lasted from March 8 at around 16:30 to March 9 at 09:00 and from March 14 at 09:30 to March 16 at 3:30 (UTC). We have no explanation why the GFC did not conduct scanning during that time.

Closer manual analysis yielded that the data exhibits a *diurnal pattern*. In order to make the pattern visible, we processed the data as four distinct time series with every 15 minutes interval forming one time series, respectively. We smoothed the time series' data points using simple exponential smoothing with a smoothing factor $\alpha = 0.05$. The result—a subset of the data ranging from March 16 to March 23—is shown in Figure 3. Each of the four diagrams represents one of the 15 minutes intervals. The diagrams show that depending on the time of the day, on average, scanners connect either close to the respective 15 minutes multiple or a little bit later.

We conjecture that the GFC maintains *scanning queues*. When the DPI boxes discover a potential Tor connection, the IP:port tuple of the suspected bridge is added to a queue. Every 15 minutes, these queues are processed and all IP:port tuples in the queue are being scanned. We believe that during the day, the GFC needs more time to process the queues since there are probably more Chinese users trying to access the Tor network which leads to more scans.

4.7 Blocking Malfunction

During our experiments we noticed several sudden disappearences of active scanning. This lack of scanning made it possible for us to successfully initiate Tor connections without causing bridges to get scanned and blocked. The Tor bridge usage statistics between January and June 2012 contain several usage spikes which confirm outages in the blocking infrastructure [19]. Another downtime was observed by Wilde [20].

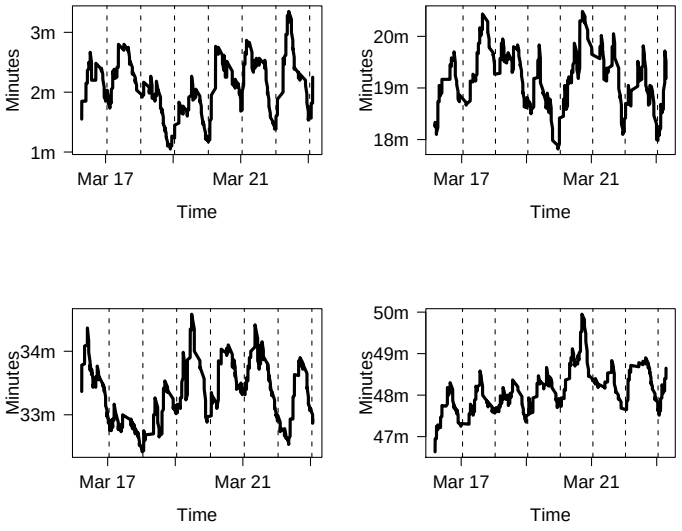


Figure 3: Diurnal scanning connection pattern.

5 Circumvention

While there is a large body of work dedicated to anti-censorship, we limit this section to the discussion of the recently developed *obfsproxy* [21] and propose a novel way to evade the GFC's DPI boxes.

5.1 Obfsproxy

The Tor Project is developing a tool called obfsproxy. The tool runs independently of Tor and is obfuscating the network traffic it receives from the Tor process. As long as both, the bridge and the client, are running the tool, the Tor traffic transmitted between them can be obfuscated so that the Chinese DPI boxes are not able to identify the TLS cipher list anymore.

Obfsproxy implements a *pluggable transport layer* which means that modules can be written that support different types of obfuscation[7]. At the time of this writing, obfsproxy contains an obfuscation module called *obfs2* [23] which is based on Leidl's obfuscated ssh [24]. Obfs2 relies on a key establishment phase which is followed by the two involved parties exchanging superenciphered messages. A passive woman-in-the-middle is able to decrypt obfs2's obfuscation layer and reveal the encapsulated data[8] but this is more complex than conducting simple pattern matching as it is frequently done by DPI boxes.

As of March 24, the official obfsproxy bundle [21] contained a list of 13 hard-coded obfsproxy bridges in its configuration file. From our VPS we

[7] An overview of currently developed modules can be found at [22].

[8] We note that this does not affect the security of the TLS connection used by Tor.

tested the reachability of all of these bridges by trying to connect to them via telnet. We found that not a single connection succeeded. One bridge seemed to be offline and the connection to the remaining 12 bridges was aborted by spoofed RST segments.

The above result raises the question whether the GFC is able to block all obfsproxy connections or just the 13 hard-coded bridges. To answer this question, we set up a private obfsproxy bridge in Sweden and connected to it from within China. We initiated several connections to it over several hours and we could always successfully establish a Tor circuit. We conclude that the IP:port tuples of the 13 hard-coded obfsproxy bridges were added to a blacklist to prevent widespread use of the official obfsproxy bundle. However, private obfsproxy bridges remain undetected by the GFC.

Similar to obfs2, Moghaddam et al. proposed a pluggable transport for Tor to mimic Skype video traffic [25]. This concept will make it significantly harder to block Tor by fingerprinting because DPI boxes would have to distinguish between legitimate and camouflaged Skype traffic.

5.2 Packet Fragmentation

One circumvention technique described by Ptacek and Newsham [26] is *packet fragmentation* which exploits the fact that some network intrusion detection systems do not conduct packet reassembly. Crandall et al. also considered fragmentation to evade the GFC's keyword-based detection [27].

We used the tool *fragroute* [28] to enforce packet fragmentation on our VPS in China. We configured fragroute to split the TCP stream to segments of 16 bytes each. In our test it took 5 TCP segments to transmit the fragmented cipher list to our bridge. Despite initiating several fragmented Tor connections, we never observed any active scanning and could use Tor without interference. This experiment indicates that the GFC does not conduct packet reassembly. A similar observation was made by Park and Crandall [29].

However, client side fragmentation is an unpractical solution given that this method must be supported by *all* connecting Chinese users. A single user who does not use fragmentation, triggers active scanning which then leads to the block of the respective bridge. Another disadvantage is the significant protocol overhead due to the shortened TCP segments which leads to a decrease in throughput.

Due to these shortcomings, we propose a way to realise *server side fragmentation*. We developed a tool[9] which transparently *rewrites the TCP window size* announced by the bridge to the client inside the SYN/ACK segment. The diminished window size makes the client split its TLS cipher list across two TCP segment. That way, the DPI boxes are not able to identify the Tor connection. At the time of this writing, the tool is already deployed to several bridges. So far, these bridges were not scanned and keep getting connections from legitimate users in China.

[9] The tool is available on our project website: http://veri.nymity.ch/gfw.

Our server side fragmentation tool has the advantage that it is easy to deploy and it only interferes with Tor connections by rewriting the announced TCP window during the TCP handshake. Hence, there are virtually no performance implications.

6 Conclusions

We showed how access to Tor is being denied in China and we conjectured how the blocking infrastructure is designed. In addition, we discussed countermeasures intended to "unblock" the Tor network. Our findings include that the Great Firewall of China might spoof IP addresses for the purpose of scanning Tor bridges and that domestic as well as ingress traffic does not seem to be subject to Tor fingerprinting. We also showed that the firewall is easily circumvented by fragmented packets. Tor traffic is currently distinguishable from what is regarded as harmless traffic in China. Since Tor is being used more and more as censorship circumvention tool, it is crucial that this distinguishability is minimised.

Acknowledgments

We thank the Tor developers for their helpful feedback and support, the anonymous reviewers for their valuable suggestions, Harald Lampesberger, Simone Fischer-Hübner and Rose-Mharie Åhlfeldt for feedback and Fabio Pietrosanti for helping with the experiments.

The work conducted by the second author has been supported by the Compare Business Innovative Centre phase 3 (C-BIC 3) project, funded partly by the European Regional Development Fund (ERDF).

Our raw data and code are available at http://veri.nymity.ch/gfw.

References

[1] The Tor Project. *Bridge easily detected by GFW*. URL: https://trac.torproject.org/projects/tor/ticket/4185.

[2] Roger Dingledine and Nick Mathewson. *Design of a blocking-resistant anonymity system*. Tech. rep. The Tor Project, 2006.

[3] The Tor Project. *Torproject.org Blocked by GFW in China: Sooner or Later?* URL: https://blog.torproject.org/blog/torprojectorg-blocked-gfw-china-sooner-or-later.

[4] The Tor Project. *Tor partially blocked in China*. URL: https://blog.torproject.org/blog/tor-partially-blocked-china.

[5] The Tor Project. *Picturing Tor censorship in China*. URL: https://blog.torproject.org/blog/picturing-tor-censorship-china.

[6] The Tor Project. *China blocking Tor: Round Two*. URL: https://blog.torproject.org/blog/china-blocking-tor-round-two.

[7] Zhen Ling et al. "Extensive Analysis and Large-Scale Empirical Evaluation of Tor Bridge Discovery". In: *International Conference on Computer Communications*. IEEE, 2012.

[8] Tim Wilde. *Great Firewall Tor Probing Circa 09 DEC 2011*. URL: https://gist.github.com/da3c7a9af01d74cd7de7.

[9] Roger Dingledine, Nick Mathewson, and Paul Syverson. "Tor: The Second-Generation Onion Router". In: *USENIX Security Symposium*. USENIX Association, 2004, pp. 303–320.

[10] Amazon Web Services LLC. *Amazon Elastic Compute Cloud (Amazon EC2)*. URL: https://aws.amazon.com/ec2/.

[11] The Tor Project. *Tor Cloud*. URL: https://cloud.torproject.org.

[12] OpenNet Initiative. *Singapore*. URL: http://opennet.net/research/profiles/singapore.

[13] The netfilter.org project. *netfilter/iptables project homepage*. URL: http://www.netfilter.org.

[14] Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson. "Ignoring the Great Firewall of China". In: *Privacy Enhancing Technologies*. Springer, 2006, pp. 20–35.

[15] Hal Roberts. *Local Control: About 95% of Chinese Web Traffic is Local*. URL: https://blogs.law.harvard.edu/hroberts/2011/08/15/local-control-about-95-of-chinese-web-traffic-is-local/.

[16] Team Cymru, Inc. *IP to ASN Mapping*. URL: https://www.team-cymru.org/Services/ip-to-asn.html.

[17] Hal Roberts et al. "Mapping Local Internet Control". In: *Computer Communications Workshop*. IEEE, 2011.

[18] Xueyang Xu, Z. Morley Mao, and J. Alex Halderman. "Internet Censorship in China: Where Does the Filtering Occur?" In: *Passive and Active Measurement Conference*. Springer, 2011, pp. 133–142.

[19] The Tor Project. *Tor users via bridges*. URL: https://metrics.torproject.org/users.html?graph=bridge-users&start=2012-01-01&end=2012-06-18&country=cn&dpi=72#bridge-users.

[20] The Tor Project. *Knock Knock Knockin' on Bridges' Doors*. URL: https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors.

[21] The Tor Project. *obfsproxy*. URL: https://www.torproject.org/projects/obfsproxy.

[22] The Tor Project. *Tor: Pluggable Transports*. URL: https://www.torproject.org/docs/pluggable-transports.

[23] The Tor Project. *obfs2 (The Twobfuscator)*. URL: https://gitweb.torproject.org/obfsproxy.git/blob/HEAD:/doc/obfs2/protocol-spec.txt.

[24] Bruce Leidl. *obfuscated-openssh*. URL: https://github.com/brl/obfuscated-openssh/blob/master/README.obfuscation.

[25] Hooman M. Moghaddam et al. *SkypeMorph: Protocol Obfuscation for Tor Bridges*. Tech. rep. University of Waterloo, 2012.

[26] Thomas H. Ptacek and Timothy N. Newsham. *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection*. Tech. rep. Secure Networks, Inc., 1998.

[27] Jedidiah R. Crandall et al. "ConceptDoppler: A Weather Tracker for Internet Censorship". In: *Computer and Communications Security*. ACM, 2007, pp. 352–365.

[28] Dug Song. *fragroute*. URL: http://monkey.org/~dugsong/fragroute/.

[29] Jong Chun Park and Jedidiah R. Crandall. "Empirical Study of a National-Scale Distributed Intrusion Detection System: Backbone-Level Filtering of HTML Responses in China". In: *Distributed Computing Systems*. IEEE, 2010, pp. 315–326.

Paper II

Towards a Censorship Analyser for Tor

Reprinted from

USENIX FOCI, 2013.

Towards a Censorship Analyser for Tor

Philipp Winter
philwint@kau.se

Abstract

Analysing censorship incidents targeting popular circumvention tools such as Tor can be a tedious task. Access to censoring networks is typically difficult to obtain and remote analysis is not always possible. Analysis is however feasible if users behind the censoring networks are given the opportunity to help.

In this paper, we propose a lightweight censorship analyser for Tor which is meant to be run by volunteering users. The analyser automatically gathers relevant data and the final report is sent back to the Tor developers. Our design builds on existing software and should be easy to bundle and deploy.

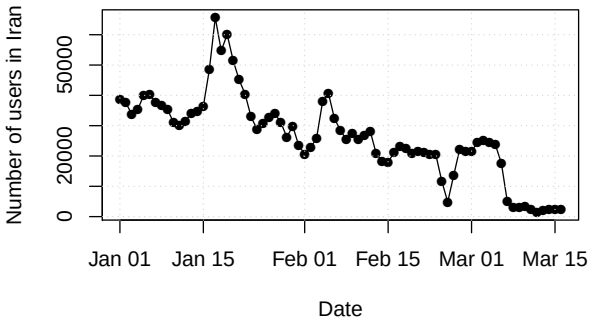


Figure 1: The amount of Iranian Tor users since the beginning of 2013 [4]. In February, the country started deploying a new strategy to block Tor.

1 Introduction

Anonymity and censorship resistance make a great couple. A good example for this is the Tor anonymity network [1]. While Tor's core competence is low-latency anonymity, over the years it developed a reputation as being a practical tool to safely circumvent censorship. Unsurprisingly, this reputation did not remain unnoticed among censors.

Due to Tor's popularity—Iran alone once accounted for more than 30,000 users as can be seen in Figure 1—censoring countries came up with techniques to identify and block Tor connections. As of March 2013, Tor was or is documented to be blocked in China, Iran, Syria, Ethiopia, the United Arab Emirates and Kazakhstan [2]. And these are only the documented censorship incidents. We expect the dark number to be higher. All these blocks greatly differ in their sophistication and range from simple IP address blacklisting to a sophisticated hybrid consisting of deep packet inspection (DPI) and active probing [3].

Censorship evasion can be seen as a *feedback loop*: circumvention tool developers learn from blocks and adapt their tools whereas censors also learn from circumvention tools and refine their blocking techniques. This feedback loop is, however, *highly asymmetric*, meaning that circumvention tool developers learn significantly less than censors. For example, the Tor project is traditionally very open in its circumvention work [2]. Code, designs, and data are available and discussed in public. Censorship systems such as the Great Firewall of China (GFW), on the other hand, are a black box. Typically, these black boxes are repeatedly queried by developers and researchers in order to unravel their inner workings. The purpose of the censorship analyser discussed in this paper is to *reduce this information asymmetry*. That way, we strive to ease censorship analysis and boost circumvention tool development.

The task of censorship analysis requires a way to observe the respective censorship system. Unfortunately, analysis is not always possible from outside the censoring networks. As a result, two popular analysis strategies are

to either *1)* obtain network traffic traces for manual inspection and/or to *2)* gain access to machines inside the censoring regime. Both of these approaches can turn out to be difficult in practice. Network traces require the cooperation of users in the censoring country and are difficult to anonymise which poses a problem of operational security. Further, remote access to machines is problematic due to the lack of volunteers or appropriate proxies such as PlanetLab, open SOCKS proxies or VPSes for rental.

The above should highlight that there is a strong need for a lightweight and easy to use tool which can assist in the process of analysing blocking incidents. In a nutshell, this tool should be run by volunteering users within the censoring country and conduct a number of networking tests in order to gain an understanding of how and if Tor is blocked in a given country or network. The analysis report should then reach the Tor developers in a safe way and thus facilitate circumvention and documentation [2]. The main contribution of this paper is the design and software architecture for such a lightweight censorship analyser for the Tor anonymity network.

The remainder of this paper is structured as follows. We discuss related work in §2. The technical requirements for our analyser are presented in §3. We proceed with the software architecture in §4 and review our concept in §6. We finally conclude this paper in §7.

2 Related Work

Lots of measurement studies have been conducted in the past in order to create "one-time snapshots" of censorship. This involves understanding the GFW [3, 5, 6, 7, 8], the backbone infrastructure of Iran [9], large-scale Internet shutdowns [10] or censorship as it is conducted world-wide [11]. These studies do not—or only to a limited degree—consider censorship as it evolves over time. This is not surprising since the censor could systematically narrow down the measurement devices and tamper with the results.

The time component was first captured by Crandall *et al.* in 2007 [12]. The authors proposed a censorship monitor called ConceptDoppler which is able to determine which keywords are filtered over time. The authors further made use of latent semantic analysis to infer keywords to probe.

In 2011, Sfakianakis, Athanasopoulos and Ioannidis proposed CensMon, a distributed web censorship monitor [13]. CensMon requires PlanetLab access in order to function. A central server receives URLs as input and instructs several distributed agents to probe these URLs in order to detect censorship.

Most recently, Filastò and Appelbaum proposed the open observatory of network interference (OONI) in 2012 [14]. OONI, which is developed by the Tor project, has broader goals than ConceptDoppler or CensMon: in addition to censorship, it is also meant to detect and document surveillance and network interference. OONI is still under heavy development but was already used to expose several censorship incidents[10].

[10] URL: https://ooni.torproject.org/archives.html.

The OpenNet Initiative [15]—a collaboration between Harvard University, the University of Toronto and the SecDev group—maintains censorship-related country profiles. In contrast to OONI, the OpenNet Initiative only publishes its gathered reports; their methods as well as the developed tools are not publicly available.

Finally, it is important to note that Internet censorship and related data sets are also of interest outside computer science. Research in the social sciences, for example, strives to understand Internet censorship on a policy level. Unfortunately, poorly documented, raw and unprocessed data sets can be a major hurdle to social science researchers as pointed out by Asghari *et al.* [16]. In particular, the authors discussed their difficulties in working with the Glasnost data provided by M-Lab[11] [17]. Based on these difficulties, Asghari *et al.* presented several suggestions for data set creators which would, when addressed, facilitate working with such data sets. We strive to consider their suggestions in the following sections.

Our approach differs from the work discussed above in that *1)* our tool is designed specifically for Tor and *2)* intended to be run by users rather than developers. This user-centric design comes with several challenges we aim to address in the following sections. By including users, we aim to be able to "shine light in dark places" and expose censorship which cannot be measured from outside the respective network.

3 Requirements and Design

At first glance, one would expect a censorship analyser to gather as much data and conduct as many experiments as possible. In practice however, this can be a bad idea since the analyser will be run by non-technical computer users who might even face repercussions for running such a "noisy" tool. Besides, users should not be linkable to the data, their copy of our tool generated. As a result, it is important to find a balance between *meaningful data* and respecting the *privacy and security* of users volunteering to run the analyser.

In §3.1 and §3.2, we will enumerate a variety of features we consider desirable in a Tor censorship analyser. These features are motivated by prior experience in debugging censorship incidents as well as by proactively probing for what censors might implement next. Naturally, some features are harder to implement than others. Therefore, the list is organised in ascending order based on the difficulty of the respective feature. While these features were designed specifically for Tor, some might be interesting for other circumvention tools as well.

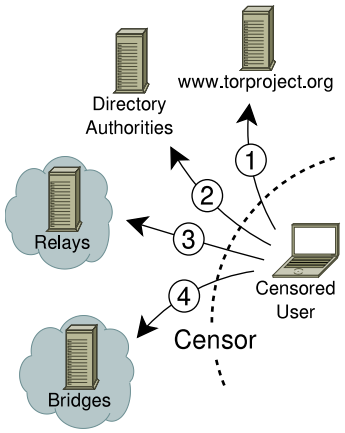


Figure 2: The Tor censorship analyser *1)* probes the official website, *2)* tries to download the consensus, *3)* tries to connect to relays *4)* and to bridges.

3.1 Analysis-centric Requirements

3.1.1 Network Trace of Analysis

Optionally, the analyser should be able to capture a network trace in pcap format. A network trace enables minute inspection of censorship techniques such as spoofed TCP reset segments injected into the user's connection. The trace must be limited to the network traffic generated by our censorship analyser and must not contain network traffic other than that. While clearly useful, pcaps contain IP addresses and can pose a threat to users having strong threat models. Techniques such as Crypto-PAn [18] could be used to pseudonymise IP addresses but packet payload could still contain identifying information. As a result, this feature should be turned off by default. If a user decides to activate it, she should be warned that she gives up a large part of her anonymity by doing so.

3.1.2 Difficult to Detect

An effort should be made to avoid obvious network activity which would make it straightforward for censors to isolate users running our analyser. Obfuscation strategies should involve random sleep periods between and during network tests and randomising the order of executed tests. We also note, however, that a fully undetectable analyser would be very difficult to implement; it would include emulation of typical user behaviour as well as steganography to transmit the final report without censors noticing.

[11] URL: http://www.measurementlab.net.

3.1.3 Probe the Website

The official website, www.torproject.org, is sometimes found to be blocked [2]. With the website being unreachable, users should be prevented from downloading the Tor Browser Bundle (TBB) [19]. To detect website blocks, the analyser should try to download the index page by emulating an HTTP GET request of a modern browser such as Firefox or Chrome. We note that it is important to craft the GET requests to resemble one of these browsers. Otherwise, the GET requests could fail if censors whitelist user agents. If the website is believed to be blocked—because the index page could not be fetched or the TLS connection failed—the following subsequent analysis steps are performed.

First, the domain www.torproject.org must be resolved by querying the user's configured DNS server. The returned IP addresses (if any) are then compared to the expected IP addresses which are hard-coded in our analyser. That way, we can detect DNS poisoning. Other open and "well-behaved" DNS servers such as Google's 8.8.8.8 could further be queried. That way, we hope to be able to detect transparent DNS rewriting assuming that we trust 8.8.8.8 to return the right DNS records.

Second, another website request should be issued for one of the official Tor mirrors. In particular, for mirrors without the strings "tor" or "torproject" in their domain name. While the official website might be blocked, mirrors could very well be reachable.

Third and finally, in a subsequent step the analyser should verify whether a TCP connection to port 443 of `www.torproject.org` is possible. Besides, a TLS connection with a modified Server Name Identifier (SNI) could unravel censorship based on SNI inspection in the TLS client hello.

The same method should be applied for bridges.torproject.org which is used to distribute so-called bridges (cf. §3.1.6) to users who find themselves unable to connect to the public Tor network.

3.1.4 Probe the Directory Authorities

As of March 2013, nine directory authorities serve the consensus for the Tor network. The consensus contains the approximately 3,000 relays which form the network. These authorities are a natural *choke point*. If a client is unable to contact any of them, a direct connection to the public Tor network can not be bootstrapped. Therefore, the authorities are sometimes blocked on the IP layer [3]. To detect such censorship attempts, our analyser should be able to connect to the authorities and try to download the consensus. Should the analyser be unable to contact any of the hard-coded authorities, two subsequent checks should be performed.

First, ICMP echo requests should be sent to the authorities. Most of the authorities reply to ping requests. If no reply is received, this could be an indicator of blocking. Note that this test could easily yield false positives if the user's network disallows ICMP in general.

Second, traceroutes should be run to the potentially blocked authorities. In case of filtering, traceroutes could help narrow down the hops on the network path responsible for the block. Further, traceroutes can be conducted using a variety of protocols such as TCP, UDP and ICMP to increase the likelihood of this test to succeed.

3.1.5 Relay Reachability

After a client successfully downloaded the consensus, the next step in bootstrapping a Tor connection consists of selecting relays which together form a circuit. The first relay in a circuit—the so-called entry guard—must be reachable for the client.

Another straightforward way to block access to the Tor network lies in periodically downloading the consensus and blacklisting all IP addresses found within. In fact, it is sufficient to blacklist only the entry guards. We can easily detect this censorship strategy by trying to initiate a TLS connection to several entry guards. However, it would also be interesting to learn whether connections to pure middle or exit relays succeed. If not, then this could be an indicator that a censor is blindly blacklisting all IP addresses found in the consensus.

Connections to entry guards can fail for a number of different reasons. Perhaps the most popular strategy among censors is to terminate or drop TLS handshakes which appear to be Tor-specific. Over time, Tor's TLS handshake exhibited a number of distinguishing elements such as the client cipher list [3, 20], the server certificates as well as the randomly generated SNIs. TLS-based filtering was or is done in Ethiopia [21], China [3] and Iran [22], just to name a few. To detect Tor-specific TLS filtering, a Tor-specific TLS client hello could be sent to machines which are not Tor relays; for example the host behind `mail.google.com`. Inferring the exact pattern matched by DPI boxes is a difficult task and was sometimes done manually [20]. An automated way would be highly desirable but is beyond the scope of this paper.

3.1.6 Bridge Reachability

Analogous to ordinary Tor relays, *bridges* should be tested as well. Bridges are relays which are not listed in the public Tor consensus. Instead, they are distributed to censored users out-of-band and serve as "hidden" stepping stones into the Tor network.

Furthermore, using *obfsproxy* it is possible to obfuscate the network traffic exchanged between clients and bridges [23]. Obfsproxy is merely an obfuscation framework enabling so-called pluggable transport modules which dictate how traffic is to be obfuscated. At the time of writing, Tor can be used with the pluggable transports obfs2 [24] and obfs3 [25]. More transports are under development [26]. In order to detect DPI targeting these pluggable transports, the analyser should be able to conduct obfs2 and obfs3 handshakes and see if they succeed.

3.1.7 Gather Debug Information

Censorship is typically not homogeneous across a country and can differ on the level of provinces, autonomous systems or even network prefixes [27]. As a result, we are interested in information which can help narrow down the respective censorship infrastructure. Also, this would help ruling out interferences and prevent jumping to wrong conclusions. Of interest would be the following information.

1. What ISP does the user have? Our analyser could obtain the whois record for the user's IP address and discard all IP address material which would otherwise reveal the user's location.

2. What is the autonomous system number? Open IP-to-ASN lookup services[12] could be used.

3. Is the user behind a captive portal? This can be difficult to verify and might require a number of heuristics.

4. Is all traffic forced to go through an HTTP proxy?

3.1.8 Anonymising Reports

Naturally, reports should not be linkable to users submitting them. We have to distinguish between *report content* and *report submission*.

The actual report is straightforward to anonymise. As an option, we can easily discard identifying information in the report by, e.g., never logging the user's IP address and redacting the first hops in traceroutes. A high degree of anonymity could be achieved by completely discarding all IP addresses, prefixes and location information such as autonomous system numbers and whois records. Network captures in the form of pcap files are very difficult to anonymise and should also be completely discarded if strong anonymity is desired.

Problematic however, is report submission. It is the task of tools such as Tor to provide anonymity on the address layer. Unfortunately, we cannot make use of Tor as the very purpose of our tool is to find out why Tor is unreachable. Users could be advised to generate one-time email addresses for submission over email. That way however, the email provider would learn the user's IP address[13]. This would place a lot of control into the hands of an untrusted third party so another—possibly better—option is automatic submission using an HTTPS POST request to infrastructure run by the Tor project. This method could be easier to block and once again, this cannot disguise the user's IP address, unfortunately.

[12] URL: http://asn.cymru.com

[13] The content of the report can be protected when the report is encrypted using the hard-coded PGP public key (see §4.1).

3.2 User-centric Requirements

While the analyser's main purpose is to gather censorship-related data, it is important to recognise that it will frequently—if not mostly—be run by users who *lack technical expertise*. These users will not be able to configure the analyser manually or run tools on the command line. This observation motivates the following requirements.

3.2.1 User-friendly Output

We emphasise that the primary purpose of the analyser is to gather data which can assist Tor developers. Still, as a side task, the analyser should inform users about the gathered data and results while avoiding too much technical jargon. Based on a decision tree inferred from the gathered data, the analyser could inform the user about possible ways to circumvent the respective censoring network.

3.2.2 Cover our Tracks

As mentioned above, it is crucial to protect users' privacy. This implies that temporary files or reports must not be stored in non-obvious locations on the user's hard drive. All temporary data must remain in the unpacked directory containing the analyser. That way, a user can easily delete the entire directory and by doing so also delete gathered data.

Nevertheless, it is difficult to hide the fact that a certain program was executed [28]. Even more so on Windows. What can be hidden is the analysis results by simply deleting them after submission[14]. The very existence of our analyser is difficult to disguise, however.

We argue that *usage diversity* can mitigate the threat arising from simply having a copy of our tool. Users could state that they were simply interested in finding out why their TBB would not bootstrap (cf. §3.2.1) rather than in assisting in circumventing their national firewall.

3.2.3 Ease of Use and Informed Consent

It is crucial that the analyser is as easy to use as possible. In particular, it should be a self-contained click-and-go executable, just like the TBB. Ease of use also involves the analyser's *bundle size*. Ideally, the analyser would only be a few megabytes in size which would also make is suitable for email distribution over GetTor[15].

Users should be informed in clear words that the analyser is performing network probing and that the gathered data can be submitted afterwards for further inspection. Users should be given the opportunity to abort the

[14] Note that simple file removal might not be sufficient when file system forensics is conducted.
[15] URL: https://www.torproject.org/projects/gettor.html.en

process prior to network probing and prior to submission. Note that internationalisation of our analyser will also be necessary since it is unrealistic to expect users to have a decent command of the English language.

4 Software Architecture

There is no need to reinvent the wheel. The analyser will be implemented as a set of tests for OONI [14]. OONI, which was already introduced in §2, is a framework for network censorship analysis and provides a Python API which can be used to develop all of the requirements discussed above. In addition, OONI defines a custom YAML format—YAMLOO—for analysis reports. At the time of writing OONI cannot, however, be invoked by running a single executable. As a result, our analyser must be packaged together with OONI to a single Windows executable. This can be done using tools such as py2exe[16]. The following sections discuss a number of architectural considerations.

4.1 Communicating Results

Eventually, the data produced by a user's analyser—mainly a text file containing YAMLOO data—has to reach the Tor developers. We believe that it would be short-sighted to define a single mechanism for the transmission to happen. This would create a single point of failure which might turn out to be easy to block for censors.

Instead, we envision several reasonable communication channels. Ideally, we would send the report in an anonymous fashion and upload it to a hidden service. But as already discussed in §3.1.8, we expect Tor to be censored and not an option for this. A report could be sent manually over email. While this requires user interaction, some sort of email is typically available; even in countries with pervasive censorship. In particular, free services such as GMail[17] are frequently reachable over HTTPS.

Another option would be to automatically transmit the report using an HTTPS POST request to a web server operated by the Tor project. While single web servers are straightforward to block, we could increase collateral damage by running the web server inside a cloud provider, the censor is hopefully unwilling to blacklist. Other communication channels could include instant messaging programs or steganographic publishing systems. Unfortunately, all of these methods have in common that the user's IP address will eventually be known by a provider of some sort.

Furthermore, the report should contain a *message digest* which is calculated over the YAMLOO report. This is particularly important when the report is being sent over email since it makes it possible to detect if the report is incomplete or the user accidentally changed parts of it.

[16] URL: http://www.py2exe.org.
[17] URL: https://mail.google.com.

Finally, the analyser should contain a hard-coded PGP public key which can be used to encrypt analysis reports. Users who decide to encrypt the report should still have the opportunity to review a report prior to encryption and submission. While encryption comes at the cost of key management, we believe that the additional management overhead is worth the increase in security in certain situations.

4.1.1 Configurable and Testable During Build

It should be possible to pass configuration parameters to the analyser during the build process. That is necessary because certain information will be subject to change. This includes hard-coded IP addresses of relays or the web servers hosting www.torproject.org. If we would be unable to change these parameters, a censor could simply whitelist the hard-coded IP addresses and render our analysis results useless.

All the features discussed in §3 should be testable in an automated way. Otherwise, we might end up shipping code which does not work in real environments or we might not notice if improvements break existing code.

5 Discussion

We point out that motivated and strongly equipped censors would be able to identify users trying to run our analyser. Such censors would also be able to actively falsify our analyser's results. Being undetectable would require a substantially more complex concept. However, we believe that most censors are more motivated to spend their resources on actual blocking technology rather than measurement and analysis technology. Of course, this will change if our analyser should turn out to be a very valuable tool in the arms race.

There are two additional important features not discussed here due to page constraints. First, our analyser could be enhanced to be able to infer the patterns used by DPI boxes to identify protocols. According to our own experience, DPI boxes do not always just use simple regular expressions but also context-sensitive languages. This is, however, a difficult problem which might require a client-server architecture which runs a grammatical inference algorithm. Precise knowledge of the patterns used for censorship would enable targeted circumvention and could be fed into tools such as format-transforming encryption [29].

Second and equally difficult, our analyser could have features to detect the type or model of DPI box used for censorship. This would make it possible to expose cases similar to the use of Bluecoat in Syria [30]. While well-designed DPI boxes not exposing their management interface can be difficult to identify, it might be possible to define a number of heuristics to cluster DPI boxes; perhaps based on injected TCP reset segments (cf. [31]).

6 Conclusion

In this paper, we outlined the design requirements for a censorship analyser for the Tor anonymity network. While the majority of these requirements discuss network probing techniques, we also consider usability aspects. We plan to implement our concept in the next step.

We believe that involving ordinary computer users in the process of censorship analysis can greatly increase the coverage of censorship documentation and shed light on previously unknown types of censorship. Great care must be taken, however, to not exploit users and inform them about the process. Informed consent must be achieved whenever possible. Furthermore, we hope to start a discussion about how to safely integrate non-technical users into the process of censorship measurement.

Acknowledgements

We want to thank Arturo Filastò, Simone Fischer-Hübner, George Kadianakis, Karsten Loesing, Tobias Pulls, Runa A. Sandvik and the anonymous reviewers for their valuable feedback.

Finally, we want to express our gratitude to Internetfonden of the Swedish Internet Infrastructure Foundation for supporting the author's work with a research grant.

References

[1] Roger Dingledine, Nick Mathewson, and Paul Syverson. "Tor: The Second-Generation Onion Router". In: *USENIX Security*. USENIX Association, 2004, pp. 303–320. URL: http://static.usenix.org/event/sec04/tech/full_papers/dingledine/dingledine.pdf.

[2] The Tor Project. *Censorship Wiki*. URL: https://censorshipwiki.torproject.org.

[3] Philipp Winter and Stefan Lindskog. "How the Great Firewall of China is Blocking Tor". In: *FOCI*. USENIX Association, 2012. URL: https://www.usenix.org/system/files/conference/foci12/foci12-final2.pdf.

[4] The Tor Project. *Directly connecting users from Iran*. URL: https://metrics.torproject.org/users.html?graph=direct-users&start=2013-01-01&end=2013-03-15&country=ir&events=off#direct-users.

[5] Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson. "Ignoring the Great Firewall of China". In: *PETS*. Springer, 2006, pp. 20–35. URL: http://www.cl.cam.ac.uk/~rnc1/ignoring.pdf.

[6] Sparks et al. "The Collateral Damage of Internet Censorship by DNS Injection". In: *SIGCOMM Computer Communication Review* 42.3 (), pp. 21–27. URL: http://conferences.sigcomm.org/sigcomm/2012/paper/ccr-paper266.pdf.

[7] Xueyang Xu, Z. Morley Mao, and J. Alex Halderman. "Internet Censorship in China: Where Does the Filtering Occur?" In: *PAM*. Springer, 2011, pp. 133–142. URL: http://www.eecs.umich.edu/~zmao/Papers/china-censorship-pam11.pdf.

[8] Jong Chun Park and Jedidiah R. Crandall. "Empirical Study of a National-Scale Distributed Intrusion Detection System". In: *ICDCS*. IEEE, 2010, pp. 315–326. URL: http://www.cs.unm.edu/~crandall/icdcs2010.pdf.

[9] Collin Anderson. *The Hidden Internet of Iran: Private Address Allocations on a National Network*. Tech. rep. 2012. URL: http://arxiv.org/pdf/1209.6398v1.

[10] Alberto Dainotti et al. "Analysis of Country-wide Internet Outages Caused by Censorship". In: *IMC*. ACM, 2011, pp. 1–18. URL: http://conferences.sigcomm.org/imc/2011/docs/p1.pdf.

[11] John-Paul Verkamp and Minaxi Gupta. "Inferring Mechanics of Web Censorship Around the World". In: *FOCI*. USENIX Association, 2012. URL: https://www.usenix.org/system/files/conference/foci12/foci12-final1.pdf.

[12] Jedidiah R. Crandall et al. "ConceptDoppler: A Weather Tracker for Internet Censorship". In: *CCS*. ACM, 2007, pp. 352–365. URL: http://www.cs.unm.edu/~crandall/concept_doppler_ccs07.pdf.

[13] Andreas Sfakianakis, Elias Athanasopoulos, and Sotiris Ioannidis. "CensMon: A Web Censorship Monitor". In: *FOCI*. USENIX Association, 2011. URL: http://static.usenix.org/event/foci11/tech/final_files/Sfakianakis.pdf.

[14] Arturo Filastò and Jacob Appelbaum. "OONI: Open Observatory of Network Interference". In: *FOCI*. USENIX Association, 2012. URL: https://www.usenix.org/system/files/conference/foci12/foci12-final12.pdf.

[15] OpenNet Initiative. URL: https://opennet.net.

[16] Hadi Asghari et al. *Making Internet Measurements Accessible for Multi-Disciplinary Research*. Tech. rep. TU Delft and Syracuse University, 2012. URL: http://dpi.ischool.syr.edu/MLab-Data_files/HA-MM-MvE-IMC.pdf.

[17] Marcel Dischinger et al. "Glasnost: Enabling End Users to Detect Traffic Differentiation". In: *NSDI*. USENIX Association, 2010. URL: http://broadband.mpi-sws.org/transparency/results/10_nsdi_glasnost.pdf.

[18] *Cryptography-based Prefix-preserving Anonymization*. URL: http://www.cc.gatech.edu/computing/Telecomm/projects/cryptopan/.

[19] The Tor Project. *Tor Browser Bundle*. URL: https://www.torproject.org/projects/torbrowser.html.en.

[20] Tim Wilde. *Great Firewall Tor Probing Circa 09 DEC 2011*. 2012. URL: https://gist.github.com/twilde/da3c7a9af01d74cd7de7.

[21] The Tor Project. *An update on the censorship in Ethiopia*. 2012. URL: https://blog.torproject.org/blog/update-censorship-ethiopia.

[22] The Tor Project. *Iran blocks Tor; Tor releases same-day fix*. 2011. URL: https://blog.torproject.org/blog/iran-blocks-tor-tor-releases-same-day-fix.

[23] The Tor Project. *obfsproxy*. URL: https://www.torproject.org/projects/obfsproxy.

[24] The Tor Project. *obfs2 (The Twobfuscator)*. URL: https://gitweb.torproject.org/obfsproxy.git/blob/HEAD:/doc/obfs2/protocol-spec.txt.

[25] The Tor Project. *obfs3 (The Threebfuscator)*. URL: https://gitweb.torproject.org/user/asn/obfsproxy.git/blob/HEAD:/doc/obfs3/obfs3-protocol-spec.txt.

[26] The Tor Project. *Tor: Pluggable Transports*. URL: https://www.torproject.org/docs/pluggable-transports.html.en.

[27] Joss Wright, Tulio de Souza, and Ian Brown. "Fine-Grained Censorship Mapping: Information Sources, Legality and Ethics". In: *FOCI*. USENIX Association, 2011. URL: http://static.usenix.org/event/foci11/tech/final_files/Wright.pdf.

[28] Runa A. Sandvik. *Forensic Analysis of the Tor Browser Bundle on OS X, Linux, and Windows*. Tech. rep. The Tor Project, 2013. URL: https://research.torproject.org/techreports/tbb-forensic-analysis-2013-06-28.pdf.

[29] Kevin P. Dyer et al. *Protocol Misidentification Made Easy with Format-Transforming Encryption*. Tech. rep. Portland State University, RedJack, LLC., and University of Wisconsin, 2012. URL: http://eprint.iacr.org/2012/494.pdf.

[30] Jillian C. York. *Government Internet Surveillance Starts With Eyes Built in the West*. 2011. URL: https://www.eff.org/deeplinks/2011/09/government-internet-surveillance-starts-eyes-built.

[31] Nicholas Weaver, Robin Sommer, and Vern Paxson. "Detecting Forged TCP Reset Packets". In: *NDSS*. The Internet Society, 2009. URL: http://www.icsi.berkeley.edu/pubs/networking/ndss09-resets.pdf.

Paper III

ScrambleSuit: A Polymorphic Network Protocol to Circumvent Censorship

Reprinted from

ACM WPES, 2013.

ScrambleSuit: A Polymorphic Network Protocol to Circumvent Censorship

Philipp Winter and Tobias Pulls and Juergen Fuss
philwint@kau.se

Abstract

Deep packet inspection technology became a cornerstone of Internet censorship by facilitating cheap and effective filtering of what censors consider undesired information. Moreover, filtering is not limited to simple pattern matching but makes use of sophisticated techniques such as active probing and protocol classification to block access to popular circumvention tools such as Tor.

In this paper, we propose ScrambleSuit; a thin protocol layer above TCP whose purpose is to obfuscate the transported application data. By using morphing techniques and a secret exchanged out-of-band, we show that ScrambleSuit can defend against active probing and other fingerprinting techniques such as protocol classification and regular expressions.

We finally demonstrate that our prototype exhibits little overhead and enables effective and lightweight obfuscation for application layer protocols.

1 Introduction

We consider deep packet inspection (DPI) harmful. While originally meant to detect attack signatures in packet payload, it is ineffective in practice due to the ease of evasion [1, 2, 3]. At the same time, DPI technology is increasingly used by censoring countries to filter the free flow of information or violate network neutrality [4]. We argue that what makes DPI particularly harmful is the *asymmetry of blocking effectiveness*, i.e., it is hard to stop motivated and skilled network intruders but very easy to censor ordinary user's Internet access. DPI technology ultimately fails to protect critical targets but succeeds in filtering the information flow of entire countries.

Numerous well-documented cases illustrate how DPI technology is used by censoring countries. Amongst others, China is using it to filter HTTP [5] and rewrite DNS responses [6]. Iran is known to use DPI technology to conduct surveillance [7]. In Syria, DPI technology is used for the same purpose [8]. Even more worrying, TLS interception proxies, an increasingly common feature of DPI boxes, are used to transparently decrypt and inspect TLS sessions which effectively breaks the confidentiality provided by TLS.

The rise of Internet censorship led to the creation of numerous circumvention tools which engage in a rapidly developing arms race with the maintainers of censorship systems. Of particular interest to censoring countries is the Tor network [9]. While originally designed as a low-latency anonymity network, it turned out to be an effective tool to circumvent censorship as well. Tor's growing success as a circumvention tool did not remain unnoticed, though. Tor is or was documented to be blocked in many countries including Iran [10], China [11], and Ethiopia [12], just to name a few. We argue that many circumvention tools—Tor included—suffer from two shortcomings which can easily be exploited by censors.

First and most importantly, they are vulnerable to *active probing* as pioneered by the Great Firewall of China (GFW) [11]: the GFW is able to block Tor by first looking for potential Tor connections based on the TLS client cipher list. If such a signature is found on the wire, the GFW reconnects to the suspected Tor bridge and tries to "speak" the Tor protocol with it. If this succeeds, the GFW blacklists the respective bridge. Active probing is not only used to discover Tor but—as we will discuss—also VPNs [13] and obfs2 [14], which is a censorship-resistant protocol. The relevance of active probing attacks is emphasised by the work of Durumeric et al. [15]. By conducting fast Internet-wide scanning, the authors were able to find approximately 80% of all active bridges at the time. From a censor's point of view, active probing is a promising strategy which greatly reduces collateral damage caused by inaccurate signatures. The attack is also non-trivial to defend against because censors can easily emulate real computer users.

Second, circumvention tools tend to exhibit a specific "flow signature" which typically remains static and is the same for all clients and servers speaking the protocol. An example is Tor's characteristic 586-byte signature (see §4.3.1). If a censor manages to deploy high-accuracy classifiers trained to

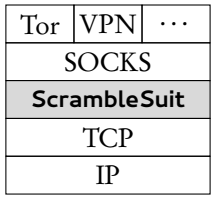


Figure 1: ScrambleSuit's protocol stack.

recognise these very flow signatures, the respective protocol could easily be spotted and blocked. Most circumvention tools are not polymorphic which makes them unable to survive such filters.

In this work, we present ScrambleSuit; a blocking-resistant transport protocol which tackles the two above mentioned problems. ScrambleSuit defines a thin protocol layer on top of TCP which provides lightweight obfuscation for the transported application layer protocol. As shown in Figure 1, ScrambleSuit is independent of its application layer protocol and works with any application supporting SOCKS. As a result, we envision ScrambleSuit to be used by, amongst other protocols, Tor and VPN to tackle the GFW's most recent censorship upgrades. In particular, ScrambleSuit exhibits the following four features:

Pseudo-random payload: To an observer, ScrambleSuit's entire traffic is computationally indistinguishable from randomness. As a result, there are no predictable patterns which would otherwise form suitable DPI fingerprints. This renders regular expressions for the purpose of identifying ScrambleSuit useless.

Polymorphic: Despite the pseudo-random traffic, a censor could still block our protocol based on flow characteristics such as the packet length distribution. ScrambleSuit is, however, able to change its shape to make it harder for classifiers to exploit flow characteristics.

Shared secret: We defend against active probing by making use of a secret which is shared between client and server and exchanged out-of-band. A server only talks to clients if knowledge of the secret is proven by the client.

Usable: We seek to maximise ScrambleSuit's usability. Our protocol integrates in Tor's existing ecosystem. Furthermore, the moderate protocol overhead discussed in §5 facilitates comfortable web surfing.

Blocking-resistant protocols can be split into two groups. While the first group strives to *mimic typically whitelisted protocols* such as HTTP [16], Skype [17, 18] and email [19], the second group aims to *look like randomness* [20, 21, 22]. Randomised protocols have the shortcoming of not being able to

survive a whitelisting censor[18]. Nevertheless, we decided in favour of randomising because mimicking comes at the cost of high overhead and we consider whitelisting on a nation scale—at least for most countries—unlikely even though it is often done in corporate networks and some countries appear to be experimenting with the concept [24, 25]. While at some point, our protocol might indeed become victim of a country's whitelisting policy, we believe that our approach provides a fast alternative in many censoring countries unwilling to go that far. So instead of maximising obfuscation while maintaining an acceptable level of usability, we seek to *maximise usability* while keeping an *acceptable level of obfuscation*.

We finally point out that unblockable network protocols do not exist. After all, censors could always "pull the plug" as it was already done in Egypt [26] and Syria [27]. By proposing ScrambleSuit, we do not claim to end the arms race in our favour but rather to raise the bar once again.

The contributions of this paper are as follows.

1. We propose ScrambleSuit, a blocking-resistant transport protocol.
2. We present two authentication mechanisms based on shared secrets and polymorphism as a practical defence against active probing and protocol classifiers.
3. We implement and evaluate a fully functional prototype of our protocol which is publicly available under a free license.

The remainder of this paper is structured as follows. In §2 we discuss related work which is followed by an architectural overview in §3. Then, §4 discusses ScrambleSuit's design in detail. The protocol is evaluated in §5 and the results are discussed in §6. We finally conclude the paper in §7.

2 Related Work

Protocol Identification: Hjelmvik and John investigated to which extent supposedly obfuscated protocols such as Skype, BitTorrent's message stream encryption, and Spotify can be identified [28]. Based on their findings, Hjelmvik and John suggest evasion techniques for protocol designers which should make it harder to identify obfuscated protocols. Some of our design decisions were motivated by their suggestions. Similar to that, Wiley proposed a framework to dynamically classify network protocols based on Bayesian models [29]. This is an important first step towards the ability to compare and evaluate blocking-resistant transport protocols.

Protocol Obfuscation: The Tor project developed a blocking-resistant protocol called obfs2 [20]. The protocol implements an obfuscation layer on top of TCP and transports Tor traffic. A passive Man-in-the-Middle (MitM)

[18]The same fate can meet mimicked protocols if the censor decides to block the cover protocol. Oman is documented to block Skype [23] which would render Skype-based circumvention protocols useless.

however can decrypt obfs2 traffic. The successor, obfs3 [21], uses a customised Diffie-Hellman handshake to solve this problem. However, both, obfs2 and obfs3 can be actively probed and do not disguise flow properties. In fact, we found that the GFW is already blocking obfs2 bridges by actively probing them [14]. Later in this paper, we extend obfs3's handshake to be resistant against active probing. Wiley's Dust protocol [22] compares to obfs2 and obfs3 in that Dust payload looks like random data but the key exchange is handled out-of-band. Dust also employs packet padding to camouflage packet lengths. But unlike ScrambleSuit, Dust does not consider inter-arrival times and is unable to change its flow signature.

Weinberg et al. presented StegoTorus [16], an obfuscation framework similar to Tor's obfsproxy [30]. StegoTorus can complicate protocol identification on the application layer as well as on the transport layer. Tor connections can be multiplexed over multiple TCP connections and the application layer is camouflaged by mimicking a cover protocol such as HTTP. SkypeMorph, as presented by Moghaddam et al. [17] disguises Tor traffic as Skype video traffic. Similar work was done by Houmansadr et al. with FreeWave which is able to carry network traffic over VoIP connections [18]. Another attempt to tunnel otherwise censored network traffic over a "whitelisted" protocol was done by Zhou et al. by proposing SWEET [19] which uses email as its cover medium. With CensorSpoofer, Wang et al. propose to decouple the upstream from the downstream channel for censorship resistance [31]. The low-bandwidth upstream channel uses steganography whereas the high-bandwidth downstream channel makes use of IP spoofing to fool censors.

Houmansadr et al. claim that mimicking a cover protocol rather than using it as an actual medium is a flawed approach [32]. The authors showed numerous aspects in which StegoTorus, SkypeMorph and CensorSpoofer differ from their respective cover medium.

Lincoln et al. proposed DEFIANCE [33]: an architecture to protect Tor bridges from being probed and their respective descriptors[19] from being harvested by crawlers. The authors accomplish these goals by developing a novel rendezvous protocol as well as a technique called address-change signaling.

Port-knocking-based Authentication: Vasserman et al. proposed SilentKnock: an undetectable authentication system based on port knocking [34]. Clients can authenticate themselves towards a server with a single TCP SYN segment. If the authentication does not succeed, the server does not reply with a SYN/ACK segment, thus appearing offline.

Smits et al. adapted SilentKnock to better work with Tor bridges [35]. The result is called BridgeSPA. Like SilentKnock, BridgeSPA does not protect against connection hijacking and faces practical problems such as the inability to cope with NAT and the dependence on Linux kernels. While ScrambleSuit can not hide its "aliveness", it is not hindered by NAT, renders connection hijacking useless, and works on every platform having a Python interpreter.

[19] A bridge descriptor is essentially a tuple containing the bridge's IP address, port and fingerprint.

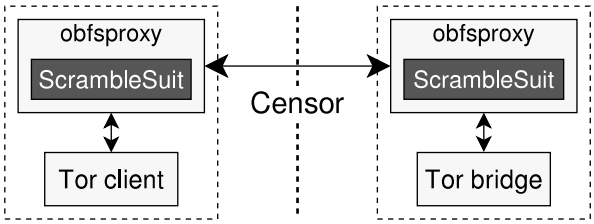


Figure 2: ScrambleSuit is a module for obfsproxy which provides a SOCKS interface for local applications. The network traffic between two obfsproxy instances is disguised by ScrambleSuit.

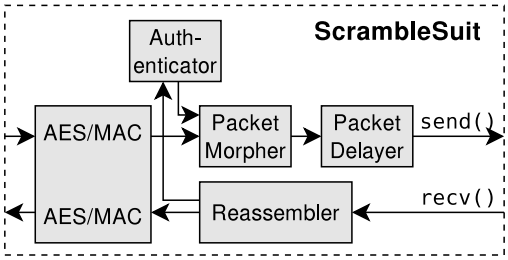


Figure 3: Internally, ScrambleSuit handles authenticated encryption of application data, client authentication as well as flow reshaping using a packet morpher and delayer.

3 Architectural Overview

ScrambleSuit is a module for obfsproxy which is an obfuscation framework developed by the Tor project [30]. As long as obfsproxy is running on the censored client as well as on the server, all network traffic in between both communication points can be obfuscated as dictated by the respective obfuscation modules. Figure 2 illustrates that obfsproxy acts as a proxy between the Tor client and the Tor bridge. While specifically designed for Tor, obfsproxy can be used by any application as long as it supports the SOCKS protocol.

Internally, ScrambleSuit is composed of several components which are depicted in Figure 3. Outgoing network data is first encrypted and then padded by the packet morpher. Before these packets are then sent over the wire, the packet delayer uses small artificial delays to disguise inter-arrival times. Finally, incoming network data is first reassembled to complete ScrambleSuit protocol messages and, after decryption, finally passed on to the local application.

3.1 Threat Model

Our adversary is a *nation-state censor* who desires to block unwanted network protocols and services which would otherwise allow users within the censor-

ing regime the retrieval of unfiltered information or to evade the national filtering system. The censor is making use of payload analysis, flow analysis as well as active probing to identify and then block undesired protocols.

Furthermore, the censor has full *active and passive* control over the national network. The censor can passively monitor all traffic entering and leaving its networks in line rate. We further expect the censor to actively tamper with traffic; namely to inject, drop, and modify traffic as well as hijack TCP sessions.

The censor can also select a subset of suspicious traffic for further inspection on the slow path[20]. This could involve *active probing* as done by the GFW in order to block the Tor network [11]. We model our censor to also conduct active MitM attacks. While we believe that passive analysis and active probing are significantly easier to deploy, there is evidence that censors are starting to—or at least have the ability to—conduct active MitM attacks as well [36].

Our adversary is also training and deploying *statistical classifiers* to identify and block protocols. While computationally expensive, it would be imaginable that a censor uses this strategy at least on the slow path and perhaps even on the fast path when using inexpensive flow features and efficient classifiers.

3.1.1 Adversary Limitations

We expect the censor to be subject to economical constraints. In particular, we assume that the censor is not using a whitelisting approach meaning that only well-defined protocols pass the national filter. Whitelisting implies significant *over-blocking* and we expect this approach to collide with the censor's economical incentives. We also expect the censor to not block protocols when there is only weak evidence for the protocol being blacklisted. This is a direct consequence of avoiding over-blocking to minimise collateral damage.

Finally, we assume that the censor does not have access to or can otherwise influence censored users' computers. We believe that such a scenario is likely to occur in corporate networks but not on a national scale.

4 Protocol Design

This section will discuss ScrambleSuit's defence against active probing, its encryption and header format as well as how we achieve polymorphism.

4.1 Thwarting Active Probing

We defend against active probing by proposing two mutual authentication mechanisms which rely on a secret which is shared *out-of-band*. A ScrambleSuit connection can only be established if both parties can prove knowledge of

[20] We define the *slow path* as the minute analysis of a small traffic subset as opposed to the *fast path* which covers the majority of all network traffic and, as a result, has to be processed quickly.

this very secret. While one authentication mechanism (see §4.1.4) is designed to work well in Tor's ecosystem, the other mechanism (see §4.1.2) provides additional security and efficiency if ScrambleSuit is used by other application protocols such as a VPN.

With respect to Tor, there *already exists* an out-of-band communication channel which is used to distribute bridge descriptors to censored users. Naturally, we make use of this channel. If, however, ScrambleSuit is used to tunnel protocols other than Tor, users have to handle out-of-band communication themselves.

4.1.1 Proof-of-Work (Again) Proves Not to Work

Before deciding in favour of using a secret exchanged out-of-band, we investigated the suitability of client puzzles. Puzzles—a variant of proof-of-work schemes—could be used by a server to time-lock a secret. This secret can then only be unlocked by clients by spending a moderate amount of computational resources on the problem. One particular puzzle construction, namely time-lock puzzles as proposed by Rivest et al. [37], provides appealing properties such as deterministic unlocking time, asymmetric work load and inherently sequential computation which means that adversaries in the possession of highly parallel architectures have no significant advantage over a client with a single CPU.

While *a single* client puzzle can not be solved in parallel, a censor is able to solve *multiple* puzzles in parallel by assigning one puzzle to every available CPU core. This is problematic because our threat model includes censors with powerful and parallel architectures. We estimated the Tor network's bridge churn rate and found that the rate of new bridge IP addresses joining the network (i.e., the amount of puzzles to solve) is not high enough to be able to increase a well-equipped censor's work load beyond the point of becoming *impractical*; at least not without becoming impractical for *clients as well*. This balancing problem is analogous to why proof-of-work schemes are also believed to be unpractical for the spam problem [38].

In summary, proof-of-work schemes would not require a shared secret but we believe that this small usability improvement would come at the cost of greatly reduced censorship resistance. A censor in the possession of powerful computational resources would certainly be slowed down but could ultimately not be stopped. Active probing would simply become a matter of investing more resources.

4.1.2 Authentication Using Session Tickets

We now present the first of our two authentication mechanisms. We envision it to be used by insecure protocols which, unlike Tor, can not protect against active MitM attacks. A client can authenticate herself towards a ScrambleSuit server by redeeming a *session ticket*. A session ticket needs to be obtained only once out-of-band. Subsequent connections are then bootstrapped using tickets issued by the server during the respective previous connection. A real

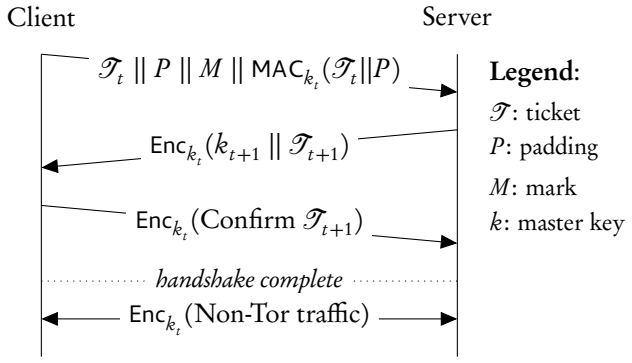


Figure 4: The client redeems a valid session ticket $\mathscr{T}_t$ containing the master key k_t. The server responds by issuing a new ticket $\mathscr{T}_{t+1}$ for future use. After the client confirmed the receipt, both parties then exchange application data.

world analogy would be Alice redeeming a ticket in order to gain access to a football stadium. Upon entering the stadium (i.e., successful authentication), the guards give Alice a new ticket so that she is able to return for the next match. The same procedure then takes place for the next match. Session tickets are standardised in RFC 5077 [39] and part of TLS since version 1.0. We employ only a subset of the standard since we do not need its full functionality.

The basic idea is illustrated in Figure 4. The three-way handshake starts with the client *redeeming* a session ticket. The server then responds by *issuing* a new ticket. Finally, the client *confirms* the receipt of the newly issued ticket.

ScrambleSuit servers issue new session tickets $\mathscr{T}_{t+1}$ which embed a future shared master key k_{t+1} and an issue date d indicating the ticket's creation time. Session tickets are symmetrically encrypted and authenticated with secret keys k_S[21] only known to the server, i.e., $\mathscr{T}_{t+1} = \mathsf{Enc}_{k_S}(k_{t+1} \,||\, d)$. As a result, a ticket is opaque to the client. Note that together with the ticket, a client also has to learn the master key k_{t+1} in order to be able to derive the same session keys as the server; so clients always obtain the tuple $(k_{t+1} \,||\, \mathscr{T}_{t+1})$ from servers.

Session tickets have the advantage that the server *does not have to keep track* of issued tickets. Instead, the server's state is outsourced and stored by clients. This reduces a server's load. To verify whether a ticket is still valid, servers simply decrypt the ticket and check the issue date d embedded in the ticket.

Whenever a client successfully connects to a ScrambleSuit server, the server issues a new ticket concatenated to the according master key $(k_{t+1} \,||\, \mathscr{T}_{t+1})$ for the client. Recall that the ticket is encrypted and opaque to the client which is the reason why the master key is prepended. This tuple is then placed in a special ScrambleSuit control message (see §4.2) which is sent immediately

[21] For simplicity, we refer to these keys as just k_S while they are in fact two symmetric keys: one for encryption and one for authentication.

after a ticket was successfully redeemed. Coming back to the real world example, Alice now has her new ticket which makes it possible for her to return for the next match.

Finally, the client confirms receipt of the new ticket by sending a dedicated confirmation message to the server. After that, the three-way ticket handshake is completed and application data can be exchanged.

Pseudo-random padding: A censor could now conduct traffic analysis by looking for TCP connections which always begin with the client sending $|\mathcal{T}|$ bytes to the server. To obfuscate the ticket's length, we introduce random padding P and authenticate the ticket $\mathcal{T}_t$ as well as the padding P by computing the message authentication code $\mathsf{MAC}_{k_t}(\mathcal{T}_t \,||\, P)$ with k_t being the shared master key obtained by the client together with the ticket. Both parties will derive session keys from k_t as discussed in §4.2.

Locating the MAC: The MAC is computationally indistinguishable from the pseudo-random padding. To facilitate localisation of the MAC, we place a *cryptographic mark* M right in front of it. The mark is defined as $M = \mathsf{MAC}_{k_t}(\mathcal{T}_t)$. After extracting the ticket from the client's first chunk of bytes, the server is now able to calculate the mark and locate the MAC. We use HMAC-SHA256-128 [40] for the MAC and the mark.

Key rotation: We mentioned earlier that a ScrambleSuit server manages secret keys k_S which are used to encrypt and authenticate session tickets. This prevents clients from tampering with tickets and the server can verify that a newly received and authenticated ticket was, in fact, issued by the server. Servers rotate their k_S keys after a period of seven days. After the generation of new k_S keys, the superseded keys are kept for another seven days in order to decrypt and verify (but not to issue!) tickets which were issued by the superseded keys. As a result, tickets are always valid and redeemable for a period of *exactly seven days*; no matter when they were issued. As a result, as long as a user keeps reconnecting to a ScrambleSuit server at least once a week, *key continuity* is ensured and there is no need for additional out-of-band communication.

Foiling replay: At this point, a censor could still intercept a client's ticket and replay it. This would make the server issue a new ticket for the censor. While the censor would not be able to read the resulting ScrambleSuit control message—the shared master key k_t would be unknown—it is sound design to prevent communication without prior authentication.

We prevent replay attacks—or in other words: ticket double spending—by caching the master key k_t embedded in a ticket. If a server encounters an already cached k_t, it does not reply. We begin to cache a key k_t after a new session ticket was issued and the client confirmed that she correctly received the new ticket by using a special ScrambleSuit message type (see §4.2). We introduced the confirmation step because otherwise a censor could *preplay* a ticket, i.e., intercept it and send it *before* the client. That way, the server would add it to the replay table and would then reject the client's ticket because it would appear to be replayed. This would allow the censor to invalidate the tickets of clients. With the confirmation step, however, censors are no longer

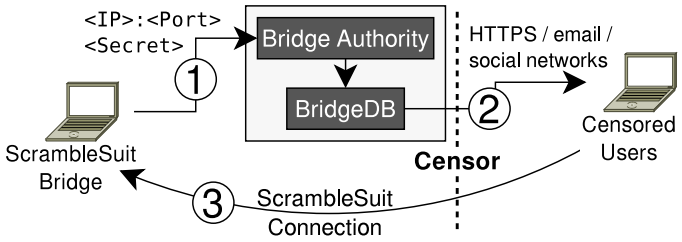


Figure 5: ScrambleSuit bridges send their descriptor to the Tor project's bridge authority ①. From there, it is distributed to censored users who learn about IP address, port and the secret *out-of-band* ②. Finally, direct connections can be established ③.

able to launch preplay attacks.

Inadequate for Tor: Session tickets provide a strong level of protection. Active probing and replay attacks are foiled while forward secrecy is provided. As a result, we envision session tickets to be satisfactory for most application protocols which do not have these properties. Session tickets alone do not, however, integrate well with Tor's existing ecosystem. The reason lies in how Tor bridges are distributed to users. The process is illustrated in Figure 5. Volunteers will set up ScrambleSuit bridges which then publish their descriptors—most notably IP address, port and shared secret—to the bridge authority which then feeds this information into the BridgeDB component ①. In the subsequent step, the gathered descriptors are distributed to censored users ②. The two primary distribution channels are email and HTTPS [41]: users can ask for bridges over email or they can visit the bridge distribution website[22] and obtain a set of bridges after solving a CAPTCHA. Finally, users can establish ScrambleSuit connections ③.

The problem is that *one bridge descriptor* is typically shared by *many users*. All these users would end up with an identical session ticket. This causes two severe problems. First, our replay protection mechanism does not allow reuse of session tickets. Only the first user would be able to authenticate herself. Second, session ticket reuse would lead to identical byte strings at the beginning of a ScrambleSuit handshake which would be a strong distinguisher. These problems lead us to an *additional authentication mechanism*, discussed in §4.1.4, which is optimised for Tor and can function with a secret which is shared by many users as shown in the scenario in Figure 5.

4.1.3 Tor-specific Session Tickets

If ScrambleSuit is used to tunnel Tor traffic (as opposed to other application protocols such as a VPN), a slightly modified session ticket handshake is used. As illustrated in Figure 6, the only two differences are:

1. The ticket confirmation message is *no longer necessary*.

[22] URL: https://bridges.torproject.org.

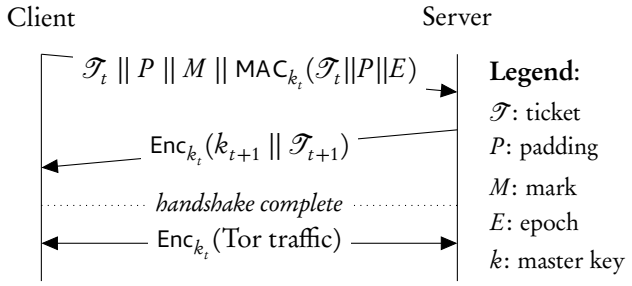


Figure 6: The Tor-specific session ticket handshake. In contrast to Figure 4, it *1)* has no ticket confirmation message and it *2)* employs a timestamp E.

2. The MAC in the first message *contains the variable* E holding a timestamp.

Recall that the purpose of the ticket confirmation message is to foil preplay attacks. When Tor is transported by ScrambleSuit, preplay attacks are no longer problematic because if a ticket is lost, the alternative authentication mechanism discussed in §4.1.4 is used instead. This possibility to "fall back" means that a lost ticket no longer implies the inability to authenticate as it does with the classical ticket scheme.

The variable E holds the current Unix timestamp divided by 3600, i.e., the number of hours which have passed since the epoch. Its purpose is to reduce the amount of keys in the replay cache. While this requires clients and servers to have loosely synchronised clocks, the server has to cache redeemed keys for a period of only one hour rather than seven days. We could not use E in the classical ticket scheme illustrated in Figure 4 because it would enable an attack. A censor could repeatedly terminate connections after a client tried to redeem its session ticket. After a while, the variable E will change since it holds a timestamp. This means that the MAC over the ticket handshake would change whereas the ticket *would not change*. We consider this a strong distinguisher.

4.1.4 Authentication Using Uniform Diffie-Hellman

Our second authentication mechanism is an extension of the Uniform Diffie-Hellman (UniformDH) handshake which was proposed in the obfs3 protocol specification [21, §3]. obfs3's handshake makes use of uniformly distributed public keys which are only negligibly different from random bytes (see Appendix A). As a result, UniformDH can be used to agree on a master key k_t without a censor knowing that Diffie-Hellman is used.

In contrast to obfs3, our version of UniformDH is based on the 4096-bit modular exponential group number 16 defined in RFC 3526 [42]. When initiating a UniformDH handshake, the client first generates a 4096-bit private key x. The least significant bit of x is then unset in order to make the num-

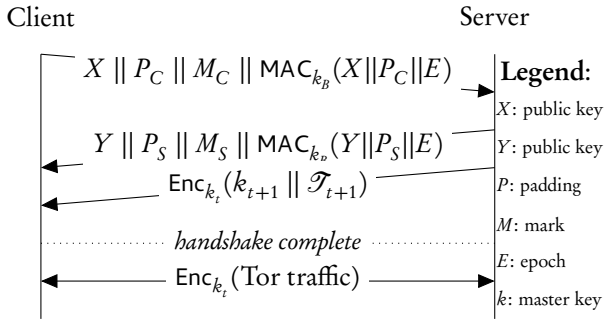


Figure 7: After client and server agreed on the master key k_t using Uniform Diffie-Hellman, the server is issuing a new session ticket for the client. Afterwards, both parties exchange Tor traffic.

ber even. The public key X is defined as $X = g^x \pmod p$ where $g = 2$. The server computes its private key y and its public key Y the same way. To prevent a censor from learning that X is a quadratic residue mod p—a clear distinguisher—the client randomly chooses to send either X or $p - X$ to the server. The server can then derive the shared master key by calculating $k_t = X^y \pmod p$. Since the private keys x and y are even, the exponentiations $X^y \pmod p$ and $(p - X)^y \pmod p$ result in the same shared master key.

4.1.5 Extending Uniform Diffie-Hellman

In its original form, the UniformDH construction does not protect against active probing. A censor who suspects UniformDH can simply probe the supposable bridge and opportunistically initiate a UniformDH handshake. To prevent that attack, we now turn UniformDH's anonymous handshake into an authenticated handshake in order to be resistant against active attacks.

We do so quite similar to the session tickets discussed in §4.1.2. As depicted in Figure 7, we concatenate pseudo-random padding P, the mark M, and a MAC. The MAC authenticates the respective public key as well as the padding. The MAC is keyed by a shared secret k_B which is distributed together with the Tor bridge's IP:port tuple over email or HTTPS (see step ① in Figure 5). Similar to tickets, the client's mark $M_C = \mathsf{MAC}_{k_B}(X)$ is used to easily locate the MAC. Note that k_B *can be reused* because it is only used to key the MAC. The handshake is conducted using UniformDH with randomly chosen public keys. As a result, two subsequent UniformDH handshakes based on the same k_B will appear different to a censor. We defend against replay attacks by adding E, the Unix epoch divided by 3600, to the MAC and cache the MAC for a period of one hour.

A successful UniformDH key agreement is followed by the server issuing a session ticket for the client. The client will then redeem this ticket upon

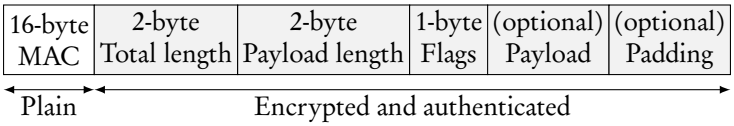


Figure 8: ScrambleSuit's message header format. The entire message is computationally indistinguishable from randomness.

connecting to the server the next time. Accordingly, we expect the UniformDH handshake to be done *only once*: namely when a Tor client connects to a bridge for the first time. From then on, session tickets as presented in §4.1.3 will be used for authentication.

To a censor, the payload of both authentication schemes is computationally indistinguishable from randomness. Furthermore, both schemes employ a two-way handshake and padding is used for the two handshakes to exhibit the same average length. As a result, a censor who is assuming that a server is running ScrambleSuit is unable to tell whether a client successfully authenticated herself by using UniformDH or by redeeming a session ticket.

We finally stress that bootstrapping ScrambleSuit using UniformDH provides *less security* than when bootstrapped using session tickets. Since the secret key k_B for UniformDH will be used by multiple clients, a malicious client in the possession of k_B who is able to eavesdrop on the connection of another client using the same ScrambleSuit server can conduct active MitM attacks. While Tor does protect against active MitM attacks[23], this can be problematic for application protocols other than Tor. Therefore, we emphasise that session tickets are the preferred authentication mechanism for insecure protocols whereas our UniformDH extension's sole purpose is to make ScrambleSuit work well in Tor's infrastructure. Finally, we discuss usability considerations of our authentication mechanisms in Appendix C.

4.2 Header Format and Confidentiality

Our protocol employs a custom message format whose header is illustrated in Figure 8. In a nutshell, ScrambleSuit exchanges variable-sized messages with optional padding. The padding is always discarded by the remote machine.

The first 16 bytes of the header are reserved for an HMAC-SHA256-128 which protects the integrity and authenticity of the protocol message. In accordance with the encrypt-then-MAC principle, the HMAC is computed over the encrypted remainder of the message. The secret key required by the HMAC is derived from the shared master key k_t.

The HMAC is followed by two bytes which specify the total length of the protocol message. ScrambleSuit's maximum transmission unit is 1448-byte-sized messages. Together with an IP and TCP header (which includes

[23] A Tor client contains hard-coded keys of the directory authorities which then sign the network consensus.

Table 1: ScrambleSuit's protocol message flags.

Flag name	Bit #	Meaning
FLAG_PAYLOAD	1	Application data.
FLAG_NEW_TICKET	2	Newly issued session ticket and master key.
FLAG_ACK_TICKET	3	Acknowledgement of receipt of the session ticket.
FLAG_PRNG_SEED	4	PRNG seed to reproduce probability distributions.
FLAG_REKEY	5	Initiate rekeying before AES' counter wraps.

the timestamping option), this adds up to 1500-byte packets which fill an Ethernet frame. In order to be able to distinguish padding from payload, the next two bytes determine the payload length. If no padding is used, the payload length equals the total length.

To separate application data from protocol signaling, we define a 1-byte message flag field. The semantics of all five flags is explained in Table 1. The first bit signals application data in the message body whereas a message with the second bit set contains a newly issued session ticket. The third bit (which can be set together with the first bit) confirms the receipt of a session ticket. Bit number four allows the server to send a pseudo-random number generator (PRNG) seed to the client which is used for our traffic analysis defence (see §4.3). Bit five initiates rekeying which happens before the counter used for AES overflows. To prevent entropy exhaustion attacks, rekeying can only be triggered by the server. We reserve the remaining three bits for future use.

The header is then followed by the message payload which contains the application protocol transported by ScrambleSuit. We employ encryption in order to hide the application protocol, the padding as well as ScrambleSuit's header. With regard to Tor, this means that the already encrypted Tor traffic is wrapped inside yet another layer of encryption. For encryption, we use 256-bit AES in counter mode. The counter mode effectively turns AES into a stream cipher. We use two symmetric keys: one for the traffic $C \rightarrow S$ and one for $S \rightarrow C$. Both symmetric keys as well as the respective nonces for the counter mode are derived from the shared 256-bit master key using HKDF based on SHA256 [43].

4.3 Polymorphic Shape

So far, we discussed defences against censors aiming to analyse packet payload or conduct active attacks to reveal ScrambleSuit's presence. However, a censor could make use of *traffic analysis*, i.e., analyse communication aspects other than the payload. In this section, we propose lightweight countermeasures to

diminish—but not to defeat!—such attacks. In particular, we will teach every ScrambleSuit server to generate its own and unique "protocol shape"[24].

Our definition of ScrambleSuit's shape is twofold: we consider *packet lengths* and *inter-arrival times*. While encrypting our protocol messages renders payload analysis useless, these two flow metrics still leak information about the transported application [44, 45, 46]. As a result, we seek to disguise these characteristics in order to decrease the accuracy of protocol classifiers trained to identify our protocol. Our approach to this problem is *protocol polymorphism*.

We achieve polymorphism by creating *one protocol shape for every server*. When a ScrambleSuit server bootstraps for the first time, it randomly generates a 256-bit seed. This seed is then fed into a PRNG which is used to obtain two discrete probability distributions. These two distributions dictate the desired shape of packet lengths and inter-arrival times. Furthermore, a server communicates its unique PRNG seed to clients (see Table 1) after successful authentication. Since the seed is shared by both parties, they can generate identical probability distributions and thus shape their traffic the same way. A censor monitoring two distinct ScrambleSuit servers will observe different distributions for packet lengths and inter-arrival times.

Once our PRNG is seeded, we generate the two distributions by first determining the amount of bins n which is uniformly chosen from the set $\{1..100\}$. In the next step, we assign each bin b_i for $1 \leq i \leq n$ a probability by randomly picking a value in the interval $]0, 1 - \sum_{j=0}^{i-1} b_j[$ with $b_0 = 0$. The following gives an example for four bins.

$$b_0 \leftarrow 0 \tag{1}$$

$$b_1 \xleftarrow{R}]0, 1 - b_0[\tag{2}$$

$$b_2 \xleftarrow{R}]0, 1 - b_0 - b_1[\tag{3}$$

$$b_n \xleftarrow{R}]0, 1 - b_0 - \ldots - b_{n-1}[\tag{4}$$

4.3.1 Packet Length Adaption

It is well known that a network flow's packet length distribution leaks information about the network protocol [28, 44] and even the content [47, 46]. For instance, a large fraction of Tor's traffic is composed of 568-byte packets which is the result of Tor's internal use of 512-byte cells plus TLS' header (see Figure 10). These 568-byte packets form a strong distinguisher which can be used to spot a Tor flow by simply capturing a few dozen network packets as shown by Weinberg et al. [16]. To defend against such simple applications of traffic analysis, we modify the packet length distribution of our transported application.

Typically, non-interactive TCP applications transmit segments filling the network link's maximum transmission unit (MTU) as long as they have

[24] This happens similar to the *scramble suits* in Philip K. Dick's novel "A Scanner Darkly".

enough to "say". Applications will only send packets smaller than the MTU if there is not enough data in the send buffer. Due to their sheer volume[25], we deem MTU-sized packets to be of no interest to censors. What we aim to disguise is only packets *smaller than the MTU*. This is done by randomly sampling a packet length from the probability distribution over all our packet lengths. The original packet length is then padded to fit the sampled packet length. The padding can be anything in between 0 and 1520 bytes. The reason for 1520 instead of 1499 bytes is that the smallest unit we can transmit is an empty ScrambleSuit message: 21 bytes.

4.3.2 Inter-Arrival Time Adaption

Analogous to packet lengths, the distribution of inter-arrival times between consecutive packets has discriminative power and can be used by censors to identify protocols [49]. While inter-arrival times are frequently distorted by jitter, overloaded middle boxes and the communicating end points, it would be no sound strategy to assume the network to be unreliable enough to render measurements difficult.

We employ an obfuscation mechanism analogous to the packet length adaption discussed earlier: first, the shared PRNG seed is used to generate a pseudo-random probability distribution. After the amount of bins is determined, each bin is assigned a probability in the range of $[0, 10[$ milliseconds. The motivation for this choice is explained in Appendix B. Random samples are then drawn from this distribution which are used to artificially delay network packets. Similar to the implementation of SkypeMorph [17], we make use of a dedicated send buffer which is processed independently of the locally incoming data. This decoupling of the incoming and outgoing data stream makes it possible to "reshape" inter-arrival times.

4.3.3 Shortcomings

It is important to note that for a censor armed with a well-chosen set of features and sufficient computational resources, traffic analysis can be a powerful attack. Robust defences, on the other hand, are believed to be expensive [45]. For instance, Dyer et al.'s simple yet powerful VNG++ classifier [45] only makes use of coarse features such as connection duration, total bytes transferred and the "burstiness" of a flow. Significantly decreasing VNG++'s accuracy would cause a drastically increased protocol overhead.

Nevertheless, traffic analysis does not give censors a *certain answer*. False positives are always a problem and can lead to over-blocking. As mentioned in our threat model, we believe that the censor might use traffic analysis to select a subset of traffic for closer inspection but not to block flows.

[25] According to a study conducted by CAIDA [48], MTU-sized packets form a significant fraction of all observed packet lengths.

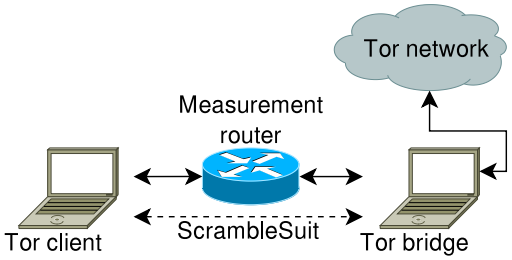


Figure 9: Our experimental setup used to measure ScrambleSuit's obfuscation and performance.

4.4 Cryptographic Assumptions

Our authentication mechanisms rely on *1)* AES-CBC, *2)* pseudo-random initialisation vectors, *3)* HMAC, *4)* pseudo-random padding and *5)* uniformly distributed Diffie-Hellman public keys. Exchanged application data is then encrypted using AES-CTR and authenticated by an HMAC. We expect all data exchanged between ScrambleSuit servers and clients to be computationally indistinguishable[26] from random data of the same length. As a result, we have the following assumptions regarding our cryptographic building blocks. We assume AES to be a pseudo-random permutation and our HMAC to be a pseudo-random function. The padding is assumed to come from a cryptographically secure PRNG and the public keys are assumed to be uniformly distributed (see obfs3 [21]).

5 Experimental Evaluation

We implemented a fully functional prototype of ScrambleSuit in the form of several Python modules for obfsproxy[27]. Our prototype consists of approximately 2,200 lines of code. We used the library PyCrypto [51] for cryptographic primitives. The measurements discussed below were all conducted using this prototype.

As illustrated in Figure 9, our experimental setup consisted of two Debian GNU/Linux machines which were connected over a router performing the measurements. All three machines were connected over 100 Mbit/s Ethernet. We expect this setup to be ideal for a censor because it minimises network interference such as jitter or packet fragmentation. As a result, we believe that a censor would do worse in practice. Both of our machines were running Tor v0.2.4.15-rc and obfsproxy. The Tor bridge was configured to remain private and only used by our client. The bridge then relayed all traffic into the public

[26] Our goal is to achieve a security level equivalent to at least 128 bits of symmetric security as defined in [50].

[27] The code is available under a free license at http://veri.nymity.ch/scramblesuit/.

Tor network. Note that ScrambleSuit was only "spoken" in between the client and the bridge.

5.1 Blocking Resistance

It is difficult to evaluate the effectiveness of our obfuscation techniques since ScrambleSuit does not have a cover protocol to mimic. Otherwise, our evaluation would simply investigate the similarity between our protocol and its cover protocol. Instead of measuring ScrambleSuit's closeness to a mimicked protocol, we measure the *deviation* from its *transported application*, i.e., Tor. Intuitively, higher deviation would imply better obfuscation.

We obtained traces of packet lengths and inter-arrival times for ScrambleSuit and Tor. In the following, we qualitatively compare both traces. To create network traffic for these traces, we repeatedly downloaded the 1 MB Linux kernel v1.0 from kernel.org[28] on the client. We downloaded the file ten times over Tor and ScrambleSuit, respectively. The measurements only covered the download and not ScrambleSuit's authentication or Tor's bootstrapping.

The two packet length distributions are illustrated in Figure 10(a) and 10(b). The dark blue lines represent Tor flows whereas bright orange lines depict ScrambleSuit. The packet length distributions clearly show the prevalence of Tor's 586-byte packets; even more so in client-to-server traffic where the purpose of these packets is flow control. While server-to-client traffic carries less 586-byte packets, they still form a significant fraction of overall packet lengths. ScrambleSuit effectively eliminates this traffic signature and transmits more MTU-sized packets. Recall that every ScrambleSuit download represents only *one specific shape*. Different servers exhibit different shapes.

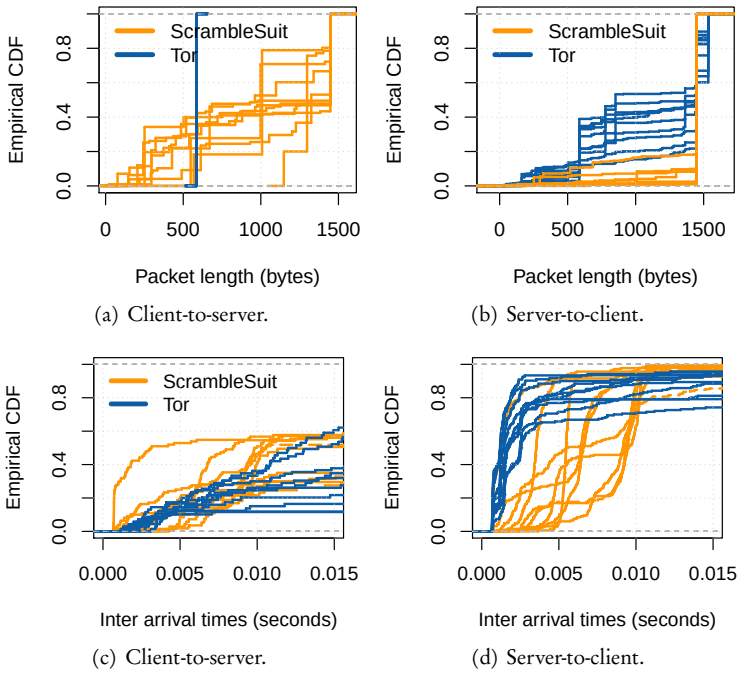
Figure 10(c) and 10(d) depict the inter-arrival times derived from the same data. Again, Tor is shown as dark blue and ScrambleSuit as bright orange lines. The inter-arrival times in Figure 10(c) tend to be rather high—only roughly 60% of Tor packets had an inter-arrival delay below 15 ms—because the bulk data was travelling from the server to the client. For this reason, the delays are significantly smaller in Figure 10(d).

Once again, ScrambleSuit visibly deviates from Tor's distribution. However, as we will show in §5.2.2, artificially increased inter-arrival times have a negative effect on throughput.

5.2 Performance

We are interested in both *computational* as well as *network* overhead. The following sections discuss the amount of overhead ScrambleSuit carries compared to a bare Tor connection.

[28] URL: https://www.kernel.org/pub/linux/kernel/v1.0/linux-1.0.tar.bz2.



(a) Client-to-server. (b) Server-to-client.

(c) Client-to-server. (d) Server-to-client.

Figure 10: Tor's and ScrambleSuit's packet length distribution and inter-arrival times for both, client-to-server and server-to-client traffic.

Table 2: Mean (μ) and standard deviation (σ) of the goodput, transferred KBytes and the total overhead. The data was generated based on the download of a 1,000,000-byte file.

	HTTP		**Tor**	
	μ	σ	μ	σ
Goodput	6.3 MB/s	3.4 MB/s	286 KB/s	227 KB/s
C→S KBytes	23.1	1.6	66.4	6.5
S→C KBytes	1047	20.7	1130	12.8
Total overhead	7%	2.2%	19.6%	1.9%
	ScrambleSuit		**ScrambleSuit-nodelay**	
	μ	σ	μ	σ
Goodput	148 KB/s	61 KB/s	321 KB/s	231 KB/s
C→S KBytes	122.9	24.1	111.9	17.9
S→C KBytes	1397.8	125.7	1342.7	112.2
Total overhead	52.1%	15%	45.5%	13%

5.2.1 Cryptographic Overhead

Our two authentication mechanisms come with small computational overhead. For both parties, UniformDH requires two modular exponentiations and two MAC generations[29]. Session tickets—which constitute the majority of authentications—are even cheaper: the client again calculates two MACs whereas the server symmetrically decrypts the ticket and verifies two MACs.

After authentication, protocol messages are symmetrically encrypted and protected by a MAC. We expect the low computational overhead to make ScrambleSuit suitable for resource-constrained devices such as smartphones.

5.2.2 Network Overhead

Our protocol's network overhead is increased by the artificial inter-arrival times, packet padding and the protocol header. In order to gain a good understanding of the exact network overhead, we created a 1,000,000-byte file containing random bytes and placed it on a web server operated by Karlstad University. We then downloaded this file using `wget`; 25 times over HTTP, Tor, ScrambleSuit and ScrambleSuit without inter-arrival time obfuscation, respectively. For Tor and ScrambleSuit, we established a new circuit for every download but we used the same entry guard to eliminate unnecessary variance. All data was captured after a Tor circuit has been established, so

[29] It is necessary to calculate the authenticating MAC as well as the mark used to locate the MAC.

handshakes are not part of the data. We then calculated the mean μ and the standard deviation σ for several performance metrics. The results are depicted in Table 2.

The goodput refers to the application layer throughput. We achieved very high values for the HTTP download because the file transfer could be carried out over the LAN. Tor averaged at roughly 280 KB/s and ScrambleSuit achieved slightly more than half of that. Just like Tor, ScrambleSuit exhibits high standard deviation. The main reason for this is differences in Tor circuit throughput. ScrambleSuit without inter-arrival time obfuscation is comparable to Tor. In fact, it exceeds Tor's throughput but this can again be explained by varying circuit throughput. This shows that the obfuscation of inter-arrival times has the biggest impact on ScrambleSuit's throughput (see also Appendix B).

The next two rows of Table 2 refer to the transferred KBytes from client to server (C→S) and server to client (S→C). Note that these metrics cover all the data which was present on the wire; including IP and TCP header. The consideration of IP and TCP overhead is important because ScrambleSuit's packet padding mechanism introduces additional TCP/IP packets. Unsurprisingly, Tor transferred more data than HTTP because of Tor's and TLS' protocol overhead. ScrambleSuit transferred the most data because of the additional protocol header as well as the varying packet lengths. Given ScrambleSuit's 1448-byte MTU, the 21-byte protocol header only accounts for 1.5% overhead.

The last row in Table 2 illustrates the total protocol overhead which is simply a function of the previous two rows. HTTP has the lowest overhead followed by Tor and finally ScrambleSuit. Our protocol exhibits 45–50% overhead which is about twice as much as Tor.

6 Discussion

Active Probing: A censor could still actively probe a ScrambleSuit server. Upon establishing a TCP connection, a censor could proceed by sending arbitrary data. However, without knowing the UniformDH shared secret or possessing a valid session ticket, authentication can not succeed and the server will remain silent.

In contrast to SilentKnock and BridgeSPA, ScrambleSuit does not disguise its "aliveness". While this approach does leak information[30], it has the benefit of making ScrambleSuit significantly easier to deploy due to lack of platform dependencies such as the kernel interface `libnetfilter_queue` to parse raw network packets in userspace.

Injection, Modification, Dropping: A censor could tamper with an established ScrambleSuit connection by injecting, modifying or dropping packets. After authentication, all exchanged data is authenticated which allows

[30] A censor learns that a server is online but unwilling to talk unless given the "correct" data.

the communicating parties to detect such tampering. If the authenticating HMAC of either party is invalid, the connection is terminated immediately.

Hijacking a ScrambleSuit connection is reduced to the same problem; a censor would bypass authentication but is unable to talk to the other party because the session keys are unknown. Finally, dropped packets would be handled by TCP's retransmission mechanism whereas terminated connections could manually be restarted by users.

Payload Analysis: Payload analysis would only yield data which is computationally indistinguishable from randomness. While most encrypted protocols negotiate session parameters in cleartext, VPNs with pre-shared keys and BitTorrent's message stream encryption also bootstrap using high-entropy network packets. Aside from that, it is difficult to survey how many "fully-random" protocols already exist in the wild. One approach would be to monitor large-scale network links and determine which fraction of TCP streams transports only high-entropy data. Similar work was done by White et al. [52] but the authors focused on per-packet rather than on per-connection measurements.

Ideally, more applications considered legitimate by censors would start transporting only high-entropy payload, thus inflating the set of protocols ScrambleSuit can hide amongst. This could be achieved by a major browser employing such encryption on the transport layer or even by an extension for the TLS protocol which would provide optional support for such a mode.

Flow Analysis: Flow analysis would yield a unique distribution of packet lengths and inter-arrival times which is different for every ScrambleSuit server. While strong traffic analysis defence is expensive, these attacks will always have a range of *uncertainty* causing false positives. Our goal was to further increase this uncertainty. We placed more value on defeating active probing attacks because in contrast to traffic analysis, they enable *deterministic* protocol identification.

Future Work: There is room for stronger traffic analysis defence. In particular, ScrambleSuit could be extended to keep track of the "burstiness" of exchanged traffic which is—amongst other features—exploited by the powerful VNG++ classifier. These bursts could then be deliberately distorted with the intention to confuse VNG++. Our shared PRNG seed could be used to make this distortion server-specific.

7 Conclusion

We presented ScrambleSuit; a lightweight transport protocol which provides obfuscation for applications such as Tor. The two major contributions of our protocol are the ability to defend against *active probing* and simple *protocol classifiers*. We achieve the former by proposing two authentication mechanisms—one general-purpose and the other specifically for Tor—and the latter by proposing morphing techniques to disguise packet lengths and inter-arrival times.

We further developed a prototype of ScrambleSuit and used it to conduct

an experimental evaluation. We discussed the effectiveness of our obfuscation techniques as well as ScrambleSuit's overhead. Our evaluation suggests that our protocol can provide decent protection against censors who do not overblock significantly. As a result, we believe that our protocol can provide a practical alternative in countries which do not whitelist Internet traffic.

In the near future, we aim to deploy ScrambleSuit as part of the Tor browser bundle.

Acknowledgements

We want to thank the anonymous reviewers, George Kadianakis, Harald Lampesberger, Stefan Lindskog, and Michael Rogers who all provided valuable feedback which improved this paper. We further want to express our gratitude to Internetfonden of the Swedish Internet Infrastructure Foundation for supporting the main author's work with a research grant.

All our data and code is freely available at:
http://veri.nymity.ch/scramblesuit/.

References

[1] Thomas H. Ptacek and Timothy N. Newsham. *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection*. Tech. rep. Secure Networks, Inc., 1998. URL: http://cs.unc.edu/~fabian/course_papers/PtacekNewsham98.pdf.

[2] Olli-Pekka Niemi, Antti Levomäki, and Jukka Manner. "Dismantling Intrusion Prevention Systems (Demo)". In: *SIGCOMM*. ACM, 2012. URL: http://conferences.sigcomm.org/sigcomm/2012/paper/sigcomm/p285.pdf.

[3] Mark Handley, Vern Paxson, and Christian Kreibich. "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics". In: *USENIX Security*. USENIX Association, 2001. URL: http://static.usenix.org/events/sec01/full_papers/handley/handley.pdf.

[4] Marcel Dischinger et al. "Detecting BitTorrent Blocking". In: *IMC*. ACM, 2008. URL: http://www.mpi-sws.org/~mdischin/papers/08_imc_blocking.pdf.

[5] Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson. "Ignoring the Great Firewall of China". In: *PETS*. Springer, 2006. URL: http://www.cl.cam.ac.uk/~rnc1/ignoring.pdf.

[6] Sparks et al. "The Collateral Damage of Internet Censorship by DNS Injection". In: *SIGCOMM Computer Communication Review* 42.3 (2012). URL: http://conferences.sigcomm.org/sigcomm/2012/paper/ccr-paper266.pdf.

[7] Christopher Rhoads and Loretta Chao. *Iran's Web Spying Aided By Western Technology*. 2009. URL: http://online.wsj.com/article/SB124562668777335653.html.

[8] Jillian C. York. *Government Internet Surveillance Starts With Eyes Built in the West*. 2011. URL: https://www.eff.org/deeplinks/2011/09/government-internet-surveillance-starts-eyes-built.

[9] Roger Dingledine, Nick Mathewson, and Paul Syverson. "Tor: The Second-Generation Onion Router". In: *USENIX Security*. USENIX Association, 2004. URL: http://static.usenix.org/event/sec04/tech/full_papers/dingledine/dingledine.pdf.

[10] The Tor Project. *Iran*. URL: https://censorshipwiki.torproject.org/CensorshipByCountry/Iran.

[11] Philipp Winter and Stefan Lindskog. "How the Great Firewall of China is Blocking Tor". In: *FOCI*. USENIX Association, 2012. URL: https://www.usenix.org/system/files/conference/foci12/foci12-final2.pdf.

[12] The Tor Project. *Ethiopia*. URL: https://censorshipwiki.torproject.org/CensorshipByCountry/Ethiopia.

[13] Charles Arthur. *China tightens 'Great Firewall' internet control with new technology*. 2012. URL: http://www.guardian.co.uk/technology/2012/dec/14/china-tightens-great-firewall-internet-control.

[14] *GFW actively probes obfs2 bridges*. 2013. URL: https://bugs.torproject.org/8591.

[15] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. "ZMap: Fast Internet-Wide Scanning and its Security Applications". In: *USENIX Security*. USENIX Association, 2013. URL: https://www.usenix.org/system/files/conference/usenixsecurity13/sec13-paper_zhu.pdf.

[16] Zachary Weinberg et al. "StegoTorus: A Camouflage Proxy for the Tor Anonymity System". In: *CCS*. ACM, 2012. URL: http://www.owlfolio.org/media/2010/05/stegotorus.pdf.

[17] Hooman Mohajeri Moghaddam et al. "SkypeMorph: Protocol Obfuscation for Tor Bridges". In: *CCS*. ACM, 2012. URL: http://www.cypherpunks.ca/~iang/pubs/skypemorph-ccs.pdf.

[18] Amir Houmansadr et al. "I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention". In: *NDSS*. The Internet Society, 2013. URL: http://www.cs.utexas.edu/~amir/papers/FreeWave.pdf.

[19] Wenxuan Zhou et al. "SWEET: Serving the Web by Exploiting Email Tunnels". In: *HotPETS*. Springer, 2013. URL: http://petsymposium.org/2013/papers/zhou-censorship.pdf.

[20] The Tor Project. *obfs2 (The Twobfuscator)*. URL: https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/blob/HEAD:/doc/obfs2/obfs2-protocol-spec.txt.

[21] The Tor Project. *obfs3 (The Threebfuscator)*. URL: https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/blob/HEAD:/doc/obfs3/obfs3-protocol-spec.txt.

[22] Brandon Wiley. *Dust: A Blocking-Resistant Internet Transport Protocol*. Tech. rep. University of Texas at Austin, 2011. URL: http://blanu.net/Dust.pdf.

[23] *Viewing cable 09MUSCAT1039, SKYPE CRACKDOWN IN OMAN*. 2009. URL: http://wikileaks.org/cable/2009/11/09MUSCAT1039.html.

[24] *Russian "Clean Internet" experiment gets green light*. 2013. URL: http://rt.com/politics/anti-pedophile-safe-internet-russian-169/.

[25] Small Media. *Iranian Internet Infrastructure and Policy Report: Election Edition 2013 (April - June)*. 2013. URL: http://smallmedia.org.uk/IIIPJune.pdf.

[26] Alberto Dainotti et al. "Analysis of Country-wide Internet Outages Caused by Censorship". In: *IMC*. ACM, 2011. URL: http://www.caida.org/publications/papers/2011/outages_censorship/outages_censorship.pdf.

[27] Eva Galperin and Jillian C. York. *Syria Goes Dark*. 2012. URL: https://www.eff.org/deeplinks/2012/11/syria-goes-dark.

[28] Erik Hjelmvik and Wolfgang John. *Breaking and Improving Protocol Obfuscation*. Tech. rep. Chalmers University of Technology, 2010. URL: http://www.iis.se/docs/hjelmvik_breaking.pdf.

[29] Brandon Wiley. *Blocking-Resistant Protocol Classification Using Bayesian Model Selection*. Tech. rep. University of Texas at Austin, 2011. URL: http://blanu.net/BayesianClassification.pdf.

[30] The Tor Project. *obfsproxy*. URL: https://www.torproject.org/projects/obfsproxy.

[31] Qiyan Wang et al. "CensorSpoofer: Asymmetric Communication using IP Spoofing for Censorship-Resistant Web Browsing". In: *CCS*. ACM, 2012.

[32] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. "The Parrot is Dead: Observing Unobservable Network Communications". In: *Security & Privacy*. IEEE, 2013. URL: http://www.cs.utexas.edu/~amir/papers/parrot.pdf.

[33] Patrick Lincoln et al. "Bootstrapping Communications into an Anti-Censorship System". In: *FOCI*. USENIX Association, 2012. URL: https://www.usenix.org/system/files/conference/foci12/foci12-final7.pdf.

[34] Eugene Y. Vasserman et al. "SilentKnock: Practical, Provably Undetectable Authentication". In: *ESORICS*. Springer, 2007. URL: http://www-users.cs.umn.edu/~hopper/silentknock_esorics.pdf.

[35] Rob Smits et al. "BridgeSPA: Improving Tor Bridges with Single Packet Authorization". In: *WPES*. ACM, 2011. URL: http://www.cypherpunks.ca/~iang/pubs/bridgespa-wpes.pdf.

[36] Martin Johnson. *China, GitHub and the man-in-the-middle*. 2013. URL: https://en.greatfire.org/blog/2013/jan/china-github-and-man-middle.

[37] Ronald L. Rivest, Adi Shamir, and David A. Wagner. *Time-lock Puzzles and Timed-release Crypto*. Tech. rep. Massachusetts Institute of Technology, 1996. URL: http://people.csail.mit.edu/rivest/RivestShamirWagner-timelock.ps.

[38] Ben Laurie and Richard Clayton. ""Proof-of-Work" Proves Not to Work". In: *WEIS*. 2004. URL: http://www.cl.cam.ac.uk/~rnc1/proofwork2.pdf.

[39] Joseph Salowey et al. *RFC 5077: Transport Layer Security (TLS) Session Resumption without Server-Side State*. 2008. URL: https://tools.ietf.org/html/rfc5077.

[40] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. *RFC 2104: HMAC: Keyed-Hashing for Message Authentication*. 1997. URL: https://www.ietf.org/rfc/rfc2104.txt.

[41] Zhen Ling et al. "Extensive Analysis and Large-Scale Empirical Evaluation of Tor Bridge Discovery". In: *INFOCOM*. IEEE, 2012. URL: http://www.cs.uml.edu/~xinwenfu/paper/Bridge.pdf.

[42] Tero Kivinen and Mika Kojo. *RFC 3526: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*. 2003. URL: http://tools.ietf.org/html/rfc3526.

[43] Hugo Krawczyk and Pasi Eronen. *RFC 5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. 2010. URL: https://tools.ietf.org/html/rfc5869.

[44] Manuel Crotti et al. "Traffic Classification through Simple Statistical Fingerprinting". In: *SIGCOMM Computer Communication Review* 37.1 (2007). URL: http://www.sigcomm.org/sites/default/files/ccr/papers/2007/January/1198255-1198257.pdf.

[45] Kevin P. Dyer et al. "Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail". In: *Security & Privacy*. IEEE, 2012. URL: http://kpdyer.com/publications/oakland2012-peekaboo.pdf.

[46] Xiang Cai et al. "Touching from a Distance: Website Fingerprinting Attacks and Defenses". In: *CCS*. ACM, 2012. URL: http://www.cs.sunysb.edu/~xcai/fp.pdf.

[47] Andriy Panchenko et al. "Website Fingerprinting in Onion Routing Based Anonymization Networks". In: *WPES*. ACM, 2011. URL: http://lorre.uni.lu/~andriy/papers/acmccs-wpes11-fingerprinting.pdf.

[48] CAIDA. *Packet size distribution comparison between Internet links in 1998 and 2008*. 2010. URL: http://www.caida.org/research/traffic-analysis/pkt_size_distribution/graphs.xml.

[49] Mohamad Jaber, Roberto G. Cascella, and Chadi Barakat. "Can we trust the inter-packet time for traffic classification?" In: *ICC*. IEEE, 2011. URL: http://www-sop.inria.fr/members/Chadi.Barakat/ICC2011.pdf.

[50] *ECRYPT II Yearly Report on Algorithms and Keysizes*. 2012. URL: http://www.ecrypt.eu.org/documents/D.SPA.20.pdf.

[51] Dwayne C. Litzenberger. *PyCrypto - The Python Cryptography Toolkit*. URL: https://www.dlitz.net/software/pycrypto/.

[52] Andrew M. White et al. "Clear and Present Data: Opaque Traffic and its Security Implications for the Future". In: *NDSS*. The Internet Society, 2013. URL: http://cs.unc.edu/~amw/resources/opaque.pdf.

A UniformDH Public Keys

UniformDH public keys form a distinguisher because they are not distributed over the entire space of 4096 bits. Public keys are always smaller than the modulus defined in RFC 3526 [42]. The unused bit space is, however, small enough for this distinguisher to be negligible. The probability P of a censor observing a UniformDH public key smaller than the 4096-bit modulus n (see reference [42] for n's value) equals:

$$P = \frac{n}{2^{4096} - 1}$$

A censor could now monitor a server's handshakes over an extended period of time in order to determine if the full 4096-bit space is never used which would be a sign of UniformDH. Doing that, a censor needs

$$\frac{ln(0.5)}{ln(P)} \approx 2^{66}$$

observations to have a confidence greater than 50% that a server is conducting UniformDH handshakes. Such an amount of observations is unpractical.

B Inter-Arrival Time Parameters

The artificial increase of inter-arrival times has a negative impact on throughput. If chosen too high, it can quickly become a nuisance to users. Figure 11 illustrates the mapping from original bandwidth (without obfuscation) to effective bandwidth (after obfuscation) under three different average obfuscation delays. The higher the average delay, the smaller is the effective bandwidth. The plotted data is based on the following equation which yields the

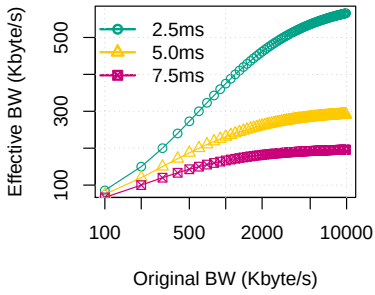


Figure 11: The cost in bandwidth when employing three different obfuscation delays.

overhead α.

$$\alpha = \left(\frac{BPS}{MTU} \cdot E[d] \right) \cdot \frac{1}{1000} + 1$$

The variable MTU refers to the maximum transmission unit which is typically 1500. BPS stands for "bytes per second" and refers to the original bandwidth of the available network link. $E[d]$ represents the expected value of the respective probability distribution d, i.e., the average obfuscation delay. In Figure 11, we plot the expected values 2.5, 5 and 7.5 milliseconds, respectively. ScrambleSuit uses the interval of $[0, 10[$ milliseconds for artificial delays. This interval has an expected value of 5ms which we believe to be a reasonable balance between this obfuscation/performance trade-off. It is ultimately limited to an effective throughput of 300 Kbyte/s and, when Tor is transported, can achieve throughputs around 150 Kbyte/s as we showed in our experimental evaluation.

C Usability Considerations

In order for a user to successfully connect to a ScrambleSuit server, she needs a *triple*: an IP address, a TCP port and a secret which is either the UniformDH secret k_B or a session ticket tuple $(k_t \parallel \mathcal{T}_t)$. We expect these triples to be distributed mostly electronically; over email, instant messaging programs or online social networks. As a result, a user can simply copy and paste the entire triple into her configuration file.

We do, however, also expect verbal distribution of ScrambleSuit triples, e.g., over a telephone line. To facilitate this, we define the encoding format of secrets and tickets to be Base32 which consists of the letters A–Z, the numbers 2–7 as well as the padding character "=". The numbers 0 and 1 are omitted to prevent confusion with the letters I and O. Since there is no distinction between uppercase and lowercase letters, we hope to make verbal distribution less confusing and error-prone. After all, a ScrambleSuit bridge descriptor would look like:

```
Bridge scramblesuit 1.2.3.4:443 password=NCA6I6GZZD42BWUB
```

We believe that the prefix `password=` will find more acceptance amongst users than simply appending the secret.