university of groningen

faculty of mathematics
and natural sciences

Expanding Domain Name System support in Tor

6th December 2016

Student: Elmer Jansema

Primary supervisor: Marco Aiello

Secondary supervisor: Frank B. Brokken

ABSTRACT

The protocols that are the foundations of the Internet have inherent privacy issues. Anonymity systems attempt to circumvent these privacy issues in a variety of ways. These systems advertise anonymity as a feature but the majority uses technology that is too simplistic. Tor is an anonymity system that works and continues to be the subject of active research. However, its implementation is unable to resolve all of the resource records (RRs) that the Domain Name System (DNS) protocol supports. Part of this thesis is research on how Tor limits the DNS protocol and the workarounds that attempt to bypass these limitations. Proposal 219 with the title *Support for full DNS and DNSSEC resolution in Tor* describes the removal of these limitations. The proposal addresses DNS resolution of all RRs by sending DNS packet data between onion routers and exit relays. Our implementation of the proposal consists of an asynchronous and a synchronous implementation. Using two implementations of the proposal enables us to measure the performance and research the differences. The results reveal that the implementations have a negative impact on the anonymity and performance of Tor and no impact on the security of Tor. Finally, we propose the asynchronous implementation for its performance and propose future work that removes the negative impact it has on anonymity and performance. These changes make the implementation suitable for inclusion into Tor.

ACKNOWLEDGEMENTS

The road that lead to this thesis was long and bumpy. From selecting a subject and creating working implementations to writing this thesis, each step was met with hurdles. The fact that you are reading this means I persevered.

Finishing this thesis and my master's degree would not have been possible without the help of my supervisors, my friends, my parents, and the Tor Project. Therefore, I thank prof. dr. ir. Marco Aiello for his guidance and support and thank dr. Frank B. Brokken for his feedback and constructive meetings. Furthermore, I give thanks to Jan-Paul Eikelenboom for proofreading drafts of this thesis. I am grateful for my parents that have giving me encouragement throughout my study. Finally, I appreciate the Tor Project for their work in developing software that protects its users from surveillance and facilitating my research by making their source code public.

CONTENTS

1 INTRODUCTION 1
- 1.1 Background 2
  - 1.1.1 Open proxies 2
  - 1.1.2 Virtual private networks 3
  - 1.1.3 Mesh networks 3
  - 1.1.4 Mix networks 3
  - 1.1.5 Onion routing 5
  - 1.1.6 Discussion 6
- 1.2 Related work 6
- 1.3 Problem statement 8
- 1.4 Thesis contribution 8
- 1.5 Thesis outline 8

2 CONCEPT AND DESIGN 11
- 2.1 The Domain Name System 11
- 2.2 Tor and the Domain Name System 12
  - 2.2.1 Workarounds 13
  - 2.2.2 Proposal 219 13

3 IMPLEMENTATION 15
- 3.1 Testing workarounds 15
- 3.2 Proposal implementation 18
  - 3.2.1 The Domain Name System library 23
  - 3.2.2 Implementation overview 23

4 RESULTS 25
- 4.1 Methodology 25
- 4.2 Impact on anonymity 26
  - 4.2.1 Traffic confirmation 26
  - 4.2.2 Fingerprinting 27
- 4.3 Impact on security 27
- 4.4 Impact on performance 28
  - 4.4.1 Latency 28
  - 4.4.2 Data consumption 31
- 4.5 Impact on usability 31

5 CONCLUSION 33

6 FUTURE WORK 35

A TOR FEATURES 37
- A.1 Perfect forward secrecy 37
  - A.1.1 Diffie-Hellman-Merkle key exchange 37

A.1.2 Key revocation 38
A.2 Integrity checking 38
A.3 Directory authorities 38
A.4 Exit policies 39
A.5 Hidden services 40

B APPLICATION CONFIGURATIONS 41

C CALL STACKS 45
C.1 Onion proxy functionality 45
C.2 Exit relay functionality 48

D DOMAIN NAME SYSTEM PACKETS 51
D.1 Domain Name System request 51
D.2 Domain Name System response 52

E RAW RESULTS 59

BIBLIOGRAPHY 63

LIST OF FIGURES

Figure 1.1 The topology of a network with *Proxy* substituting the IP address of *Workstation*. The substitution prevents the rest of the Internet from knowing the IP address of *Workstation* and assumes its IP address is 203.0.113.2. Icons by Cisco Systems, Inc. 2

Figure 1.2 Examples of partially and fully connected network topologies. Each node acts as a client and a server which makes it possible for the partially connected network to fully function without having 6 edges. Icons by Cisco Systems, Inc. 4

Figure 1.3 Visualisation of onion routing of Alice wrapping her message in layers of encryption and each onion router removing one encryption layer until the message reaches Bob. Source: *Spread the word about Tor* [73]. 5

Figure 2.1 The structures of the RELAY_RESOLVE and RELAY_RESOLVED cells where the fully qualified domain name (FQDN) field contains regular domains or special in-addr.arpa domains for reverse DNS resolution, the type field contains the type of the value field such as FQDN, IPv4 address, IPv6 address or errors, the length field contains the length of the value field in octets, the value field contains the FQDN, IPv4 address, IPv6 address, or errors depending on the type field, and the time to live (TTL) field contains the number of seconds until the DNS response expires. 12

Figure 2.2 The structures of the RELAY_DNS_BEGIN and RELAY_DNS_RESPONSE cells where the flags field is for future use and must be set to zero, the status field sets its first bit when its the last RELAY_DNS_RESPONSE cell for a previous RELAY_DNS_BEGIN cell and the other bits are for future use and must be set to zero, and the length field contains the length of the DNS response packet data in octets. 14

Figure 3.1 The two architectures for testing workarounds that perform Domain Name System resolution through Tor. 17
Figure 4.1 The average latencies in milliseconds when resolving the resource record A 250 times for each fully qualified domain name and implementation. The fully qualified domain names are the top 3 from the *Alexa Top 500 Global Sites* and one from the Internet Assigned Numbers Authority (IANA) list [2]. The implementations are the current Domain Name System implementation, and the asynchronous and synchronous proposal implementations. 29
Figure 4.2 The average latencies in milliseconds when applying a threshold of 10 milliseconds to the latency samples that Figure 4.1 uses. 30
Figure E.1 The average latencies in milliseconds including the minimum and maximum values when resolving the resource record A 250 times for each fully qualified domain name and implementation. The fully qualified domain names are the top 3 from the *Alexa Top 500 Global Sites* and one from the IANA list [2]. The implementations are the current Domain Name System implementation, and the asynchronous and synchronous proposal implementations. 60

Figure E.2 The latencies in milliseconds when resolving the resource record A for each fully qualified domain name and implementation. The fully qualified domain names are the top 3 from the *Alexa Top 500 Global Sites* and one from the IANA list [2]. The implementations are the current Domain Name System implementation, and the asynchronous and synchronous proposal implementations. 62

LIST OF TABLES

Table 1.1 The comparison between the goals of recent Tor papers where the A column represents anonymity, the P column represents performance, the S column represents security and the U column represents usability. The $+$ indicates a positive impact on the goal while the $-$ indicates a negative impact on the goal, and the $\pm$ indicates a balance between two or more goals. 7
Table 3.1 The fully qualified domain names are the top three from the *Alexa Top 500 Global Sites* [4] and one example fully qualified domain name from the Internet Assigned Numbers Authority list [2, section 3]. Domain Name System resolution uses the Domain Name System servers from Google Inc. and OpenDNS. The resource record PTR of `example.org` is empty because it is unavailable. 16
Table 3.2 The IP addresses while using the DNSPort. Emphasis signifies differences from the results in Table 3.1. 18
Table 3.3 The IP addresses while using the SocksPort. Emphasis signifies differences from the results in Table 3.1. 19
Table 3.4 The procedures that describe the current Domain Name System functionality and the functionality of proposal 219. 20
Table 3.5 Summary of the call stack which sets up the DNSPort to be able to receive Domain Name System requests from users (`PROC01`). 21

Table 3.6 Summary of the call stack which processes Domain Name System requests from the DNSPort and forwards them to the exit relay (`PROC02`). . 21
Table 3.7 Summary of the call stack which processes Domain Name System responses from the exit relay and forwards them to the DNSPort (`PROC03`). 21
Table 3.8 Summary of the call stack which sets up the listening socket (or ORPort) to be able to receive Domain Name System requests from onion proxies (`PROC04`). 22
Table 3.9 Summary of the call stack which processes Domain Name System requests from the ORPort and forwards them to the Domain Name System server (`PROC05`). 22
Table 3.10 Summary of the call stack which processes Domain Name System responses from the Domain Name System server and forwards them to the ORPort (`PROC06`). 22
Table 4.1 The number of samples are either below, equal, or above the threshold of 10 milliseconds. . . . 30
Table C.1 The call stack when setting up the DNSPort so it is able to receive Domain Name System requests from users (`PROC01`). 45
Table C.2 The call stack when processing Domain Name System requests from the DNSPort and forwarding them to the exit relay (`PROC02`). 46
Table C.3 The call stack when processing Domain Name System responses from the exit relay and forwarding them to the DNSPort (`PROC03`). 48
Table C.4 The call stack when setting up the listening socket (or ORPort) so it is able to receive Domain Name System requests from onion proxies (`PROC04`). 48
Table C.5 The call stack when processing Domain Name System requests from the ORPort and forwarding them to the Domain Name System server (`PROC05`). 50
Table C.6 The call stack when processing Domain Name System responses from the Domain Name System server and forwarding them to the ORPort (`PROC06`). 50
Table D.1 The hexadecimal representation of the Domain Name System request for the resource record A of `example.org`. 51

Table D.2 The dissection showing the fields inside the header section of the Domain Name System packet from Table D.1. 52
Table D.3 The dissection showing the fields inside the question section of the Domain Name System packet from Table D.1. 52
Table D.4 The hexadecimal representation of the Domain Name System response as answer to the Domain Name System request in Table D.1. 53
Table D.5 The dissection showing the fields inside the header section of the Domain Name System packet from Table D.4. 53
Table D.6 The dissection showing the fields inside the question section of the Domain Name System packet from Table D.4. 54
Table D.7 The dissection showing the fields inside the answer section of the Domain Name System packet from Table D.4. 54
Table D.8 The dissection showing the fields inside the first entry of the authority section of the Domain Name System packet from Table D.4. 55
Table D.9 The dissection showing the fields inside the second entry of the authority section of the Domain Name System packet from Table D.4. 55
Table D.10 The dissection showing the fields inside the first entry of the additional section of the Domain Name System packet from Table D.4. 56
Table D.11 The dissection showing the fields inside the second entry of the additional section of the Domain Name System packet from Table D.4. 56
Table D.12 The dissection showing the fields inside the third entry of the additional section of the Domain Name System packet from Table D.4. . . 57

Table D.13 The dissection showing the fields inside the fourth entry of the additional section of the Domain Name System packet from Table D.4. 57

LISTINGS

Listing B.1 Shell script for performing DNS resolution on the top 3 from the *Alexa Top 500 Global Sites* and one from the IANA list [2]. The DNS servers are from Google Inc., OpenDNS, or the local Tor DNSPort. 41
Listing B.2 The configuration file with defaults values for Tor options. The Tor instance with the open DNSPort and the Tor instance with the open SocksPort share these settings. 42
Listing B.3 The configuration file for running the Tor instance with an open DNSPort. The logging settings make it possible to see the DNS requests. 42
Listing B.4 The configuration file for running the Tor instance with an open SocksPort. The logging settings make it possible to verify that the exit relay is responding to request. 42
Listing B.5 The configuration file which contains the default Unbound options. The Unbound instances that forward requests to the open DNSPort and the external DNS server include this file into their configuration files. 42
Listing B.6 The configuration file for running the Unbound instance that forwards requests to the open DNSPort. 43
Listing B.7 The configuration file for running the Unbound instance that forwards requests to the external DNS server from the list of Public Access (Tier 2) servers that OpenNIC [87] offers. The torsocks application forwards outgoing DNS requests by Unbound through Tor. 43

Listing B.8 The configuration file for running Unbound instances that return the IPv4 address 127.0.0.1 for the FQDNs from Section 3.1. 43

GLOSSARY

Bridge relay	A guard relay that excludes itself from the consensus status document. Bridge relays obfuscate Tor traffic to make it harder to block connections to the Tor network [38]. xv, 5, 39
Cell	The message packet which is either 512 or 514 bytes in size depending on the link protocol version [see 41, sections 0.2, 3]. ix, xvi, 12–15, 23, 26, 38, 39
Circuit	The tunnel through the Tor network which consists of one guard relay, non-exit relay, and one exit relay. xv, xvi, 6, 12, 23, 25, 26, 30, 31, 38, 39
Directory authority	For the concept behind of directory authorities see Appendix A.3. xv, 5, 20, 25, 38, 39
Directory cache	For the concept behind of directory caches see Appendix A.3. xv, 39
Exit relay	The third and last onion router in a circuit which is on the edge of the Tor network. Its function is to connect to IP addresses and ports outside of the Tor network if their exit policy permits it (see Appendix A.4). iii, xii, xiv–xvi, 5, 7, 8, 12, 17, 20, 21, 25–27, 33, 35, 39–42, 45, 46, 48
Extra-info document	The document containing non-essential information about the onion router, such as its bandwidth history, the number of connections to the onion router per country, and other pieces of information [see 114, section 2.1.2]. 38
Guard relay	The first onion router in a circuit and prevents certain profiling attacks [see 40, section 5]. xv, xvi, 5, 25, 26
IP address	The numeric value that uniquely identifies computers inside a computer network [95, 32]. ix, xi, xv, xvi, 1–3, 6, 11, 12, 15, 17–19, 39, 40

IPv4 address	The numeric value that uniquely identifies computers inside a computer network [95]. The numeric value is 32 bits long and its representation uses the dot-decimal notation. For example, the IPv4 address of `www.example.org` is 93.184.216.34. ix, xv, 7, 12, 26, 41, 43
IPv6 address	The numeric value that uniquely identifies computers inside a computer network [32]. The numeric value is 128 bits long and its representation uses eight groups of four hexadecimal digits with a colon as the separator. For example, the IPv6 address of `www.example.org` is 2606:2800:220:1:248:1893:25c8:1946. ix, 7, 12
Mix	A node within a mix network. 3, 4
Non-exit relay	The second onion router in a circuit which relays cells between the guard relay and the exit relay. xv, 5, 25, 41
Onion proxy	The client communicating with onion routers. xii, xvi, 12, 20, 22, 25–27, 39, 45, 48
Onion router	A node within the Tor network. iii, ix, xv, xvi, 5, 6, 20, 25–27, 38, 39
Server descriptor	The document containing information about the onion router, such as its nickname, IP address, fingerprint, and other pieces of information [see 114, section 2.1.1]. 38, 39
Stream	The anonymous connection between the onion proxy and a destination outside of the Tor network. 12, 39

ACRONYMS

API	application programming interface. 31, 33, 35
cjdns	Caleb James DeLisle's Network Suite. 3
CPU	Central Processing Unit. 25, 33, 35
DH	Diffie-Hellman. 37
DHM	Diffie-Hellman-Merkle. 37
DHT	distributed hash table. 3
DIG	domain information groper. 15, 17, 25

DNS	Domain Name System. iii, ix–xiv, xvii, 7–9, 11–18, 20–23, 25–31, 33, 35, 41–43, 45, 46, 48–57, 60, 62
DNSSEC	DNS Security Extensions. 7, 8, 11, 13, 26, 27, 31, 33
DoS	denial of service. 8, 33, 35
EDNS	Extension Mechanisms for DNS. 31
FQDN	fully qualified domain name. ix–xi, xv, 7, 9, 11–13, 15, 16, 25–29, 31, 39–41, 43, 51, 59, 60, 62
HTTP	Hypertext Transfer Protocol. xvii, 2, 27
HTTPS	HTTP Secure. 3, 11, 27
I2P	Invisible Internet Project. 5, 6
IANA	Internet Assigned Numbers Authority. x, xi, xiv, 15, 16, 29, 42, 60, 62
IPS	Internet Protocol Suite. 3
IPv4	Internet Protocol version 4. 41, 54, 56
IPv6	Internet Protocol version 6. 3, 56, 57
KSK	key signing key. 11, 13
LAN	local area network. 39
LTS	Long Term Support. 25
MITM	man in the middle. 6, 11, 33, 38
MTA	message transfer agent. 7, 32
NSA	National Security Agency. 1, 6
OCSP	Online Certificate Status Protocol. 38
PFS	perfect forward secrecy. 4, 5, 38
PII	personally identifiable information. 1
RR	resource record. iii, x–xii, 7, 9, 11–13, 16–18, 21, 23, 25–29, 31–33, 51, 60, 62
RSA	Ron Rivest, Adi Shamir, and Leonard Adleman. 31
SHA-1	160-bit Secure Hash Algorithm. 31
SHA-256	256-bit Secure Hash Algorithm. 38
SMTP	Simple Mail Transfer Protocol. 4
SOCKS	Socket Secure. 2, 12, 15, 17, 41
SPoF	single point of failure. 6, 13, 18
TCP	Transmission Control Protocol. 5, 12, 13, 31, 33, 39, 51, 53
TLD	top-level domain. 40
TLS	Transport Layer Security. 7, 28, 37, 38
TTL	time to live. ix, 12, 27, 54–57
UDP	User Datagram Protocol. 5, 12, 13
VPN	virtual private network. 2, 3, 6

WWW	World Wide Web. 4, 5
XMPP	Extensible Messaging and Presence Protocol. 7, 32
ZSK	zone signing key. 11, 13

1

INTRODUCTION

Legislation and whistle-blowing have invigorated the discussion on privacy and the Internet and the impact they have on each other. Laws, such as implementations of the Data Retention Directive [46], focus on using the Internet to aid in fighting crime. At the same time key disclosure laws [80] try to combat privacy-enhancing technologies on the Internet. These laws are under scrutiny by digital rights groups [127] and law experts [11, 26] because of their effects on privacy. The Data Retention Directive is invalid according to a declaration by the Court of Justice of the European Union [25]. Furthermore, recent disclosures show how the foundations of the Internet have a negative impact on privacy. Among these disclosures are the publication of private surveillance industry documents by Wikileaks [104] and the release of National Security Agency (NSA) documents by Edward Joseph Snowden [110]. These disclosures show how the Internet is helping and at the same time obstructing surveillance operations. Finally, information brokers use the Internet for building profiles of consumers by recording and analysing their browsing habits. Companies buy these profiles to show specific advertisements on websites, verify identities, and perform background checks [13]. The laws, surveillance operations and commercial information market affect Internet privacy. According to Westin, '[p]rivacy is the claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others' [125].

The foundations of the Internet connect separate computer networks together thus creating one global computer network [17, section 1.1.2a]. Computers have addresses according to the Internet Protocol specifications [95, 32]. The packet switching method sends messages as packets and enables computers to communicate with each other. Packets include the IP addresses of the source and destination computers to make it possible for routers to send the packets through the network. Including the IP addresses makes it possible to passively track connections between computers and contradicts the privacy claim by Westin. Law experts suggest that IP addresses are personally identifiable information (PII) [66] and research describes that people have reasons for concealing their PII [74]. Therefore, controlling privacy includes being able to prevent IP address disclosures.

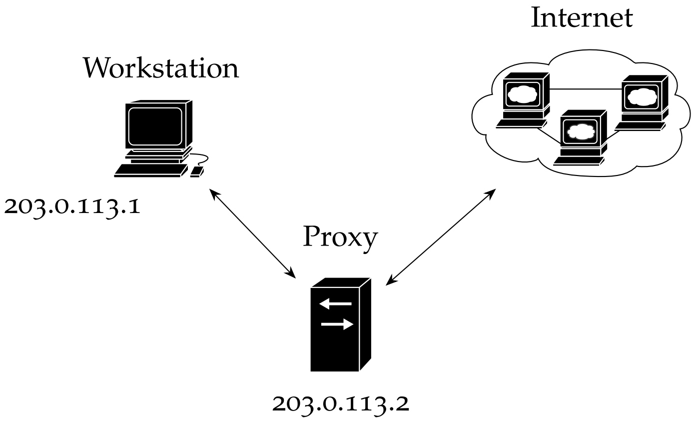


Figure 1.1: The topology of a network with *Proxy* substituting the IP address of *Workstation*. The substitution prevents the rest of the Internet from knowing the IP address of *Workstation* and assumes its IP address is 203.0.113.2. Icons by Cisco Systems, Inc.

1.1 BACKGROUND

The desire for anonymity on the Internet creates a demand for systems that hide the IP addresses of their users. Among these systems are open proxies, virtual private networks (VPNs), mesh networks, mix networks, and onion routing networks. The majority of these systems (such as open proxies and VPNs) are simplistic and have different use cases. Therefore, they offer weak anonymity and provide their users with a false sense of security while offering no protection against adversaries trying to undermine their anonymity.

1.1.1 *Open proxies*

The basic way of hiding source IP addresses are open proxies. Open proxies forward network traffic and mask the source IP addresses by substituting them with their own IP addresses. For the topology of a network with one proxy, see Figure 1.1. The three common open proxy types are Hypertext Transfer Protocol (HTTP) proxies [47], Socket Secure (SOCKS) proxies [71], and web proxies. These open proxy types differ by using different communication protocols. The majority of the networking applications and operating systems have support for HTTP and SOCKS proxies by default. Interaction with web proxies is through web interfaces where users submit the address of the website they want to visit. These web interfaces are user-friendly but only permit users to visit websites The strength of open proxies is their simplicity and easy of use. Furthermore, the majority of open proxies require no authentication and are free to use.

The HTTP and SOCKS proxy specifications lack methods for encrypting the network traffic which makes them susceptible to passive and active network attacks. Web proxies are able to support HTTP Secure (HTTPS) because they use web interfaces. In addition to external adversaries, the proxies themselves are able to log and modify network traffic. Therefore, malicious open proxies are able to compromise the anonymity of users by recording IP addresses. One way to decrease this risk is to use proxy chaining where one proxy connects to another proxy and so forth. However, the communication between proxies uses no encryption which makes the whole proxy chain susceptible to passive and active network attacks.

1.1.2 *Virtual private networks*

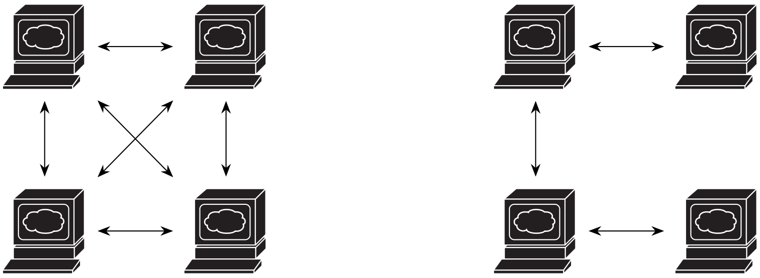
Using VPNs hides the source IP addresses. The VPN network topology is equal to the proxy network topology (see Figure 1.1) where the VPN replaces the proxy. Servers running VPN software are gateways to private networks which are accessible over the Internet. Access to the majority of VPNs includes pricing because they are part of a business model. These gateways use encryption and authentication which makes the network traffic unreadable and unmodifiable by external adversaries. Like the open proxies, the VPNs are able to log and modify network traffic. Finally, VPNs are an invention for connecting private networks together and have no intention of providing anonymity. Research by Appelbaum et al. describes that VPNs are susceptible to a variety of practical user deanonymisation attacks [5].

1.1.3 *Mesh networks*

Open proxies and VPNs use the Internet Protocol Suite (IPS) [102] while mesh networks use their own protocol suites. As the name implies a mesh network has the topology of a mesh which is either partially or fully connected (see Figure 1.2). Furthermore, mesh nodes act as clients and servers simultaneously which is a characterisation of a mesh network. Caleb James DeLisle's Network Suite (cjdns) is a protocol suite which '[...] implements an encrypted IPv6 network using public-key cryptography for address allocation and a distributed hash table for routing' [33]. The largest mesh network using cjdns is Hyperboria [57]. The protocol suite guarantees protection against eavesdropping and modification of network traffic but offers no anonymity [111].

1.1.4 *Mix networks*

Mix networks are similar to open proxies and VPNs but instead of using a single proxy they relay network traffic through mixes [21]. Mix cascades are series of mixes which hide the relation between the source and destination of messages. Layers of public-key cryptography wrap



(a) A fully connected network with 4 nodes and 6 edges.

(b) A partially connected network with 4 nodes and 3 edges.

Figure 1.2: Examples of partially and fully connected network topologies. Each node acts as a client and a server which makes it possible for the partially connected network to fully function without having 6 edges. Icons by Cisco Systems, Inc.

the messages with one layer for each consecutive mix in the cascade. When mixes receive messages, they remove one layer of encryption and place the resulting messages in a queue with other messages. At random intervals, the mixes reorder their queues and forward the messages in batches to other mixes in the network. The last mix in the cascade is able to remove the last encryption layer and forward the messages to their destination. The combination of encryption layers, and delaying, reordering and sending out messages in batches makes it difficult to correlate the source and destination of the messages. Modern mix networks improve anonymity by adding larger delays, dividing messages in fixed-sized blocks, using perfect forward secrecy (PFS)[1], and generating dummy network traffic.

Protocols using mix networks must be unidirectional and able to handle large delays between messages which limits the number of use cases. The use case Chaum describes is electronic mail because the Simple Mail Transfer Protocol (SMTP) [64] handles high latencies. The focus on sending anonymous electronic mail lead to the creation of anonymous remailers. Modern anonymous remailers implement mix network protocols such as cypherpunk (type I) [91], Mixmaster (type II) [82], and Mixminion (type III) [30].

The exception to being unidirectional is JonDonym [61] which offers World Wide Web (WWW) navigation through their Web MIXes system [15]. The company JonDos GmbH develops the JonDonym software, and other companies and individuals are running the mixes [60].

---

[1] Perfect forward secrecy prevents key disclosures from affecting past sessions by using independent encryption keys for each session. Appendix A.1 contains details regarding perfect forward secrecy.

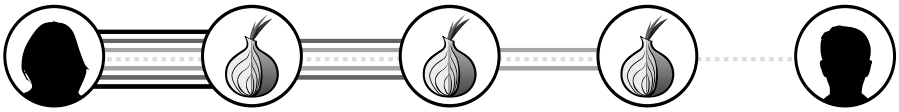


Figure 1.3: Visualisation of onion routing of Alice wrapping her message in layers of encryption and each onion router removing one encryption layer until the message reaches Bob. Source: *Spread the word about Tor* [73].

1.1.5 *Onion routing*

Mix networks are unidirectional and add latencies which makes them unusable for navigating the WWW, instant messaging, or streaming media. Onion routing [51] uses the encryption layers concept from mix networks but is otherwise different as Syverson mentions in their paper 'Why I'm Not an Entropist' [106, sections 8-9]. In fact, onion routing gets its name from the encryption layers because decryption acts like peeling onions. The differences make onion routing capable of offering bidirectional communication with lower latencies but affect anonymity. The removal of artificial latencies and the lack of message reordering makes onion routing susceptible to traffic confirmation attacks by global adversaries [29].

Second generation onion routing

Tor [42] is the second generation of onion routing [105] and improves its security and anonymity by adding PFS, integrity checking, directory authorities, exit policies and hidden services[2]. Tor uses the virtual circuit switching method as a layer on top of the Transmission Control Protocol (TCP) [96] which makes it incompatible with the User Datagram Protocol (UDP) [97]. The Tor Project and numerous volunteers develop the Tor software [117]. People which affiliate themselves with The Tor Project are responsible for the directory authorities [23] while volunteers contribute the guard relays, non-exit relays, exit relays, and bridge relays [115].

Garlic routing

Garlic routing works similar to onion routing but with the ability to group messages. According to Dingledine et al., '[g]arlic routing provides a few benefits. Delivery reliability and robustness is therefore increased through path redundancy' [37]. The Invisible Internet Project (I2P) [63] uses garlic routing and the packet switching method to make it possible to have anonymous communication using TCP and UDP. Volunteers carry out software development and network operations [59].

---

[2] Appendix A contains additional information about these improvements.

1.1.6 *Discussion*

Open proxies and VPNs provide weak anonymity because they are single points of failure (SPoFs) and users disclose their IP addresses by directly communicating with them. Furthermore, these systems know the destinations because they directly connect to them. Their positioning between the users and their destinations is similar to a man in the middle (MITM) and gives them the same capabilities. On the basis of these arguments, we disregard them as anonymity systems. Existing mesh networks provide no anonymity. Adding anonymity to them is a substantial amount of work and outside the scope of this thesis. Mix networks have a low number of use cases and are unpopular [90, 89]. On the other hand, onion routing networks and garlic routing networks have numerous use cases (because of the lower latencies) and are popular [115, 31].

The remaining choice is between onion routing networks and garlic routing networks (i.e. Tor versus I2P). The scientific community is doing active research on Tor [108] while there is minor attention for I2P [58]. Furthermore, the number of Tor users surpasses the number of I2P users by a factor of 140[3]. Finally, the release of NSA documents by Edward Joseph Snowden includes slides that crown Tor as '[...] the King of high secure, low latency Internet Anonymity' [12]. These arguments make Tor suitable for further research.

1.2 RELATED WORK

Prior work on Tor tends to focus on the four goals of anonymity systems, namely anonymity, performance, security, and usability. The work on Tor typically involves two goals with the aim of improving them simultaneously or improving one while minimising the negative impact on the other.

Research by Snader and Borisov [103] proposes an bandwidth measurement algorithm and a path selection algorithm[4] with a focus on throughput. These algorithms improve the performance and security of Tor. Research by Geddes, Jansen and Hopper [50] determines that changing congestion control algorithms has implications for anonymity by enabling existing and new attacks. The congestion control algorithms are able to improve performance but decrease anonymity. Research by Akhoondi, Yu and Madhyastha [3] proposes protections against malicious autonomous systems and a path selection algorithm with a focus on latency. The protections and algorithm uses parameters to change the balance between performance and anonymity.

---

[3] Research into the I2P network mentions around 142 thousand unique IP addresses [31] while Tor had around 2 million users during the same time period [115].

[4] The path selection algorithm [40] selects a set of onion routers suitable for creating circuits.

PAPER	A	P	S	U
Snader and Borisov [103]		+	+	
Geddes, Jansen and Hopper [50]	−	+		
Akhoondi, Yu and Madhyastha [3]	±	±		
Norcie et al. [85]				+
Winter et al. [126]		+		
Greschbach et al. [53]	−			

Table 1.1: The comparison between the goals of recent Tor papers where the *A* column represents anonymity, the *P* column represents performance, the *S* column represents security and the *U* column represents usability. The + indicates a positive impact on the goal while the − indicates a negative impact on the goal, and the ± indicates a balance between two or more goals.

Other research focuses on a single goal such as research by Norcie et al. [85] which proposes changes to the Tor Browser Bundle[5] to improve its usability. Research by Winter et al. [126] exposes malicious exit relays which improves the security of Tor Finally, research by Greschbach et al. [53] describes an attack that uses DNS traffic to decrease the anonymity of Tor. The comparison in Table 1.1 exemplifies the focus on performance and anonymity with minor attention for the security and usability issues of Tor.

Existing security and usability issues exist in the form of proposals which document the issue and its solutions [76]. The proposals are part of an process that the Tor Project uses for changing the Tor specifications [77] One of the open proposals is proposal 219 by Mikle with the title *Support for full DNS and DNSSEC resolution in Tor* [79]. Proposal 219 addresses the lack of support for RRs other than A, AAAA, and PTR which are the IPv4 address, the IPv6 address and the FQDN, respectively [81, section 3, 112]. The rudimentary support for DNS enables Internet browsing but has no support for the other capabilities that DNS offers. These missing capabilities include DNS Security Extensions (DNSSEC) [6, 8, 7] for securing DNS responses and DNS-based Authentication of Named Entities [56] which enables Transport Layer Security (TLS) connections without depending on certificate authorities. In addition, message transfer agents (MTAs) are able to use RR MX and Extensible Messaging and Presence Protocol (XMPP) clients and servers are able to use RR SRV [54] instead of both falling back to RR A [64, section 5.1, 100, section 3.2.2].

---

[5] The Tor Browser Bundle is a fork of the Mozilla Firefox browser that includes modifications to improve the anonymity of its users and has Tor integration.

1.3 PROBLEM STATEMENT

The goal of this paper is to implement proposal 219 and measure its impact on the Tor network. The impact measurements will focus on the four goals of anonymity systems, namely anonymity, performance, security, and usability. Additional features and work on existing features must refrain from reducing anonymity because it is the primary key driver of Tor. Introducing new source code also creates the opportunity of introducing security vulnerabilities so secure coding practices are applicable. Tor distinguishes itself from other anonymity systems by offering low latencies so the performance impact is important. Finally, the usability of proposal 219 is important to ensure its features attract users. Focusing on each goal separately creates the following research questions;

> Does extending DNS support disclose information that decreases user anonymity?
>
> Does extending DNS support increase the security of Tor or applications using Tor?
>
> Does extending DNS support decrease the performance of Tor or applications using Tor?
>
> Does extending DNS support increase the usability of Tor or applications using Tor?

1.4 THESIS CONTRIBUTION

The contribution of this thesis is an implementation of proposal 219 and measurements showing its impact on the Tor network. Measurements of the implementation indicate that the implementation impacts a subset of the goals of anonymity systems. Fingerprinting issues decrease the level of anonymity that Tor provides and therefore require future work. The handling of DNS requests makes exit relays susceptible to denial of service (DoS) attacks and impacts the performance of Tor. Fixing the fingerprinting issues and changing how exit relays handle incoming DNS requests is necessary for making the implementation robust.

1.5 THESIS OUTLINE

In Chapter 2 we describe DNS, DNSSEC, how Tor integrates DNS, and the proposal that removes the limitations that Tor places on DNS. Chapter 3 mentions the workarounds for the DNS limitations within Tor and discusses their reliability and usability. Furthermore, it includes an analysis of the source code of Tor in order to find the relevant functionality for implementing proposal 219. Lastly, we implement proposal 219. In Chapter 4 we use the implementation to measure its impact on the goals of Tor. Chapter 5 uses the measurements to answer the research questions. In Appendix A we describe how Tor improves upon onion routing with regards to security and anonymity. Appendix B contains the shell script and configuration files for implementing the workarounds for the DNS limitations within Tor. In Appendix C contains the call stacks of the Tor application which implements the current DNS functionality. Appendix D contains the hexadecimal representations and dissections of DNS packets when attempting to resolve RR A for the FQDN `example.org`. Finally, Appendix E contains raw data plots which we use to compute average latencies and measure the impact on performance.

2

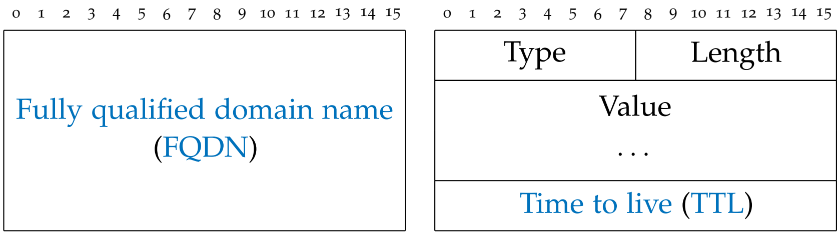
CONCEPT AND DESIGN

The writers of this paper assume the reader has basic knowledge of DNS so this chapter only outlines the privacy issues of DNS and how DNSSEC addresses them. Furthermore, we describe how DNS works within the Tor network and the limitations that Tor places on the capabilities of DNS. There are workarounds for these limitations which we describe accompanied with their usability issues. Finally, we discuss proposal 219 and explain how its solutions differ from the current Tor implementation.

2.1 THE DOMAIN NAME SYSTEM

The DNS protocol has privacy issues because it uses plain text messages without cryptographic signatures [10]. Plain text communication is susceptible to MITM attacks such as the interception of network traffic which makes it possible to passively track website visits. The lack of cryptographic signatures makes it is impossible to perform integrity and authentication checks which allows adversaries to modify DNS messages. These message modifications are able to force users to use malicious name servers and receive IP addresses to malicious servers when resolving FQDNs.

Using DNSSEC partially mitigates the privacy issues by adding cryptographic signatures to DNS responses [6]. The DNSSEC specification introduces RRs to create a public key infrastructure, namely the RR RRSIG for storing the cryptographic signatures, the RR DNSKEY for storing the zone signing keys (ZSKs) and the key signing keys (KSKs), and the RR DS for storing the chains of trust [8]. Verification of the cryptographic signature requires the ZSK of the FQDN. The KSK certifies the validity of the ZSK and the ZSK of the parent zone certifies the KSK of the current zone. The cryptographic signatures that sign the ZSK and the KSK are similar to certificates in the HTTPS architecture. The RR DS of each respective zone links the chains of trust by referring back to the RR DNSKEY of that zone. However, DNSSEC lacks data confidentiality which makes it possible to interpret the network traffic. There is no data confidentiality because the security working group considers '[...] data in the DNS [...] public information. This [...] assumption means that discussions and proposals involving data confidentiality and access control are explicitly outside the scope of this working group' [48].



(a) The RELAY_RESOLVE cell. (b) The RELAY_RESOLVED cell.

Figure 2.1: The structures of the RELAY_RESOLVE and RELAY_RESOLVED cells where the FQDN field contains regular domains or special in-addr.arpa domains for reverse DNS resolution, the type field contains the type of the value field such as FQDN, IPv4 address, IPv6 address or errors, the length field contains the length of the value field in octets, the value field contains the FQDN, IPv4 address, IPv6 address, or errors depending on the type field, and the TTL field contains the number of seconds until the DNS response expires.

2.2 TOR AND THE DOMAIN NAME SYSTEM

Internet browsing requires DNS resolution for retrieving the IP addresses of web servers but Tor is incompatible with UDP. The DNS protocol supports TCP and UDP but recommends use of the latter for standard queries [81, section 4.2]. The use of TCP is only a requirement when response data exceeds 512 bytes or when transferring zones. The incompatibility with UDP prevents onion proxies from performing DNS resolution natively. Tor overcomes this limitation by moving DNS resolution to the exit relays. See Appendix A.4 for more details.

The RELAY_RESOLVE cell enables onion proxies to perform DNS resolution without the exit relay creating a stream to the resulting IP address [41, section 6.4]. The specification of the RELAY_RESOLVE cell limits DNS resolution to the RRs A, AAAA and PTR which are the IPv4 address, the IPv6 address and the FQDN, respectively [81, section 3, 112]. In response the exit relays send back RELAY_RESOLVED cells which contain the IP addresses, FQDNs, or errors. Figure 2.1 visualises the structure of these cells.

The SocksPort, DNSPort, and ControlPort interfaces use the RELAY_RESOLVE cell to offer DNS resolution to their clients. The SocksPort interface uses the SOCKS protocol with private methods [71, section 3] to support DNS requests [119]. The DNSPort interface acts as a DNS server and listens for DNS requests. Finally, the ControlPort interface gives external processes control over the Tor process so they are able to make DNS requests, change the configuration of the Tor instance, and manage circuits and streams.

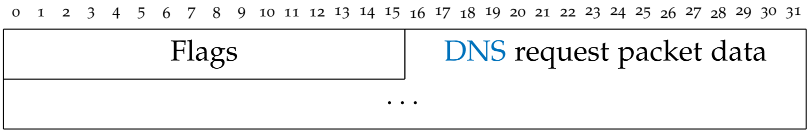
2.2.1 *Workarounds*

Creating support for DNS resolution of additional RRs is achievable by using TCP and sending the DNS requests through Tor to an external DNS server. The disadvantage of this workaround is the requirement for an external DNS server that supports TCP. There are DNS implementations that only support UDP because of a misinterpretation of the DNS specification which states that 'DNS resolvers and recursive servers MUST support UDP, and SHOULD support TCP [...]' [16]. An update to the DNS specification fixes the ambiguity [14, section 4] but legacy DNS implementations still exist. For this reason users have to be careful when choosing their DNS servers. Another disadvantage is the static list of DNS servers being a SPoF. In situations where all of the DNS servers in the list are offline, DNS resolution becomes impossible and requires additional DNS servers.

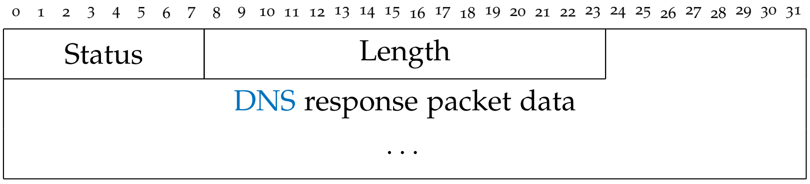
2.2.2 *Proposal 219*

The Tor developers recognise the limitations that the current Tor implementation places on the DNS protocol. Therefore, Mikle wrote proposal 219 with the title *Support for full DNS and DNSSEC resolution in Tor* [79]. The proposal describes two additional cells, namely RELAY_DNS_BEGIN and RELAY_DNS_RESPONSE. These two cells replace the RELAY_RESOLVE and RELAY_RESOLVED cells. The difference between the cells is the inclusion of DNS packet data instead of FQDNs (see Figure 2.2). Using DNS packet data removes the limitation of being able to resolve only RR A, AAAA, and PTR. The DNS packet data includes additional information about the DNS request such as whether to use recursive resolution, use DNSSEC, or disable signature validation.

The cells in Tor are either 512 or 514 bytes in size depending on the link protocol version [41, sections 0.2, 3]. When FQDNs use large ZSK and KSK sizes, the response sizes of the RRs DNSKEY and RRSIG [8] are larger than the maximum cell size. Proposal 219 suggests splitting the DNS response packet across RELAY_DNS_RESPONSE cells.



(a) The RELAY_DNS_BEGIN cell.



(b) The RELAY_DNS_RESPONSE cell.

Figure 2.2: The structures of the RELAY_DNS_BEGIN and RELAY_DNS_RESPONSE cells where the flags field is for future use and must be set to zero, the status field sets its first bit when its the last RELAY_DNS_RESPONSE cell for a previous RELAY_DNS_BEGIN cell and the other bits are for future use and must be set to zero, and the length field contains the length of the DNS response packet data in octets.

3

IMPLEMENTATION

The discussion on workarounds in Section 2.2.1 uses theoretical arguments for its conclusion. In this chapter we measure the reliability and usability by implementing the workarounds. After proving the necessity of proposal 219, we analyse the source code of Tor to find relevant functionality and implement the proposal.

3.1 TESTING WORKAROUNDS

Implementing and testing DNS resolution workarounds enables us to measure their reliability and usability. The implementations use the domain information groper (DIG) DNS client, the Unbound DNS resolver [122], and the torsocks application wrapper [121]. The configurations of these applications are available in Appendix B. The DNS client generates DNS queries and sends them to the DNS resolver which acts as a redirection layer. Changing the configuration of the DNS resolver enables switching between workarounds. The application wrapper ensures that applications without support for SOCKS [71] (such as the DNS resolver) are able to send their network traffic through Tor. The SocksPort, the DNSPort, and the ControlPort use the same RELAY_RESOLVE cell for DNS resolution, thus the resulting IP addresses are the same regardless of the port type. The SocksPort and the ControlPort change the DNS responses to conform to their respective protocols (SOCKS [71] for the SocksPort and a custom protocol for the ControlPort [107]). These protocols remove additional information such as whether the DNS response is an authoritative answer[1]. When applicable, the tests only use the DNSPort.

Testing the reliability of the workarounds involves comparing their DNS responses with the DNS responses from the Google Inc. and OpenDNS DNS servers (see Table 3.1). For testing DNS resolution we use the top three FQDNs from the *Alexa Top 500 Global Sites* [4], namely `google.com`, `facebook.com`, and `youtube.com`, and `example.org` from the IANA list of example FQDNs [2, section 3].

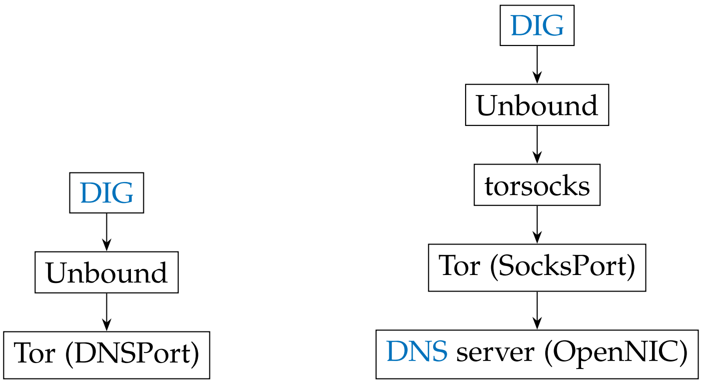
The components for testing the DNSPort are DIG, Unbound, and Tor. The DIG instance sends DNS requests to the Unbound instance which forwards the DNS requests to the DNSPort of the Tor instance. For a visual representation of the relation between these components, see Figure 3.1a.

---

[1] When the DNS server is an authority for the FQDN, the DNS response is an authoritative answer.

FULLY QUALIFIED DOMAIN NAME	RESOURCE RECORD A AND AAAA	RESOURCE RECORD PTR
`google.com`	173.194.65.100	ee-in-f100.1e100.net
	173.194.65.101	ee-in-f101.1e100.net
	173.194.65.102	ee-in-f102.1e100.net
	173.194.65.113	ee-in-f113.1e100.net
	173.194.65.138	ee-in-f138.1e100.net
	173.194.65.139	ee-in-f139.1e100.net
	2a00:1450:4013:c00::66	ee-in-x66.1e100.net
`facebook.com`	173.252.120.6	edge-star-shv-12-frc3.facebook.com
	2a03:2880:2130:cf05:face:b00c:0:1	edge-star6-shv-12-frc3.facebook.com
`youtube.com`	173.194.65.91	ee-in-f91.1e100.net
	173.194.65.93	ee-in-f93.1e100.net
	173.194.65.136	ee-in-f136.1e100.net
	173.194.65.190	ee-in-f190.1e100.net
	2a00:1450:4013:c00::5b	ee-in-x5b.1e100.net
`example.org`	93.184.216.34	
	2606:2800:220:1:248:1893:25c8:1946	

Table 3.1: The fully qualified domain names are the top three from the *Alexa Top 500 Global Sites* [4] and one example fully qualified domain name from the Internet Assigned Numbers Authority list [2, section 3]. Domain Name System resolution uses the Domain Name System servers from Google Inc. and OpenDNS. The resource record PTR of `example.org` is empty because it is unavailable.



(a) The architecture for testing Domain Name System resolution through the DNSPort.

(b) The architecture for testing Domain Name System resolution with an external Domain Name System server through Tor.

Figure 3.1: The two architectures for testing workarounds that perform Domain Name System resolution through Tor.

The results contain differences in the IP addresses of Google and YouTube (see Table 3.2). Split-horizon DNS [92, section 4] is the cause of these differences by returning IP addresses of servers closest to the geographic location of the originating IP address. Decreasing the physical distance between the client and the server decreases the latency. The onion routing technology of Tor makes us appear to be coming from different geographic locations which results in different IP addresses. Furthermore, Tor only returns the first IP address from all of the IP addresses that the DNS resolver of the exit relay returns [39, lines 1527–1588]. Returning incomplete DNS responses decreases the redundancy that DNS is able to provide. Finally, the results for RR AAAA are inconsistent because the support for it depends on the exit relay configuration. For generating the results (see Table 3.2) we were using exit relay `B7EC0C02D7D9F1E31B0C251A6B058880778A0CD1`[2].

The components for testing an external DNS server through Tor are DIG, Unbound, torsocks, Tor, and an external DNS server. The DIG instance sends DNS requests to the Unbound instance which forwards the DNS requests to the external DNS server through the SocksPort of the Tor instance. Unbound has no support for SOCKS and requires torsocks to redirect outgoing network traffic of the DNS resolver to the SocksPort of the Tor instance. The external DNS server is from the list of Public Access (Tier 2) servers that OpenNIC [87] provides, in particular `ns1.vwv.be.dns.opennic.glue` or `192.71.249.249`[3]. For a visual representation of the components, see Figure 3.1b.

The results are similar to the results of the DNSPort with differences in the IP addresses of Google and YouTube (see Table 3.3). As with the previous workaround, split-horizon DNS is the cause. Another differ-

---

[2] Use `https://atlas.torproject.org/` to get information about this exit relay.

[3] For information about the DNS server, see `http://meo.ws/dnsrec.php`.

FULLY QUALIFIED DOMAIN NAME	RESOURCE RECORD A AND AAAA	RESOURCE RECORD PTR
`google.com`	*74.125.136.138*	*ea-in-f138.1e100.net*
	2a00:1450:4013:c01::71	*ea-in-x71.1e100.net*
`facebook.com`	173.252.120.6	edge-star-shv-12-frc3.facebook.com
	2a03:2880:2130:cf05:face:b00c:0:1	edge-star6-shv-12-frc3.facebook.com
`youtube.com`	*74.125.136.93*	*ea-in-f93.1e100.net*
	2a00:1450:4013:c01::5d	*ea-in-x5d.1e100.net*
`example.org`	93.184.216.34	
	2606:2800:220:1:248:1893:25c8:1946	

Table 3.2: The IP addresses while using the DNSPort. Emphasis signifies differences from the results in Table 3.1.

ence between the results of the DNSPort is the number of IP addresses. Using the external DNS server circumvents the DNS functionality of Tor thus the responses contain all of the IP addresses instead of only the first IP address.

After performing the tests, we conclude that the reliability of the DNSPort is insufficient because it truncates DNS responses and has inconsistent support for the RR AAAA. The alternative is to use an external DNS server through Tor which applies no truncation to the DNS responses. However, setting it up is non-trivial and the DNS server is a SPoF. The usability of the DNSPort is sufficient because it acts like a DNS server and only requires the user to specify the port it must listen on. Setting up an external DNS server requires finding suitable DNS servers and changing the DNS resolver configuration. Therefore, using an external DNS server is an inappropriate workaround for beginners. These problems highlight the necessity of proposal 219 which combines the usability of the DNSPort with the reliability of the DNS responses that an external DNS server returns.

3.2 PROPOSAL IMPLEMENTATION

Adding the proposal functionality starts with identifying the files and functions that contain and implement the current DNS functionality. The code structure of Tor [120] divides the source code into generic functionality, such as cryptographic algorithms, data storage

FULLY QUALIFIED DOMAIN NAME	RESOURCE RECORD A AND AAAA	RESOURCE RECORD PTR
`google.com`	*74.125.136.100*	*ea-in-f100.1e100.net*
	74.125.136.101	*ea-in-f101.1e100.net*
	74.125.136.102	*ea-in-f102.1e100.net*
	74.125.136.113	*ea-in-f113.1e100.net*
	74.125.136.138	*ea-in-f138.1e100.net*
	74.125.136.139	*ea-in-f139.1e100.net*
	2a00:1450:4013:c01::65	*ea-in-x65.1e100.net*
`facebook.com`	173.252.120.6	edge-star-shv-12-frc3.facebook.com
	2a03:2880:2130:cf05:face:b00c:0:1	edge-star6-shv-12-frc3.facebook.com
`youtube.com`	*74.125.136.91*	*ea-in-f91.1e100.net*
	74.125.136.93	*ea-in-f93.1e100.net*
	74.125.136.136	*ea-in-f136.1e100.net*
	74.125.136.190	*ea-in-f190.1e100.net*
	2a00:1450:4013:c01::88	*ea-in-x88.1e100.net*
`example.org`	93.184.216.34	
	2606:2800:220:1:248:1893:25c8:1946	

Table 3.3: The IP addresses while using the SocksPort. Emphasis signifies differences from the results in Table 3.1.

PROCEDURE	CODE
ONION PROXY FUNCTIONALITY	
Setting up the DNSPort to be able to receive DNS requests from users.	`PROC01`
Processing DNS requests from the DNSPort and forwarding them to the exit relay.	`PROC02`
Processing DNS responses from the exit relay and forwarding them to the DNSPort.	`PROC03`
EXIT RELAY FUNCTIONALITY	
Setting up the listening socket (which provides the ORPort) to be able to receive DNS requests from onion proxies.	`PROC04`
Processing DNS requests from the ORPort and forwarding them to the DNS server.	`PROC05`
Processing DNS responses from the DNS server and forwarding them to the ORPort.	`PROC06`

Table 3.4: The procedures that describe the current Domain Name System functionality and the functionality of proposal 219.

structures, and threading; Tor-specific functionality, such as directory authorities (see Appendix A.3), onion routers, and hidden services (see Appendix A.5); and tests, that verify the correctness of the implementation according to the Tor specifications [116].

The current DNS functionality is dividable into two categories with each three procedures (see Table 3.4). These procedures are similar to the workings of proposal 219 despite their implementations being different.

The file names and the comments at the top of the files describe their functionality and helps us with identifying the two files that relate to the DNS functionality. These files are dnsserv.c and dns.c and implement the DNS server of the onion proxy (which provides the DNSPort) and the DNS client of the exit relay, respectively. With these files we have the starting point for identifying the functions that implement the current DNS functionality.

The descriptive nature of the function names makes it possible to determine the functionality they offer. In addition, the function comments contain functionality descriptions. The next step is generating call stacks which visualise how the functions connect to other parts of Tor. Call stack generation involves using GNU Debugger [49], applying breakpoints to relevant functions, and printing the call stack when reaching the breakpoints. Tables 3.5 to 3.10 provide descriptive sum-

FUNCTION	DESCRIPTION
`tor_main`	The entry point of Tor.
`options_init_from_torrc`	The configuration files parser.
`connection_listener_new`	The DNSPort creation function.
`evdns_add_server_port_with_base`	The callback initialisation function.

Table 3.5: Summary of the call stack which sets up the DNSPort to be able to receive Domain Name System requests from users (`PROC01`).

FUNCTION	DESCRIPTION
`evdns_server_callback`	The callback function.
`connection_ap_handshake_rewrite_and_attach`	The circuit selector.
`connection_edge_send_command`	The DNS request forwarder.

Table 3.6: Summary of the call stack which processes Domain Name System requests from the DNSPort and forwards them to the exit relay (`PROC02`).

maries of the call stacks that implement the procedures in Table 3.4. The complete call stacks are available in Appendix C.

Analysis of the call stacks indicates that exit relays use the Libevent library [78] for resolving DNS requests. The Libevent library reflects the limitations of the RELAY_RESOLVE cell as it only supports DNS resolution of the RRs A, AAAA, and PTR[4] [75]. Removing the RR limitation means either patching the Libevent library or replacing the

FUNCTION	DESCRIPTION
`conn_read_callback`	The callback function.
`connection_edge_process_relay_cell`	The cell parser
`connection_edge_process_resolved_cell`	The RELAY_RESOLVED cell parser.
`dnsserv_resolved`	The DNS response forwarder.

Table 3.7: Summary of the call stack which processes Domain Name System responses from the exit relay and forwards them to the DNSPort (`PROC03`).

---

[4] The development leader of Tor is one of the Libevent maintainers. Adding support for DNS resolution to Libevent was by request from Tor.

FUNCTION	DESCRIPTION
`tor_main`	The entry point of Tor.
`options_init_from_torrc`	The configuration files parser.
`connection_listener_new`	The ORPort creation function.
`connection_add_impl`	The callback initialisation function.

Table 3.8: Summary of the call stack which sets up the listening socket (or ORPort) to be able to receive Domain Name System requests from onion proxies (`PROC04`).

FUNCTION	DESCRIPTION
`conn_read_callback`	The callback function.
`connection_edge_process_relay_cell`	The cell parser
`connection_exit_begin_resolve`	The RELAY_RESOLVE cell parser.
`dns_resolve`	The DNS request resolver.

Table 3.9: Summary of the call stack which processes Domain Name System requests from the ORPort and forwards them to the Domain Name System server (`PROC05`).

FUNCTION	DESCRIPTION
`evdns_callback`	The callback function.
`dns_found_answer`	The DNS response cache.
`connection_edge_send_command`	The DNS response forwarder.

Table 3.10: Summary of the call stack which processes Domain Name System responses from the Domain Name System server and forwards them to the ORPort (`PROC06`).

Libevent library. Choosing the later only requires changes to the Tor code base.

3.2.1 *The Domain Name System library*

The DNS library replacement has to meet certain requirements to be able to perform the functionality in proposal 219. These requirements are being able to perform asynchronous DNS resolution, supporting all RR types, and giving access to the DNS response packets. Asynchronous DNS resolution makes the Tor application able to continue operating while the DNS server resolves the DNS request. Support for all RR types is essential because it is the primary goal of proposal 219. Proposal 219 mentions that RELAY_DNS_RESPONSE cells contain DNS packet data therefore requiring that the DNS library provides access to the DNS response packets. The Unbound library [123] is the only DNS library which meets all of these requirements.

3.2.2 *Implementation overview*

The modular structure of Tor simplifies the process of adding features. In our case, adding cell types involves writing parsing functionality that handles these cell types, integrating the Unbound library, and changing the current DNSPort implementation. The onion routing code, which is a core component and responsible for creating circuits, requires no changes. Furthermore, the current DNS functionality converts DNS requests and responses to conform to the specification of the RELAY_RESOLVE and RELAY_RESOLVED cells. The proposal makes this conversion obsolete by adding the DNS request and response packets directly to the cells.

The implementation is built on top of the Tor source code which uses the GNU's Not Unix Build System [45, 69]. Its dependencies are the Libevent [75], OpenSSL [88], zlib [98], and Unbound [123] libraries. After installing the dependencies, building the software involves generating the configuration files (with the autoreconf tool), configuring the build system (with the configure script), and executing the build system (with the make tool). Testing the implementation involves running the make tool with the *check* argument which executes the test suite to verify correctness. Installing the software involves running the make tool as an superuser with the argument *install*.

4

RESULTS

The proposal implementation enables us to apply methods and metrics to it. With these methods and metrics we are able to measure and interpret the impact that the proposal has on anonymity, security, performance, and usability. These measurements and interpretations help us answer the research questions from Section 1.3.

4.1 METHODOLOGY

Each of our research questions address a goal of anonymity systems, namely anonymity, security, performance, or usability. These goals require distinct methods for retrieving measurements.

Measuring the impact on anonymity involves researching how expanding DNS support affects existing types of attacks. For our measurements we research traffic confirmation and fingerprinting attacks and use code analysis to measure their impact on our implementation.

The impact on security also consists of a code analysis which researches our implementation code in comparison to the Tor source code. There is also an analysis of the impact on external applications that use our implementation.

The performance measurements consist of the latency between DNS requests and their responses, and the amount of data that DNS traffic consumes. For retrieving latency samples we use the Chutney integration testing suite [22] which creates a local Tor network without disrupting the official Tor network. Performing sampling in a local Tor network removes the influence that global Internet routing has on latency. The local Tor network consists of two directory authorities which act as guard relays and non-exit relays, one exit relay, and one onion proxy. Using three onion routers is the minimum number that the onion proxy requires to be able to construct a circuit. The local Tor network runs on a computer with an Intel Core i3-3220 Central Processing Unit (CPU) running at 3.30 gigahertz, and 8 gigabytes of memory. The computer uses the operating system Ubuntu 14.04.3 LTS (code name Trusty Tahr). The DNS client is DIG version 9.9.5. Caching affects the accuracy of the samples so during sampling we disable caching in the Tor implementation, the Unbound library, and the DNS client.

Sampling involves having the DNS client send DNS requests to the DNSPort of the onion proxy. The DNS requests are for the RR A of each FQDN in Section 3.1. The samples are the latency in milliseconds between the DNS requests and their responses (see Figure E.2). There is artificial latency because the exit relay forwards DNS requests to the DNS server. For latency sampling the contents of the DNS responses are unimportant thus the Unbound library uses a hosts file. The hosts file prevents the exit relay from using the DNS server and returns the IPv4 address 127.0.0.1 for the FQDNs from Section 3.1. Listing B.8 contains the configuration file for this Unbound instance.

The data consumption measurements are theoretical and use DNS specifications for computing the data consumption. Using DNS specifications makes our analysis independent from DNS implementations and avoids measurement inaccuracies due to implementation errors. The number of RRs that DNS supports makes it difficult to analyse all RRs thus we only focus on the RR A of `example.org` and its DNSSEC extensions.

Finally, the usability measurements include an analysis of the differences between our implementation and the Tor source code and their impact on usability. The measurements also include the usability impact on external applications that use our implementation.

4.2 IMPACT ON ANONYMITY

For measurements on the impact of anonymity we look at traffic confirmation and fingerprinting attacks. Traffic confirmation attacks (or end-to-end correlation attacks) are the type of attacks that focus on finding the source and destination of messages within the Tor network. Fingerprinting attacks are the type of attacks that identify users by looking at the communication between onion proxies and onion routers.

4.2.1 *Traffic confirmation*

Traffic confirmation attacks are a group of attacks that aim to locate the endpoints of a circuit. These attacks require that adversaries are able to control or monitor the guard relay and the exit relay of a circuit. When the adversary controls the guard relay and the exit relay they are able to perform tagging attacks. Tagging attacks inject data into cells to make them traceable. Data injection is difficult because it changes the cells which is detectable with integrity checking. Passive attacks only monitor network traffic and their effectiveness is equal to active attacks. The threat model of Tor mentions that it has no protection against global adversaries which are able to monitor the endpoints of a circuit [42, section 3.1, 9].

There is no impact on anonymity because the proposal implementation makes no changes to the onion routing code. Introducing cell types has no impact on anonymity because the onion routing code encloses them in layers of encryption and uses integrity checking.

4.2.2 *Fingerprinting*

Being able to fingerprint users breaks the anonymity that Tor provides. The resulting fingerprints are abstract representations of users on the basis of their network traffic. Tor encrypts its messages which makes generating fingerprints using the communication data between onion routers difficult. Exit relays are able to intercept data when the connections to destinations use plain text protocols (such as HTTP instead of HTTPS).

Under the assumption that users visit the same websites, DNS requests contain no fingerprints that uniquely identify users. The requests are fingerprintable when users request unique FQDNs. Websites that contain resources with unique FQDNs are able to force users to send unique requests. Onion proxies negate this fingerprinting technique by disabling caching of DNS responses by default. When exit relays use DNS caches, the fingerprints are only for those exit relays and has no significant impact on the anonymity of the onion routers.

Another fingerprinting technique uses the TTL DNS packet field. The TTL field contains the number of seconds that represents how long the RR is valid. Caching DNS resolvers use the field for computing the validity of their cache. Using different TTL values for each DNS response makes the authoritative DNS server able to fingerprint its users when they request RRs again. The fingerprint technique relies on a DNS cache which is inactive by default so there is no impact on anonymity.

The final fingerprinting technique uses the TTL field but relies on the DNS cache of exit relay. When the exit relay caches RRs and returns its cache entries to any onion proxy, it exposes previous requests for that RR through the TTL field. For example, onion proxy A requests RR A of `example.org` which has a default TTL of 86400 seconds. Now onion proxy B requests the same RR for the same FQDN. When the TTL is lower than the initial TTL, onion proxy B is able to conclude that there was a previous request for the RR A of `example.org`. Exit relays lower the impact of this fingerprint technique by changing the TTL to be within the interval $[60, 10600]$. Proposal 219 suggests lowering the interval to $[5, 600]$ [79, section 7]. Our implementation is vulnerable to this fingerprinting technique because it makes no changes to the DNS packet data.

4.3 IMPACT ON SECURITY

No changes were made to the onion routing code so the security of Tor remains the same (see Section 3.2.2).

The proposal implementation increases the security of DNS by being able to support the DNSSEC RRs. Verifying the chain of trust is paramount for trusting the DNS response. Without verification the trust in the DNS responses is similar to the current implementation.

Expanding DNS support makes RRs with a focus on security outside of DNS usable. For example, the RR TLSA [56] stores the TLS certificate of the server. Using the RR TLSA deprecates the certificate authorities in favour of the signer of the DNS root zone. The article *Understanding DNSSEC* by Cardwell [20] contains additional security-enhancing RRs.

4.4 IMPACT ON PERFORMANCE

For measuring the performance impact of our implementation we use the latency and the data consumption metrics. Latency measures the delay between requests and their responses in milliseconds. Data consumption measures the amount of data in bytes that the DNS requests and DNS responses are consuming.

4.4.1 *Latency*

During initial performance testing of the proposal implementation the average latency between DNS requests and responses was ≈ 900 milliseconds. In comparison, the current DNS functionality has an average latency of ≈ 1 millisecond. The cause was processing of DNS responses only every 1000 milliseconds. In other words, when a DNS response arrives after a processing cycle it has to wait ≈ 1000 milliseconds for the next processing cycle. These processing cycles make DNS resolution asynchronous by periodically checking for DNS responses instead of waiting for them. The solution is to decrease the delay between processing cycles from 1000 milliseconds to 10 milliseconds. Another solution is removing the processing cycles and using synchronous DNS resolution.

The following latency measurements include the asynchronous implementation with 10 milliseconds between its processing cycles and the synchronous implementation in order to show their performance differences. Figure 4.1 contains the average latency for each FQDN and implementation. The average latencies are between 1.048 and 5.304 milliseconds. The current DNS functionality has the lowest latency across all FQDNs and the asynchronous proposal implementation has the highest latency across all FQDNs. In Section 3.2.2 we explain that there is a delay of 10 milliseconds between processing cycles in the asynchronous implementation and this delay is partially causing the average latency increases. When comparing the results of the current DNS functionality with the synchronous proposal implementation, the synchronous proposal implementation has a higher average latency while having no processing cycles. The primary implementation difference between the current DNS functionality and the proposal implementations is the Unbound library. Finding the lines of

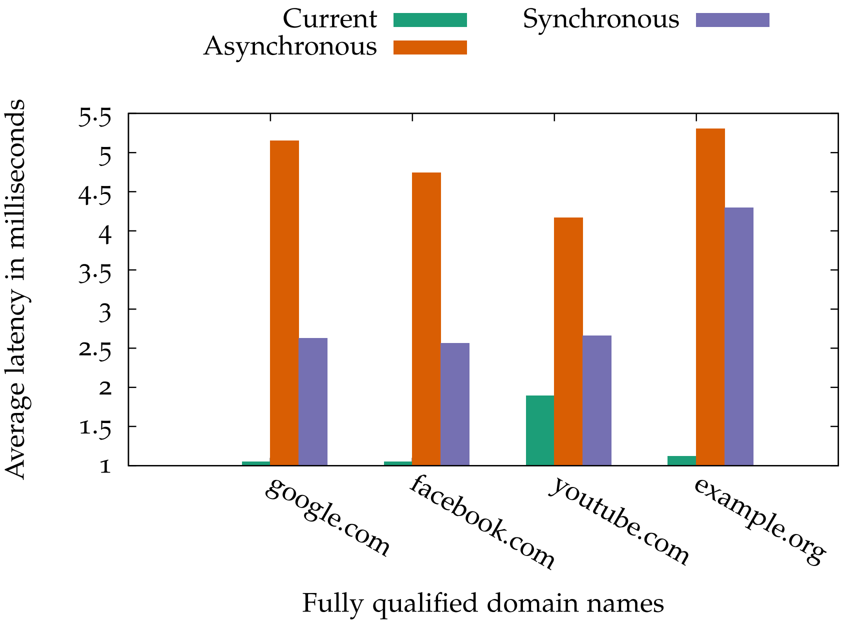


Figure 4.1: The average latencies in milliseconds when resolving the resource record A 250 times for each fully qualified domain name and implementation. The fully qualified domain names are the top 3 from the *Alexa Top 500 Global Sites* and one from the IANA list [2]. The implementations are the current Domain Name System implementation, and the asynchronous and synchronous proposal implementations.

IMPLEMENTATION	BELOW	EQUAL	ABOVE	TOTAL
Current	993	1	6	1000
Asynchronous	992	0	8	1000
Synchronous	989	2	9	1000
Total	2974	3	23	3000

Table 4.1: The number of samples are either below, equal, or above the threshold of 10 milliseconds.

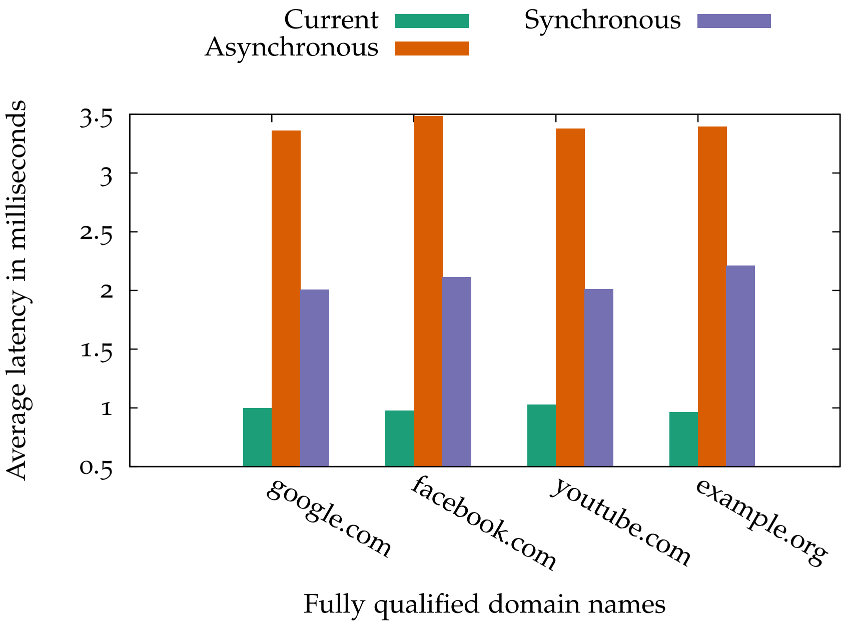


Figure 4.2: The average latencies in milliseconds when applying a threshold of 10 milliseconds to the latency samples that Figure 4.1 uses.

code in the Unbound library that increase the latency is outside the scope of our research and remains an open question.

The raw data plots in Figures E.2a to E.2d contain peaks which increase the latency averages. Using a 10 millisecond threshold and counting the number of samples that are either below, equal, or above the threshold measures the impact of these peaks. The threshold results in Table 4.1 show that values above 10 milliseconds account for 0.767 percent of the total number of samples. Differences between the threshold and the values above the threshold are higher than between the threshold and the values below the threshold (see Figure E.1). Without these values, the average latencies decrease to values between 0.964 and 3.482 milliseconds (see Figure 4.2).

The cause of these values appearing near the beginning of the sampling phase in unknown. The theory is that building circuits adds latency for the initial DNS requests. Subsequent DNS requests reuse these circuits and therefore show no additional latency. Testing the theory involves using additional DNS requests outside the sampling phase to force Tor to build circuits. The subsequent DNS requests, which are part of the sampling phase, still have the additional latency. These results disprove the theory and keep the cause is unknown.

4.4.2 *Data consumption*

The size of a DNS request packet depends on the length of the FQDN. For example, the packet sizes of DNS requests for RR A of the FQDN `example.org` and `facebook.com` are 29 and 30 bytes, respectively. Using TCP increases the packet length by 2 bytes [81, section 4.2.2]. For hexadecimal representations and dissections of the DNS request and response packets for RR A of `example.org`, see Appendix D. The maximum length of FQDNs is 255 bytes [81, section 2.3.4].

Using DNSSEC increases requests and responses by requiring Extension Mechanisms for DNS (EDNS) and including cryptographic signatures. The requirement to use EDNS [7, section 4.1] enables DNS packets to be larger than 512 bytes. Using EDNS adds a pseudo-RR to the DNS packets which increases their size by 11 bytes [28, section 6.1.2]. Adding cryptographic signatures to the DNS responses enables integrity and authentication checking (see Chapter 2). The RR RRSIG represents these cryptographic signatures with two fields of variable lengths [8, section 3]. The first field contains the name of the signer and depends on the length of the FQDN. The second field contains the cryptographic signature and depends on the cryptographic signature algorithm. To simplify our calculations we focus on RSA/SHA-1 which is the only mandatory cryptographic signature algorithm [99, section 2.3]. The output length of the algorithm is equal to the length of the RSA modulus and has a maximum length of 512 bytes [1, section 2]. The static fields are 18 bytes in total. In the worst case (meaning the longest possible FQDN and RSA modulus) the RR RRSIG is $255 + 512 + 18 = 785$ bytes. Therefore, the additional data that DNSSEC requires increases the DNS requests by 11 bytes and DNS responses by $11 + 785 = 796$ bytes.

As we describe in Chapter 2, the cryptographic signatures are part of a chain of trust. Following the chain of trust requires additional DNS requests which increase the data consumption.

4.5 IMPACT ON USABILITY

There were no changes to the protocols of the SocksPort, DNSPort, and ControlPort interfaces so their usability remains the same. The implementation only changes the DNSPort to use the new DNS application programming interface (API). The SocksPort and the ControlPort interfaces use protocols that differ significantly from the DNS protocol.

Supporting additional RRs requires extensions to these protocols to be able to support this additional information.

The usability of applications using the DNSPort interface increases because they are able to resolve previously unavailable RRs. For example, MTAs are able to use RR MX and XMPP clients and servers are able to use RR SRV.

5

CONCLUSION

With the results from Chapter 4 we are able to answer our research questions from Section 1.3. These research questions are based on the goals of anonymity systems, namely anonymity, security, performance, and usability.

The proposal implementation permits fingerprinting of the DNS requests which impacts anonymity. The fingerprinting prevention techniques from Section 4.2.2 require packet data modifications which the Unbound library is unable to perform. The NLnet Labs organisation maintains the Unbound library but also maintains the ldns library [70] which has a low-level DNS API.

Removing the limitation on DNS resolution increases the security of Tor by making it possible to use DNSSEC. Using DNSSEC makes it harder for exit relays to alter DNS responses and prevents certain MITM attacks. Furthermore, users no longer have to search for DNS servers that support TCP to be able to resolve RRs other than A, AAAA, and PTR.

The impact on the performance of Tor depends on the proposal implementation. The synchronous implementation has no processing cycles and its performance depends on the latency of the DNS server. Exit relays that use the synchronous implementation block other requests until DNS resolution is complete. Blocking other requests makes the exit relays slow and susceptible to DoS attacks and for that reason we discourage the use of the synchronous implementation. The asynchronous implementation requires decreasing the delay between processing cycles to 10 milliseconds to get reasonable performance. Decreasing the delay between processing cycles has adverse effects on other processing functions. These processing functions require significant amounts of CPU time and calling them more than once per second has an noticeable impact on performance. The recommendation is to use the asynchronous implementation because it handles DNS requests without blocking other requests which results in better performance over time.

Without changing the DNSPort interface, it has become possible to offer additional RRs. The SocksPort and ControlPort interfaces require extensions to be able to support the additional DNS information that has become available and are out of scope. The usability increase of these interfaces also increases the usability of the applications that use them because there is no fallback to the RR A. Avoiding the fallback RRs removes one round trip between applications and the Tor network and establishes connections faster.

6

FUTURE WORK

Following the discussion in Chapters 4 and 5 we discourage the use of the synchronous implementation because it makes exit relays susceptible to DoS attacks. The asynchronous implementation has better performance than the synchronous implementation, but comparing it with the current implementation shows a negative impact on the performance and anonymity of Tor. Minimising the impact increases the robustness of the asynchronous implementation and is necessary for inclusion in Tor.

The proposal implementation uses the Libevent library for creating the DNSPort because the Unbound library has no functionality for setting up listening sockets. However, the Libevent API has no functionality for sending the DNS packet data from the exit relay through the DNSPort. Retrieving access to the DNSPort sockets involves using pointer manipulation which is fragile and breaks when Libevent changes its internal data structures. Our suggestion is to create the listening sockets within Tor and to stop offloading this task to the Libevent library.

In Section 4.4.1 we suspect Unbound of adding latency by using the process of elimination. Finding the code responsible for the latency was out of our scope and remains an open question. Answering this question and improving the responsible code would decrease the latency and increase the performance of the proposal implementation.

The asynchronous implementation depends on its processing cycles to have reasonable performance. Decreasing the delay between processing cycles increases the amount CPU time other processing functions use. Moving DNS resolution to a separate thread would increase the performance of the DNS resolution without having to decrease the delay between processing cycles.

The lack of fingerprinting prevention techniques impacts the level of anonymity that the proposal implementation provides. Our suggestion is to replace the Unbound library with the ldns library and implement the fingerprinting prevention techniques.

A

TOR FEATURES

Tor improves upon onion routing with features that increase security and anonymity (see Section 1.1.5). The following sections explain these features in detail.

A.1 PERFECT FORWARD SECRECY

Electronic communication is able to remain confidential when people use encryption. Encryption algorithms are either symmetric or asymmetric. Symmetric encryption uses one key for encryption and decryption, while asymmetric encryption uses public keys for encryption and private keys for decryption. Furthermore, symmetric encryption is efficient when encrypting large amounts of data but requires the exchange of keys. Asymmetric encryption has opposing properties (i.e. obviates key exchanges but is inefficient when encrypting large amounts of data).

Hybrid cryptographic systems nullify the limitations of these encryption types by using symmetric encryption for the data and asymmetric encryption for the symmetric keys. The combination requires no key exchange and the encryption of large amounts of data is efficient. It is because of these properties that the TLS protocol [34] and the OpenPGP standard [19] use hybrid cryptographic systems. However, when private key disclosures occur hybrid cryptographic systems are fragile because they provide no protection against retroactive decryption.

A.1.1 *Diffie-Hellman-Merkle key exchange*

The ephemeral Diffie-Hellman-Merkle (DHM) key exchange[1] prevents retroactive decryption by using the private key only for authentication and separate ephemeral keys for encryption [35]. Negotiation of the encryption key is at the start of each session. Furthermore, the encryption key is unique for each session and ephemeral[2]. The properties of the ephemeral DHM key exchange guarantee forward secrecy. Independent generation of encryption keys and preventing

---

[1] By using the term Diffie-Hellman-Merkle key exchange instead of Diffie-Hellman key exchange we '[...] recognise Merkle's equal contribution to the invention of public key cryptography' [55].

[2] The article *How to botch TLS forward secrecy* by Langley discusses how software designs affect the properties of ephemeral encryption keys [67].

one encryption key disclosure from affecting other encryption keys guarantees PFS.

A.1.2 *Key revocation*

PFS is susceptible to impersonation attacks because it uses long-term private keys for authentication. Private key disclosures make it possible for third parties to impersonate the owner and perform MITM attacks. Certificate revocation lists [24] and the Online Certificate Status Protocol (OCSP) [101] try to prevent impersonation attacks by notifying users of key compromises and key rotations. However, these solutions have issues with regards to security, scalability, and performance [68].

A.2 INTEGRITY CHECKING

The first iteration of onion routing has no integrity checking which makes it susceptible to MITM attacks. Tor solves this issue by using the TLS protocol, which ensures data integrity between two endpoints (in this case, two onion routers) [34, section 5]. However, onion routing works by relaying data between onion routers to hide the relation between the source and destination of messages. The TLS protocol permits malicious onion routers within the circuit to modify cells before forwarding them to the next onion router. Tor addresses this issue by using message digests Df and Db. The HMAC-based KDF [65] uses the 256-bit Secure Hash Algorithm (SHA-256) [18] and initialises Df and Db. The payloads in the relay cells which the onion router sends and receives amends Df and Db, respectively. Onion routers include the first four bytes of Df when sending relay cells and verify the integrity of relay cells they receive with Db.

A.3 DIRECTORY AUTHORITIES

The directory authorities keep track of the state of the network by receiving server descriptors and extra-info documents from the onion routers. For integrity and authentication purposes the onion routers sign the server descriptors and extra-info documents with their private key. Every hour the directory authorities submit votes to the rest of the directory authorities. The votes represent the state of the network according to each individual directory authority. The directory authorities sign their votes with their private key so the integrity and authenticity is verifiable. Each directory authority adds the consensus methods they support to their vote (by their numeric value). The mutual consensus method is the highest value for which two-thirds of the voting directory authorities has support. The consensus algorithm uses the mutual consensus method to generate deterministic consensus status documents. Each directory authority generates a cryptographic signature for each consensus status document and distributes it among the other directory authorities. There are 18 consensus methods which add increasing amounts of extra information to the consensus status documents [114, section 3.8.1].

Moreover, there are 10 directory authorities [93, lines 826–851], one of which acts as the only bridge relay directory authority in the network [36, line 99]. Directory caches redistribute the consensus status documents to decrease the load on the directory authorities. Onion routers are by default directory caches. The cryptographic signatures make the redistribution of consensus status documents and server descriptors by malicious directory caches detectable.

The consensus status documents and server descriptors are public domain and there are applications that analyse the data and detect anomalies within the Tor network. One such application is DocTor [43] which detects Sybil attacks [44, 94] and issues with the consensus status documents and server descriptors. Another application, exitmap [126], detects malicious and unreliable exit relays.

A.4 EXIT POLICIES

The last onion router in a circuit is the exit relay. Exit relays are on the edge of the Tor network and connect to destinations outside of the Tor network. The onion proxy initiates connections by sending relay cells to exit relays and uses circuits to stay anonymous. Relay cells include the FQDN or IP address of the destination, and a port number. Upon receiving the relay cell, the exit relay resolves the IP address, connects to it, and notifies the onion proxy about the status of the stream. The stream virtually connects the onion proxy to the destination server with its communication going through the circuit.

The exit relays directly communicate with servers outside the Tor network. These servers could assume that the exit relays are the source of the requests, leading to Digital Millennium Copyright Act notifications [72] and arrests [113]. Exit policies offer exit relay operators the capability to describe which IP addresses and ports are accessible through their exit relays. The default exit policy blacklists the private IP address space, the IP address of the exit relay and a range of port numbers [118]. These limitations prevent users from communicating with devices within the local area network (LAN) of the exit relay or with other networking applications running on the exit relay. The port numbers restrict electronic mail and file sharing.

The default exit policy is a blacklist which blocks its entries and permits everything else. The alternative is the reduced exit policy which is a whitelist that '[. . .] allows as many Internet services as possible while still blocking the majority of TCP ports' [109]. Additionally, it blocks addresses within the private IP address space and the IP address of the exit relay.

A.5 HIDDEN SERVICES

The objective of onion routing is hiding the relation between the source and destination of messages. The first iteration of onion routing hides the relation by making it difficult to find the source IP addresses. Tor extends anonymity by offering the same protection to destinations in the form of hidden services.

Hidden services enable destinations to hide their IP addresses by placing them within the `.onion` top-level domain (TLD) which makes them only reachable through Tor. Generating a FQDN within the `.onion` TLD involves taking the first half of the 160-bit Secure Hash Algorithm [18] digest of the public key and encoding the result with Base32 [62]. Using public keys for FQDN generation ensures that the FQDNs are self-authenticating. The FQDN generation algorithm outputs FQDNs with 16 characters in the range `a-z2-7` (in accordance with Base32) and with the suffix `.onion`. For example, the FQDNs of the search engine DuckDuckGo, the social networking service Facebook, and bitcoin service Blockchain are `http://3g2upl4pq6kufc4m.onion/` [124], `https://blockchainbdgpzk.onion/` [27] and `https://facebookcorewwwi.onion/` [83], respectively.

B

APPLICATION CONFIGURATIONS

The shell script in Listing B.1 generates the results in Tables 3.1 to 3.3. The script requires one parameter which specifies whether to use the DNS server from Google Inc., OpenDNS, or the local Tor DNSPort.

The Tor configuration file in Listing B.2 contains default settings and supplements the configuration files in Listings B.3 and B.4. The file specifies that the Tor instances act as non-exit relays, are running in the foreground, and avoid disk writes when possible. Listing B.3 contains the Tor configuration file for opening the DNSPort that listens on port 5301, and changes logging to include DNS requests. Listing B.4 contains the Tor configuration file for opening the SocksPort that listens on port 9050 (which is the default), and changes logging to be able to verify that the exit relay is responding to requests.

The Unbound configuration file in Listing B.5 contains the default settings and supplements the configuration files in Listings B.6 and B.7. The file specifies that the Unbound instances have no cache, are running in the foreground, listen on port 5300, and only use Internet Protocol version 4 (IPv4). Listing B.6 contains the Unbound configuration file for forwarding requests to the DNSPort of the Tor instance using Listing B.3. Listing B.7 contains the Unbound configuration file for forwarding requests to the DNS servers from OpenNIC [87]. Unbound has no support for SOCKS so the application wrapper torsocks redirects outgoing network traffic from Unbound to the SocksPort of the Tor instance using Listing B.3. Listing B.8 contains the Unbound configuration file returning the IPv4 address 127.0.0.1 for the FQDNs from Section 3.1.

```
#!/usr/bin/env sh

# The top 3 from the Alexa Top 500 Global Sites on 25-03-2015.
domains="google.com facebook.com youtube.com example.org"

# Only display the resolved IP addresses and the DNS server.
dig_flags="+short"

if test "x${1}" = "x"; then
    printf -- "Usage: %s [google|opendns|tor]\n" "${0}"
    exit 1
elif test "x${1}" = "xgoogle"; then
    dnsserver="8.8.8.8"
    dnsport="53"
    tcp="+notcp"
elif test "x${1}" = "xopendns"; then
    dnsserver="208.67.222.222"
    dnsport="53"
    tcp="+notcp"
elif test "x${1}" = "xtor"; then
    dnsserver="127.0.0.1"
    dnsport="5300"
```

```
    tcp="+tcp"
fi

printf -- "Server: %s:%s\n" "${dnsserver}" "${dnsport}"
for domain in ${domains}; do
    printf -- "Domain: %s\n" "${domain}"
    ips="$(dig "${dig_flags}" "${tcp}" -p "${dnsport}" @"${dnsserver}" \
        "${domain}" A "${domain}" AAAA | grep -v "^;;" | cut -f 1 -d ’ ’ | \
        LC_ALL=C sort -u -t . -k 1,1n -k 2,2n -k 3,3n -k 4,4n)"
    for ip in ${ips}; do
        reverse="$(dig "${dig_flags}" "${tcp}" -p "${dnsport}" \
            @"${dnsserver}" -x "${ip}" | grep -v "^;;" | cut -f 1 -d ’ ’ | \
            LC_ALL=C sort -u)"
        printf -- "IP: %s (%s)\n" "${ip}" "${reverse}"
    done
done
```

Listing B.1: Shell script for performing DNS resolution on the top 3 from the *Alexa Top 500 Global Sites* and one from the IANA list [2]. The DNS servers are from Google Inc., OpenDNS, or the local Tor DNSPort.

```
ExitRelay 0
ExitPolicy reject *:*
RunAsDaemon 0
AvoidDiskWrites 1
```

Listing B.2: The configuration file with defaults values for Tor options. The Tor instance with the open DNSPort and the Tor instance with the open SocksPort share these settings.

```
SOCKSPort 0
DNSPort 5301
Log [app]info stderr
LogMessageDomains 1

# This exit node is known to do IPv6 address resolution.
ExitNodes B7EC0C02D7D9F1E31B0C251A6B058880778A0CD1
```

Listing B.3: The configuration file for running the Tor instance with an open DNSPort. The logging settings make it possible to see the DNS requests.

```
Log [net]info stderr
Log [circ]info stderr
LogMessageDomains 1
```

Listing B.4: The configuration file for running the Tor instance with an open SocksPort. The logging settings make it possible to verify that the exit relay is responding to request.

```
server:
    verbosity: 2
    port: 5300
    cache-min-ttl: 0
    cache-max-ttl: 0
    do-daemonize: no
    chroot: ""
    username: ""
    logfile: ""
    log-time-ascii: yes
    log-queries: yes
```

```
    hide-identity: yes
    hide-version: yes
    do-ip4: yes
    do-ip6: no
```

Listing B.5: The configuration file which contains the default Unbound options. The Unbound instances that forward requests to the open DNSPort and the external DNS server include this file into their configuration files.

```
include: "unbound.conf"

server:
    root-hints: "named.cache"
    do-not-query-localhost: no
forward-zone:
    name: "."
    forward-addr: 127.0.0.1@5301
```

Listing B.6: The configuration file for running the Unbound instance that forwards requests to the open DNSPort.

```
include: "unbound.conf"

server:
    root-hints: "opennic.hints"
    # Disable udp because torsocks does not allow it.
    do-udp: no
    do-tcp: yes
    tcp-upstream: yes
forward-zone:
    name: "."
    forward-addr: 192.71.249.249@53
```

Listing B.7: The configuration file for running the Unbound instance that forwards requests to the external DNS server from the list of Public Access (Tier 2) servers that OpenNIC [87] offers. The torsocks application forwards outgoing DNS requests by Unbound through Tor.

```
include: "unbound.conf"

server:
    interface: 127.0.1.2
    root-hints: "named.cache"
    do-not-query-localhost: no

local-zone: "google.com" redirect
local-data: "google.com A 127.0.0.1"
local-zone: "facebook.com" redirect
local-data: "facebook.com A 127.0.0.1"
local-zone: "youtube.com" redirect
local-data: "youtube.com A 127.0.0.1"
local-zone: "example.org" redirect
local-data: "example.org A 127.0.0.1"
```

Listing B.8: The configuration file for running Unbound instances that return the IPv4 address 127.0.0.1 for the FQDNs from Section 3.1.

C

CALL STACKS

The call stacks in this chapter implement the current DNS functionality which is dividable into two categories with each three procedures, see Table 3.4.

C.1 ONION PROXY FUNCTIONALITY

The call stacks in this section implement the current DNS functionality of the onion proxy. The call stack in Table C.1 sets up the DNSPort to be able to receive DNS requests from users. The call stack in Table C.2 processes DNS requests from the DNSPort and forwards them to the exit relay. Finally, the call stack in Table C.3 processes DNS responses from the exit relay and forwards them to the DNSPort.

FILE	FUNCTION
`tor/src/or/tor_main.c`	main
`tor/src/or/main.c`	tor_main
`tor/src/or/main.c`	tor_init
`tor/src/or/config.c`	options_init_from_torrc
`tor/src/or/config.c`	options_init_from_string
`tor/src/or/config.c`	set_options
`tor/src/or/config.c`	options_act_reversible
`tor/src/or/connection.c`	retry_all_listeners
`tor/src/or/connection.c`	retry_listener_ports
`tor/src/or/connection.c`	connection_listener_new
`tor/src/or/dnsserv.c`	dnsserv_configure_listener
`libevent/evdns.c`	evdns_add_server_port_with_base

Table C.1: The call stack when setting up the DNSPort so it is able to receive Domain Name System requests from users (`PROC01`).

FILE	FUNCTION
`tor/src/or/tor_main.c`	main
`tor/src/or/main.c`	tor_main
`tor/src/or/main.c`	do_main_loop

`tor/src/or/main.c`	run_main_loop_until_done
`tor/src/or/main.c`	run_main_loop_once
`libevent/event.c`	event_base_loop
`libevent/event.c`	event_process_active
`libevent/event.c`	event_process_active_single_queue
`libevent/event.c`	event_persist_closure
`libevent/evdns.c`	server_port_ready_callback
`libevent/evdns.c`	server_port_read
`libevent/evdns.c`	request_parse
`tor/src/or/dnsserv.c`	evdns_server_callback
`tor/src/or/connection_edge.c`	connection_ap_rewrite_and_attach_if_allowed
`tor/src/or/connection_edge.c`	connection_ap_handshake_rewrite_and_attach
`tor/src/or/circuituse.c`	connection_ap_handshake_attach_circuit
`tor/src/or/circuituse.c`	connection_ap_handshake_attach_chosen_circuit
`tor/src/or/connection_edge.c`	connection_ap_handshake_send_resolve
`tor/src/or/relay.c`	connection_edge_send_command
`tor/src/or/relay.c`	relay_send_command_from_edge_
`tor/src/or/relay.c`	circuit_package_relay_cell
`tor/src/or/relay.c`	append_cell_to_circuit_queue
`tor/src/or/scheduler.c`	scheduler_channel_has_waiting_cells
`tor/src/or/scheduler.c`	scheduler_retrigger
`libevent/event.c`	event_active

Table C.2: The call stack when processing Domain Name System requests from the DNSPort and forwarding them to the exit relay (`PROC02`).

FILE	FUNCTION
`tor/src/or/tor_main.c`	main
`tor/src/or/main.c`	tor_main

`tor/src/or/main.c`	do_main_loop
`tor/src/or/main.c`	run_main_loop_until_done
`tor/src/or/main.c`	run_main_loop_once
`libevent/event.c`	event_base_loop
`libevent/event.c`	event_process_active
`libevent/event.c`	event_process_active_single_queue
`libevent/event.c`	event_persist_closure
`tor/src/or/main.c`	conn_read_callback
`tor/src/or/connection.c`	connection_handle_read
`tor/src/or/connection.c`	connection_handle_read_impl
`tor/src/or/connection.c`	connection_process_inbuf
`tor/src/or/connection_or.c`	connection_or_process_inbuf
`tor/src/or/connection_or.c`	connection_or_process_cells_from_inbuf
`tor/src/or/channeltls.c`	channel_tls_handle_cell
`tor/src/or/channel.c`	channel_queue_cell
`tor/src/or/command.c`	command_process_cell
`tor/src/or/command.c`	command_process_relay_cell
`tor/src/or/relay.c`	circuit_receive_relay_cell
`tor/src/or/relay.c`	connection_edge_process_relay_cell
`tor/src/or/relay.c`	connection_edge_process_relay_cell_not_open
`tor/src/or/relay.c`	connection_edge_process_resolved_cell
`tor/src/or/relay.c`	connection_ap_handshake_socks_got_resolved_cell
`tor/src/or/connection_edge.c`	connection_ap_handshake_socks_resolved_addr
`tor/src/or/connection_edge.c`	connection_ap_handshake_socks_resolved
`tor/src/or/dnsserv.c`	dnsserv_resolved

`libevent/evdns.c`	evdns_server_request_add_aaaa_reply or evdns_server_request_add_a_reply or evdns_server_request_add_ptr_reply (depends on the type of the DNS request) (called in dnsserv_resolved before calling evdns_server_request_respond)
`libevent/evdns.c`	evdns_server_request_respond
`system call`	sendto

Table C.3: The call stack when processing Domain Name System responses from the exit relay and forwarding them to the DNSPort (`PROC03`).

C.2 EXIT RELAY FUNCTIONALITY

The call stacks in this section implement the current DNS functionality of the exit relay. The call stack in Table C.4 sets up the listening socket (or ORPort) to be able to receive DNS requests from onion proxies. The call stack in Table C.5 processes DNS requests from the ORPort and forwards them to the DNS server. Finally, the call stack in Table C.6 processes DNS responses from the DNS server and forwards them to the ORPort.

FILE	FUNCTION
`tor/src/or/tor_main.c`	main
`tor/src/or/main.c`	tor_main
`tor/src/or/main.c`	tor_init
`tor/src/or/config.c`	options_init_from_torrc
`tor/src/or/config.c`	options_init_from_string
`tor/src/or/config.c`	set_options
`tor/src/or/config.c`	options_act_reversible
`tor/src/or/connection.c`	retry_all_listeners
`tor/src/or/connection.c`	retry_listener_ports
`tor/src/or/connection.c`	connection_listener_new
`tor/src/or/main.c`	connection_add_impl
`libevent/event.c`	event_new

Table C.4: The call stack when setting up the listening socket (or ORPort) so it is able to receive Domain Name System requests from onion proxies (`PROC04`).

FILE	FUNCTION
`tor/src/or/tor_main.c`	main
`tor/src/or/main.c`	tor_main
`tor/src/or/main.c`	do_main_loop
`tor/src/or/main.c`	run_main_loop_until_done
`tor/src/or/main.c`	run_main_loop_once
`libevent/event.c`	event_base_loop
`libevent/event.c`	event_process_active
`libevent/event.c`	event_process_active_single_queue
`libevent/event.c`	event_persist_closure
`tor/src/or/main.c`	conn_read_callback
`tor/src/or/connection.c`	connection_handle_read
`tor/src/or/connection.c`	connection_handle_read_impl
`tor/src/or/connection.c`	connection_process_inbuf
`tor/src/or/connection_or.c`	connection_or_process_inbuf
`tor/src/or/connection_or.c`	connection_or_process_cells_from_inbuf
`tor/src/or/channeltls.c`	channel_tls_handle_cell
`tor/src/or/channel.c`	channel_queue_cell
`tor/src/or/command.c`	command_process_cell
`tor/src/or/command.c`	command_process_relay_cell
`tor/src/or/relay.c`	circuit_receive_relay_cell
`tor/src/or/relay.c`	connection_edge_process_relay_cell
`tor/src/or/connection_edge.c`	connection_exit_begin_resolve
`tor/src/or/dns.c`	dns_resolve
`tor/src/or/dns.c`	dns_resolve_impl
`tor/src/or/dns.c`	launch_resolve
`tor/src/or/dns.c`	launch_one_resolve
`libevent/evdns.c`	evdns_base_resolve_ipv4 or evdns_base_resolve_ipv6 or evdns_base_resolve_reverse or evdns_base_resolve_reverse_ipv6 (depends on the type of the DNS request)

Table C.5: The call stack when processing Domain Name System requests from the ORPort and forwarding them to the Domain Name System server (`PROC05`).

FILE	FUNCTION
`tor/src/or/tor_main.c`	main
`tor/src/or/main.c`	tor_main
`tor/src/or/main.c`	do_main_loop
`tor/src/or/main.c`	run_main_loop_until_done
`tor/src/or/main.c`	run_main_loop_once
`libevent/event.c`	event_base_loop
`libevent/event.c`	event_process_active
`libevent/event.c`	event_process_active_single_queue
`libevent/evdns.c`	reply_run_callback
`tor/src/or/dns.c`	evdns_callback
`tor/src/or/dns.c`	dns_found_answer
`tor/src/or/dns.c`	inform_pending_connections
`tor/src/or/dns.c`	send_resolved_cell or send_resolved_hostname_cell (depends on the type of the DNS request)
`tor/src/or/relay.c`	connection_edge_send_command
`tor/src/or/relay.c`	relay_send_command_from_edge_
`tor/src/or/relay.c`	circuit_package_relay_cell
`tor/src/or/relay.c`	append_cell_to_circuit_queue
`tor/src/or/scheduler.c`	scheduler_channel_has_waiting_cells
`tor/src/or/scheduler.c`	scheduler_retrigger
`libevent/event.c`	event_active

Table C.6: The call stack when processing Domain Name System responses from the Domain Name System server and forwarding them to the ORPort (`PROC06`).

D

DOMAIN NAME SYSTEM PACKETS

The following tables describe the hexadecimal representations and dissections of DNS packets when attempting to resolve RR A for the FQDN `example.org`.

D.1 DOMAIN NAME SYSTEM REQUEST

OFFSET	HEXADECIMAL	ASCII
`00000000`	`00 1d 30 39 05 20 00 01 00 00` `00 00 00 00 07 65`	`..09. .........e`
`00000010`	`78 61 6d 70 6c 65 03 6f 72 67` `00 00 01 00 01`	`xample.org.....`
`0000001f`		

Table D.1: The hexadecimal representation of the Domain Name System request for the resource record A of `example.org`.

FIELD	SUBFIELD	DATA	VALUE
Packet length		`0x001d`	29 (only added when using TCP)
ID		`0x3039`	12345
Flags		`0x0520`	
	QR		0
	OPCODE		0
	AA		1
	TC		0
	RD		1
	RA		0
	Z		0
	AD		1
	CD		0
	RCODE		0
QDCOUNT		`0x0001`	1
ANCOUNT		`0x0000`	0

NSCOUNT	`0x0000`	0
ARCOUNT	`0x0000`	0

Table D.2: The dissection showing the fields inside the header section of the Domain Name System packet from Table D.1.

FIELD	SUBFIELD	DATA	VALUE
QNAME		`0x076578616d706c65036f726700`	
	Length	`0x07`	7
	Name	`0x6578616d706c65`	example
	Length	`0x03`	3
	Name	`0x6f7267`	org
	Length	`0x00`	0
QTYPE		`0x0001`	1
QCLASS		`0x0001`	1

Table D.3: The dissection showing the fields inside the question section of the Domain Name System packet from Table D.1.

D.2 DOMAIN NAME SYSTEM RESPONSE

OFFSET	HEXADECIMAL	ASCII
`00000000`	`00 b5 30 39 81 80 00 01 00 01 00 02 00 04 07 65`	`..09...........e`
`00000010`	`78 61 6d 70 6c 65 03 6f 72 67 00 00 01 00 01 c0`	`xample.org......`
`00000020`	`0c 00 01 00 01 00 00 00 05 00 04 5d b8 d8 22 c0`	`...........].."." `
`00000030`	`0c 00 02 00 01 00 00 00 04 00 14 01 61 0c 69 61`	`............a.ia`
`00000040`	`6e 61 2d 73 65 72 76 65 72 73 03 6e 65 74 00 c0`	`na-servers.net..`
`00000050`	`0c 00 02 00 01 00 00 00 04 00 04 01 62 c0 3b c0`	`............b.;.`
`00000060`	`39 00 01 00 01 00 00 03 88 00 04 c7 2b 84 35 c0`	`9...........+.5.`
`00000070`	`39 00 1c 00 01 00 00 03 88 00 10 20 01 05 00 00`	`9.......... ....`

```
00000080  8c 00 00 00 00 00 00 00 00 00   ..........S.Y...
          53 c0 59 00 01 00
00000090  01 00 00 03 88 00 04 c7 2b 85   ........+.5.Y...
          35 c0 59 00 1c 00
000000a0  01 00 00 03 88 00 10 20 01 05   ....... ........
          00 00 8d 00 00 00
000000b0  00 00 00 00 00 00 53            ......S|
000000b7
```

Table D.4: The hexadecimal representation of the Domain Name System response as answer to the Domain Name System request in Table D.1.

FIELD	SUBFIELD	DATA	VALUE
Packet length		`0x00b5`	181 (only added when using TCP)
ID		`0x3039`	12345
Flags		`0x8180`	
	QR		1
	OPCODE		0
	AA		0
	TC		0
	RD		1
	RA		1
	Z		0
	AD		0
	CD		0
	RCODE		0
QDCOUNT		`0x0001`	1
ANCOUNT		`0x0001`	1
NSCOUNT		`0x0002`	2
ARCOUNT		`0x0004`	4

Table D.5: The dissection showing the fields inside the header section of the Domain Name System packet from Table D.4.

FIELD	SUBFIELD	DATA	VALUE
QNAME		`0x076578616d706c-65036f726700`	

	Length	`0x07`	7
	Name	`0x6578616d706c65`	example
	Length	`0x03`	3
	Name	`0x6f7267`	org
	Length	`0x00`	0
QTYPE		`0x0001`	1
QCLASS		`0x0001`	1

Table D.6: The dissection showing the fields inside the question section of the Domain Name System packet from Table D.4.

FIELD	SUBFIELD	DATA	VALUE
QNAME		`0xc00c`	
	Pointer	`0b11`	
	Offset	`0b1100`	12 or example.org
QTYPE		`0x0001`	1 or A
QCLASS		`0x0001`	1 or IN
TTL		`0x00000005`	5
RDLENGTH		`0x0004`	4
RDATA		`0x5db8d822`	1572395042 or IPv4 93.184.216.34

Table D.7: The dissection showing the fields inside the answer section of the Domain Name System packet from Table D.4.

FIELD	SUBFIELD	DATA	VALUE
QNAME		`0xc00c`	
	Pointer	`0b11`	
	Offset	`0b1100`	12 or example.org
QTYPE		`0x0002`	2 or NS
QCLASS		`0x0001`	1 or IN
TTL		`0x00000004`	4
RDLENGTH		`0x0014`	20
RDATA		`0x01610c69616e612d73657276657273036e657400`	
	Length	`0x01`	1

	Name	`0x61`	a
	Length	`0x0c`	12
	Name	`0x69616e612d73657276657273`	iana-servers
	Length	`0x03`	3
	Name	`0x6e6574`	net
	Length	`0x00`	0

Table D.8: The dissection showing the fields inside the first entry of the authority section of the Domain Name System packet from Table D.4.

FIELD	SUBFIELD	DATA	VALUE
QNAME		`0xc00c`	
	Pointer	`0b11`	
	Offset	`0b1100`	12 or example.org
QTYPE		`0x0002`	2 or NS
QCLASS		`0x0001`	1 or IN
TTL		`0x00000004`	4
RDLENGTH		`0x0004`	4
RDATA		`0x0162c03b`	
	Length	`0x01`	1
	Name	`0x62`	b
	Pointer	`0b11`	
	Offset	`0b111011`	59 or iana-servers.net

Table D.9: The dissection showing the fields inside the second entry of the authority section of the Domain Name System packet from Table D.4.

FIELD	SUBFIELD	DATA	VALUE
QNAME		`0xc039`	
	Pointer	`0b11`	
	Offset	`0b111001`	57 or a.iana-servers.net
QTYPE		`0x0001`	1 or A
QCLASS		`0x0001`	1 or IN
TTL		`0x00000388`	904
RDLENGTH		`0x0004`	4

RDATA		0xc72b8435	3341517877 or IPv4 199.43.132.53

Table D.10: The dissection showing the fields inside the first entry of the additional section of the Domain Name System packet from Table D.4.

FIELD	SUBFIELD	DATA	VALUE
QNAME		0xc039	
	Pointer	0b11	
	Offset	0b111001	57 or a.iana-servers.net
QTYPE		0x001c	28 or AAAA
QCLASS		0x0001	1 or IN
TTL		0x00000388	904
RDLENGTH		0x0010	16
RDATA		0x20010500008c00000000000000000053	IPv6 2001:0500:008c:0000:0000:0000:0000:0053 or IPv6 2001:500:8c::53

Table D.11: The dissection showing the fields inside the second entry of the additional section of the Domain Name System packet from Table D.4.

FIELD	SUBFIELD	DATA	VALUE
QNAME		0xc059	
	Pointer	0b11	
	Offset	0b1011001	89 or b.iana-servers.net
QTYPE		0x0001	1 or A
QCLASS		0x0001	1 or IN
TTL		0x00000388	904
RDLENGTH		0x0004	4
RDATA		0xc72b8535	3341518133 or IPv4 199.43.133.53

Table D.12: The dissection showing the fields inside the third entry of the additional section of the Domain Name System packet from Table D.4.

FIELD	SUBFIELD	DATA	VALUE
QNAME		`0xc059`	
	Pointer	`0b11`	
	Offset	`0b1011001`	89 or b.iana-servers.net
QTYPE		`0x001c`	28 or AAAA
QCLASS		`0x0001`	1 or IN
TTL		`0x00000388`	904
RDLENGTH		`0x0010`	16
RDATA		`0x20010500008d00000000000000000053`	IPv6 2001:0500:008d:0000:0000:0000:0000:0053 or IPv6 2001:500:8d::53

Table D.13: The dissection showing the fields inside the fourth entry of the additional section of the Domain Name System packet from Table D.4.

E

RAW RESULTS

Figure E.1 graphs the differences between the average latency in milliseconds, and the minimum and maximum values. The maximum values add biases towards higher average latencies which Section 4.4.1 explains further. Figures E.2a to E.2d show the latency samples in milliseconds per FQDN which are the basis for the average latencies in Figures 4.1 and E.1.

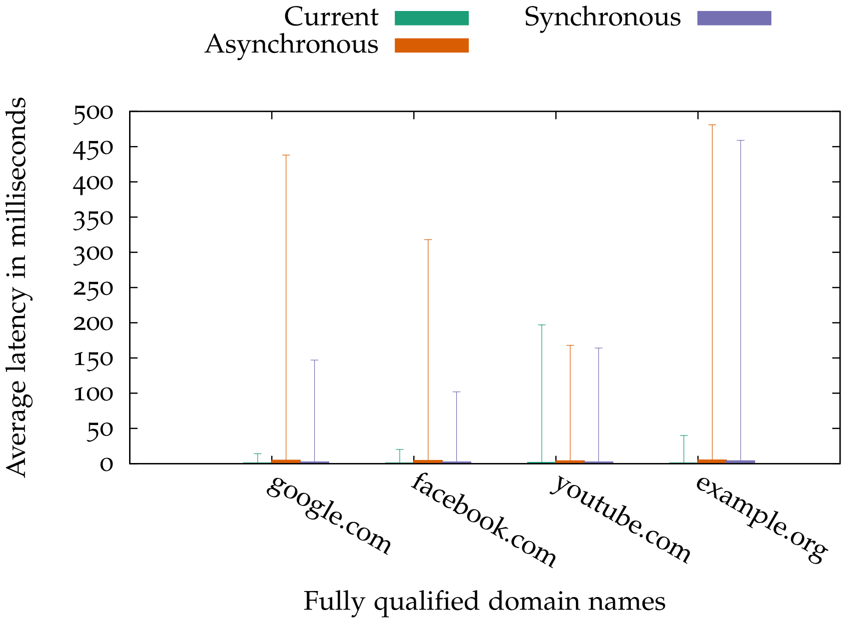
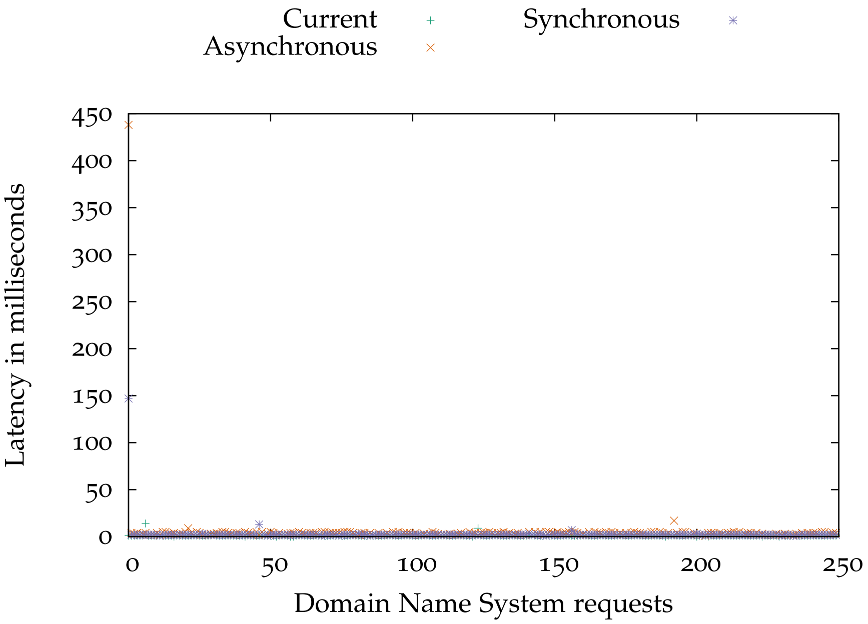
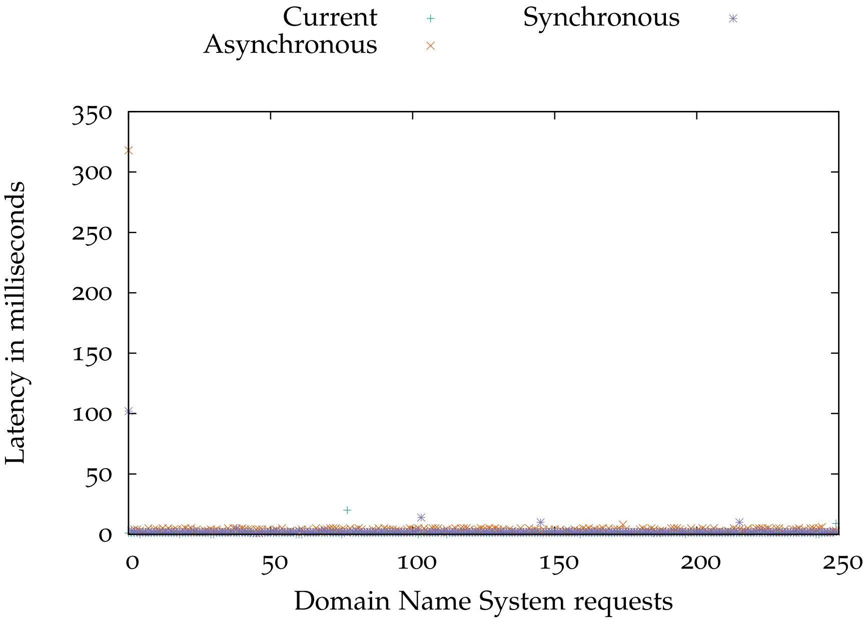


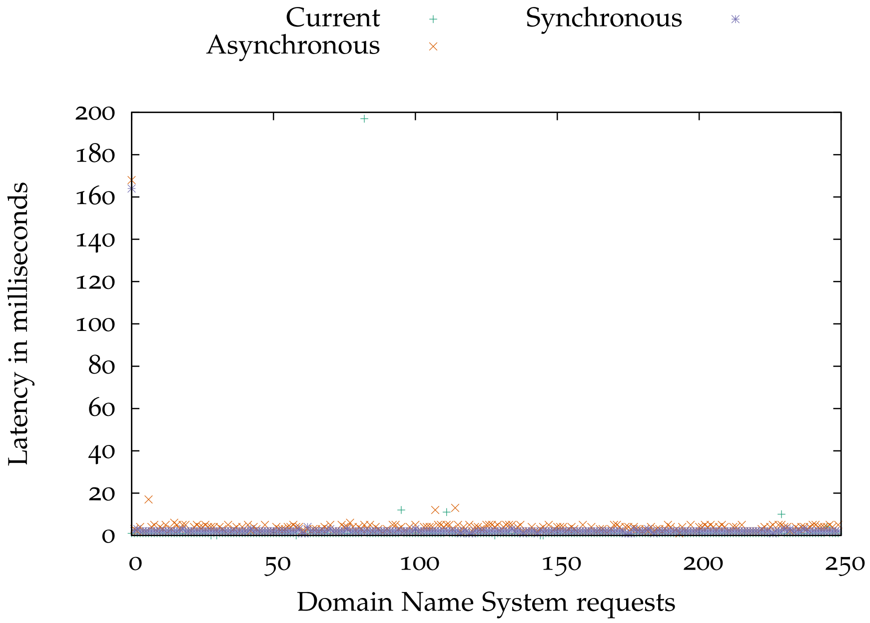
Figure E.1: The average latencies in milliseconds including the minimum and maximum values when resolving the resource record A 250 times for each fully qualified domain name and implementation. The fully qualified domain names are the top 3 from the *Alexa Top 500 Global Sites* and one from the IANA list [2]. The implementations are the current Domain Name System implementation, and the asynchronous and synchronous proposal implementations.



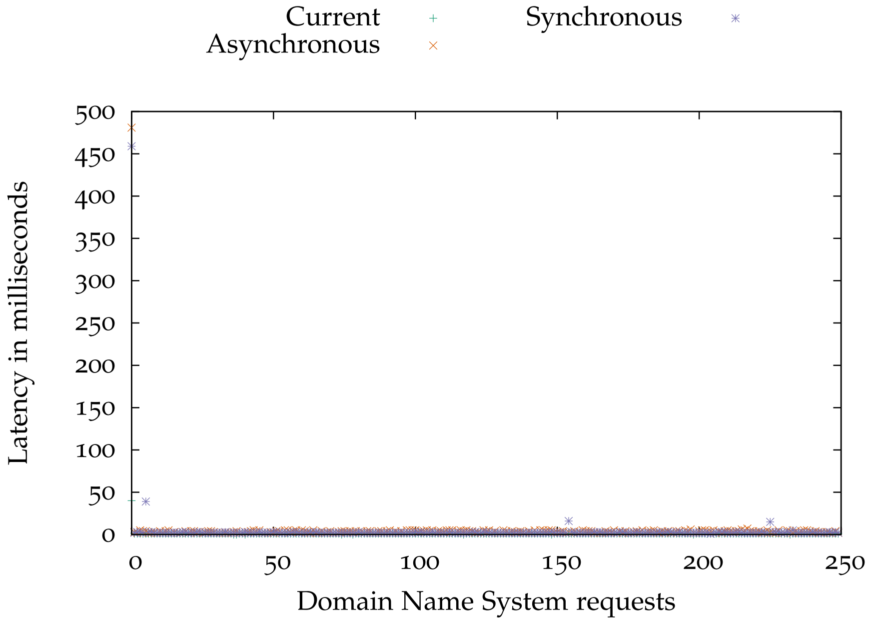
(a) The latencies in milliseconds when resolving `google.com`.



(b) The latencies in milliseconds when resolving `facebook.com`.



(c) The latencies in milliseconds when resolving `youtube.com`.



(d) The latencies in milliseconds when resolving `example.org`.

Figure E.2: The latencies in milliseconds when resolving the resource record A for each fully qualified domain name and implementation. The fully qualified domain names are the top 3 from the *Alexa Top 500 Global Sites* and one from the IANA list [2]. The implementations are the current Domain Name System implementation, and the asynchronous and synchronous proposal implementations.

BIBLIOGRAPHY

[1] Donald E. Eastlake 3rd. *RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS)*. Request for Comments 3110. Network Working Group. May 2001. DOI: 10.17487/RFC3110 (cited on page 31).

[2] Donald E. Eastlake 3rd and Aliza R. Panitz. *Reserved Top Level DNS Names*. Request for Comments 2606. Network Working Group. June 1999. DOI: 10.17487/RFC2606 (cited on pages xiv, 15, 16, 29, 42, 60, 62).

[3] Masoud Akhoondi, Curtis Yu and Harsha V. Madhyastha. 'LASTor: A Low-Latency AS-Aware Tor Client'. In: *Proceedings of the 2012 IEEE Symposium on Security and Privacy*. (San Francisco, California, United States of America). May 2012, pages 476–490. DOI: 10.1109/SP.2012.35 (cited on pages 6, 7).

[4] *Alexa Top 500 Global Sites*. Alexa. March 2015. URL: http://www.alexa.com/topsites (visited on 26/03/2015) (cited on pages xiv, 15, 16, 29, 42, 60, 62).

[5] Jacob Appelbaum, Marsh Ray, Ian Finder and Karl Koscher. 'vpwns: Virtual Pwned Networks'. In: *Proceedings of the 2nd USENIX Workshop on Free and Open Communications on the Internet*. Edited by Roger Dingledine and Joss Wright. USENIX Assocation, August 2012 (cited on page 3).

[6] Roy Arends, Rob Austein, Matt Larson, Dan Massey and Scott Rose. *DNS Security Introduction and Requirements*. Request for Comments 4033. Network Working Group. March 2005. DOI: 10.17487/RFC4033 (cited on pages 7, 11).

[7] Roy Arends, Rob Austein, Matt Larson, Dan Massey and Scott Rose. *Protocol Modifications for the DNS Security Extensions*. Request for Comments 4035. Network Working Group. March 2005. DOI: 10.17487/RFC4035 (cited on pages 7, 31).

[8] Roy Arends, Rob Austein, Matt Larson, Dan Massey and Scott Rose. *Resource Records for the DNS Security Extensions*. Request for Comments 4034. Network Working Group. March 2005. DOI: 10.17487/RFC4034 (cited on pages 7, 11, 13, 31).

[9] Roger Dingledine (arma). *"One cell is enough to break Tor's anonymity"*. The Tor Project, Inc. February 2009. URL: https://blog.torproject.org/blog/one-cell-enough (visited on 20/05/2016) (cited on page 26).

[10] Derek Atkins and Rob Austein. *Threat Analysis of the Domain Name System (DNS)*. Request for Comments 3833. Network Working Group. August 2004. DOI: `10.17487/RFC3833` (cited on page 11).

[11] Frits C. Bakker. *Advies Wetsvoorstel computercriminaliteit III*. July 2013 (cited on page 1).

[12] James Ball, Bruce Schneier and Glenn Greenwald. *NSA and GCHQ target Tor network that protects anonymity of web users*. October 2013. URL: `http://www.theguardian.com/world/2013/oct/04/nsa-gchq-attack-tor-network-encryption` (visited on 27/01/2015) (cited on page 6).

[13] Lois Beckett. *Everything We Know About What Data Brokers Know About You*. ProPublica. September 2013. URL: `https://www.propublica.org/article/everything-we-know-about-what-data-brokers-know-about-you` (visited on 30/04/2014) (cited on page 1).

[14] Ray Bellis. *DNS Transport over TCP – Implementation Requirements*. Request for Comments 5966. Internet Engineering Task Force (IETF). August 2010. DOI: `10.17487/RFC5966` (cited on page 13).

[15] Oliver Berthold, Hannes Federrath and Stefan Köpsell. 'Web MIXes: A System for Anonymous and Unobservable Internet Access'. In: *Designing Privacy Enhancing Technologies*. International Workshop on Design Issues in Anonymity and Unobservability. (International Computer Science Institute (ICSI), Berkeley, California, 25th–26th July 2000). Edited by Hannes Federrath. Volume 2009. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, pages 115–129. ISBN: 978-3-540-44702-3. DOI: `10.1007/3-540-44702-4_7` (cited on page 4).

[16] Robert Braden. *Requirements for Internet Hosts – Application and Support*. Request for Comments 1123. Network Working Group. October 1989. DOI: `10.17487/RFC1123` (cited on page 13).

[17] Robert Braden. *Requirements for Internet Hosts – Communication Layers*. Request for Comments 1122. Network Working Group. October 1989. DOI: `10.17487/RFC1122` (cited on page 1).

[18] John Bryson and Patrick Gallagher. *Secure Hash Standard (SHS)*. Federal Information Processing Standards Publication (FIPS PUB) 180-4. National Institute of Standards and Technology. March 2012. 37 pages (cited on pages 38, 40).

[19] Jon Callas, Lutz Donnerhacke, Hal Finney, David Shaw and Rodney Thayer. *OpenPGP Message Format*. Request for Comments 4880. Network Working Group. November 2007. DOI: `10.17487/RFC4880` (cited on page 37).

[20] Mike Cardwell. *Understanding DNSSEC*. December 2011. URL: https://grepular.com/Understanding_DNSSEC (visited on 28/06/2016) (cited on page 28).

[21] David Lee Chaum. 'Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms'. In: *Communications of the ACM* 24.2 (February 1981), pages 84–90. DOI: 10.1145/358549.358563 (cited on pages 3, 4).

[22] *chutney. The chutney tool for testing and automating Tor network setup*. The Tor Project, Inc. October 2015. URL: https://gitweb.torproject.org/chutney.git (visited on 10/11/2015) (cited on page 25).

[23] *Consensus health*. The Tor Project, Inc. May 2014. URL: https://consensus-health.torproject.org/ (visited on 30/05/2014) (cited on page 5).

[24] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley and Tim Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. Request for Comments 5280. Network Working Group. May 2008. DOI: 10.17487/RFC5280 (cited on page 38).

[25] Court of Justice of the European Union. *Judgment of the Court (Grand Chamber) of 8 April 2014*. April 2014 (cited on page 1).

[26] Court of Justice of the European Union. *Opinion of Mr Advocate General Cruz Villalón delivered on 12 December 2013*. December 2013 (cited on page 1).

[27] Nik Cubrilovic. *Securing Blockchain.info Users with Tor and SSL*. New Web Order. December 2014. URL: https://www.nikcub.com/posts/securing-blockchain-users-with-tor-and-ssl/ (visited on 04/12/2014) (cited on page 40).

[28] Joao Damas, Michael Graff and Paul Vixie. *Extension Mechanisms for DNS (EDNS(0))*. Request for Comments 6891. Internet Engineering Task Force (IETF). December 2012. DOI: 10.17487/RFC6891 (cited on page 31).

[29] George Danezis. 'Statistical Disclosure Attacks. Traffic Confirmation in Open Environments'. In: *Security and Privacy in the Age of Uncertainty*. International Information Security Conference (SEC2003). (Athens Chamber of Commerce and Industry, Greece, 26th–28th May 2003). Edited by Dimitris Gritzalis, Sabrina De Capitani di Vimercati, Pierangela Samarati and Sokratis Katsikas. Volume 122. IFIP Advances in Information and Communication Technology. International Federation for Information Processing (IFIP) Technical Committee (TC) 11. Springer US, 2003, pages 412–426. ISBN: 978-0-387-35691-4. DOI: 10.1007/978-0-387-35691-4_40 (cited on page 5).

[30] George Danezis, Roger Dingledine and Nick Mathewson. 'Mixminion: Design of a Type III Anonymous Remailer Protocol'. In: *Proceedings of the 2003 IEEE Symposium on Security and Privacy*. Edited by Bob Blakley and Lee Badger. May 2003, pages 2–15. DOI: `10.1109/SECPRI.2003.1199323` (cited on page 4).

[31] David. *Through A Network, Darkly. A Geographic Look At I2P*. January 2015. URL: `https://thetinhat.com/articles/2015/i2p-survey.html` (visited on 27/01/2015) (cited on page 6).

[32] Stephen E. Deering and Robert M. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. Request for Comments 2460. Network Working Group. December 1998. DOI: `10.17487/RFC2460` (cited on pages xv, xvi, 1).

[33] Caleb James DeLisle. *cjdns*. GitHub. May 2014. URL: `https://github.com/cjdelisle/cjdns` (visited on 08/05/2014) (cited on page 3).

[34] Tim Dierks and Eric Rescorla. *The Transport Layer Security (TLS) Protocol – Version 1.2*. Request for Comments 5246. Network Working Group. August 2008. DOI: `10.17487/RFC5246` (cited on pages 37, 38).

[35] Whitfield Diffie and Martin Edward Hellman. 'New directions in cryptography'. In: *IEEE Transactions on Information Theory* 22.6 (November 1976), pages 644–654. DOI: `10.1109/TIT.1976.1055638` (cited on page 37).

[36] Roger Dingledine. *Behavior for bridge users, bridge relays, and bridge authorities*. The Tor Project, Inc. May 2009. URL: `https://gitweb.torproject.org/torspec.git/tree/proposals/125-bridges.txt` (visited on 08/12/2014). Commit `ebc5a935ee4a-a0c123829706671a3f43da82f11f` (cited on page 39).

[37] Roger R. Dingledine, Michael Freedman, David Molnar, Brian Sniffen and Todd Kamin. 'The Free Haven Project: Design and Deployment of an Anonymous Secure Data Haven'. Master's thesis. Massachusetts Institute of Technology, 22nd May 2000 (cited on page 5).

[38] Roger Dingledine and Nick Mathewson. *Design of a blocking-resistant anonymity system*. Technical report. The Tor Project, Inc, November 2006 (cited on page xv).

[39] Roger Dingledine and Nick Mathewson. *Implements a local cache for DNS results for Tor servers*. The Tor Project, Inc. February 2016. URL: `https://gitweb.torproject.org/tor.git/tree/src/or/dns.c` (visited on 28/06/2016). Commit `57699de005b2-033a1886cecf7a1ed17e2bf95b81` (cited on page 17).

[40] Roger Dingledine and Nick Mathewson. *Tor Path Specification*. The Tor Project, Inc. April 2013. URL: https://gitweb.torproject.org/torspec.git/tree/path-spec.txt (visited on 08/12/2014). Commit ebc5a935ee4aa0c123829706671a3f43da82f11f (cited on pages xv, 6).

[41] Roger Dingledine and Nick Mathewson. *Tor Protocol Specification*. The Tor Project, Inc. August 2014. URL: https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt (visited on 08/12/2014). Commit ebc5a935ee4aa0c123829706671a3f43da82f11f (cited on pages xv, 12, 13).

[42] Roger Dingledine, Nick Mathewson and Paul Syverson. 'Tor: The Second-Generation Onion Router'. In: *Proceedings of the 13th USENIX Security Symposium*. Edited by Matt Blaze. August 2004, pages 303–320 (cited on pages 5, 26).

[43] *DocTor*. The Tor Project, Inc. October 2014. URL: https://gitweb.torproject.org/doctor.git (visited on 14/10/2014) (cited on page 39).

[44] John R. Douceur. 'The Sybil Attack'. In: *Peer-to-Peer Systems*. International Workshop on Peer-to-Peer Systems (IPTPS). (Faculty Club, Cambridge, Massachusetts, United States of America, 7th–8th March 2002). Edited by Peter Druschel, Frans Kaashoek and Antony Rowstron. Volume 2429. Lecture Notes in Computer Science. Massachusetts Institute of Technology (MIT). Springer Berlin Heidelberg, 2002, pages 251–260. ISBN: 978-3-540-45748-0. DOI: 10.1007/3-540-45748-8_24 (cited on page 39).

[45] Paul Eggert and Eric Blake. *Autoconf*. Free Software Foundation. May 2016. URL: https://www.gnu.org/software/autoconf/autoconf.html (visited on 22/11/2016) (cited on page 23).

[46] European Parliament and Council of the European Union. 'Directive 2006/24/EC'. In: *Official Journal of the European Union*. L 49.105 (March 2006), pages 54–63. ISSN: 1725-2555 (cited on page 1).

[47] Roy T. Fielding and Julian F. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. Request for Comments 7230. Internet Engineering Task Force (IETF). June 2014. DOI: 10.17487/RFC7230 (cited on page 2).

[48] James M. Galvin. 'DNS Security: A Historical Perspective'. In: *IETF Journal* 2.2 (September 2006). Edited by Mirjam Kühne, pages 25–28 (cited on page 11).

[49] *GDB: The GNU Project Debugger*. Free Software Foundation. August 2015. URL: https://www.gnu.org/software/gdb/ (visited on 01/09/2015) (cited on page 20).

[50] John Geddes, Rob Jansen and Nicholas Hopper. 'How Low Can You Go: Balancing Performance with Anonymity in Tor'. In: *Privacy Enhancing Technologies*. Privacy Enhancing Technologies Symposium (PETS). (Indiana Memorial Union (IMU), Bloomington, Indiana, United States of America, 10th–12th July 2013). Edited by Emiliano De Cristofaro and Matthew Wright. Volume 7981. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pages 164–184. ISBN: 978-3-642-39076-0. DOI: `10.1007/978-3-642-39077-7_9` (cited on pages 6, 7).

[51] David M. Goldschlag, Michael G. Reed and Paul F. Syverson. 'Hiding Routing Information'. In: *Information Hiding*. Edited by Ross Anderson. Volume 1174. Lecture Notes in Computer Science. Springer Berlin Heidelberg, May 1996, pages 137–150. DOI: `10.1007/3-540-61996-8_37` (cited on page 5).

[52] *Google Public DNS*. Google Inc. May 2014. URL: `https://developers.google.com/speed/public-dns/` (visited on 26/03/2015) (cited on pages xiv, 15, 16, 41, 42).

[53] Benjamin Greschbach, Tobias Pulls, Laura M. Roberts, Philipp Winter and Nick Feamster. 'The Impact of DNS on Tor's Anonymity'. In: *Computing Research Repository (CoRR)* (September 2016). arXiv: `1609.08187` (cited on page 7).

[54] Arnt Gulbrandsen, Paul Vixie and Levon Esibov. *A DNS RR for specifying the location of services (DNS SRV)*. Request for Comments 2782. Network Working Group. February 2000. DOI: `10.17487/RFC2782` (cited on page 7).

[55] Martin Edward Hellman. 'An overview of public key cryptography'. In: *IEEE Communications Magazine* 40.5 (May 2002), pages 42–49. DOI: `10.1109/MCOM.2002.1006971` (cited on page 37).

[56] Paul Hoffman and Jakob Schlyter. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. Request for Comments 6698. Internet Engineering Task Force (IETF). August 2012. DOI: `10.17487/RFC6698` (cited on pages 7, 28).

[57] *Hyperboria*. Project Meshnet. March 2015. URL: `https://wiki.projectmeshnet.org/index.php?title=Hyperboria&oldid=6168` (visited on 13/03/2015) (cited on page 3).

[58] *I2P Bibliography*. The Invisible Internet Project. July 2015. URL: `https://geti2p.net/en/papers/` (visited on 30/07/2015) (cited on page 6).

[59] *I2P Project Members – I2P*. The Invisible Internet Project. May 2014. URL: `https://geti2p.net/en/about/team` (visited on 30/05/2014) (cited on page 5).

[60] *JonDonym Mix Operators*. JonDos GmbH. October 2013. URL: `https://anonymous-proxy-servers.net/en/operators.html` (visited on 30/05/2014) (cited on page 4).

[61] *JonDonym – the anonymisation service*. JonDos GmbH. November 2013. URL: `https://anonymous-proxy-servers.net/` (visited on 09/05/2014) (cited on page 4).

[62] Simon Josefsson. *The Base16, Base32, and Base64 Data Encodings*. Request for Comments 4648. Network Working Group. October 2006. DOI: `10.17487/RFC4648` (cited on page 40).

[63] jrandom. *Invisible Internet Project (I2P). Project Overview*. The Invisible Internet Project. 28th August 2003 (cited on page 5).

[64] John C. Klensin. *Simple Mail Transfer Protocol*. Request for Comments 5321. Network Working Group. October 2008. DOI: `10.17487/RFC5321` (cited on pages 4, 7).

[65] Hugo Krawczyk. 'Cryptographic Extraction and Key Derivation: The HKDF Scheme'. In: *Advances in Cryptology – CRYPTO 2010*. Edited by Tal Rabin. Volume 6223. Lecture Notes in Computer Science. Springer Berlin Heidelberg, May 2010, pages 631–648. DOI: `10.1007/978-3-642-14623-7_34` (cited on page 38).

[66] Frederick Lah. 'Are IP Addresses "Personally Identifiable Information?"' In: *I/S: A Journal of Law and Policy for the Information Society* 4.3 (2008): *2008 Privacy Year in Review*, pages 676–703 (cited on page 1).

[67] Adam Langley. *How to botch TLS forward secrecy*. ImperialViolet. June 2013. URL: `https://www.imperialviolet.org/2013/06/27/botchingpfs.html` (visited on 06/06/2014) (cited on page 37).

[68] Adam Langley. *No, don't enable revocation checking*. ImperialViolet. April 2014. URL: `https://www.imperialviolet.org/2014/04/19/revchecking.html` (visited on 10/07/2014) (cited on page 38).

[69] Stefano Lattarini, Ralf Wildenhues and Jim Meyering. *Automake*. May 2016. URL: `https://www.gnu.org/software/automake/` (visited on 22/11/2016) (cited on page 23).

[70] *ldns*. NLnet Labs. March 2014. URL: `https://www.nlnetlabs.nl/projects/ldns/` (visited on 07/06/2016) (cited on page 33).

[71] Marcus Leech, Matt Ganis, Ying-Da Lee, Ron Kuris, David Koblas and LaMont Jones. *SOCKS Protocol Version 5*. Request for Comments 1928. Network Working Group. March 1996. DOI: `10.17487/RFC1928` (cited on pages 2, 12, 15).

[72] Andrew Lewman. *Five Years as an Exit Node Operator*. The Tor Project, Inc. November 2008. URL: `https://blog.torproject.org/blog/five-years-exit-node-operator` (visited on 24/10/2014) (cited on page 39).

[73] Karsten Loesing. *Spread the word about Tor*. The Tor Project, Inc. March 2015. URL: `https://blog.torproject.org/blog/spread-word-about-tor` (visited on 09/04/2015) (cited on page 5).

[74] Gary T. Marx. 'What's in a Name? Some Reflections on the Sociology of Anonymity'. In: *The Information Society: An International Journal* 15.2 (1999), pages 99–112. DOI: `10.1080/019722499128565` (cited on page 1).

[75] Nick Mathewson. *Fast portable non-blocking network programming with Libevent*. January 2012. URL: `http://www.wangafu.net/~nickm/libevent-book/` (visited on 01/09/2015) (cited on pages 21, 23).

[76] Nick Mathewson. *Index of Tor Proposals*. The Tor Project, Inc. January 2007. URL: `https://gitweb.torproject.org/torspec.git/tree/proposals/000-index.txt` (visited on 05/10/2016). Commit `39dbb69a11b3706a89ffdd517fc4392f700e4cc3` (cited on page 7).

[77] Nick Mathewson. *The Tor Proposal Process*. The Tor Project, Inc. January 2007. URL: `https://gitweb.torproject.org/torspec.git/tree/proposals/001-process.txt` (visited on 05/10/2016). Commit `39dbb69a11b3706a89ffdd517fc4392f70-0e4cc3` (cited on page 7).

[78] Nick Mathewson and Niels Provos. *libevent – an event notification library*. January 2015. URL: `http://libevent.org/` (visited on 01/09/2015) (cited on page 21).

[79] Ondrej Mikle. *Support for full DNS and DNSSEC resolution in Tor*. The Tor Project, Inc. February 2012. URL: `https://gitweb.torproject.org/torspec.git/tree/proposals/219-expanded-dns.txt` (visited on 08/12/2014). Commit `ebc5a935-ee4aa0c123829706671a3f43da82f11f` (cited on pages iii, 7, 13, 27).

[80] Minister of Security and Justice. *Wijziging van het Wetboek van Strafrecht en het Wetboek van Strafvordering in verband met de verbetering en versterking van de opsporing en vervolging van computercriminaliteit (computercriminaliteit III)*. May 2013 (cited on page 1).

[81] Paul Mockapetris. *Domain Names – Implementation and Specification*. Request for Comments 1035. Network Working Group. November 1987. DOI: `10.17487/RFC1035` (cited on pages 7, 12, 31).

[82] Ulf Moeller, Lance Cottrell, Peter Palfrader and Len Sassaman. *Mixmaster Protocol*. Internet draft. Version 2. Network Working Group, December 2004 (cited on page 4).

[83] Alec Muffett. *Making Connections to Facebook more Secure*. October 2014. URL: `https://www.facebook.com/notes/protect-the-graph/1526085754298237` (visited on 13/11/2014) (cited on page 40).

[84] *Network Topology Icons*. Cisco Systems, Inc. January 2015. URL: `https://www.cisco.com/web/about/ac50/ac47/2.html` (visited on 11/03/2015) (cited on pages 2, 4).

[85] Greg Norcie, Jim Blythe, Kelly Caine and L. Jean Camp. 'Why Johnny Can't Blow the Whistle: Identifying and Reducing Usability Issues in Anonymity Systems'. In: *Proceedings of the 2014 Workshop on Usable Security (USEC)*. (San Diego, California, United States of America). February 2014. ISBN: 1-891562-37-1. DOI: `10.14722/usec.2014.23022` (cited on page 7).

[86] *OpenDNS IP Addresses*. OpenDNS. March 2014. URL: `https://www.opendns.com/home-internet-security/opendns-ip-addresses/` (visited on 26/03/2015) (cited on pages xiv, 15, 16, 41, 42).

[87] *OpenNIC Public Servers*. OpenNIC. October 2016. URL: `https://servers.opennicproject.org/` (visited on 28/10/2016) (cited on pages xiv, 17, 41, 43).

[88] *OpenSSL*. OpenSSL Software Foundation. November 2016. URL: `https://www.openssl.org/` (visited on 22/11/2016) (cited on page 23).

[89] Peter Palfrader. *Number of Mixminion Nodes*. September 2012. URL: `http://www.noreply.org/mixminion-nodes/` (visited on 27/01/2015) (cited on page 6).

[90] Peter Palfrader. *Remailer Reliability Stats*. January 2015. URL: `http://www.noreply.org/echolot/` (visited on 27/01/2015) (cited on page 6).

[91] Sameer Parekh. 'Prospects for Remailers'. In: *First Monday* 1.2 (August 1996). DOI: `10.5210/fm.v1i2.476` (cited on page 4).

[92] Jon Peterson, Olaf Kolkman, Hannes Tschofenig and Bernard Aboba. *Architectural Considerations on Application Features in the DNS*. Request for Comments 6950. Internet Architecture Board (IAB). October 2013. DOI: `10.17487/RFC6950` (cited on page 17).

[93] Matej Pfajfar, Roger Dingledine and Nick Mathewson. *Code to parse and interpret configuration files*. The Tor Project, Inc. September 2014. URL: `https://gitweb.torproject.org/tor.git/tree/src/or/config.c` (visited on 08/12/2014). Commit `b448ec195dd8687d2d5f363e12fec046eb2d1677` (cited on page 39).

[94] *Possible Sybil Attack*. The Tor Project, Inc. January 2014. URL: https://lists.torproject.org/pipermail/tor-consensus-health/2014-January/004134.html (visited on 14/10/2014) (cited on page 39).

[95] Jon Postel. *Internet Protocol*. Request for Comments 791. Information Sciences Institute. September 1981. DOI: 10.17487/RFC0791 (cited on pages xv, xvi, 1).

[96] Jon Postel. *Transmission Control Protocol*. Request for Comments 793. Information Sciences Institute. November 1987. DOI: 10.17487/RFC793 (cited on page 5).

[97] Jon Postel. *User Datagram Protocol*. Request for Comments 768. Information Sciences Institute. August 1980. DOI: 10.17487/RFC0768 (cited on page 5).

[98] Greg Roelofs and Mark Adler. *zlib Home Site*. September 2016. URL: http://zlib.net/ (visited on 22/11/2016) (cited on page 23).

[99] Scott Rose. *Applicability Statement: DNS Security (DNSSEC) DNSKEY Algorithm Implementation Status*. Request for Comments 6944. Internet Engineering Task Force (IETF). April 2013. DOI: 10.17487/RFC6944 (cited on page 31).

[100] Peter Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Core*. Request for Comments 6120. Internet Engineering Task Force (IETF). March 2011. DOI: 10.17487/RFC6120 (cited on page 7).

[101] Stefan Santesson, Ambarish Malpani, Carlisle Adams, Michael Myers, Rich Ankney and Slava Galperin. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP*. Request for Comments 6960. Internet Engineering Task Force (IETF). June 2013. DOI: 10.17487/RFC6960 (cited on page 38).

[102] Robert W. Shirey. *Internet Security Glossary – Version 2*. Request for Comments 4949. Network Working Group. August 2007. DOI: 10.17487/RFC4949 (cited on page 3).

[103] Robin Snader and Nikita Borisov. ‘A Tune-up for Tor: Improving Security and Performance in the Tor Network’. In: *Proceedings of the 2008 Network and Distributed Security Symposium (NDSS)*. (San Diego, California, United States of America). Internet Society, February 2008 (cited on pages 6, 7).

[104] *Spy Files*. WikiLeaks. December 2011. URL: https://wikileaks.org/spyfiles/ (visited on 28/04/2014) (cited on page 1).

[105] Paul Syverson. *Onion Routing. A note about the use of ‘generation’*. October 2006. URL: http://www.onion-router.net/ (visited on 23/01/2015) (cited on page 5).

[106] Paul Syverson. 'Why I'm Not an Entropist'. In: *Security Protocols XVII*. International Workshop on Security Protocols. (Cambridge, England, United Kingdom, 1st–3rd April 2009). Edited by Bruce Christianson, James A. Malcolm, Vashek Matyáš and Michael Roe. Volume 7028. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pages 213–230. ISBN: 978-3-642-36213-2. DOI: `10.1007/978-3-642-36213-2_25` (cited on page 5).

[107] *TC. A Tor control protocol*. Version 1. The Tor Project, Inc. July 2015. URL: `https://gitweb.torproject.org/torspec.git/tree/control-spec.txt` (visited on 12/08/2015). Commit `acd48dd75e6b657fb5585531d04693c0a6e62fa7` (cited on page 15).

[108] *The Free Haven Anonymity Bibliography. Tor Performance*. Free Haven. July 2015. URL: `http://freehaven.net/anonbib/full/topic.html#Tor_20Performance` (visited on 30/07/2015) (cited on page 6).

[109] *The Reduced Exit Policy*. The Tor Project, Inc. October 2014. URL: `https://trac.torproject.org/projects/tor/wiki/doc/ReducedExitPolicy?version=19` (visited on 24/10/2014) (cited on page 39).

[110] *The Snowden Digital Surveillance Archive*. Canadian Journalists for Free Expression (CJFE) and the Faculty of Information at the University of Toronto. May 2015. URL: `https://snowdenarchive.cjfe.org` (visited on 07/05/2014) (cited on page 1).

[111] William Theaker. *Interview with Caleb James DeLisle of cjdns*. Free Software Foundation. September 2013. URL: `https://www.fsf.org/blogs/licensing/interview-with-caleb-james-delisle-of-cjdns` (visited on 11/09/2014) (cited on page 3).

[112] Susan Thomson, Christian Huitema, Vladimir Ksinant and Mohsen Souissi. *DNS Extensions to Support IP Version 6*. Request for Comments 3596. Network Working Group. October 2003. DOI: `10.17487/RFC3596` (cited on pages 7, 12).

[113] *To Whom It May Concern – English version*. Blackout Austria. July 2014. URL: `https://network23.org/blackoutaustria/2014/07/01/to-whom-it-may-concern-english-version/` (visited on 24/10/2014) (cited on page 39).

[114] *Tor directory protocol*. Version 3. The Tor Project, Inc. September 2014. URL: `https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt` (visited on 08/12/2014). Commit `ebc5a935ee4aa0c123829706671a3f43da82f11f` (cited on pages xv, xvi, 39).

[115] *Tor Metrics. Direct users by country*. The Tor Project, Inc. January 2015. URL: https://metrics.torproject.org/userstats-relay-country.html?graph=userstats-relay-country&start=2014-10-29&end=2015-01-27&country=all&events=off (visited on 27/01/2014) (cited on pages 5, 6).

[116] *Tor Permalinks*. The Tor Project, Inc. August 2015. URL: https://spec.torproject.org/ (visited on 26/08/2015) (cited on page 20).

[117] *Tor Project: Core People*. The Tor Project, Inc. May 2014. URL: https://www.torproject.org/about/corepeople.html.en (visited on 30/05/2014) (cited on page 5).

[118] *Tor -stable Manual*. The Tor Project, Inc. October 2014. URL: https://www.torproject.org/docs/tor-manual.html.en#ExitPolicy (visited on 24/10/2014) (cited on page 39).

[119] *Tor's extensions to the SOCKS protocol*. The Tor Project, Inc. February 2014. URL: https://gitweb.torproject.org/torspec.git/tree/socks-extensions.txt (visited on 12/08/2015). Commit `5b875a19f2de7da4a7c7cf66a810a4f33b81f57a` (cited on page 12).

[120] *Tor's source code*. The Tor Project, Inc. August 2015. URL: https://gitweb.torproject.org/tor.git/ (visited on 26/08/2015). Commit `1eb210637539ce71794ff49fc462c35511b8a141` (cited on page 18).

[121] *torsocks. Wrapper to safely torify applications*. The Tor Project, Inc. May 2015. URL: https://gitweb.torproject.org/torsocks.git (visited on 01/09/2015) (cited on page 15).

[122] *Unbound*. NLnet Labs. September 2015. URL: https://www.unbound.net/ (visited on 01/09/2015) (cited on pages 15, 33).

[123] *Unbound*. NLnet Labs. July 2015. URL: https://www.unbound.net/documentation/libunbound.html (visited on 02/09/2015) (cited on page 23).

[124] Gabriel Weinberg. *DuckDuckGo now operates a Tor exit enclave*. August 2010. URL: http://www.gabrielweinberg.com/blog/2010/08/duckduckgo-now-operates-a-tor-exit-enclave.html (visited on 13/11/2014) (cited on page 40).

[125] Alan F. Westin. *Privacy and Freedom*. The Bodley Head Ltd., April 1970 (cited on page 1).

[126] Philipp Winter, Richard Köwer, Martin Mulazzani, Markus Huber, Sebastian Schrittwieser, Stefan Lindskog and Edgar Weippl. 'Spoiled Onions: Exposing Malicious Tor Exit Relays'. In: *Privacy Enhancing Technologies*. Privacy Enhancing Technologies Symposium (PETS). (The Royal Tropical Institute, Linnaeusstraat 2, 1092 CK Amsterdam, Netherlands, 16th–

18th July 2014). Edited by Emiliano De Cristofaro and Steven J. Murdoch. Volume 8555. Lecture Notes in Computer Science. Springer International Publishing, 2014, pages 304–331. ISBN: 978-3-319-08506-7. DOI: 10.1007/978-3-319-08506-7_16 (cited on pages 7, 39).

[127] Rejo Zenger. *Bewaarplicht: wrong, on so many levels*. May 2014. URL: https://www.bof.nl/2014/04/16/bewaarplicht-wrong-on-so-many-levels/ (visited on 29/04/2014) (cited on page 1).