JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

AN ONLINE PRIVACY ENHANCING FRAMEWORK BASED ON THE PRINCIPLES AND CAPABILITIES OF THE ONION ROUTER (TOR)

ABDULLAHI MODIBBO ABDULLAHI
18/ PGC /SCI01/1002

DEPARTMENT OF MATHEMATICAL AND PHYSICAL SCIENCES
(COMPUTER SCIENCE PROGRAMME),
COLLEGE OF SCIENCES,
AFE BABALOLA UNIVERSITY ADO EKITI, EKITI STATE.

SUPERVISOR: DR. O. M. AWEH
CO-SUPERVISOR: DR. (Mrs.) O. ABIOLA

CHAPTER ONE

INTRODUCTION

1.0 Introduction

This chapter outlines the research area and specifies the boundaries of the study. It provides a preface to the current challenges and risks of preserving online privacy and anonymity with emphasis on known mitigation techniques and tools that can be used by a non-specialized audience. Consequently, it provides a synopsis of the study aim and the optimal objectives towards addressing the aim.

1.1 Online Privacy Concerns

Nowadays, the proliferation of novel and innovative ideas along with the ongoing exponential evolution of technology constitute the dominant characteristics of the current generation (Silverstone, 2017, Davenport,

2013). However, the increased dependency on computer based systems has generated a new, previously unseen, category of complexities (Acquisti et al., 2015). Online privacy and protection of personal data such as legal name, location, behaviour patterns, political beliefs and pseudonyms (Forte et al., 2017), also known as Personally Identifiable Information (PII), have become of major value and importance (Gottschalk Jr et al., 2017).

According to a 2016 U.S. Consumer Privacy Index survey conducted by TRUSTe, user privacy awareness was estimated at 31% (TRUSTe, 2016b), presenting a 24% reduction when compared to the 55% result obtained during the 2015 U.S. Consumer Confidence Edition survey (TRUSTe, 2015b). Similar U.K. studies have shown a drop of 26%, from 51% in 2015 (TRUSTe, 2015a) to 25% in 2016 (TRUSTe, 2016a). A separate large-scale research by the University of Hohenheim, Germany with a sample of 26 761 Europeans, among 27 member-states, reported that 63% of the participants had major concerns regarding personal data disclosure (Trepte et al., 2015). Although a straightforward statistical analysis of the attained results clearly indicates an increase in user concerns regarding online privacy and personal data protection, it is important to understand the reasons that have led to this.

As some believe, a major decisive factor that has contributed to the increase in privacy concerns and a cultural shift towards increased awareness regarding online security, might be the recent revelations regarding certain surveillance practices, made by former NSA (National Security Agency) contractor Edward J. Snowden (Preibusch, 2015). Indeed, an immense, global-scale media coverage broke out on June 6th, 2013 when newspaper The Guardian, in collaboration with Snowden, reported on a previously secret NSA program (PRISM) targeting all metadata – the who, what and when of phone calls – in domestic communications of *Verizon* customers (Greenwald, 2013). During the days that followed, more cyber-intelligence issues, involving various tools and techniques, came to light such as Tempora (Landau, 2014), XKeyScore (Greenwald, 2013), monitoring by Internet Service Providers (ISP) (Gellman and Poitras, 2013), a U.K. and U.S. diplomat tracking program (Shane and Somaiya, 2013) and an *Apple – Google* user data monitoring (Greenwald and MacAskill, 2013). The revelations prompted heated discussions among privacy experts, advocates and politicians, while the public observed powerless and unable to react (Greenwald, 2014). Another factor having considerable significance in the public's concerns equation is adversaries, more widely known as hacktivists. In recent years, the capabilities of

hacktivists have rapidly increased as demonstrated by a variety of cyber-attacks. From stealing 76 million JPMorgan customer records at 2014 (Radin, 2014), to halting a Ukrainian Power Plant in 2015 (Lee et al., 2016) and raiding Tesco Bank in 2016 (Arthur, 2016), these groups have manage to draw attention and severely impact the public's perception regarding online security raising doubts on a government's ability to safeguards its citizens (Shackelford and Craig, 2014). Although this might be the case to some extent, in reality a lot of cyber security related investments have taken place at national-level over the past years in order to augment stability and security for the public (Fielder et al., 2016, Gordon et al., 2015). A prime example of such investment is the £1.9 billion announced by UK's chancellor, Philip Hammond in November 2016 (Hammond, 2016) in order to enhance the country's cyber security capabilities.

Individuals have also resorted to the use of several techniques and tools in order to defend themselves. These efforts have widely promoted online privacy and anonymity. Tor (The Onion Router), I2P (Idea to Product), Freepto, IprediaOS and LPS (Lightweight Portable Security) are only a few of such techniques aiming at addressing public doubts. However, the lack of familiarity and basic technical knowledge exhibited by most audiences acted as a major adoption obstacle (Ball et al., 2013). Recognizing the existing knowledge gap, researchers and journalists attempted to educate and support the public by presenting and explaining some of the recommended solutions (Lee, 2013, Shillair et al., 2015). At the same time, software developers tried to bridge this knowledge gap by creating preconfigured, easy-to-use, variations of their applications (Rajagopal et al., 2016) such as the Tor-Bundle (TorProject, 2017a), the AvANa BBS open source application by Freepto (Freepto, 2016) and the automated I2P mobile application (Villa, 2017).

Large organizations also noted the gradual increase of online privacy concerns and the formation of a new cyber security market, and acted by adopting increased security measures and techniques. Examples include (i) the 2016 Facebook deployment of Signal protocol in their Messenger application to provide end-to-end encryption capabilities (Marlinspike, 2016), (ii) the 2016 launch of Watson for cyber security by IBM to enhance security analyst decision making (IBM, 2017) and (iii) the 2015 Microsoft initiative to create a Cyber Defense Operation Center (Choney, 2015).

However, as Horsman has observed the revelations made by Greenwald in 2013 regarding government surveillance mechanisms along with the exposure of data collection methods followed by Apple and Google have negatively affected the perceptions of people when using communication technologies (Horsman, 2016). According to Forte et al. (2017), besides the problem of technology understanding, another factor capable to increase privacy concerns was the perception of the public towards organisations providing "secure" services (Forte et al., 2017). Due to the closed source nature of these services, there has been a shift towards the use of open source security solutions, such as Tor (Jardine, 2016).

1.2 The Onion Router (TOR)

The Onion Router (TOR) was developed in the mid-1990s at the U.S. Naval Research Laboratory aiming to protect online U.S. intelligence communications. It was further expanded by DARPA (Defense Advanced Research Projects Agency) in 1998 until it became publicly available. The majority of researchers working on the project, founded "The Tor Project" as a non-profit organisation in 2006 (TorProject, 2017b), with the financial support of EFF (Electronic Frontier Foundation) (Shavers and Bair, 2016a). To date, Tor remains active with an average of 2 million users per day (0.1% of the global internet users) as illustrated in Figure 1 (TorProject, 2017i). There are also more than 20 organizations financially supporting it, including Google, Mozilla, NSF (National Science Foundation) and SRI (Stanford Research Institute) (TorProject, 2017j).

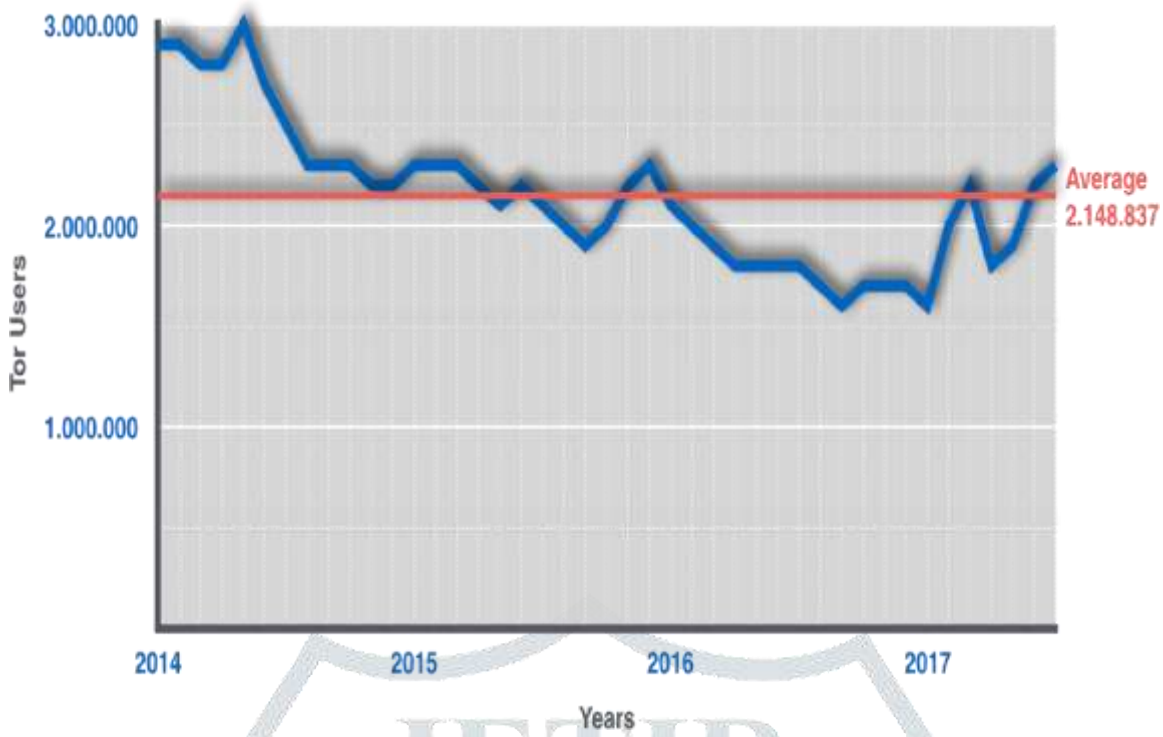


Figure1.1 Number of directly-connecting Tor clients in the past years (Source: National Science Foundation (NSF) 2017)

The aim of Tor is to provide online anonymity to the end-user by using a unique architectural network topology, which involves a number of nodes deployed around the world by volunteers and using state of the art encryption techniques (Shavers and Bair, 2016b). However, due to its advanced security features, it has also been heavily used by organized crime and terrorist organizations, (Mansfield-Devine, 2014) as an online shopping website for drugs, weapons and pornography, This has been ttracting much attention among the intelligence and the researcher communities (Iovation, 2013).

Over the years, a number of vulnerabilities have been identified either by public or private research (Ling et al., 2013b) or by governmental projects (Mansfield-Devine, 2014) aiming to de-anonymize some of the illicit users. In 2013, Carnegie Mellon University received funding from the U.S. Department of Defense in a research on how to unmask IP addresses of Tor users. The results of this project were first revealed in the 2014 Black Hat hacking conference. It was later disclosed through the 2015 court documents presented in the Manhattan Federal Court, that the specific research was used to compromise the operating team of an online Tor black market called Silk Road 2 (Hen, 2016). This resulted in the .onion domain's seizure as illustrated in Figure 1.2 and the confiscation of 170 000 Bitcoin, worth approximately $110 million in mid-2015 Bitcoin rates, by the FBI (Reuters, 2015).

Figure 1.2 Silk Road 2 domain after FBI's seizure

As a synopsis, extensive research has targeted a combination of Tor vulnerabilities and a considerable number of experiments have been conducted during the last decade aiming at degrading anonymous communications over Tor. Table 1.1 attempts to outline chronologically the OSI (Open Systems Interconnection) layers associated with these vulnerabilities and present the approaches followed by Tor researchers from 2006 until today.

Table 1.1 Tor research domains and approaches for the last decade

NO	PAPER	CITATION	OSI LAYER	APPROACH
1	Locating hidden servers	(Overlier and Syverson, 2006)	- Application - Session	- Live Experiments - Simulations
2	Network flow watermarking attack on low-latency anonymous communication systems	(Wang et al., 2007)	- Session - Transport - Network	- Live Experiments

3	A new cell counter-based attack against tor	(Ling et al., 2009)	- Session - Transport	- Theory - Simulations
4	Website fingerprinting in onion routing based anonymization networks	(Panchenko et al., 2011)	- Application - Session	- Live Experiments - Simulations
5	Trawling for tor hidden services: Detection, measurement, deanonymization	(Biryukov et al., 2013)	- Session - Transport - Network	- Live Experiments
6	Traveling the Silk Road: A measurement analysis of a large anonymous online marketplace	(Christin, 2013)	- Application - Network	- Live Experiments
7	Spoiled onions: Exposing malicious Tor exit relays	(Winter et al., 2014)	- Session - Network	- Theory - Statistics - Simulation
8 9	RAPTOR: Routing Attacks on Privacy in Tor	(Sun et al., 2015)	- Session - Transport - Network - Datalink	- Theory - Statistics
10	How Anonymous Is the Tor	(Koch et al.,	- Network	- Theory - Simulations

Network? A Long-Term	2016)		
Black-Box Investigation		- Physical	- Simulations
	(Matic et al.,	- Datalink	
Dissecting Tor Bridges: A	2017)		
Security Evaluation of			
Their Private and Public			
Infrastructures			

1.3 Statement of the Problem

For the purpose of the project, there is a need to develop an isolated experimental network that can incorporate the principles and capabilities of Tor. Online privacy enhancing tools such as Tor, require a significant amount of thorough experimentation and service evaluation to assess the risks of cyber-attacks successfully, investigate the effects of modifications in Tor software components and collect accurate, scientific network data (Shirazi et al., 2015).

Research should not be performed on the live Tor network for a variety of ethical and personal security reasons (Gelinas et al., 2017, Forte et al., 2017). Tor is under constant development and maintenance by the Tor project team and receives significant funding and support from a number of large organisations (TorProject, 2017j), in order to ensure the anonymity of its users. As a result, researchers are not able to perform large-scale modifications, as it could have an impact on the overall quality of service (Shirazi et al., 2015). It is possible for an individual to run Tor relays and collect encrypted user data passing through them. However such act is not ethical as the users are neither aware of their participation in any kind of research nor have they provided their informed consent (Biryukov et al., 2014). Although there are safeguards in place, it is also possible for an individual to run a Tor exit relay and collect unencrypted data while exiting or entering the Tor network. In such case, besides the ethical concerns there are also operational research risks, as the collected data might be used to compromise the identity of the end user or

link a criminal activities with a specific group or individual (Johnson et al., 2013), resulting in possible retaliations against the researcher (Biryukov et al., 2013).

Furthermore, the conduction of experiments on the live Tor network might raise ethical concerns (Al Nabki et al., 2017) and expose Tor users or the research to in dangerous situations, conflicts or undesirable communications with illegal organizations or criminal groups (Byrne and Kimball, 2017). In addition, the collection and analysis of personal data passing through any network are considered illegal in most countries, including the United Kingdom. Consequently, such actions might place the researcher in a situation entailing the risk of investigation by the police or secret service (Kugler, 2015).

The development of an isolated Tor experimental network is crucial, in ensuring the conduction of Tor related research experiments in an ethically and legally appropriate manner, which in turn can guarantee the validity and repeatability of the adopted scientific methodology. It is therefore necessary to investigate in depth the problems of various techniques and tools employed by researchers to perform research experiments, including their limitations, before developing a solution to address them. It is also imperative to evaluate the robustness and efficiency of the solution, through experimentation in order to determine its strengths and limitations. This procedure provides researchers with adequate information and developing material for selecting the optimal setup, depending on their needs and requirements. In mind of the above, the following research question was formed to initiate the research in this domain.

1.4 Aim and Objectives

The aim of this study is to develop a framework based on the principles and capabilities of Tor for enhancing privacy in online transactions

The following three research objectives will be used to achieve this aim:

i. An isolated experimental network that can incorporate the principles and capabilities of TOR will be developed.
ii. The experimental network architecture and routing processes will be analysed, and also, the networh equipped with TOR capabilities will be analysed to understand in-depth the TOR network architecture, cryptographic controls, detailed design and implementation.

iii. The knowledge gained will be used to establish the significant requirements for the technical implementation of online privacy and anonymity.

1.5 Scope of the Study

This study is limited to developing a framework for online privacy enhancement for common browsers based on TOR concepts.

1.6 Limitations to this Study

The greatest impediments to this study were:

i. The tools required to setup, simulate and evaluate the experimental network were very expensive and difficult to get. And as a result of this, only the TOR network could be setup. The other browsers (Mozilla, chrome and opera) parallel netwoks could not be setup.

ii. Mastering the expertise required for the experiment took a great deal of time and efforts and this affected other aspects of the study.

iii. Funding and restriction on movement was another major impediment in this study. It limited the capacity to hold the desired consultations for a study such as this.

1.7 Organization of the Study

This study consists of five chapters which are respectively divided into subsections.

Chapter 1: Provides a preface to the current challenges and significance of preserving online privacy, discusses the advantages and limitations of the Tor network, defines the aim and objectives of the study.

Chapter 2: Reviewed sundry literature on the characteristics of online privacy and anonymity and provides an in-depth analysis and evaluation of the Tor network, with an emphasis on network architecture, cryptographic controls, design and implementation.

Chapter 3: Review the objectives outlined and justified the adopted research methodology. It discussed the adopted tools, techniques and processes used for the study, and documents the procedures, steps and research choices that took place during the development of an isolated network infrastructure. In addition,

it collects measurements and statics from each produced virtual network and evaluates the overall system performance.

Chapter 4: It proceed further to documents the procedures, steps and research choices that took place during the integration of TOR in the produced isolated network infrastructure and presents the actions leading to the development of a TOR isolated experimental network. In terminates with an evaluation of the produced TOR isolated experimental network, defines its limitations and discuss a series of network experiments, and then provides an overview of the study and discusses the system limitations.

Chapter 5: Provides a concise, justifiable, evidence-based response to the study aim and objectives and discusses future work directions.

CHAPTER TWO

LITERATURE REVIEW

2.1 The Onion Router

The Onion Router (TOR) belongs to the selective elite of cyberspace technologies, providing for anonymous internet use. Its sophisticated functionality can be matched by only a handful of alternative technologies. A simplistic interpretation of Tor is that of an open-source internet browser, which is actually an adaptation of the well-known Firefox internet browser. These modifications significantly enhance the online privacy and security by concealing the Internet Protocol (IP) address of Tor users while surfing the web or exchanging emails. Without the dedication of tremendous resources and the use of unprecedented techniques, it is practically impossible to back trace or identify the originated IP.

Despite its advanced features and capabilities, a Tor Browser deployment is a relatively straightforward process that requires no technical knowledge or differs from traditional browser installations. Its cross-platform compatibility with Windows, Linux or Apple operating systems (TorProject, 2017a) and the lack of specific hardware dependencies makes Tor, the simplest solution to online privacy by far. In addition, its recent support at Android (TheTorProject, 2017a) and iOS (Tigas, 2017) mobile operating systems have established it as a universal solution to anonymous, secure communications.

The goal of onion routing was to have a way to use the internet with as much privacy as possible, and the idea was to route traffic through multiple servers and encrypt it each step of the way. This is still a simple explanation for how Tor works today.

In the early 2000s, Roger Dingledine, a recent Massachusetts Institute of Technology (MIT) graduate, began working on an (NRL) onion routing project with Paul Syverson. To distinguish this original work at NRL from other onion routing efforts that were starting to pop up elsewhere, Roger called the project Tor, which stood for The Onion Routing. Nick Mathewson, a classmate of Roger's at MIT, joined the project soon after.

From its inception in the 1990s, onion routing was conceived to rely on a decentralized network. The network needed to be operated by entities with diverse interests and trust assumptions, and the software needed to be free and open to maximize transparency and separation. That is why in October 2002 when the Tor network was initially deployed, its code was released under a free and open software license. By the end of 2003, the network had about a dozen volunteer nodes, mostly in the U.S., plus one in Germany.

In 2007, the organization began developing bridges to the Tor network to address censorship, such as the need to get around government firewalls, in order for its users to access the open web. Tor began gaining popularity among activists and tech-savvy users interested in privacy, but it was still difficult for less-technically savvy people to use, so starting from 2005, development of tools beyond just the Tor proxy began and development of Tor Browser began in 2008. With Tor Browser having made Tor more accessible to everyday internet users and activists, Tor was an instrumental tool during the Arab Spring beginning in late 2010. It not only protected people's identity online but also allowed them to access critical resources, social media, and websites which were blocked (TorProject, 2016a).

The need for tools safeguarding against mass surveillance became a mainstream concern thanks to the Snowden revelations in 2013. Not only was Tor instrumental to Snowden's whistleblowing, but content of the documents also upheld assurances that, at that time, Tor could not be cracked.

Tor Project, fight every day for everyone to have private access to an uncensored internet, and Tor has become the world's strongest tool for privacy and freedom online.

But Tor is more than just software. It is a labor of love produced by an international community of people devoted to human rights. The Tor Project is deeply committed to transparency and the safety of its users (TorProject, 2016a).

2.2 Historical Perspective of TOR

Tor was developed in the mid-1990s at the U.S. Naval Research Laboratory. Its aim was to protect U.S. intelligence communications online. It was further expanded by DARPA in 1998, until it became publicly available in 2002. In 2006, the majority of researchers working on the project founded "The Tor Project" as a non-profit organization (TorProject, 2017b), having the financial support of EFF (Electronic Frontier Foundation) (Shavers and Bair, 2016a). Although Tor is constantly updated and maintained by "The Tor Project", it still carries its innate open-source characteristics, meaning that it is practically not controlled by any one entity. In addition, anyone with sufficient technical skills is able to contribute to the project. This characteristic, led to Tor attracting many attributions by privacy motivated individuals, located all over the world.

The objective of Tor is to provide users with the capability to browse anonymously the worldwide web. This online anonymity carries additional benefits and further extends the potential of Tor in certain contexts. For instance, it enables users located in dictatorship regimes or controlled by oppressive governments, to access online content that might be geographically blocked. There are innumerous examples and case studies demonstrating the real potential of Tor, varying from permitting whistleblowers to anonymously communicate with officials, to enabling organizations interested in preserving their private conversations secret.

2.3 Basic TOR Workflow

At its core, Tor constructs a randomly selected path of internet nodes, which form a network routing tunnel. It then encrypts the user-generated traffic by applying an elliptic curve cryptographic algorithm (Kumari et al., 2017), using as input the sequentially obtained session keys from each path node. The procedure continues with the transmission of the encrypted data stream to the initial path node ("entry relay"), which strips the outermost layer of encryption and then retransmits the outcome to its predetermined adjacent path node. As the encrypted data stream passes through each of the intermediate nodes ("middle relays"),

different layers of encryption are detached until it reaches the final node ("exit relay"). The exit node decrypts the data stream, establishes an unencrypted session with the desired destination IP address and delivers the original message. Figure 2.1 illustrates a graphical representation of the basic Tor operations.

Figure 2.1 Visual representations of the basic Tor operations. (Source: TorProject 2016)

However, what has been discussed constitutes only part of the complex functionality and advanced procedures taking place in the network. Furthermore, there are certain key characteristics that substantially affect the robustness and security of the workflow of Tor and significantly impact the provided Quality of Service (QoS).

Besides knowing the originated preceding and following successive node, all Tor relays do not know or retain any information related with the traffic routing path. This significantly increases the privacy and anonymity characteristics of the network, making it impossible to determine the selected routing path in the case of a Tor compromise. However, by design there exist two exceptions to this rule.

Entry relays, also referred as guard nodes, are aware of the IP address that the traffic originated from and consequently the user's identity, as a session is established during the initial 3-way handshake (Hayes and Danezis, 2015). In addition, exit relays have knowledge of the IP address of their predecessor relay. Another aspect of an exit relay that significantly differentiates it from the rest of Tor nodes, it is its ability to establish unencrypted sessions with nodes outside the Tor infrastructure. However, this ability carries the inherited characteristics of any publicly visible internet node, including being identifiable by traditional back-tracking techniques (Winter et al., 2014). A typical example is the 2011 case of Nolan King's, in which the ICE (Immigration and Customs Enforcement) broke into his house and confiscated his private server and computer, accusing him of distribution of child pornography (Hofmann, 2011). A later

investigation revealed that the accused did not possess or merchandise any pornography material but he was simply running a Tor exit node. As a result, conventional traffic analysis techniques, mistakenly led to his doorstep.

As it was previously discussed, Tor constructs a randomly selected path of relays that will be used during a specific user's interaction with the network. However, the formulated path does not endure indefinitely; meaning is not a permanent one. By default, each Tor routing path fully resets every ten minutes (Edman and Syverson, 2009), providing an adversary with a negligible timeframe to attempt compromising even a single relay. This characteristic has an enormous impact on the overall network security, as it eliminates a primary surface of attack.

Although the Tor path reset might be perceived as a straightforward activity, in practice it is a relatively complex operation which plays fundamental role in the development and architecture of a secure whole Tor infrastructure (Elahi et al., 2012). Strangely enough, this process also contributes significantly to the advance encryption capabilities that Tor employs (Wacek et al., 2013). However, before proceeding to in-depth analysis, there are certain key characteristics and procedures that one must comprehend.

2.4 Online Privacy

Online privacy is a broad area that encompasses a broad range of user concerns when interacting with the internet. Personal information is constantly collected, stored, mined and shared depending on the application, the Internet Service Provider (ISP), the browser and all related third-party entities (Bergström, 2015). Although the collection of this information was considered a relatively grey area, in light of recent terrorist's attacks such practices have now been legitimised and even mandatory in some cases (Ball et al., 2013).

In describing this web of issues, Solove says that privacy is "the relief from a range of kinds of social friction" that "enables individuals to involve in worthwhile activities in ways that they would otherwise find difficult or impossible" and "protection from a cluster of related activities that impinge upon people in related ways" (Solove, 2006). He has identified and organized these problematic and harmful activities in a taxonomy of four groups with subgroups: (1) information collection (including surveillance, interrogation), (2) information processing (including aggregation, identification, insecurity, secondary use, exclusion), (3)

information dissemination (including breach of confidentiality, disclosure, exposure, increased accessibility, blackmail, appropriation, distortion), and (4) invasion (including intrusion, decisional interference). Solove suggests that his taxonomy allows for the recognition of "structural" privacy problems, or the creation of risk that a person might be harmed in the future in the same way that environmental harms and pollution accumulate over time and increase the risk that harm will occur. He explains: "In many instances, privacy is threatened not by singular egregious acts but by a slow series of relatively minor acts, which gradually begin to add up" (Solove, 2008).

It is likely that the difficulty in defining privacy stems in part from the fact that the notion of privacy is culturally relative and contingent on factors such as economic status and availability of technology (DeCew, 2006). Alan Westin claims that although particular behaviors vary considerably across cultures, "needs for individual and group privacy and resulting social norms are present in virtually every society" (Westin, 1967), and he cites a host of anthropological studies documenting the various ways societies achieve privacy.

In our society we simultaneously seek privacy while having to disclose personal information in order to receive services and establish friendships. Online communication and the Social Web have led us into the habit of sharing large amounts of information with a great number of people, yet many do not feel threatened when doing so (Trepte and Reinecke, 2011). The problem is that the same technology that makes it easy to share personal details has also led to what Moor (1997) refers to as greased information – data that moves like lightening and is difficult to hold on to. Moor (1997) states that "once information is captured electronically for whatever purpose, it is greased and ready to go for any purpose".

As a consequence, the safety of our personal information has become of great importance and a major topic of interest to the business and IT sectors, as well as the general public. Reports focused on the issues of privacy and personal information has become more numerous and prominent in popular media:

I. In June 2013, The Guardian published a story on how the National Security Agency (NSA) is collecting the phone records of millions of Verizon customers on a daily basis (Greenwald, 2013). The information came from a document leaked by an NSA contract employee, the now infamous Edward Snowden.

II. In September 2014, several public celebrities had their personal photographs stolen from Apple's iCloud service (Satariano and Strohm, 2014). In November 2014, Sony Pictures was hacked and thousands of confidential documents containing the personal and private information of employees and celebrities were stolen and posted online (Brandom, 2014; McCormick, 2014).

III. RadioShack, an iconic US electronics retail chain, filed for bankruptcy in February 2015. The data it collected on over 100 million customers was sold via auction. This sale is being contested by several parties, one claiming that the data does not belong to RadioShack, several others claiming that the company is violating its own privacy policies (Brustein, 2015).

IV. Early in July 2015 it was disclosed that breaches of databases managed by the US government's Office of Personnel Management had exposed the sensitive information of at least 22.1 million individuals (Nakashima, 2015). Later on in July 2015, Ashley Madison – an online dating website that targets married people – was hacked and personal details on its 37 million users stolen (Krebs, 2015) and in August 2015 these details were released on to the Internet (Gibbs, 2015).

V. In February 2016, the Federal Bureau of Investigation (FBI) obtained a court order to compel Apple to break into an iPhone belonging to the perpetrator of a mass shooting (Edwards, 2016). Apple said that the only way this can be done is by creating a special version of Apple's iOS operating system that bypasses the phone's security, and opted to fight the order in court rather than comply. Ultimately, the FBI withdrew its request after finding a third party to assist in unlocking the phone, but the issue re-sparked debate about many aspects of privacy and state surveillance (Johnson, Swartz, & della Cava, 2016).

VI. In September 2016, Yahoo revealed that in 2014 hackers penetrated its network and stole personal data related to more than 500 million accounts (McMillan, 2016). This is believed to be the largest breach ever publicly disclosed by a company. These are only a few examples that are spurring global discussion of privacy and the need for adequate legislation to govern it. More than a hundred countries have privacy laws in place or in the process of development (Greenleaf, 2014).

An example is the November, 2016 UK Investigatory Powers Bill Act (Amber Rudd, 2016) which mandates that police and intelligence officers are allowed to access the user's browsing history content without a warrant. The act also contains suggestions on how the UK law enforcement officers can remotely

hack into devices or sweep up huge amounts of internet information passing through the public internet (BBC, 2015), Such GCHQ secret practices were uncovered in early 2013 by whistleblower Edward Snowden (Landau, 2014).

During an unencrypted digital communication between a user and a website “http://site.com”, data streams containing personal user information such as password, IP address, physical location and data content are exchanged through the public internet. Although the communication session is established between the two entities (user, website), the traffic is passing through a number of hops and cables until it reaches the destination IP address.

As previously discussed, ISPs are able to monitor and in some cases, store the full content of the communication, identify the originating IP address and link the traffic to a specific individual. As the traffic travels through the internet, intelligence agencies such as NSA (National Security Agency) and GCHQ (Government Communications Headquarters), are able to observe and store data that might be useful to them in the future (Ball et al., 2013).

Continuing, the data stream is delivered to the website’s ISP which enjoys the same privileges as the user’s ISP, enabling it to maintain data logs of the digital communication. Finally, the data stream is delivered by the website’s ISP to “http://site.com”, which in turn shares data with third-party business entities. In addition, law enforcement agencies, system administrators and legal advisors are allowed access to the information passing through ISPs and the log files maintained by websites, upon a court order or search warrant. Furthermore, malicious attackers are also able to tap and observe traffic passing through the user’s wireless or wired communication means using a variety of techniques.

The user - website digital communication is thus passing through an increasingly worried number of entities and is recorded legally or illegally by a number of authorities. The unencrypted nature (http) of the communication means that the data payload contained within the data stream is propagated through the internet in clear text format. This enables all internet hops to fully read or store data at any given moment, significantly increasing the possibility of data manipulation. Figure 2.2 illustrates the scenario of unencrypted internet communication between a user and a website “site.com” and documents the type of information that can be accessed or recorded by each internet entity (Foundation, 2017).

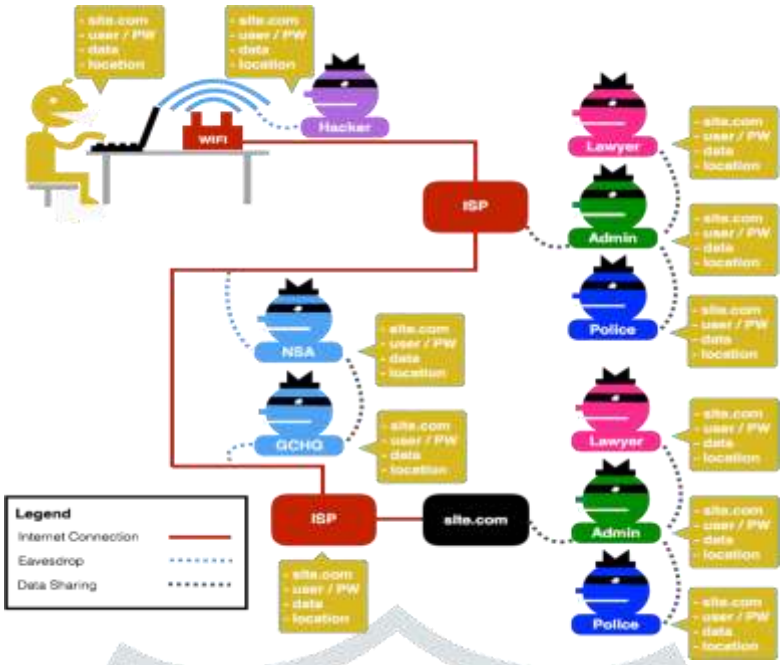


Figure 2.2: Recorded information during an unencrypted internet communication session between a user and a site.com (Foundation, 2017)

2.5 Anonymity and Pseudonymity

Anonymity is another broad term that encompasses different principles and carries distinct characteristics based on the context that is used. A recently conducted research by the University of Kentucky has classified anonymity into three categories: (i) visual anonymity, (ii) disassociation and (iii) identifiability (Winkler and Zeadally, 2015).

Visual anonymity refers to the level of visualization means, such as photograph or video stream, used during an online communication between two entities. Disassociation involves the use of pseudonyms or the submission of fake information during an online registration procedure. Lack of identifiability extends the disassociation category with the addition of ensuring that the user's digital persona cannot be linked to the real user (Rashid et al., 2013). Although all three categories address significant anonymity aspects, from the perspective of computer science are not considered significant to guarantee that a user will remain anonymous during a digital communication session (Morio and Buchholz, 2009). As illustrated in the example provided in section 2.1, digital communication carries supplementary information such as IP address that can directly be linked to the individual, even if the traffic is encrypted.

Anonymity and pseudonymity are not neutral states. When anonymous Internet users are the subject of mainstream news articles, it is often in the context of either large-scale political protests, or hacking, trolling, deceiving, or abusing others on the Internet through inflammatory posts. A few examples include

calling for harassment laws to be enforced more thoroughly, justified with an example of child exploitation material posted on the Facebook pages of two murdered children (Stafford, 2012); blaming anonymity for the "explosion of cybercrime that has swept across the Web" (Markoff, 2010); proposing uniform online identities that would function like a "driver's license" to increase trust online (Geist, 2014); and describing "hackers' collective" Anonymous' sabotage of the official Web site and Twitter account of North Korea (Alexander, 2013). Taken together, articles like these link anonymity and pseudonymity with criminality and chaos, perpetrating mistrust of those who do not wish to reveal their "real" identity online. But while extreme cases such as these do exist, this article argues that both anonymity and pseudonymity allow people to enact specific, and arguably valuable, identity practices online.

Now, in the age of the hacker group Anonymous and other social activist groups, a related phenomenon has arisen: doxing, which involves groups of anonymous or pseudonymous users researching an individual and then publishing identifiable facts about that person. Doxing has been used for supposed social good, as in the case of the Anonymous offshoot KnightSec exposing the identities of people involved in the Steubenville rape case (Almasy, 2013), but anonymous and pseudonymous Internet.

Tor seeks to address the issues surrounding online anonymity and pseudonymity and then enable users to completely hide their true IP address through its unique architectural topology and implementation. Furthermore, Tor offers additional mechanisms that disable the execution of scripts or the collection of user data through cookies during the session, ensuring that the user's identity will remain hidden (He et al., 2014). Figure 2.3 illustrates an internet communication between a Tor user and a website "site.com" and documents the type of information which could be accessed or recorded by each internet entity (Foundation, 2017).

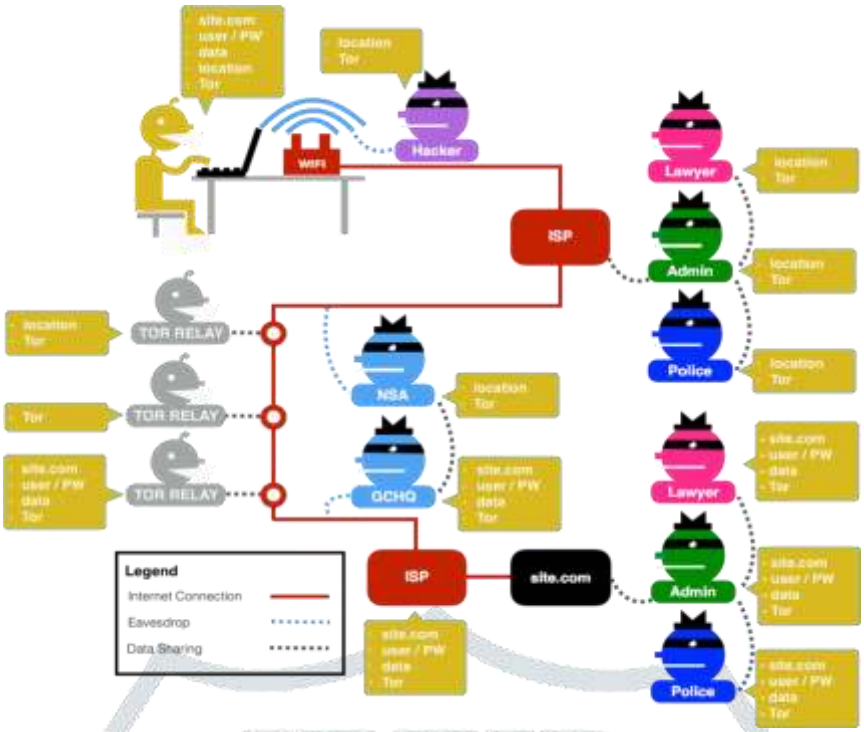


Figure 2.3 Recorded information during an encrypted communication session between a Tor user and a site.com (Foundation, 2017).

2.6 Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) resides in the category of Public Key Cryptography (PKI), also known as Asymmetric Key Cryptography. Despite the common origin with other PKI algorithms, ECC differentiates substantially in its methodological approach, involves an algebraically defined structure of elliptic curves, over a set of finite fields consisting of integer arrays of prime numbers (Hankerson et al., 2006). In a simplistic manner, during the execution of ECC, a number of discreet and randomly selected points of a curve form a multidimensional dataset, which is then divided by a set of prime numbers. The obtained array of elliptic curve data points over the finite fields of integers is used, in combination with the imported algorithmically-based generated key, to form a single layer of encryption (Cohen et al., 2005).

The original Tor team developed the onion routing system (Goldschlag et al., 1996, Goldschlag et al., 1999), known today as The Onion Router, based on the research network concepts of low and high-latency communications, firstly introduced by Chaum (Chaum, 1981) and the principles of ECC. In the initial onion routing algorithm, a user randomly selects and builds a circuit path, consisting of a sequence of onion routing nodes. Each node is assigned a permanent public key which is shared with the user during the preliminary process of the random network path construction. The user's message is encrypted using

the recently obtained public keys in the sequential order that the message will be propagated through the onion routers. Each individual node decrypts the message using its locally maintained private key, resulting in the sequential stripping of encryption layers until the final node is reached. However, there are major drawbacks in this approach which are significantly reducing the security and privacy of Tor.

Consider a path consisting of S ↔X ↔ Y ↔ Z ↔D, where S is the source, D is the destination and X, Y, Z are intermediate onion nodes. Based on the original Tor approach, the public keys of X, Y, Z are shared with S during the process of network path construction. However, only X maintains a direct connection with S. Consequently, all public keys pass through X to reach S. If an attacker compromises X is able to use the private key of X to decrypt the outer layer of encryption revealing the identity of its successor Y node. The adversaries will then attack Y and perform the same operation, reaching Z. At this point, the attacker has obtained X ↔ Y ↔Z, which is the randomly selected, secret path of onion nodes used by S. Figure 2.4 illustrates the lack of Perfect Forward Secrecy (PFS) in Tor, as is defined by cryptographic terms (Adrian et al., 2015).

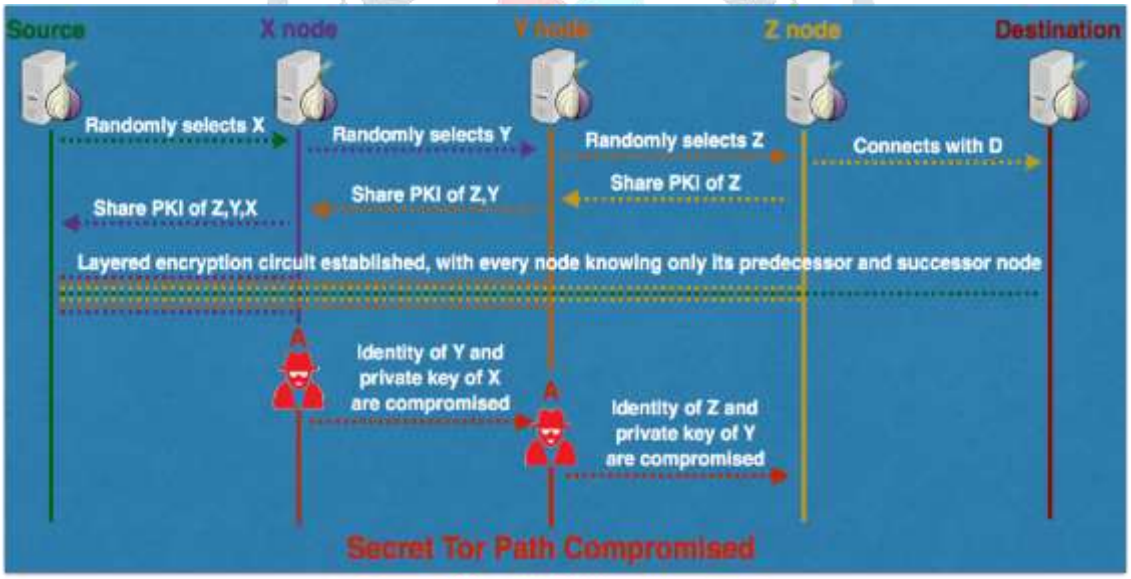


Figure 2.4 The problem of Perfect Forward Secrecy in the initial version of Tor. (Adrian et al., 2015)

2.7 The Second-Generation Onion Router

The Tor team has recognised the need for security improvements, including the lack of PFS protection (Kate et al., 2007). In 2004, Dingledine Roger, Mathewson Nick and Syverson Paul published with the permission of NRLW (Naval Research Lab Washington) and the supervision of DTIC (Defense Technical

Information Center) the specification "Tor: The Second-Generation Onion Router", which is the Tor implementation used until today (Dingledine et al., 2004). The second-generation Onion Router significantly improves the original design, by introducing, replacing and integrating new capabilities and features, as follows:

2.7.1 Perfect Forward Secrecy

As seen in the original design, onion routers used permanent and long-lasting public keys to both initiate a conversation and decrypt the outer layer of traffic passing through them. The compromise of a single node could lead to the exposure of subsequent nodes and ultimately to the decryption of the traffic, as keys were locally maintained.

Recognizing the above design flaws, the second-generation Tor changed the authentication procedure and introduced a different way of encryption by introducing telescoping path design (Snader and Borisov, 2008, Snader and Borisov, 2011).

Suppose a user (U) randomly selects nodes X and Y to build a telescoping path. U initiates a three-way TCP (Transmission Control Protocol) handshake communication with X in order to exchange symmetric keys. After the keys are verified and the communication is encrypted, U and X negotiate a temporary session key with X being the initiator, meaning that the session key is generated by X. Once the session key has been confirmed between the two nodes, the nature of encryption changes and uses as encryption key the newly generated and negotiated session key. Up to this point a single Tor hop (connection) has been established. The same procedure is repeated between U and all successive nodes until the circuit is complete, as it is illustrated in Figure 2.5.

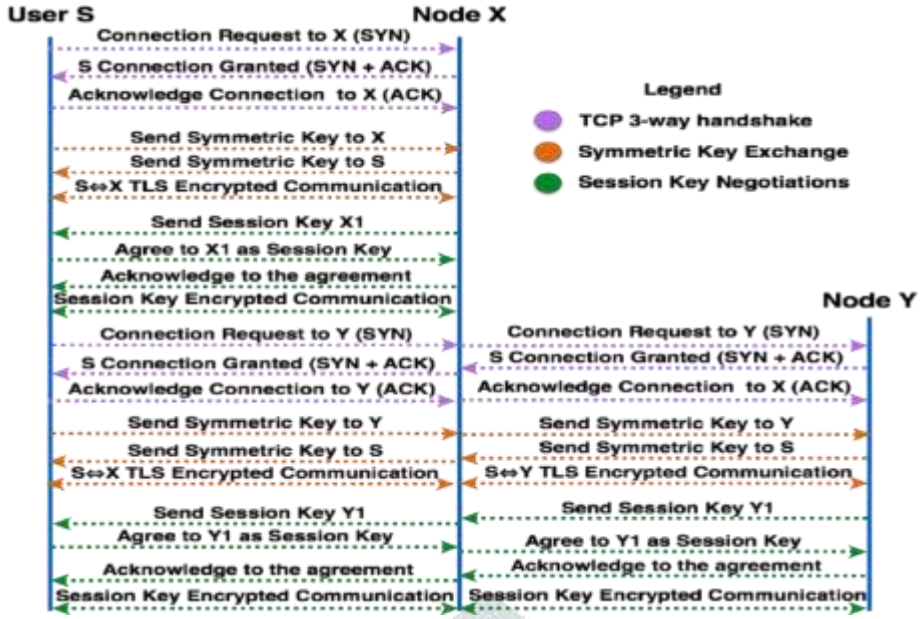


Figure 2.5 Tor implementation of telescoping path-building design (Source: Naval Research Lab Washington (NRLW) 2007).

2.7.2 Tor Proxy Interface

Tor was originally designed to serve as an encrypted military communication channel by utilising its own unique network protocol (Goldberg, 2006), named TAP (Tor Authentication Protocol). However, TAP was exclusively used in the Tor network and required each application to provide its own "application proxy" software, to achieve compatibility. As a result, when Tor became publicly available, it assumed that application developers would implement their own "application interpreters" of TAP protocol, something that never happened.

Recognizing the significant impact of this limitation, the second-generation onion router incorporated the SOCKS (Haraty and Zantout, 2014) standard, cross-platform application proxy which is supported by the vast majority of TCP-based applications. In recent years, Tor has allowed users to optionally select their preferred application proxy software, such as StegoTorus (Weinberg et al., 2012), Privatoria (Privatoria, 2017) and AirVPN (AirVPN, 2017), though the default is still SOCKS (Socks-proxy, 2017) with sock4 and sock5 being the de-facto network protocol standards.

2.7.3 Multiplex TCP Streams

Due to the lack of a universally compatible application proxy and the initial utilisation of Tor by a small number of applications, the original Tor design involved the establishment of separate network paths for individual applications. In a hypothetical scenario where a user was simultaneously browsing the web, chatting and emailing through Tor, three separate network paths would be constructed and maintained.

Consequently, this operation would require the simultaneous use of multiple public keys depending on the request that would arrive to the user. Additionally, a single user would consume a network bandwidth three times bigger than a single network path and would require the computational power of approximately x^3 onion nodes, where x is the computational power required by a single Tor path (McCoy et al., 2008). To address these deficiencies, the second generation Tor introduced a multiplexed approach of TCP connections, meaning that a single circuit it is used to propagate all of the user's traffic through the Tor network. This choice reduced significantly the amount of overall network resources required and substantially improved network efficiency and flexibility (Girry et al., 2015).

2.7.4 Directory Authority Servers

One of the most significant modifications of the original Tor design is the introduction of a new trusted category of servers known as Directory Authority (Haraty and Zantout, 2014). As illustrated in Figure 2.6, when a user initiates the construction of a Tor network path (No. 6-9), the first step always involves the user's connection with a group of Directory Authority servers (No. 1-4). That is, the locally maintained Tor proxy software establishes an encrypted and secure communication and downloads through HTTP or SOCKS a list of all available Tor relays (No. 6).

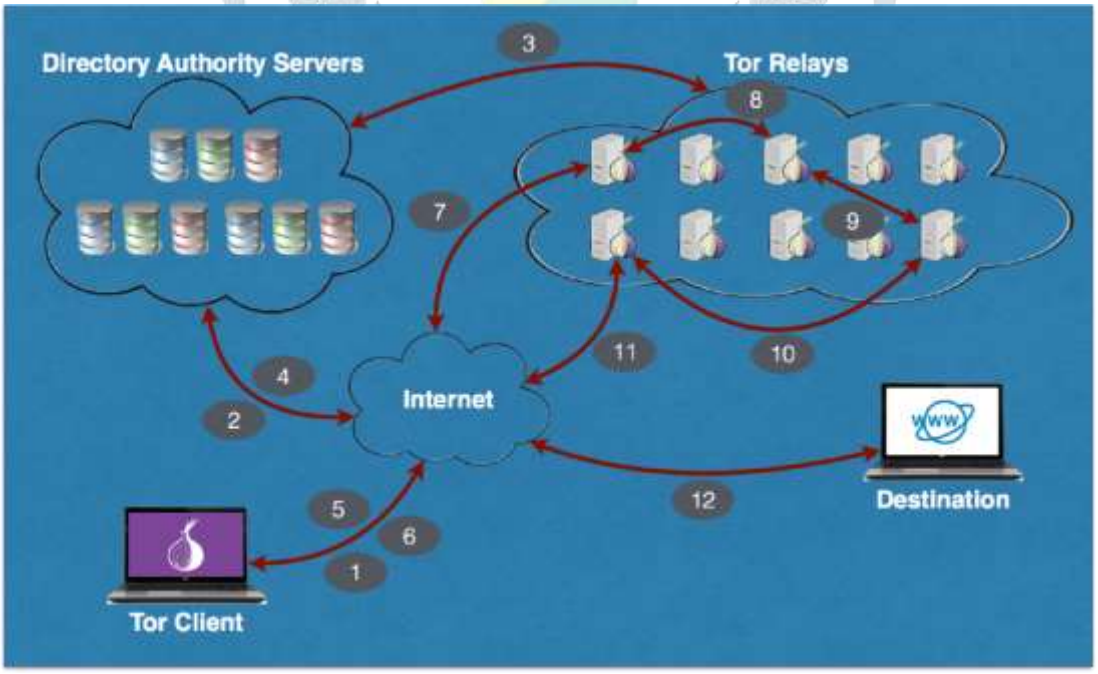


Figure 2.6 The place of Directory Authority server in the Tor infrastructure (Source: Electronic Frontier Foundation (EFF) 2012)

In the current implementation of Tor, there exist 9 Directory Authority servers distributed around the globe and maintained by trusted, volunteer entities working for the Tor project (Atlas, 2017). All Directory Authority servers are responsible for maintaining a Tor live documentation named Consensus (TorProject, 2017d). Consensus contains a master list of all available Tor Relays, with the exception of Bifroest (TorProject, 2016) Directory Authority which contains a master list of all available Tor bridges. Besides the name and IP address of all relays, consensus also provides a variety of other important characteristics related to specific nodes. These characteristics include, but not limited to, cryptographic signatures, server status, available bandwidth, open ports and accessible services.

Consensus is constantly maintained, updated and upgraded every 15 minutes based on the Directory Authority voting system. The voting system workflow is as follows:

i. Each DA receive diagnostics and, calculates and counts the behaviour of each Tor relay
ii. Each DA compiles its own version of a master list of Tor relays consisting of all previously computed elements
iii. Each DA forwards its master list to the all other DAs with a “status-vote” flag enabled
iv. Each DA waits for an acknowledgement from all recipient DA
v. Each DA compares and evaluates the master lists of all other DAs against its own version. It then generates a new master list which consists of the most voted characteristics of each Tor relay and cryptographically signs it.
vi. The detached signatures of the master list of Tor relays is published on all Das
vii. There should be a majority of detached signatures matching, meaning that the same values and attributes of each Tor relay have been verified by distinct authorities.
viii. The most voted consensus is then published by all DAs

Consensus can be obtained by anyone through any Tor Directory Authority.

Table 2 illustrates a visual representation of the basic attributes of each Tor Directory Authority (Atlas, 2017).

Table 2.1 Tor Directory Authority Servers (June 2017)

Nickname	Bandwidth	Uptime	Country	IP	Flag	ORPort	DirPort	Type
tor26	75 KiB/s	6d 23h		86.59.21.38		443	80	Relay
moria1	500 KiB/s	11d 14h		128.31.0.34		9101	9131	Relay
maatuska	50 KiB/s	4d 16h		171.25.193.9		80	443	Relay
longclaw	38 KiB/s	15h 59m		199.254.238.53		443	80	Relay
gabelmoo	40 KiB/s	22d 12h		131.188.40.189		443	80	Relay
Faravahar	336.45 KiB/s	8d 9h		154.35.175.225		443	80	Relay
dizum	3.3 MiB/s	13d 18h		194.109.206.212		443	80	Relay
dannenberg	2.89 MiB/s	30d 5h		193.23.244.244		443	80	Relay
Bifroest	469.82 KiB/s	143d 11h		37.218.247.217		443	80	Relay

Authority This Relay is Directory Authority
Running This Relay is currently running
Valid This Relay have been "validated"
Stable This Relay is suitable for long-lived circuits
V2Dir This Relay implement the v2 directory protocol or higher

2.8 Entry and Middle Relays

As previously discussed, one of the most essential aspects of Tor, which significantly differentiates it from other existing anonymity software, is that no single entity is controlling the Tor network. The vast majority of existing Tor relays is run by volunteers. The Tor documentation provides detailed information that enables anyone to configure a Tor relay and join the thousands of existing nodes, used by hundreds of thousands of people on a daily basis (TorProject, 2017c). Running a Tor node does not provide special privileges to the owner and imposes no risks to the privacy and anonymity services offered by Tor. A volunteer server simply receives encrypted Tor traffic, removes the outer layer of encryption and transmits the message to a successive node.

The vast majority of volunteer Tor relays are used as intermediate (middle) nodes. However, the functionality and configuration between middle and entry nodes does not differ significantly. A small proportion of volunteer Tor relays that meet three specific criteria might be used as a guard (entry) nodes (Arma, 2011). The criteria are as follows:

i. The relay needs to have an uptime bigger than 12.5% of all Tor relays or be active for at least 8 days.

ii. The relay needs to provide at least the median available bandwidth within the Tor network or 250KB/s.

iii. The relay needs to provide at least the median WFU (Weighted Fractional Uptime) over Tor relays or 98% WFU.

WFU is calculated by dividing the time since the node's first appearance over the time period during which a node has been unavailable. The equation is calculated using seconds as primary unit. Consequently, WFU is a metric measurement that defines the time fraction a relay has been operational.

2.9 Exit Relays

The process of correctly configuring and managing a Tor exit relay is relevantly more complex than administering an entry or middle relay. Exit relays are responsible to remove the last layer of Tor encryption. Once this layer has been removed, the obtained traffic outcome does not incorporate any Tor security features and does not differ from traditional internet traffic. Furthermore, an exit node is responsible to connect to the public internet and establish a connection with the destination address in order to deliver the recently obtained and decrypted traffic. This procedure is achieved through the use of public DNS (Domain Name System) servers that advertise the location of distinct IP ranges.

In a nutshell, exit relays act as bridges between the Tor and the open internet, presenting their IP address as the source of any Tor traffic passing through them (Winter et al., 2014). As a result, volunteers running Tor exit relays might become the subject of police investigations or receive abuse complaints by ISPs (Internet Service Provider). Recognizing the increased legal risks, the Tor Project has provided a detailed list of guidelines for running Tor exit relays (TorProject, 2010b), as follows:

- **Collaboration with ISP:**

It is recommended to run a Tor exit relay from colocation data centers (i.e. data centers where equipment, space and bandwidth are rented) and avoid exit relay deployment from the owner's house as this might draw increased attention to the owner's physical location. In addition, it is a good practice to select a trustworthy ISP and read carefully its enacted AUP (Acceptable User Policy) to identify whether required services such as additional IP, SWIP (Shared Whois Project) and reverse DNS are available. To assist volunteers, Tor documentation provides a list of suggested, previously used ISPs and a list of not recommended ISPs (TorProject, 2017e).

The final step during the selection process is for the volunteer to communicate with the ISP and explain in depth the Tor functionalities and its importance, by providing useful Tor links such as Tor Users, Tor Overview, Tor Legal FAQ and Tor Abuse FAQ (TorProject, 2017b).

- **IP Segregation:**

The use of a separate IP enables both the volunteer and the ISP to better monitor and manage the traffic passing through a Tor exit relay. Since the volunteer has no control of the type or context of traffic passing through, any abuse complains and DMCA (Digital Millennium Copyright Act) notices can be linked directly to the exit relay and not to the individual.

Consequently, IP segregation provides the volunteer the necessary time to respond and reduce the risk of personal internet suspension by the ISP. To further assist volunteers, the Tor documentation provides a boilerplate response to abuse complaints that can be slightly modified by the volunteer and quickly submitted to the ISP (TorProject, 2014). In addition, the use of a separate IP block enables the volunteer to perform a SWIP (Zhou et al., 2015) registration and display contact information which can be used in the case of abuse complaints. This functionality enables the volunteer to personally control and respond to abuse complaints, which in turn significantly reduces the risks of legal complications with ISPs.

- **Reduced Exit Policy:**

The setup of a legally acceptable and technically feasible exit policy is a necessary evil that requires careful considerations by the volunteer. However, once efficiently completed, it provides an effective mitigation technique that addresses the vast majority of legal complications. An important step, which precedes the exit policy configuration, is the selection of a practical and meaningful reverse DNS name for the exit relay. Good examples are the “tor-exit.yourdomain.org” or “tor-proxy-readme.yourdomain.org”. In addition, Tor allows exit relays, with DirPort service enabled on port 80, to publish a disclaimer notice which informs the users about the purpose of the server and its available services. These disclaimer notices can be customised directly, through the Tor configuration file, using the option “DirPortFrontPage” which imports an html file to the server’s web directory. Country specific disclaimer notice templates are available through GitHub (chgans, 2014). Figure 2.7 illustrates a disclaimer notice obtained from an exit relay with IP 107.191.56.192 (Choopa, 2017).

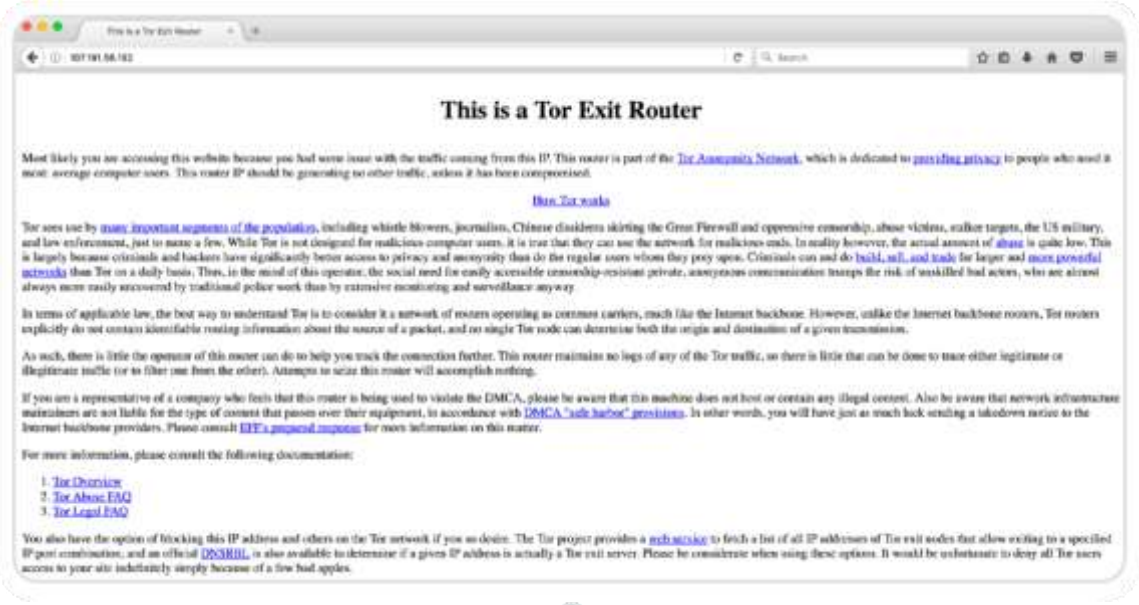


Figure 2.7: Tor exit relay notice disclaimer (Choopa, 2017).

Tor's default exit policy is to allow all available internet services which, in certain cases can result to an increase of abuse complaints. In particular, an extensive use of BitTorrent has been observed in the exit relays that adopt the default allow policy (TorProject, 2010). This can significantly increase the likelihood of receiving a DMCA abuse complaint. It is fairly difficult to prohibit BitTorrent traffic passing through an exit relay as BitTorrent clients can be run on any port. Nonetheless, the Tor team has recognized the necessity for a more restrictive exit policy in certain legal jurisdictions. The Tor documentation provides a number of reduced exit policies based on the level of restrictions a volunteer might want to employ (TorProject, 2017g). An example of a basic web browsing reduced exit policy is as follows:

```
ExitPolicy accept *:53 # DNS
ExitPolicy accept *:80 # HTTP
ExitPolicy accept *:443 # HTTPS
ExitPolicy reject *:*
```

2.10 Circuit Construction

The construction of a Tor circuit is a fairly complicated process that requires the concurrent collaboration of distinct network entities in order to be successfully completed. Each onion relay (OR) generates, over short periods of time, a relay "descriptor" file which contains contact information, measurements and statistics related to the server. This information includes, but not limited to, the server's IP address, open ports, public key and available bandwidth. All ORs commit their "descriptor" files to the Tor Directory Authorities (DA), which are responsible to agree and then publish the latest version of consensus.

The first step of creating a Tor circuit is for a Tor Client, also known as Onion Proxy (OP), to download the latest version of consensus from any available DA. Once the consensus is downloaded, the OP initiates an offline OR selection process aiming at generating an array of distinct network paths that fulfill four specific criteria:

- An OR must exist only once per circuit.
- ORs that belong to the same class B network (/16) or belong to the same family (operator) are not allowed on the same circuit.
- A circuit must consist of both "fast" and "stable" ORs. "Fast" flags are assigned by DAs to ORs with low down time (offline) and available bandwidth among the top 7/8 of known active ORs. Additionally, "stable" flags are assigned to ORs if their weighted mean time between failures is at least the median of all known active ORs (Winter et al., 2016).
- Only three entry relays are selected and used as guards for all generated circuits (Dingledine et al., 2014).

During the selection process the OP executes a bandwidth-weighted based algorithm which produces Tor network paths consisting of an exit, an entry and a middle relay. The generated paths are cross checked against the above four criteria, before being imported into an offline list of acceptable Tor circuits. By design, each circuit will not be used for a period longer than 10 minutes.

In summary, the OP generates an index list of all available ORs based on an $\sum_{x=1}^{n} \frac{f(x)}{n}$, evaluation function where n is the maximum number of criteria and $f(x)$ is the criteria's measurement (such as bandwidth), obtained through the consensus (AlSabah and Goldberg, 2016). By design, the algorithm favors the selection of higher-bandwidth ORs. However, lower-bandwidth ORs are also selected to reduced traffic congestion over faster network routes. In general, the probability of an OR being selected is $b_i \sum_{w=1}^{r} b_w$, were b_2 is the OR's available bandwidth and r is the total number of ORs in the Tor Network. Once the list of acceptable Tor circuits is built, the OP performs a preliminary end-to-end integrity checking to verify the server's availability and then initiates the construction of the Tor circuit as illustrated in Figure 2.8.

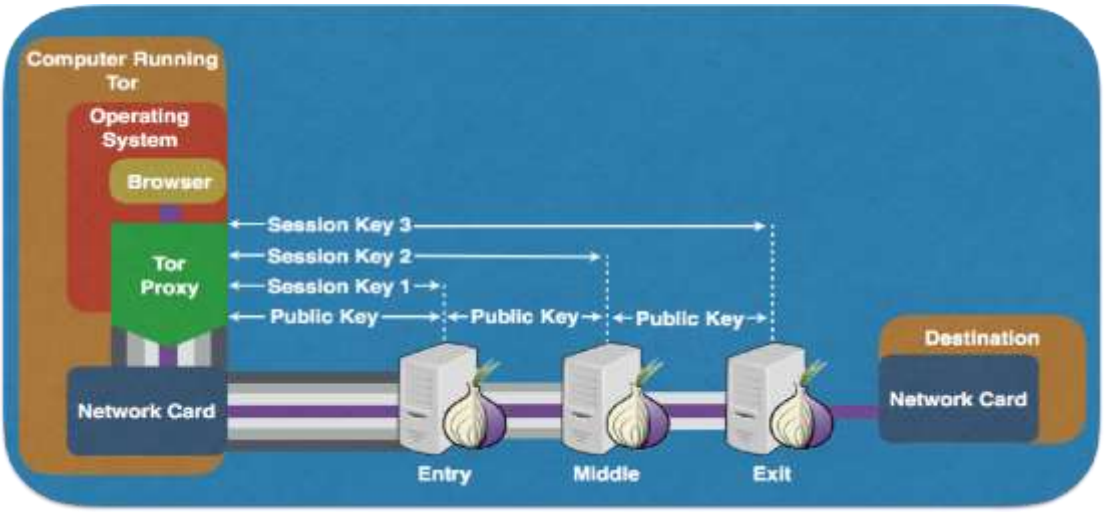


Figure 2.8 Tor circuit construction and data transmission (Source: Electronic Frontier Foundation (EFF) 2012)

2.11 Cells

As illustrated in Figure 2.8, a Tor proxy receives data from the application layer (browser), encrypts it as discussed in section 2.3 and propagates it through the Tor network using network frames, known as cells. Upon arrival to a Tor relay, cells are processed by the kernel TCP receiver buffer (AlSabah and Goldberg, 2016) and then are stored in an input buffer for further processing. Once cells have been encrypted / decrypted based on the server's type and the network route, they are placed into a FIFO (First In First Out) queue. The kernel scheduler then retrieves cells from the queue and stores them in an output buffer, used by the kernel TCP sender buffer to forward cells to the successive node. As previously discussed, Tor can support multiple TCP connections over a single circuit. Subsequently, each Tor relay is required to maintain several inputs and output buffers in order to avoid cross – circuit interferences (AlSabah and Goldberg, 2013). Figure 2.9 illustrates the exchange of cells at the Tor application-level queuing architecture.

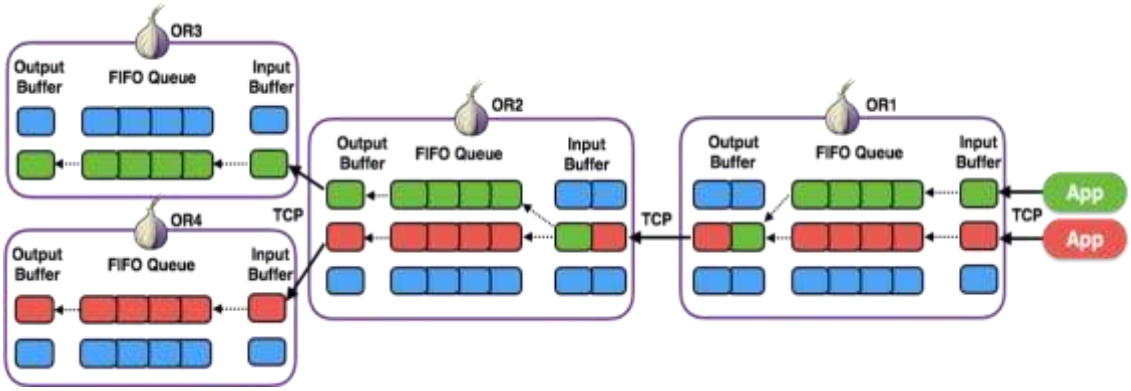


Figure 2.9 Tor application-level queuing architecture (Source: Electronic Frontier Foundation (EFF) 2012)

Each cell in the Tor network has a fixed-size length of 512 bytes and consists of a cell header (3 bytes) and a cell payload (509 bytes). The cell header is subdivided into a circuit identifier (2 bytes) and a command (1 byte) that specifies how the cell payload will be used. Cells can be categorized as either control or relay cells based on the command variable. Control cells are used by Tor nodes to maintain the active circuit while relay cells carry the user's data stream through the Tor network.

The major difference between the two types of cells is the additional relay header (11 bytes) included within the data payload of relay cells. A relay header consists of: (i) a stream identifier (2 bytes) used to distinguish different data streams that pass through a single circuit, (ii) a checksum variable (6 bytes) used to conduct end-to-end integrity checking, (iii) a data stream length (2 bytes) and (iv) an additional relay command (1 byte) used to manage the data stream. Relay headers and relay payloads (498) are transmitted through the network in the form of a cipher stream generated using a 128-bit AES (Advanced Encryption Standard) cipher in CTR (counter) mode. Figure 2.10 illustrates a visual structure of a cell and provides two tables depicting the most frequent control and relay cell commands. Additional information is available at TorSpec documentation on GitHub (Dingledine, 2011).

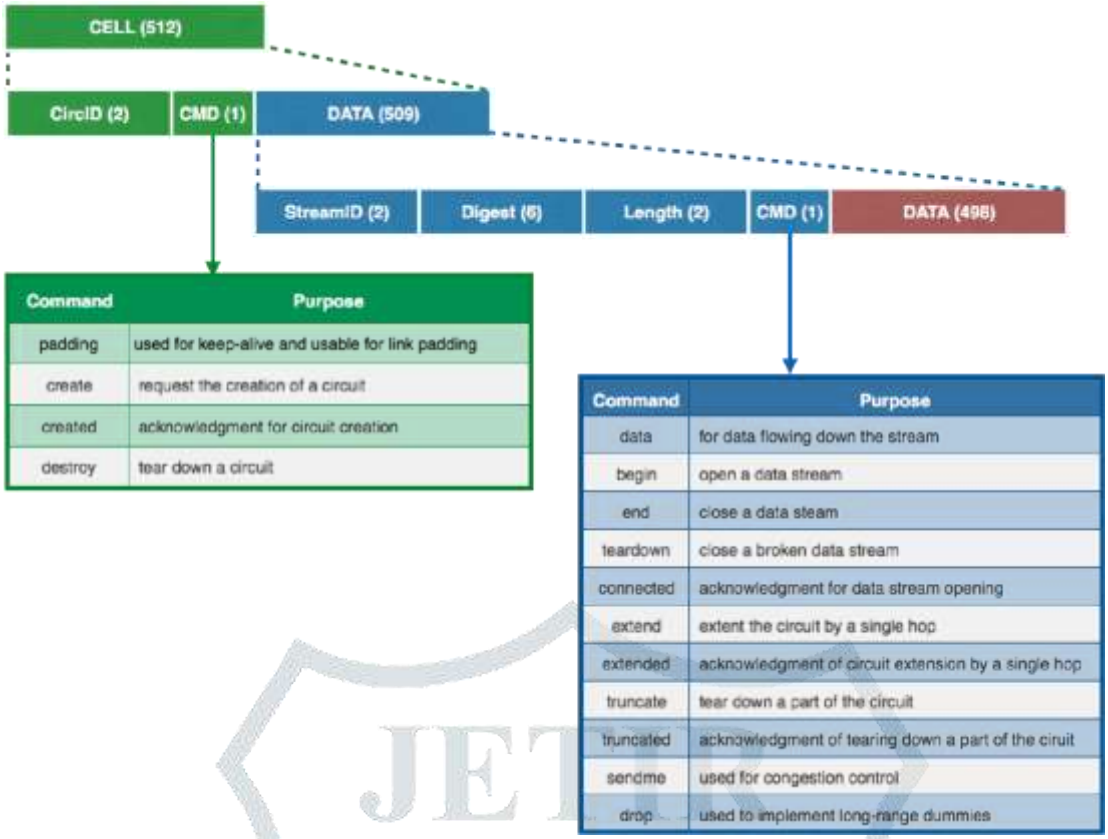


Figure 2.10 Relay cell structure and components (Source: Electronic Frontier Foundation (EFF) 2012)

2.12 Current State of Tor

To date, Tor continues to be one of the most prevalent and supported anonymity networks in the world and is used by approximately two million users every day (TorProject, 2017i). Over the years, a lot of researchers have embraced the open source nature of the project and have suggested ideas and improvements to further augment Tor's capabilities. Figure 2.11 summarises research ideas and improvements suggested by the research community up to 2016 (AlSabah and Goldberg, 2016) and presents them as branches of a mind map diagram.

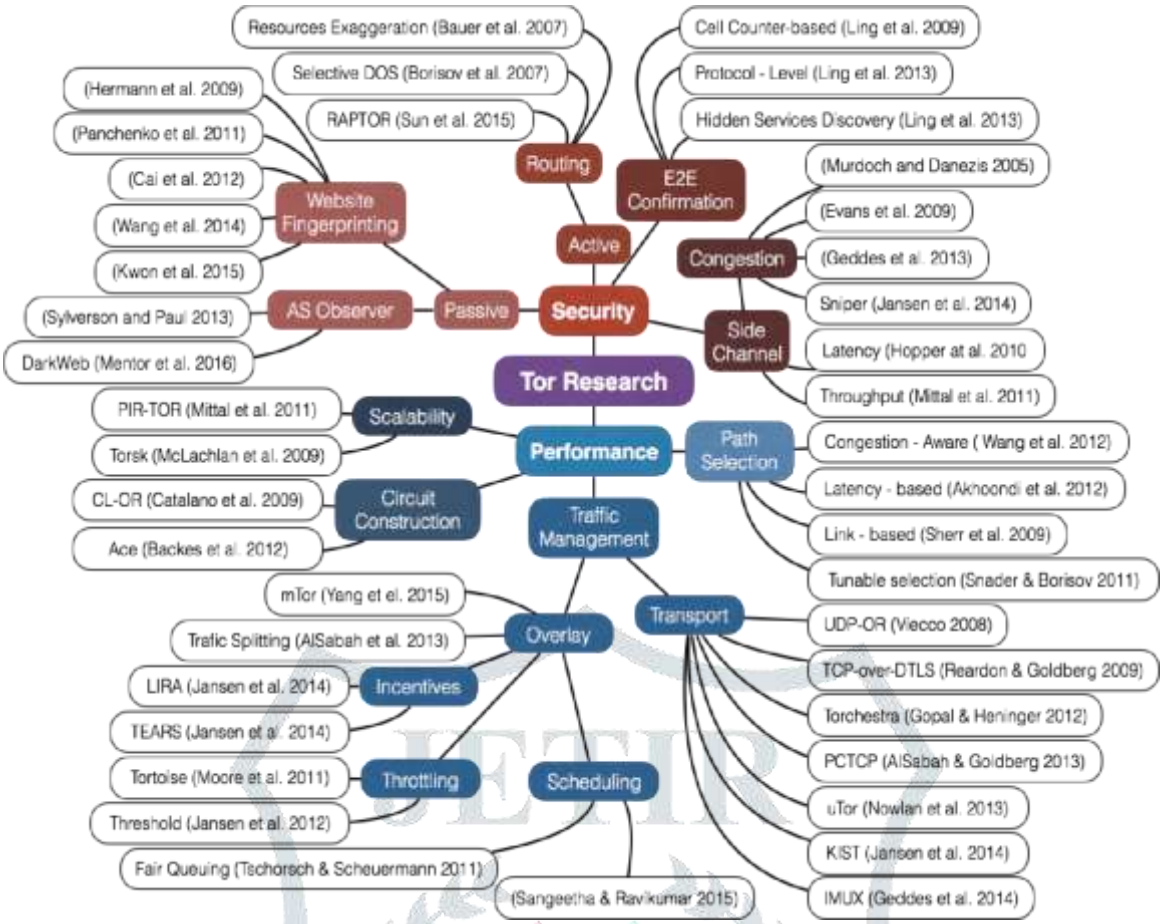


Figure 2.11 Contributions to the Tor open source project (Source: AlSabah and Goldberg 2016).

2.12.1 Cryptography

The most recent cryptographic controls and function used by Tor (Martin, 2017) are:

- **Public Key Cryptography**

As discussed in sections 2.3 and illustrated in Figures 2.3 and 2.7, Tor facilitates symmetric key cryptography between network entities. The current Tor version supports PKI of 1024-bit RSA and 256-bit elliptic curve cryptography based on Curve25519.

- **Symmetric encryption**

Cell payloads are further encrypted using 128-bit AES in CTR mode, as discussed in section 2.8.

- **Hashing Algorithm**

The digest variable, included in the encrypted cell payload, utilises the SHA1 hashing function, which is used for end-to-end integrity checking as discussed in section 2.8. According to the Tor team, the algorithm will gradually shift to the SHA256 hashing function until the end of 2017 (TorProject, 2017f).

- **SSL / TLS**

To further increase confidentiality, data authenticity, entity authentication and guaranteed Perfect Forward Secrecy.Tor is utilising SSL (Secure Socket Layer) over TLS (Transport Layer Security) in all network communications. The supported cipher suites are either AES (Advance Encryption Standard) or 3TDES (Triple Data Encryption Standard) in CBC (Cipher Block Chain) mode.

Figure 2.12 provides a visual structure of the major encryption algorithms used in a Tor communication channel.

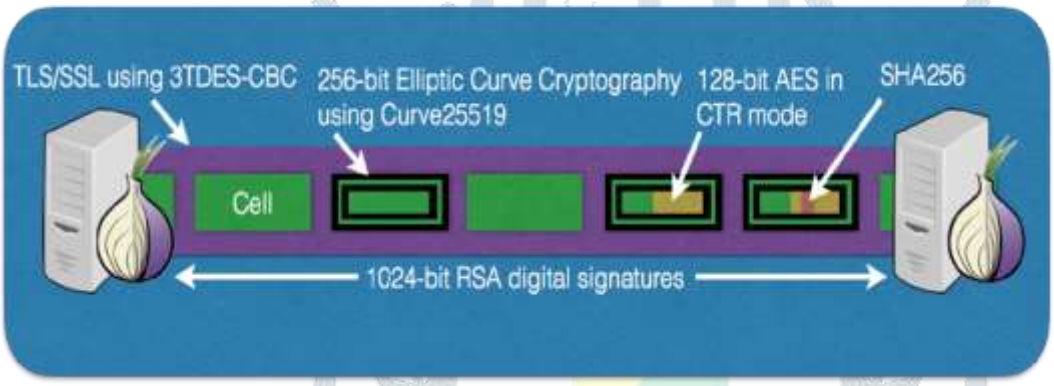


Figure 2.12 Major encryption algorithms used in the Tor network (Source: Electronic Frontier Foundation (EFF) 2012)

2.12.2 Network Infrastructure

As of June 2017, the Tor network consists of 7157 relays, including 9 Directory authorities, 2499 entry relays and 798 exit relays (Gabert, 2017). Figure 2.13 illustrates additional Tor network statistics obtained on the 20th of June, 2017 through the TorStatus open source application project.

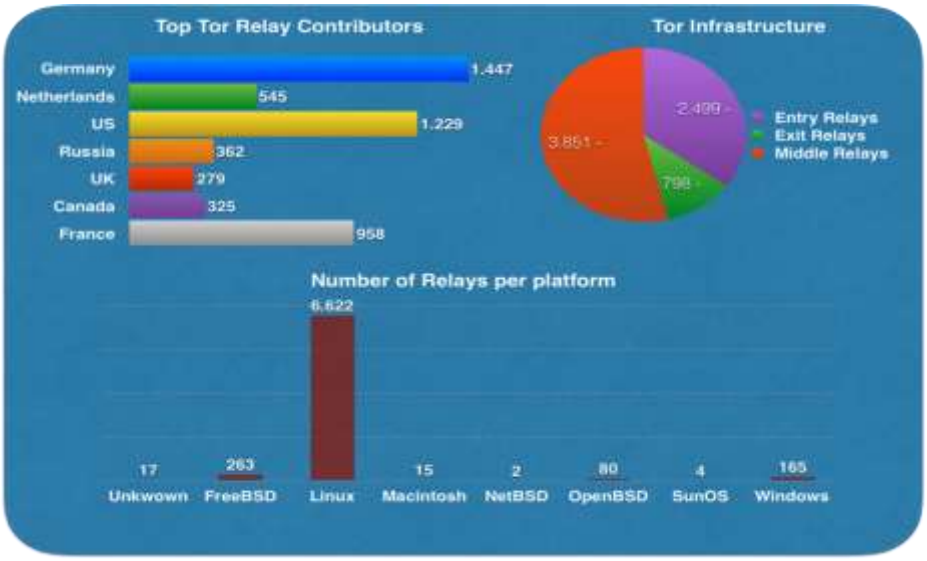


Figure 2.13 TorStatus statistics (Source: Electronic Frontier Foundation (EFF) 2012)

CHAPTER THREE

RESEARCH METHODOLOGY

3.0 Introduction

This chapter presents the research design and formulates the research question whose answer will assist in addressing the aim and objectives of the study. The adopted research methodology is justified along with the tools, techniques, approaches and processes used for the duration of the research. The ethical dilemmas and concerns involved are discussed along with mitigation techniques which reduce to the minimum any residual risks.

3.1 Research Question

- **To what extent can the concepts and principles of Tor help in minimizing privacy concerns in online interactions?**

3.2 Research Objectives

The following three research objectives (re-aticulated with sub objectives) are redefined to align with the research question and is used to define the boundaries of the study.

1. To develop an isolated Tor experimental network

    a. To develop an isolated network infrastructure

b. To integrate Tor in the produced isolated network infrastructure

2. To analyze and understand in depth the Tor capabilities regarding its implementation, network architecture, cryptographic controls and design.
3. To critically evaluate the performance of the developed isolated Tor experimental network and define its limitations.

Each research objective contributes towards addressing the research question, as follows: Establish the significant characteristics of online privacy and anonymity and achieve a deep understanding of the technical implementation of Tor. Select the necessary components required in the accomplishment of the study aim.

3.3 Research Approach

The specified research objectives were divided into two distinct categories based on the best methodological approach to address them. The first objective was addressed using an experimental-based approach in accordance with the description provided by (Kothari, 2004) and it is consistent with the approach provided by Yanow (2015). Objective 2 is addressed through a literature perusal as advocated by Fixsen et al (2005) and Yanow (2015). A literature-based methodology, also known as the "state-of-the-art assessment" (Boudreau et al., 2001), defined it as the process of reading, analyzing, critically evaluating and thoroughly summarizing research-related scholarly materials, a view consistent with that of Saunders (2011). These authors formed data clusters from secondry data sources in line with the suggestions of Heaton (2008). According to Kothari et al., (2004), a literature based methodology enables the researcher to cover a large proportion of the research surface and justifies that the collected data is accurate and reliable. In contrast, empirical studies promote the gathering of information through questionnaires, surveys or interviews. Such practices face several challenges as it is necessary to verify the trustworthiness of the data and remove any bias introduced by the behavior of participants (Ajzen, 2002).

The knowledge gained from the literature based objective 2, is used for the duration of the project to provide justification and reasoning during the development phase, validity during the testing and verification during the analysis. Besides any assumptions, which are clearly highlighted and emphasized, this research attempts to strengthen the obtained results by relating them back to the literature.

The second phase of the research (objective 1 and 3) consolidated the knowledge gained from objective 2 and involves the development of a flexible, efficient network infrastructure which utilizes the majority of capabilities and features provided by Tor. The process of conception, construction and deployment of infrastructure in this study took a significant amount of experimentation and testing. Additionally, the researcher's prior technical knowledge and understanding of computer networks, cryptographic controls, architectural design and programming logic, was a crucial precondition, that aidded the successful completion of research objective 1.These second phase of the research embraces the practices and principles of an experimental-based methodology.

Experimental research was divided into two mutually inclusive schemes (Kothari, 2004), as follows:

i. Informal experimental design: The use of observation-based comparison analysis that differentiates the initial experimental state, also known as "benchmark", from a collection of divergent states (Ryan and Morgan, 2007).

ii. Formal experimental design: The use of sophisticated techniques and precise measurements to identify and analyze the transformation of controlled variable and define the rate and time frame in which the changes occurred (Montgomery, 2008).

Therefore, phase two, which is the development of the Tor network adopted the process of an informal experimental design, by establishing an initial simulation environment as benchmark and then building on it. Finally, the evaluation of the suggested Tor experimental network (phase 3) endorsed the formal experimental approach by assessing whether the proposed infrastructure fulfills the privacy and anonymity characteristics that Tor seeks to provide. Hence, the extent to which the proposed experimental Tor network supports these features and the way it combines crypto and networking to achieve its goals, are evaluated in order to unveil any limitations.

The combination of experimental and literature based, research approaches and the sequential network development, analysis and evaluation process assure the verifiability, accuracy and consistence of the research results. Subsequently, this research exhibits traits from the post-positivist research philosophy.

3.4 Literature Overview

About a third of the overall research weight has been allocated to the conducted literature review, as the expected outcomes are crucial in initiating the second phase of the project. The literature examined centers around two main areas: the characteristics and importance of online privacy and anonymity, and how Tor addresses these through its unique architectural implementation. Since both domains incorporate a broad range of components and characteristics, a content selection was performed, based on research relevance and scope.

3.5 Experimental Design

The first objective of the research is the development of a Tor experimental network that integrates the majority of principles and characteristics, identified through the literature review. Due to its complexity and size, objective one is divided into two smaller sub-objectives. Chapters three and four provide an insight into the network development process and illustrate the sequence of steps, procedures and tools followed in order to integrate Tor in an isolated network infrastructure. Both chapters are presented in a documentation format aim at providing the reader with a comprehensive picture of the development process.

3.5.1 Design Choices

The development, evaluation and analysis of the proposed Tor experimental network is limited by the available time and resources. Due to cost limitations, the research was conducted using publicly available, free and open source software.

The nature of the preliminary development phase of the project requires, as a first step the selection of appropriate tools, techniques and processes that are able to provide the necessary functionality to the researcher. This subsection aims at explaining the logic behind each choice and provide in depth information for all software and hardware used. The information that follows is crucial in safeguarding the study's scientific validity and repeatability:

- **Computer System**

The entire experiment was conducted on a 15-inch, 2015 MacBook Pro laptop, with a 2.8 GHz Intel Core i7 processor overclocking up to 3.4 GHz, 16 GB of RAM, 1 TB SSD, an AMD Radeon R9 M370X and an Intel Iris Pro graphics cards, which is the researcher's personal computer.

- **Host Operating System**

The laptop's host operating system is OSX High Sierra version 10.14x, which is not the latest Apple operating system available. The researcher decided not to upgrade to the latest MacOS Mojave released in early 2018, as there was a slight risk of software incompatibility and operating system bugs.

- **Virtualization Software**

There are three major virtualization software: (i) VMWare (VMWare, 2018), (ii) VirtualBox (VirtualBox, 2018) and (iii) Parallels (Parallels, 2018). Due to existing bugs, latency, limitations and the researcher's prior unpleasant experience with it, VMWare Fusion was rejected. Both VirtualBox and Parallels were evaluated as suitable for this research project with each carrying its own benefits and limitations. VirtualBox is free in comparison with Parallels, as the latter requires a one-time subscription of $80. Despite this cost, Parallels was selected as the virtualization software as it is designed specifically to run on Mac operating systems. It also provides advanced access control, encryption and cloud backup storage capabilities, establishing it as the ideal virtualization software for apple products according to the researcher's opinion.

- **Guest Operating System**

The guest operating system deployed through Parallels virtualization software is Ubuntu 18.04 LTS (Canonical, 2018a), with a Sha256 checksum of "f5ce20686a2f3201f04a309d04171ee15757f00954b33b87f3f1d36b3b0f5356". The researcher's familiarity with the specific operating system was the main reason for its selection. In addition, 18.04 was the most stable and more convenient version of Ubuntu as of now, which is the project's starting month.

- **Network Simulation Software**

The creation of a customized network infrastructure and the subsequent Tor integration to produce an isolated Tor experimental network require the use of a flexible network simulation environment. Three distinct network simulation software options were considered prior to the project's initiation: (i) the Shadow network simulator (Jansen and Hopper, 2012), (ii) the Docker container platform (Hykes and Docker, 2017) and (iii) the low-cost experimental network simulator Netkit-ng (Cartigny, 2014).

Shadow was rejected due to the limited customization options and the preconfigured nature of the generated network. On the other hand, both Docker and Netkit-ng offer increased flexibility and allow the user to create a customized simulation network with no significant limitations. Although there exist big differences regarding the implementation of these two options (such as the amount of resources required to run a simulation environment and the development process to generate the needed infrastructure), these have been judged as not important for the purposes of this research.

Nonetheless, the researchers selected to use Netkit-ng as the network simulation software, due to the existence of a similar Tor network implemented using Docker (Antitree, 2016).

- **Traffic Capturing Software**

Traffic capturing and analysis is one of the most essential steps in reviewing and comprehending in depth the actions and operations performed on a network infrastructure. Due to its cross-platform compatibility and the increased use of a Linux terminal interface to interact with Netkit-ng virtual machines, tcpdump (Van Jacobson et al., 2017) is selected as the main traffic capturing tool. The results of tcpdump are verified through comparison with the output of uml_dump (Julien Iguchi-Cartigny et al., 2014) tool. Wireshark network protocol analyzer (Combs, 2017) is used to visualize the results in a more human readable form.

- **Network Enumeration**

Routings paths, open ports, running services, IP ranges and enacted systems can provide a comprehensive overview and are important aspects of understanding and evaluating a network infrastructure. Due to its

substantial flexibility and well documented nature, nmap (Lyon, 2017) is used as network enumeration software.

3.6 Ethics

Over the years, researchers have recognized a number of concerns (Kerlinger and Lee, 1999) and ethical dilemmas (Flicker et al., 2007), that might occur during a research process. A large part of these concerns circles around ethical issues stemming from the attendance of participants. Emphasis was given to the fact that this concerns were taken into consedration (Bryman, 2015; Holloway and Galvin, 2016).

It is important to note that ethical concerns have attracted much attention in the legal domain (Biggs, 2010) and sundry regulations are been promogated (Sieber and Tolich, 2013) which mostly concentrate on data anonymization and protection. For example in the United Kingdom Article 29 reflectes this development, “Opinion on Anonymization Techniques” (Costa et al., 2016) for General Data Protection Regulation (PARLIAMENT, 2016).

The design of this research does not involve any participants, as has been thoroughly discussed and justified. However, the absence of attendees does not eliminate the need for ethical considerations. On the contrary, this fact increases the level of complexity towards identifying potential ethical issues (Hewson and Stewart, 2016).

CHAPTER FOUR

ISOLATED EXPERIMENTAL NETWORK IMPLEMENTATION

4.0 Introduction

The real practical demonstration of the proposed concept is initiated, described in depth, setup, implemented and tested in the section following. The sections presents the sequence of events, decisions taken and actions performed during the preliminary development of an isolated network infrastructure. With the intent to provide a comprehensive, detailed documentation of the exact steps and procedures followed. Additionally, it aims at providing reasoning and logic in the decision-making process of development and illustrating the gradual network infrastructure and system functionality expansion.

Specifically, this section explains in depth the required configuration settings to set up a Netkit-ng (Cartigny, 2014) development environment in a virtual machine running

Ubuntu 18.04 LTS Linux distribution (Canonical, 2017a) and deploying it through the Parallels for Mac (Parallels, 2017) virtualization system. The section also presents and explains the steps followed to build a LAN (Local Area Network) infrastructure using Netkit-ng. The produced LAN is used as a basis for further expansion into a WAN (Wide Area Network) topology. Finally, a custom DNS (Domain Name System) is integrated into the produced WAN. Upon completion, of the above steps an isolated network infrastructure was setup that was used as a basis for the development of an isolated Tor experimental network.

4.1 Preliminary Steps

The following actions were performed on the host OSX operating system (section 3.5.1), using the default tcsh (Enhanced C-shell) Unix shell (Joy et al., 2009), in order to initiate the process of setting up an ideal development environment. The objective was to build a virtual machine that will be used for the duration of the project. Although the steps presented in this subsection can be characterized as fairly simplistic and straightforward, they play an essential role in the process of conducting a scientifically verified project. Specifically, the project's validity, reliability and repeatability heavily depend on these initial steps.

A new directory, named ***ISO***, was created in the ***Downloads*** directory, to be used as a storage location for the Ubuntu image file (line 1-2). Additionally, an optional environment variable, named ***dpath***, was set to contain the absolute directory path for ***Downloads*** using the shell command ***pwd*** (line 3), which returns the current directory.

Continuing, ***wget*** (Scrivano, 2017) was used to download ***Ubuntu 18.04.5*** (line 4) from one of the official Ubuntu repositories (Canonical, 2017b). A separate text file containing the sha256 checksums of all images was also downloaded from the same repository (line 5). The parameters ***–v***, which stands for verbose output and **–O** which specifies the directory and name of the selected downloaded files were passed as arguments to ***wget***.

```
1: cd ~/Downloads/
2: mkdir ISO
3: dpath= $(echo $(pwd))
```

```
4: wget -v -O $dpath/ISO/ubuntu-18.04.5-desktop amd64.iso
   http://releases.ubuntu.com/18.04.5/ubuntu-14.04.5desktop-amd64.iso
5: wget -v -O $dpath/ISO/sha256sums
   http://releases.ubuntu.com/18.04.5/SHA256SUMS
```

To avoid building and running a malicious and tampered virtual machine, it is always a good practice to verify the hash check sum. This simple step can verify the authenticity and data integrity of the downloaded file. The ***Shasum*** (Shelor, 2013) package, which is the apple variation of ***sha256sum*** (Ulrich Drepper et al., 2010) Linux package, was used to verify the sha256 of the Ubuntu image file (line 7). Using the ***–a 256*** parameter, the program computes and returns the ***sha256*** (line 6), but if the ***–c*** parameter is specified then the program takes as input a text file containing sha256 hashes and compares them with the hashes generated from the files located in the current directory.

It is worth mentioning that in order for the program to automatically match the two sha256 hashes, the name of the file that will be used to generate the comparison hash must be identical to the original. In simpler words, if while downloading the file (line 4), the user modifies the name from ***Ubuntu 18.04.5-desktop-amd64.iso*** to anything else, then the ***– c*** parameter (line 7) will fail to match the hashes, despite the fact that a manual comparison of the generated hash and the hash located in the text file will be identical.

```
6: shasum –a 256 ubuntu-18.04.5-desktop-amd64.iso
7: shasum –c sha256sums
```

The final step of the preliminary configuration phase was the use of the downloaded ***Ubuntu*** image file to build a virtual machine using ***Parallels for Mac*** virtualization software (section 3.5.1). The option ***Install Windows or another OS from a DVD or image file***, was selected from the GUI (Graphical User Interface) provided through Parallels, to generate an Ubuntu virtual machine. The default configuration of the virtual machine was altered to suit the required needs of the development environment. Table 4.1 presents the defaults virtual machine settings and the modified ones, providing the reasoning behind their modification.

Table 4.1 Ubuntu virtual machine configuration

No	DEAULT SETTINGS	MODIFIED SETTINGS	REASONING
1	Shared Home Directory	Disabled	Virtual Machine Isolation
2	Share iCloud	Disabled	Not Required
3	Share Dropbox	Disabled	Not Required
4	Share Google Drive	Disabled	Not Required
5	Share Linux Applications with Mac	Disabled	Virtual Machine Isolation
6	Share Mac Clipboard	Disabled	Virtual Machine Isolation
7	Apple Remote	Disabled	Not Required
8	Share Mac Printers with Ubuntu	Disabled	Not Required
9	2CPU	4CPU	Increase system processing power and speed

10	1GB RAM	8GB RAM	Reduces memory allocation/ Increase overall system performance
11	128GB GPU	1GB GPU	Optimization to assist in the use of a dual monitor setup
12	Shared network Card	Thunderbolt Ethernet	Virtual Machine Isolation
13	10GB static SSD	40GB dynamic SSD	It is currently unknown how much storage will be needed
14	Automatic update	Manual update	Maintain virtual machine in a known, stable state
15	Coherence Mode	Disabled	Virtual Machine Isolation
16	Backup disabled	Smart Guard enabled	Maintain daily virtual machine snapshots on the cloud

4.2 Development Environment Configuration

The following actions were performed on the newly created ***Ubuntu*** virtual machine, aiming at establishing an ideal development environment for the deployment of ***Netkitng***. Following the virtual

machine boot and the setting of an administrator password, ***Parallels Tools*** which is a set of operating system drivers, were automatically downloaded and installed. The installation was completed with a system reboot.

It is typically a good practice to update the system to the most recently available packages, especially in the case of ***Unix-like systems***. The procedure was easily executed using the ***update***, ***upgrade*** and ***autoremove*** commands (line 1-3) via terminal. Since this was the first interaction of the Operating System with the ***Ubuntu repositories***, the procedure might require some time to complete (10-15 minutes), depending always on the available bandwidth.

```
1: sudo apt-get update
2: sudo apt-get upgrade -y
3: sudo apt-get autoremove
```

The next step involved the use of ***wget*** to download three zip files (line 4-9) which, when extracted into a single directory, will form the ***Netkit-ng directory***. The three files included: (i) the ***core*** that contains a series of shell scripts, UML commands and Netkitng documentation, (ii) the ***filesystem*** which contains a reduced ***Debian Wheezy*** file system and (iii) a modified Debian Wheezy ***kernel*** that will be used by the virtual machines.

```
4: cd ~/Downloads/
5: wget -v -O ~/Downloads/netkit-ng-core-32-3.0.4.tar.bz2
   https://github.com/netkit-ng/netkit-ng-
   core/releases/download/3.0.4/netkit-ng-core-
   323.0.4.tar.bz2
6: wget -v -O ~/Downloads/netkit-ng-filesystem-i386-
   F7.00.1.3.tar.bz2 https://github.com/netkit-ng/netkit-
   ngbuild/releases/download/0.1.3/netkit-ng-filesystem-
   i386F7.0-0.1.3.tar.bz2
7: wget -v -O ~/Downloads/netkit-ng-kernel-i386-K3.2-
   0.1.3.tar.bz2 https://github.com/netkit-ng/netkit-
   ngbuild/releases/download/0.1.3/netkit-ng-kernel-i386-
   K3.20.1.3.tar.bz2
```

Once the three files were downloaded, the ***tar*** command was then used to unpack them into a single directory, named netkit-ng (line 8-10). The parameters ***-xjSf*** were specified, with –x used to extract files from an achieve, -j used to filter the extracted archive through ***bzip2*** which is the downloaded format, -S to handle sparse files efficiently and –f to import files into the same directory.

```
8: tar -xjSf netkit-ng-core-32-3.0.4.tar.bz2
9: tar -xjSf netkit-ng-filesystem-i386-F7.0-0.1.3.tar.bz2
10: tar -xjSf netkit-ng-kernel-i386-K3.2-0.1.3.tar.bz2
```

As specified in the Netkit-ng documentation (Cartigny, 2014), the final configuration step was setting up three environment variables (lines 11-13): ***NETKIT_HOME*** which is set to the netkit-ng directory, ***MANPATH*** which points to the ***man directory*** located within the netkit-ng directory and ***PATH*** which points to the ***bin directory*** located again in the netkit-ng directory.

```
11: export NETKIT_HOME=/home/parallels/Downloads/netkit-ng
12: export MANPATH=$NETKIT_HOME/man:$MANPATH
13: export PATH=$NETKIT_HOME/bin:$PATH
```

However, the above method imported these three environmental variables to the current ***bash profile*** making them unavailable after session termination or system reboot. To further increase the robustness and efficiency of the development environment and avoid repetition, the above environmental variables were permanently added to the system. In an Ubuntu operating system, this can be achieved by modifying the ~/***.bashrc*** file (line 14-15).

```
14: echo '#netkit-ng environmental variables' >> ~/.bashrc
15: echo export NETKIT_HOME=/home/parallels/Downloads/netkitng >>
   ~/.bashrc
16: echo export MANPATH=$NETKIT_HOME/man:$MANPATH >> ~/.bashrc
17: echo export PATH=$NETKIT_HOME/bin:$PATH >> ~/.bashrc
18: source ~/.bashrc
```

Line 18 reloads the modified ~/.bashrc profile without the need for a system reboot. It is also worth mentioning that one must be careful ***not to use single >,*** when importing lines to the bashrc file using ***echo***, as this command will overwrite the existing bashrc file.

At this point the default Netkit-ng configuration had been completed. To verify the correctness of the configuration and check whether all required packages existed in the underlying Ubuntu operating system, a script located in the netkit-ng directory, named ***check_configuration.sh***, was provided. Once the script was executed (line 19-20), it verified the existence and correctness of the environmental variables, checked if the current shell session was allowed to execute ***nineteen distinct system calls*** and verified the presence of ***six additional packages***.

Upon running it, the script could not locate ***xterm***, ***konsole*** and ***Wireshark***, since these three tools are not part of the default configuration of Ubuntu. The tools were installed via terminal (line 21-23) and, after completion; the ***check_configuration.sh*** provided an error message "***System appears not to be able to run the Linux kernel***". As it was later discovered ***Netkit-ng*** requires two additional libraries (libc6:i386, libstdc++6:i386) that were installed (line 24). At this point Netkit-ng has been successfully installed and configured.

```
19: chmod +x check_configuration.sh
20: ./check_configuration.sh
21: sudo apt-get install wireshark
22: sudo apt-get install konsole
23: sudo apt-get install xterm
24: apt-get install libc6:i386 libstdc++6:i386
```

4.3 Netkit-ng Basic Workflow

Netkit-ng is a software-based network emulator with the ability to reproduce complex computer network scenarios and efficiently virtualize them on a single machine. Although it is possible to customise the filesystems located within each virtual machine, by design all virtual machines share the same core, kernel and filesystem, establishing Netkit-ng as one of the most light-weighted, open source network emulation tools (Pizzonia and Rimondini, 2016).

The creation of a Netkit-ng network is a relatively easy process with few restrictions, as listed below:

- All configuration files and subdirectories must exist in a ***single directory*** (e.g. lab1)
- A text file named ***lab.conf*** must exist
- Each virtual machine requires a separate directory (e.g.vm1) and a text file with the extension ***.startup*** (e.g. vm1.startup)

Additional functionalities and capabilities can be optionally added to the virtualized network through other pre-defined directories and file extensions. Detailed documentation can be found at the online manual of Netkit-ng (Rimondini, 2010).

4.4 Network Evaluation Criteria

To better understand the quality of each produced emulation network and establish means of comparison between different network versions, a list of network evaluation criteria have been defined and used for the full duration of the network development, as follows:

- **Accuracy**

Each produced network emulation environment is expected to behave in a certain way, based on the actions performed and option selected during its configuration. Upon completion of each network version, a series of short experiments are performed in each virtual machine aiming at collecting information that can be used at a later point to evaluate and compare different network versions. The commands and techniques used to collect information are as follows:

i. Routing paths

Initially the ***route –n*** (Phil Blundell and Eckenfels, 2014) command is used in all virtual machines to obtain the default routing path. The results are validated by comparing them with the output of the ***netstat -rn*** (Microsoft, 2012) command. Wherever the complexity of the network allows it, ***traceroute <IP> / domain*** (Wood, 1996) command is used to further extend the validation and reliability of the results.

ii. Running services

One the most common ways of obtaining running services on a Linux operating system is through the ***service –status-all*** command. The command lists the current state of all services controlled by ***System V***. The ***+ symbol*** indicates that the service is running, the ***– symbol*** that the service is stopped and the ***? symbol*** indicates that the service is managed by the ***Upstart***, which is executed during the operating system boot up.

iii. Open ports / connections

The ***netstat –tulpn*** is used to obtained all TCP and UDP listening ports from a Linux operating system. ***Nmap*** (Lyon, 2017) is another tool that can be used to scan an entire network and provide useful network information such as open ports and associated services.

- **Portability**

One the main purposes of developing an emulation network is to provide individuals with a portable, light-weight environment that can be easily shared. One of the factors affecting the portability of an emulation environment is the storage consumption of the system. The command ***du –ah*** (Torbjorn Granlund et al., 2016) is used on the host operating system (Ubuntu) to extract the memory allocated to each virtual machine (***.disk file***) immediately after the execution of the ***lstart*** command.

- **Speed**

Another important factor that defines the quality of a network emulation environment, is the time required for the network to boot up and come into an operational state. Although there are efficient ways of measuring the time taken for ***lstart*** to be completed, the obtained results will not be realistic since the host operating system is not able to determine the state (***booting / operational***) of each netkit-ng virtual machine.

Therefore, the ***Clock*** (Meanterm, 2016) mobile application running on an ***iPhone 8*** was used to count the time taken for each virtual machine to boot up. The timer was initiated upon the execution of the ***lstart*** command and was terminated the moment each virtual machine had finished booting up and was ready to receive input ***(e.g. root@pc1:~#)***.

- **Hardware limitations**

The hardware resources required (memory, CPU) by a network emulation environment can severely affect availability and user experience. Memory allocation and percentage of CPU consumption can be obtained through the ***htop*** command, which is not included by default on the Ubuntu distribution. Therefore, ***sudo apt-get install htop*** was used to download and install the htop package.

It is also worth mentioning that each netkit-ng virtual machine is correlated with ***six operating system processes*** as follows:

i. A terminal session that executes the ***vstart.sh*** script located in ***$NETKIT_HOME/bin/***.

ii. An instantiation of Xterm terminal emulator which exists in ***$NETKIT_HOME/kernel/netkit-kernel.***

iii. A ***hypervisor process*** that controls the available netkit-ng modules and is located in ***$NETKIT_HOME/kernel/modules.***

iv. ***Three sub-processes*** that are responsible for the execution of netkit-ng modules such as the creation of a virtual ***network hub*** and the I/O in the allocated ***.disk file.***

Furthermore, two additional operating system processes are used for each existing netkitng virtual network hub, as follows:

i. A ***bin/sh*** script responsible to extract system variables such as time and current date which are then passed as attributes to all virtual machines connected to the designated network hub.

ii. A ***uml_switch*** located in ***$NETKIT_HOME/bin/uml_switch.***

For the purposes of this project, the resources allocated to individual operating system processes and are related with the same netkit-ng virtual machine, are added and the final result is presented. The statistics are obtained after the network boot up and while the machines remain idle.

4.5 Version 1: Local Area Network

This subsection marks the beginning of the actual network development phase of the project and aims upon completion to produce a ***solid network foundation***, on which the rest of the network infrastructure will be built. To better understand the development environment and gain experience in Netkit-ng, the researcher decided to initially create a small LAN (Local Area Network). The ***LAN*** consisted of three virtual machines (***pc1***, ***pc2***, ***gw1***) connected to the same ***/24*** network, named ***A***. Each machine was assigned a static ***IP address*** and a network interface. The additional network details of the devised LAN are illustrated in Figure 4.1

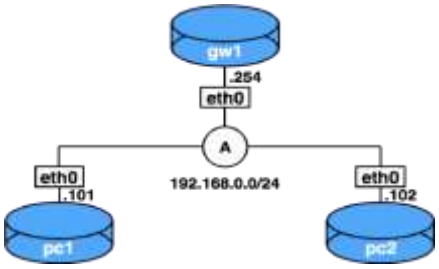


Figure 4.1 Version 1: Local Area Network

4.5.1 Development

The first step in deploying a Netkit-ng virtual network was the creation of lab directory that contained the minimum required sub-directories and files (lines 1-5). As previously discussed, each virtual machine requires a unique directory and a configuration file. Netkit-ng extracts the names of existing subdirectory and allocates them to individual virtual machines. It is vital for each ***.startup*** file to maintain an ***identical name*** as the related ***subdirectory***. This allows Netkit-ng to match each virtual machine with an existing configuration file.

```
1: cd ~/Desktop/
2: mkdir net1
3: cd net1
4: mkdir pc1 pc2 gw1
5: touch pc1.startup pc2.startup gw1.startup lab.conf
```

It is possible for a virtual machine to temporarily boot up without the existence of a subdirectory and configuration file through the ***vstart*** command. However, the machine uses the default Netkit-ng configuration which will not be sufficient as the development process progresses. Furthermore, this project selects to utilise the functionalities provided through the ***L-tools***, being the most recent. The next step after the creation of the necessary Netkit-ng subdirectories and files was to configure each virtual machine.

There are two major text editors available via the GNU terminal, ***nano*** (Allegretta et al., 2017) and ***vi*** (Joy, 2016), with the former being selected due to the researcher's prior experience. The remaining of this subsection discusses the imported configuration settings in each file and provides the logic and reasoning behind them.

- **lab.conf**

```
1: LAB_DESCRIPTION="LAN"
2: LAB_VERSION=1
3: LAB_AUTHOR="Abdullahi Modibbo"
4: LAB_EMAIL="modibbo04@gmail.com"
5: gw1[0]=A
6: pc1[0]=A
7: pc2[0]=A
```

This text file contains a series of configuration settings which enable Netkit-ng to correlate and connect individual virtual machine into distinct virtual networks. The name of each machine, originated from the name of existing subdirectory, and was used as a ***unique identifier***. Furthermore, by-design, Netkit-ng allows each virtual machine to deploy up to ***40 different network interfaces***. The use of opening and closing brackets appended next to the machine's name and enclosing a numeric value ***(e.g. [0])***, indicates that this virtual machine ***(e.g. pc1)*** is connected, using the enclosed network interface number, to a specified network ***(e.g. A)*** (lines 5-7).

Optional Netkit-ng variables (lines 1-4) can also be substituted in the lab.conf file. These string variables are presented by default within each Netkit-ng virtual machine, after the Netkit-ng configuration script completion. Besides the virtual configuration, additional network functionalities and machine capabilities, to be used at a later stage, can be integrated through the lab.conf file.

- **pc1.startup**

```
1: ifconfig eth0 192.168.0.101/24
2: route add default gw 192.168.0.254
```

Each .startup file was executed within the relevant Netkit-ng virtual machine as part of the boot up script. Besides a series of customised Netkit-ng commands, normal shell scripting commands were also supported, as long as the packages used, existed within the virtual machine's filesystem.

Using the ***ifconfig*** command, a static IP address was assigned to the ***eth0*** network interface (line 1). As illustrated in the ***lab.conf*** file, ***pc1*** used the ***0 interface*** to connect to a network labelled as ***A***, which was the reason of selecting the ***eth0*** interface as the network interface to be used in pc1.

In addition, the allocation of a ***class C network*** (192.168.0.0) to a LAN is a universally acceptable practice, as discussed in the ***RFC-1918*** (IETF, 1996). Also, the use of a ***/24*** IP range network (netmask) can provide additional information regarding the subnet as listed in Table 4.2

Table 4.2 Class C network information

	DECIMAL	BINARY
NETMASK	255.255.255.0 = 24	11111111.11111111.11111111.00000000
WILDCARD	0.0.0.255	00000000.00000000.00000000.11111111
NETWORK	192.168.0.0/24 = C	11000000.10101000.00000000.00000000
BROADCAST	192.168.0.255	11000000.10101000.00000000.11111111
HOSTMIN	192.168.0.1	11000000.10101000.00000000.00000001
HOSTMAX	192.168.0.254	11000000.10101000.00000000.11111110
HOSTS/NET	254	-

Although the above variables can explicitly be specified in each .startup file, the use of ***/24*** allows Netkit-ng to interpret and automatically assign these values to the network interface of ***pc1*** and by extension to the ***A subnet***.

Finally, the use of the ***route add*** command with the parameters ***default gw*** (gateway) and an IP address specifies the default network routing path of the virtual machine (line 2). In simpler words, pc1 will transmit all generated traffic to the virtual machine with IP address ***192.168.0.254*** provided that the destinations address is not located within the same ***A subnet***.

- **pc2.startup**

```
1: ifconfig eth0 192.168.0.102/24
2: route add default gw 192.168.0.254
```

An identical configuration file and set of commands as ***pc1***, were used for ***pc2*** with the only difference being the assigned IP address (192.168.0.102).

- **gw1.startup**

```
1: ifconfig eth0 192.168.0.254/24
```

As the name implies, this virtual machine was set as the ***default gateway*** for the ***192.168.0.0/24 network***. Similarly, a static IP address was allocated using the ***ifconfig*** command. Figure 4.2 illustrates the default routing path for subnet A.

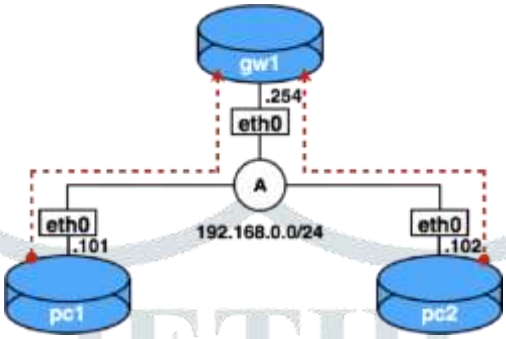


Figure 4.2 Default routing path for subnet A

4.5.2 Data collection

This small virtual LAN was the first iteration in a series of virtual networks. Therefore, there were no means of comparison with the measurements and statistics obtained during the network evaluation. However, the obtained data can be used as a benchmark for the network version that will follow. The data presented below were acquired based on the previously defined network evaluation criteria (section 4.4).

Table 4.3 Version 1: LAN – routing paths

NO	VIRTUAL MACHINE	I P	NETWORK INTERFACE	DEFAULT GATEWAY
1.	gw1	192.168.0.254	eth0	-
2.	pc1	192.168.0.101	eth0	192.168.0.254
3.	pc2	192.168.0.102	eth0	192.168.0.254

Table 4.4 Version 1: LAN – running services

NO	VIRTUAL MACHINE	RUNNING SERVICES
1.	All virtual machines	ebtables, lxc, resolvconf, rsyslog

Table 4.5 Version 1: LAN – portability

NO	DIRECTORIES / FILES	STORAGE ALLOCATION
1.	Lab directory (7 items)	12.8 KB
2.	netkit-ng (827 items)	2.1 GB = 2 100 000 KB
TOTAL	-	2 100 013 KB

Table 4.6 Version 1: LAN – speed

	VM	VM BOOT UP TIME (min: sec. millisec)	NETWORK BOOT TIME (millisec. millisec)
1.	gw1	00:14.44	00:14.44
2.	pc1	00:12.44	00:26.88
3.	pc2	00:12.24	00:39.32

Table 4.7 Version 1: LAN – hardware limitations

	VIRTUAL MACHINE / SYSTEM	MEMORY ALLOC. ON RUN TIME	STORAGE ALLOC. ON RUN TIME	CPU USAGE ON RUN TIME
1.	gw1	223616 KiB ≈ 28.6 MB	684 KB	0.6% – 1.3%
2.	pc1	223616 KiB ≈ 28.6 MB	680 KB	0.6% – 1.3%
3.	pc2	223616 KiB ≈ 28.6 MB	680 KB	0.6% – 1.3%
4.	Network Hop A	7552 KiB ≈ 7.7 MB	-	-
5.	UML Switch A	2172 KiB ≈ 2.2 MB	-	-
TOTAL	-	95.7 MB	2044 KB ≈ 2 MB	1.8% – 3.9%

No open ports were obtained at any of the virtual machines using the ***netstat –tulpn*** command. The results were partially verified using the ***nmap 192.168.0.0/24*** command to scan the entire network. However, by default nmap scans the first ***1000 ports***. The command ***nmap –p- 192.168.0.0/24*** was used to scan all

65535 ports in each virtual machine, however the command was unexpectedly terminated before completion, since there was not enough virtual memory allocated to perform the task. The error message provided was ***Out of memory: Kill process 1509 (nmap) score 753 or sacrifice child***.

4.6 Version 2: Wide Area Network

Although the creation of a virtual LAN infrastructure provided a basic level of experience in Netkit-ng, the knowledge gained from it, was considered insufficient for the purpose of this project. Aiming at acquiring a deeper understanding and extending the already gained knowledge, the researcher deemed necessary to increase the network complexity by expanding the constructed LAN into a WAN (Wide Area Network).

This virtual WAN infrastructure consisted of two local area networks ***(A, B)*** connected via a central node aiming to represent the ***internet***. Three additional virtual machines were connected to the central node. The virtual network topology was chosen based on the end goals of the project, which is to create an isolated Tor experimental network.

Consequently, two of the virtual machines connected to the ***internet*** were named as ***dirauth1*** and ***dirauth2*** aiming to be used as ***Tor Directory Authorities*** in one of the forthcoming network versions. The third virtual machines connected to the internet, named ***dns***, aimed to serve as a Domain Name System for the entire infrastructure. However, the objective of this network version was to produce the WAN infrastructure and not to configure the Tor Directory Authorities or the DNS. Figure 4.3 illustrates the WAN topology and provides additional network information.

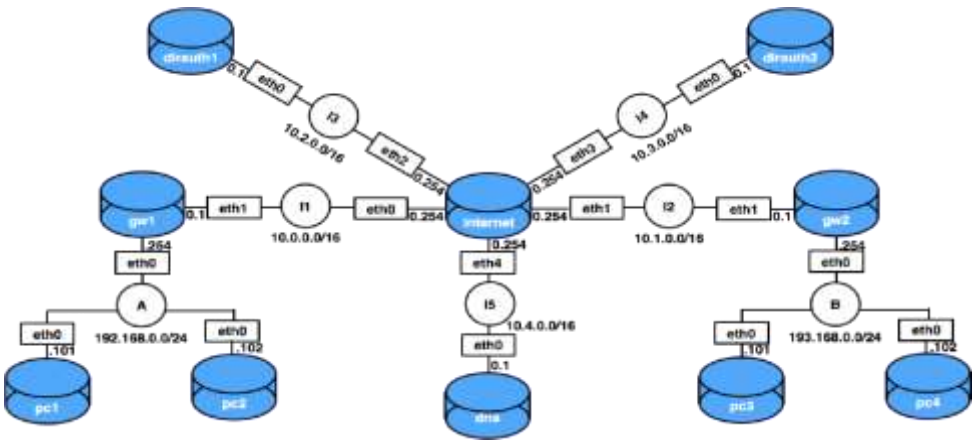


Figure 4.3 Version 2: WAN

4.6.1 Development

The development of the second network version relied heavily on the techniques and processes used during the first network version. Consequently, this subsection is focusing on the steps that are different from the LAN development (section 4.5.1). A copy of the LAN lab directory, renamed to ***net2***, was used as the WAN lab directory (line 1). Seven additional netkit-ng virtual machines ***(directories & .startup files)*** were created in the net2 directory (line 2-4), as illustrated in Figure 4.3.

```
1: cp -R ~/Desktop/net1/ ~/Desktop/net2
2: cd ~/Desktop/net2/
3: mkdir internet gw2 pc3 pc4 dirauth1 dirauth2 dns
```

```
4: touch internet.startup gw2.startup pc3.startup pc4.startup
   dirauth1.startup dirauth2.startup dns.startup
```

The remaining of this subsection discusses the ***lab.conf***, ***gw1.startup*** and ***internet.startup*** configuration files and provides the reasoning behind each choice. All configuration files are available in Appendix B.

- **lab.conf**

```
……
6: gw1[0]=A
7: pc1[0]=A
8: pc2[0]=A
9: gw2[0]=B
10: pc3[0]=B
11: pc4[0]=B
12: gw1[1]=I1
13: internet[0]=I1
14: gw2[1]=I2
15: internet[1]=I2
16: dirauth1[0]=I3
17: internet[2]=I3
18: dirauth2[0]=I4
19: internet[3]=I4
20: dns[0]=I5
21: internet[4]=I5
```

Through the ***lab.conf*** configuration file, all active network interfaces ***(e.g. eth0)*** were allocated to individual subnets. As can be observed, ***gw1*** (line 12) and ***gw2*** (line 14) were assigned a second network

interface which enabled virtual machines located in ***LAN A (pc1, pc2)*** and ***LAN B (pc3, pc4)*** to connect to the central node ***(internet)*** and thus with the entire virtual network infrastructure.

Also, all existing subnets were connected to the central virtual machine ***(internet)***, thus creating a virtual network that represents an actual network topology of an authentic WAN infrastructure (line 13-21).

- **gw1.startup**

```
1: ifconfig eth0 192.168.0.254/24
2: ifconfig eth1 10.0.0.1/16
3: route add default gw 10.0.0.254
```

In addition to the previously configured ***eth0*** network interface (section 4.5.1) which acts as a default gateway for ***subnet A*** (line 1), a separate ***eth1*** network interface was enabled and connected to the ***subnet I1*** (line 2). Also, the ***IP 10.0.0.254*** which is the ***eth0*** network interface of the ***internet*** virtual machine was set as the default gateway for ***gw1*** (line 3), resulting in the formation of a continuous routing path from all virtual machines located in subnet A to the internet. A similar configuration was used in ***gw2*** aiming at achieving the same network configuration and routing paths for ***subnet B***.

- **internet.startup**

```
1: ifconfig eth0 10.0.0.254/16
2: ifconfig eth1 10.1.0.254/16
3: ifconfig eth2 10.2.0.254/16
4: ifconfig eth3 10.3.0.254/16
5: ifconfig eth4 10.4.0.254/16
6:
7: route add -net 192.168.0.0/24 gw 10.0.0.1
8: route add -net 193.168.0.0/24 gw 10.1.0.1
9: route add -net 10.2.0.0/16 gw 10.2.0.1
10: route add -net 10.3.0.0/16 gw 10.3.0.1
11: route add -net 10.4.0.0/16 gw 10.4.0.230
```

The ***internet*** virtual machine act as a central node that enables connectivity between all virtual subnets, which is a realistic representation of what actually exist in the real world. A number of virtual machines could be used to represent the internet, with the purpose of segregating the network into smaller subnetworks that would be able to represent different geographic locations. However, for the purposes of

this project, the researcher decided for the time being, to represent the internet as a single virtual machine, as it is easier to observe the interactions between distinct virtual network entities by capturing traffic at a single virtual machine.

Having in mind the above, ***five network interfaces*** were allocated to the internet virtual machine (line 1-5) and were connected to individual networks. In addition, all virtual subnets were configured to use the respective internet's network interfaces as their default subnet gateway. As a result, the internet virtual machine required a way of knowing in which subnet to forward any inbound network traffic. This was achieved by adding the IP ranges of the non-directly accessible subnets and then specifying the directly connected network gateway that will be used to forward traffic to the desired subnet.

For example, the ***internet*** virtual machine did not maintain a direct connection with the ***subnet A***, but was connected to it through the ***gw1*** virtual machine. As a result, the ***internet*** virtual machine was not able to identify where the ***subnet A*** was located in order to forward network traffic with ***destination IP*** address in the ***192.168.0.0/24*** network range. Using the command ***route add –net 192.160.0.0/24 gw 10.0.0.1***, the internet virtual machine was able to identify that, in order for inbound traffic to reach ***subnet A***, the traffic had to be forwarded to the ***IP address 10.0.0.1***, which was the ***eth1*** network interface of **gw1** (line 7). A similar configuration was used for ***subnet B*** (line 8).

Since the remaining network subnets ***(I3, I4, I5)*** maintained a direct connection with the ***internet*** virtual machine, the addition of default gateways in these designated subnets was consider optional and, based on the current network topology, did not provide significant functionality. However, for consistency reasons and since the network infrastructure was still under development, the researcher selected to also define default gateways for the remaining subnets (line 9-11).

4.6.2 Data Collection

The data presented below was acquired based on the network evaluation criteria (section 4.4) and was used to compare and evaluate the two existing virtual network versions.

Table 4.8 Version 2: WAN – routing paths

NO	IP	VIRTUAL MACHINE	NETWORK INTERFACE	DEFAULT GATEWAY
1.	gw1	192.168.0.254 10.0.0.1	eth0 eth1	- 10.0.0.254
2.	pc1	192.168.0.101	eth0	192.168.0.254
3.	pc2	192.168.0.102	eth0	192.168.0.254
4.	gw2	193.168.0.254 10.1.0.1	eth0 eth1	- 10.1.0.254
5.	pc3	192.168.0.101	eth0	193.168.0.254
6.	pc4	192.168.0.102	eth0	193.1 68.0.254
7.	dirauth1	10.2.0.1	eth0	10.2.0.254
8.	dirauth2	10.3.0.1	eth0	10.3.0.254
9.	dns	10.4.0.230	eth0	10.4.0.254
10.	internet	10.0.0.254 10.1.0.254 10.2.0.254 10.3.0.254 10.4.0.254	eth0 eth1 eth2 eth3 eth4	10.0.0.1 10.1.0.1 10.2.0.1 10.3.0.1 10.4.0 .230

Table 4.9 Version 2: WAN – running services

NO	VIRTUAL MACHINE	RUNNING SERVICES
1.	All virtual machines	ebtables, lxc, resolvconf, rsyslog

Table 4.10 Version 2: WAN – portability

	DIRECTORIES / FILES	STORAGE ALLOCATION
1.	Lab directory (7 items)	88 KB
2.	netkit-ng (827 items)	2.1 GB = 2 100 000 KB
TOTAL	-	2 100 088 KB

Table 4.11 Version 2: WAN – speed

	VM	VM BOOT UP TIME (min: sec. millisec)	NETWORK BOOT TIME (min: sec. millisec)
1.	dirauth1	00:14.03	00:14.03
2.	dirauth2	00:12.84	00:26.87
3.	dns	00:12.90	00:39.77
4.	gw1	00:14.13	00:53.90
5.	gw2	00:13.95	01:07.85
6.	internet	00:12.61	01:20.46
7.	pc1	00:12.16	01:32.62
8.	pc2	00:11.94	01:44.56
9.	pc3	00:11.90	01: 56.46
10.	pc4	00:12.34	02:08.80

Table 4.12 Version 2: WAN – hardware limitations

	VIRTUAL MACHINE / SYSTEM	MEMORY ALLOC.NO RUN TIME	STORAGE ALLOC.ON RUN TIME	CPU USAGE ON RUN TIME
1.	dirauth1	223616 KiB ≈ 28.6 MB	684 KB	0.7% – 2%
2.	dirauth2	223616 KiB ≈ 28.6 MB	684 KB	0.7% – 2%
3.	dns	223616 KiB ≈ 28.6 MB	684 KB	0.7% – 2%
4.	gw1	223616 KiB ≈ 28.6 MB	684 KB	0.7% – 2%
5.	gw2	223616 KiB ≈ 28.6 MB	680 KB	0.7% – 2%
6.	internet	223616 KiB ≈ 28.6 MB	700 KB	0.7% – 2%
7.	pc1	223616 KiB ≈ 28.6 MB	684 KB	0.7% – 2%
8.	pc2	223616 KiB ≈ 28.6 MB	676 KB	0.7% – 2%
9.	pc3	223616 KiB ≈ 28.6 MB	684 KB	0.7% – 2%
10.	pc4	223616 KiB ≈ 28.6 MB	680 KB	0.7% – 2%
11.	7 Network Hops	52864 KiB ≈ 53.9 MB	-	0%
12.	7 UML Swit ches	15204 KiB ≈ 15.4 MB	-	0%
TOTAL	-	355.2 MB	6840 KB ≈ 6.8 MB	6% – 13%

No open ports were obtained at any of the virtual machines using the ***netstat –tulpn*** command or ***nmap*** tool.

4.6.3 Network Evaluation

The goal of this subsection was to compare and critically evaluate the two produced virtual networks based on the measurements and statistics obtained during the data collection process and also discuss unexpected

system behaviors that have been observed during the development phase. Particular attention was given in distinguishing the diversities and similarities between each version and evaluating their impact on the overall virtual infrastructure and host operating system. The evaluation took into account that the ***WAN*** virtual network differentiates significantly in ***size*** and ***complexity***, as it consisted of ten virtual machines and seven subnets, compared to the ***LAN*** virtual network that enclosed three virtual machines and a single subnet.

As expected, the running services in all virtual machines were identical since the same core, filesystem and kernel were used and no additional services were enabled through any of the .startup files. At this point, it is worth mentioning that besides the four running services illustrated in Table 4.4 and Table 4.9, an additional ***33 services*** were marked with ***[?]***, meaning that these services had been executed as part of the boot up process and their status remained unknown since they were not managed by the session user, even though it was root. According to the ***Netkit-ng documentation*** (Cartigny, 2014) and a 2016 paper which discusses the ***emulating capabilities of netkit*** (Pizzonia and Rimondini, 2016), it can be assumed that the above mentioned services are part of the ***modified Debian*** operating system used by netkit-ng. A full list of additional running services is provided in Appendix C.

Since none of the above five running services required the use of a network port and additional ports had not been manually opened, all virtual machines in both networks did not maintain open ports. These expectations were verified through the use of ***netstat*** and ***nmap***. The ***error message*** provided by ***nmap*** regarding the insufficient amount of memory to execute a network scan was taken into consideration and future network versions aimed also to address this issue.

In addition, routing path information and default gateway configurations obtained during the data collection and presented in Table 4.3 and Table 4.8 match the expected network behaviour and do not deviate at any point.

The most interesting information obtained during the ***LAN*** and ***WAN*** data collection process of both networks, concerned the virtual network portability and hardware limitations. Although the two produced networks differentiated significantly in context, the storage required for both netkit-ng lab directories, is ***extremely low***. Also, the size of the netkit-ng filesystem, kernel and core remained exactly the same, since

no changes had been made. Consequently, both virtual networks consumed approximately ***2.1 GB***, which is an acceptable size that allows fast network sharing among peers. The acceptable storage size of both networks, combined with the low storage allocation ***(676 KB – 700KB)*** for each virtual machine during both network run times, verified the light-weighted nature of the netkit-ng emulation software regarding its storage consumption.

Nevertheless, unexpected results were obtained regarding the CPU consumption and memory allocation. Specifically, the CPU consumption for the LAN ranged between ***1.8% and 3.9%***, in comparison with the WAN that required ***6% to 13%***. As can be observed, the WAN consumed approximately three times more CPU than the LAN, which can be characterized as a significant increase compared to the relatively small size of both networks.

However, the most worrying results pertained to the total memory allocation. During run time, the LAN consumed ***95.7 MB of RAM,*** in comparison with the ***352.2 MB of RAM*** consumed by the WAN, (approximately three times more). Nonetheless, the host operating system was allocated ***4GB of RAM***, which is substantially more compared with the ***352.2 MB*** used by Netkit-ng. Based on the measurements and statistics illustrated in Table 4.7 and Table 4.13, each virtual machine required approximately ***29MB*** regardless of its network topology and active network interfaces. Furthermore, each deployed network hub ***(subnet)*** required an additional amount of approximately ***10 MB***, which is considered not very substantial, but when all network hubs were substituted the overall amount increased significantly.

Finally, the LAN required a total of ***39.32 seconds*** to boot up, compared with the ***2 minutes and 8.8 seconds*** needed by the WAN. Although the two time-frames differentiated significantly and the first impression was that the overall WAN performance was worst, this was not the case. Specifically, the ***average*** boot up time for the LAN virtual machines was ***13.04 seconds***, which was more than the ***12.88 seconds***, average boot up time of the virtual machines in the WAN. Similarly, the ***standard deviation*** ***(***$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$ where ;< is one sample value, $\bar{x}$ is the sample mean and ***N*** is the sample size***)*** of the LAN was ***1.22*** compared to the ***0.87*** of the WAN, which was interpreted as positive for the WAN. Based on the obtained results of Table 9 and Table 14 and the calculated average and standard deviation, it can be assumed that a greater number of virtual machines required less individual time to boot up.

However, this assumption cannot be verified with the existing data but will be under consideration in future evaluation sections.

4.7 Version 3: Wide Area Network with DNS

The two previous development phases have substantially increased the researcher's knowledge and experience in the Netkit-ng emulator. Therefore, from this subsection onwards, the focus shifts to the actual network functionality and capabilities, rather than the virtualization aspects of the system. However, in order to preserve the ***reliability*** and ***repeatability*** of the project, and provide a ***means of comparison*** with the previously produced virtual networks, the development process preserves a similar approach.

A number of decisions were made based on the end goal of the project, which is the development of an isolated Tor experimental network. The second version of the network contained two virtual machines which in future network versions aim to represent Tor ***directory authorities***. However, as discussed in Chapter 2, a fully functional Tor network requires the existence and use of additional network entities such as ***Entry***, ***Middle*** and ***Exit*** relays.

Consequently, the third virtual network version integrated ***thirty additional virtual machines***, labelled as ***relay1 to relay30*** in a sequential manner. In order to comply with the ***Tor circuit construction criteria***, all ***relay*** virtual machines were placed in ***/16 different subnets*** and were connected to the ***internet*** virtual machine. The numbers of relay virtual machines were selected by having in mind both the available ***hardware resource*** allocated to the host operating system and also the provision of substantial ***network complexity*** for the ***creation of Tor circuits***. Another step aiming at representing an actual WAN infrastructure was the configuration of a ***Domain Name System*** in the ***dns*** virtual machine that will be used by the entire virtual network infrastructure. Figure 4.4 illustrates the network topology of the third virtual network version and provides additional network information such as subnet names, IP ranges and active network interfaces.

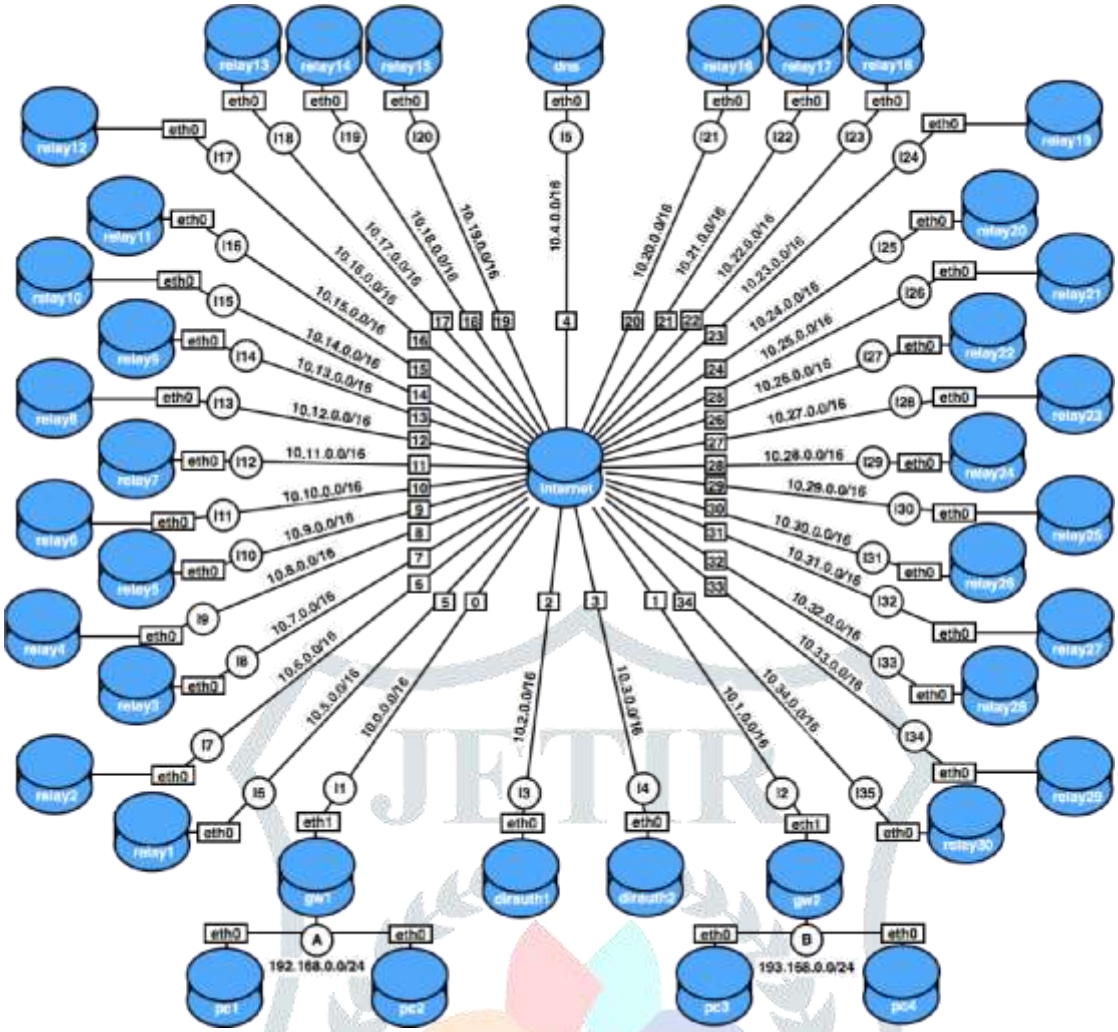


Figure 4.4. Version 3: WAN with DNS

Besides the addition of ***thirty virtual machines*** and the implementation of a ***DNS***, the third network version utilized additional Netkit-ng functionalities through the ***shared.startup*** and ***lab.dep*** text files.

As discussed in section 4.3, each Netkit-ng virtual machine requires an associated ***.startup*** file that is executed during boot up time. Although it is mandatory for these files to exist, Netkit-ng provides another way of configuring all virtual machines through the ***shared.startup*** text file. The content of this file is executed with a ***level 4 priority*** in comparison with the ***level 5 priority*** of each individual .startup file, meaning that if the ***shared.startup*** file exists in the lab directory, it will be executed before the machine dependent .startup file.

Another aspect of Netkit-ng that has not been touched until now is the boot up sequence of all virtual machines when using the command ***lstart***. By design, Netkit-ng uses the name of each virtual machine to create an ***ascending***, ***alphabetically sorted*** list that is used as the boot up sequence. However, if the lab.dep file exists in the lab directory, then a parallel boot up sequence is enabled, meaning all virtual machines are

booted simultaneously. Additional network dependencies can be specified in this file. For instance, initiate machines **B** and **C** in parallel ***only* if** machine **A** is already **running**.

4.7.1 Development

The second virtual network version (WAN) was used as the basis for the development of the third network version, which was the WAN with a DNS. Similar steps as the first and second network version were used to create the ***relay1 to relay30*** virtual machines. Also, the ***shared.startup*** and ***lab.deb*** text files were added to the lab directory aiming at providing additional network functionality and improving future network configuration procedures.

According to the Netkit-ng documentation (Cartigny, 2014), the main purpose of shared.startup file is to provide Netkit-ng users the ability to pass common configuration settings to all virtual machines. However, for the purposes of this project, this file was used in a rather unconventional way. A variable ***machine_name*** was declared inside the ***shared.startup*** file and was assigned the output of the ***uname –n*** (MacKenzie, 2010) command, which returns a string that contains the name of a Linux system (line 1). Then, a series of nested if statements compared the ***machine_name*** variable with hardcoded strings (line 2, 14). Although the ***shared.startup*** script was executed by default in all Netkit-ng virtual machines, every script iteration assigned a different value to the ***machine_name*** variable, resulting in the execution of different actions in each virtual machine.

```
shared. startup
1:   machine_name=$(echo $(uname -n))
2:   if [ "$machine_name" = "dns" ]; then
……

14:  elif [ "$machine_name" = "internet" ]; then

……

200: fi
```

The unusual use of the ***shared.startup*** significantly improves the developing experience while working with Netkit-ng, since a ***single text file*** can contain an entire virtual network configuration making it ***easier*** and ***faster*** for the user to edit changes or add commands.

Another major network modification was the implementation of network dependency rules specified in the ***lab.dep*** text file. All virtual machines were divided into four main categories: (i) ***clients***, (ii) ***gateways***, (iii) ***servers*** and (iv) ***others***, which were then used to select dependency rules. The format used to specify a dependency in the ***lab.dep*** file was ***"A: B",*** which is interpreted as ***start A only if B is running***. ***Dns*** was selected as the first virtual machines to boot up, followed by the ***internet*** (line 4). Both virtual machines provide significant functionality to the network infrastructure with the former used to match domain names with IP addresses and the latter being the central network node. The boot up sequence continued with ***all directly connected nodes*** to the internet, which was basically all relay, Directory Authority and gateway machines (line3). ***Client*** virtual machines were the last to boot up (line 1-2). The complete shared.startup and lab.conf files appear in Appendix D

```
lab.dep 1: pc1 pc2: gw1

2: pc3 pc4: gw2
```

```
3: gw1 gw2 dirauth1 dirauth2 relay1 relay2 relay3 relay4 relay5
   relay6 relay7 relay8 relay9 relay10 relay11 relay12 relay13
   relay14 relay15 relay16 relay17 relay18 relay19 relay20 relay21
   relay22 relay23 relay24 relay25 relay26 relay27 relay28 relay29
   relay30: internet
```

```
4: internet: dns
```

4.7.2 DNS Configuration

One of the most common ways of managing a network infrastructure includes the ability to look up ***network interfaces*** and ***IP addresses*** by name. The use of ***fully qualified domain names*** (FQDNs) eases the configurations of running network services and increases the overall ***robustness*** and ***maintainability***, particularly in the case of private network infrastructures. This can be achieved through the use of a ***DNS server***.

By design, Netkit-ng comes with the ***dnsmasq*** tool (Kelley, 2017), which is a domain name software that allows easy deployment of a ***DNS*** and ***DHCP*** (Dynamic Host Configuration Protocol) server. However, having in mind that Tor network entities are correlated with IP addresses that are assigned individual cryptographic fingerprints, the researcher deemed necessary to use ***static IP addresses*** instead of dynamic. Consequently, ***dnsmasq*** was not considered suitable for the purposes of this project. Therefore, a custom DNS server was developed using ***bind9*** (Consortium, 2016), which is an open source software that enables

the sharing and resolving of DNS network queries. The remaining of this subsection briefly explains the necessary steps and procedures to deploy a custom DNS server on Netkit-ng using ***bind9***.

This particular type of DNS configuration requires the installation of two Linux packages: (i) ***bind9*** and (ii) ***bind9utils***. Both packages already existed in the Netkit-ng filesystem. Under normal circumstances, a ***bind9 DNS configuration*** involves editing files located in the ***/etc/bind/*** directory. However, any system modifications that occur in Netkit-ng virtual machines during runtime are erased during shutdown with the following two exceptions:

- If the command ***lhalt*** is used to terminate a running Netkit-ng network, then all ***.disk files persist*** on shutdown along with any system modification that happened. However, this option significantly increases the consumption of disk storage on the host machine and makes the overall management of the virtual infrastructure extremely difficult since each machine is able to maintain a different system configuration.
- The second action that is able to perform changes to the entire network infrastructure is the ***modification of the Netkit-ng filesystem***. Although this option is significantly better than the first, it is not appropriate in the current situation since any changes will affect all virtual machines and not just the ***dns***, as desired.

A third way that allows users to selectively modify files located in a Netkit-ng virtual machine is through the associated sub-directory located in the lab directory. Any substituted directories or file that exist in the associated sub-directory are copied to the ***root directory (/)*** of the associated virtual machine during boot up. If the files already exist in the virtual machine then they are ***overwritten***. This Netkit-ng functionality was used to overwrite the ***named.conf.local*** (defines all DNS zones) and the ***named.conf.options*** (additional DNS functionality) text files located in the ***/etc/bind/*** directory. Four additional text files that contained ***DNS zone*** and reverse ***DNS lookup*** information were added to the ***/etc/bind*** directory. The complete DNS configuration files are available in Appendix E

The final step of configuring a private DNS server in a network infrastructure is to inform all existing network host that the specific virtual machine is used as a DNS server and all ***DNS queries*** should be forward to this specific IP address. This can be achieved by adding the ***IP address*** of the DNS server and

the ***FQDN*** of the private network to the ***/etc/resolv.conf*** text file and then ***restarting networking services*** in order to ***reload*** this network configuration file. The shared.startup file was used for the modification of /***etc/resolv.conf*** in all virtual machines line (201 – 206).

```
shared.startup
201: echo "Configuring /etc/resolv.conf file"
202: echo "search torlab.test." > /etc/resolv.conf
203: echo "nameserver 10.4.0.230" >> /etc/resolv.conf
204: echo "Restarting networking service"
205: service networking stop
206: service networking start
```

At this point is worth mentioning that initially the command ***service network restart*** was used but, because of a system's warning that ***restart is deprecated because it may not enable again some interfaces***, the command was altered to ***stop | start*** (lines 205-206).

4.7.3 Data Collection

The data presented below was acquired based on the network evaluation criteria (section 4.4) and was used to evaluate the third network version and compare it with the second.

Due to the increased number of virtual machines, the complete data tables are presented in Appendix F, G and H and only a selective number of machines are presented in this subsection.

Table 4.13 Version 3: WAN with DNS – routing paths

NO	VIRTUAL MACHINE	IP	NETWORK INTERFACE	DEFAULT GATEWAY
1.	gw1	192.168.0. 254 10.0.0.1	eth0 eth1	- 10.0.0.254
2.	pc1	192.168.0.101	eth0	192.168.0.254
3.	pc2	192.168.0.102	eth0	192.168.0.254
4	gw2	193.168.0.254 10.1.0.1	eth0 eth1	- 10.1.0.254
5.	pc3	192.168.0.101	eth0	193.168.0.254
6.	pc4	192.168.0.102	eth0	193.168.0.254
7.	dirauth1	10.2.0.1	eth0	10.2.0.254
8.	dirauth2	10.3.0.1	eth0	10.3.0.254
……	……	……	……	……
35.	relay26	10.30.0.1	eth0	10.30.0.254
36.	relay27	10.31.0.1	eth0	10.33.0.254
37,	relay28	10.32.0.1	eth0	10.32.0.254
38.	relay29	10.33.0.1	eth0	10.33.0.254
……	……	……	……	……

Table 4.14 Version 3: WAN with DNS – running services

NO	VIRTUAL MACHINE	RUNNING SERVICES
1.	All virtual machines	ebtables, lxc, resolvconf, rsyslog
2.	dns	bind9, ebtables, lxc, resolvconf, rsyslog

Table 4.15 Version 3: WAN with DNS – portability

	DIRECTORIES / FILES	STORAGE ALLOCATION
1.	Lab directory (7 items)	236 KB
2.	netkit-ng (827 items)	2.1 GB = 2 100 000 KB
TOTAL	-	2 100 236 KB

Table 4.16 Version 3: WAN with DNS – speed

	VM	VM BOOT UP TIME (min: sec. millisec)	NETWORK BOOT TIME (min: sec. millisec)
1.	dns	00:18.05	00:18.05
2.	internet	00:49.50	01:07.55
.....	.....	.....	.....
38.	pc2	00:22.01	
39.	pc3	00:22.01	02:37.97
40.	pc4	00:22.01	

Table 4.17 Version 3: WAN with DNS – hardware limitations

	VIRTUAL MACHINE / SYSTEM	MEMORY ALLOC. ON RUN TIME	STORAGE ALLOC. ON RUN TIME	CPU USAGE ON RUN TIME
1.	dirauth1	264156 KiB ≈ 33.8 MB	700 KB	0.7% – 1.3%
2.	dirauth2	264156 KiB ≈ 33.8 MB	704 KB	0.7% – 1.3%
3.	dns	264156 KiB ≈ 33.8 MB	792 KB	0.7% – 1.3%
4.	gw1	264156 KiB ≈ 33.8 MB	700 KB	0.7% – 1.3%
……	……	……	……	……
TOTAL	-	1718.3 MB ≈ 1.7 GB	28164 KB ≈ 28.1 MB	28% – 52%

Table 4.18 Version 3: WAN with DNS – Open Ports

	VM	OPEN PORT	PROTOCOL	SERVICE
1.	All virtual machines	none	-	-
2.	dns	10.4.0.230: 53	tcp / udp	DNS
		127.0.0.1:953	tcp	rndc
		::1:953	tcp6	rndc
		:::53	udp6	rndc

4.7.4 Network Evaluation

The goal of this subsection was to compare and critically evaluate the three produced virtual networks aiming at identifying network ***improvements*** and ***deficiencies*** which can be supported by the obtained measurements and statistics. Additionally, the main purpose of this evaluation procedure was to determine whether the latest virtual network infrastructure (***WAN with DNS***) contained the necessary network components that would allowed the commencement of the next development phase, which is the integration of Tor.

When comparing the three produced virtual networks the most distinguishable network characteristic is the ***number of virtual machines*** and consequently the overall ***network size*** and ***complexity***. Nonetheless, this

is expected in most development procedure and in particularly when developing a network infrastructure. Considering that the initial network version (***LAN***) consisted of ***three*** virtual machines and the latest network version (***WAN with DNS***) consisted of ***forty*** virtual machines, one can conclude that the virtual network infrastructure has substantially evolved. Another aspect of the network with a significant growth is the ***number of network subnets*** as it is observed in ***Tables 4.3, 4.8 and 4.12***, which is another indication of the substantial network expansion.

In contrast, the ***running service*** in all virtual networks remained the ***same*** with the ***exception*** of the dns virtual machine, which in the latest version was configured to run a ***bind9*** DNS server. Based on the current virtual network topology and having in mind the end goals of the virtual network, at the moment there is no other service which is able to provide substantial network functionality.

The ***portability*** of the three virtual network infrastructures remained stable at approximately ***2.1 GB***, which is what was expected since no modification has taken place in the Netkit-ng file system, core and kernel and the size of the text files added to the lab directory (section 4.7.2) is minor (***40KB***).

In a similar fashion to the previous evaluation subsection (section 4.6.3), the most unexpected results concerned the total memory consumption. The ***WAN*** consumed ***355.2 MB*** of RAM which was approximately three times more than the LAN (***95.7 MB***).

However, the memory consumption of the ***WAN with DNS*** was measured at ***1.7 GB of RAM***, which significantly exceeded any expectations. Specifically, the WAN with DNS consumed approximately ***42.5%*** out of the ***4 GB of available RAM*** of the host operating system. Besides the abnormal memory consumption, the third network version also required a significant amount of computational power that ranges between ***28% to 52%***, compared with the former two network version that required ***1.8% to 3.9%*** and ***6% to 13%*** respectively.

The most significant improvement in the third virtual network was the overall time required to boot up. The network size and complexity of **WAN with DNS** differed substantially ***(40 virtual machines, 37 subnets)*** when compared to the first ***(3 virtual machines, 1 subnet)*** and second ***(10 virtual machines, 7 subnets)*** network versions. Nonetheless, the WAN with DNS required approximately ***2:38 minutes*** to boot up,

which was a major improvement was considering that the second network version, which consisted of only 10 virtual machines, required ***2:09 minutes***. Based on the measurements of ***Table 4.17*** and the simultaneous boot up of some virtual machines, it was assumed that the reason behind this significant improvement was the use of the ***lab.dep*** text file and the activation of the ***parallel boot up*** functionality of Netkit-ng (section 4.7.1).

At this point, it is worth mentioning an ***interesting Netkit-ng behaviour*** that was observed during the boot up of the third network version. In order to ***verify*** the observations, the ***WAN with DNS*** was initiated ***four*** times. As specified in the ***lab.dep*** the first virtual machine to boot up was the dns followed by the internet. Continuing, all virtual machines that were directly connected to the internet were configured to boot up ***in parallel***. According to the Netkit-ng documentation (Rimondini, 2010), the maximum number of virtual machines that can be started simultaneously can be configured by the variable ***MAX_SIMULTANEOUS_VMS*** located in the netkit.conf file and set to five by default. However, it can be assumed that the boot up sequence does not rely only on the ***lab.dep*** dependencies and the ***MAX_SIMULTANIEOUS_VMS*** variable but takes into consideration more variables. The reasoning behind this assumption is based on the fact that every time the ***WAN with DNS*** was started, the same groups of five virtual machines were booted up using an alphabetically based order (***as expected***), with the exception of Group 1 for which ordering was random and different in every iteration. Figure 4.5 illustrates four boot up sequences aiming to provide a clear understanding of what happened.

Figure 4.5 Version 3: WAN with DNS – parallel boot up sequence

4.7.5 DNS Server Evaluation

For the duration of the third virtual network development, significant emphasis was given to the implementation of a DNS server. This subsection aims to present and briefly explain the basic DNS workflow and functionality in the third virtual network through experimentation even though, for the purpose of this project, this is considered common knowledge.

The first way of verifying the correct configuration of a ***bind9*** DNS server is using the command ***named-checkconf /etc/bind/named.conf***. If there is no output, then the configuration is considered correct and the ***bind9*** service can be safely ***restarted*** through the ***/etc/init.d/bind9 stop | start*** command that reloads any changes.

During DNS resolution, ***DNS queries*** are sent over ***UDP*** protocol from ***DNS clients*** to the ***DNS server*** that binds on ***port 53***. Whenever a DNS response exceeds the normal UDP length (***512 bytes***), the message is ***truncated*** and the first segment of the response is send over ***UDP***. The client can then reissue the same DNS query over ***TCP***. This functionality reduced the network traffic significantly and increases the overall network performance by utilising the absence of acknowledgements and speed of ***UDP*** while providing an alternative solution (***TCP***) whenever it is necessary (RFC1123, 1989, RFC1034, 1987, RFC1035, 1987). In general, all DNS queries are sent from ***highnumbered ports*** (above 49152) to the DNS server destination port 53, and all DNS responses are sent from ***port 53*** to the ***same*** high-numbered client port.

In addition, ***bind9*** includes ***rndc*** which is an optional utility used to manage the DNS server ***remotely*** via a command line administration. Since it is running on the ***loopback*** IP address ***port 953***, the service is accessible only to users that are logged into the server and own a ***shared private key*** able to grant them privileges to the service. The key is stored in a separate file owned by root and the file path is ***included*** in both ***/etc/named.conf*** and ***/etc/rndc.conf*** text files. Additional information is available in the Reference Guide: Chapter 12 of Red Hat (RedHat, 2017).

As it is illustrated in ***Table 4.19***, the dns virtual machine allows TCP / UDP on port 53 and 953, which are running the dns and the rndc services respectively. Also, the ***listen-on-v6*** configuration setting specified in the ***named.conf.options*** text file, allows the use of ***IPv6*** addresses running on ***::1:953*** using ***TCP6*** (tcp over IPv6) and ***:::53*** using ***UDP6*** (UDP over IPv6).

In order to confirm that the deployed DNS server was working as expected, a brief experiment was implemented. The ***experiment*** involved the ***capturing*** and ***analysis*** of ***network traffic*** passing through the network while the ***pc1*** virtual machine executed a series of commands that generated DNS queries. The command ***tcpdump*** (Van Jacobson et al., 2017) was used on the ***pc1***, ***gw1***, ***internet*** and ***dns*** (line 1-3) virtual machines (Figure 4.6), being on the expected traffic path, to capture traffic that arrived and leave each network node. Additionally, the results were verified using the ***vdump*** Netkit-ng command (Julien Iguchi-Cartigny et al., 2014), which allowed the capturing of network traffic from the host operating system on running Netkit-ng subnets (line 5-8). The subnets monitored were ***A***, ***I1*** and ***I5*** (Figure 4.6).

```
1: #Executed on pc1, gw1, internet and dns
2: filename =$(echo $(date + "%d-%m-%y_%T"))
3: machine_name=$(echo $(uname -n))
4: tcpdump -i any -w /hostlab/"$machine_name"_"$filename.cap" &
5: #Executed on the host operating system
6: vdump A | wireshark -i - -k &
7: vdump I1 | wireshark -i - -k &
8: vdump I5 | wireshark -i - -k &
```

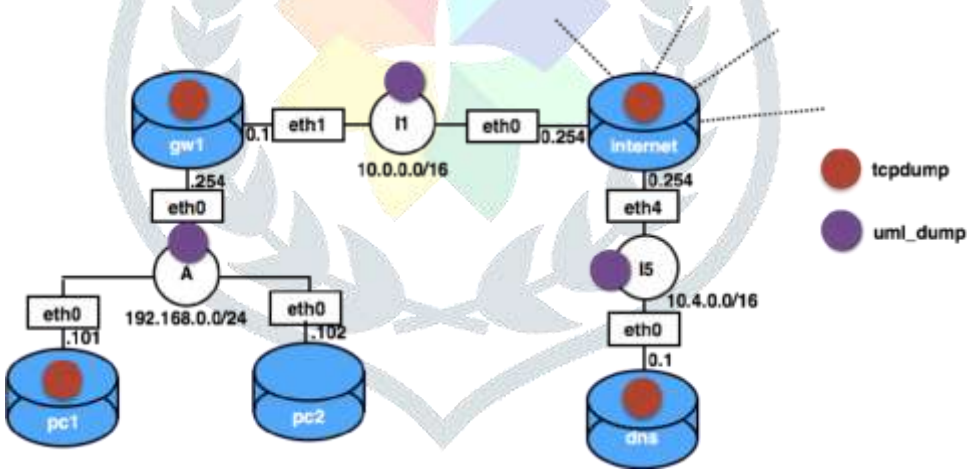


Figure 4.6 DNS server evaluation – tcpdump and vdump deployment

Initially the command ***host 10.4.0.230*** (Peter Tobias et al., 2013) which performs a ***reverse DNS lookup*** on the specified IP address, was executed on the ***pc1*** virtual machine and a DNS query ***(83 bytes)*** was send to the DNS server ***over UDP*** from port ***60988***. The DNS server sent a DNS response (***142 bytes***) over UDP ***port 53***. The final output of the command was ***230.0.4.10.in-addr.arpa domain name pointer dns.torlab.test.*** The results were ***verified*** through a comparison of the seven .pcap files, obtained from the use of ***tcpdump*** and ***vdump***. The only difference between each file was the recording of network-specific

arp (address resolution protocol) requests (RFC826, 1982) which, for the purposes of this project is considered common knowledge. The captured network traffic is available in Appendix I.

A similar experiment was implemented using the ***host internet.torlab.test*** command. However, instead of passing an IP address, a ***FQDN*** was used to generate a ***forward DNS lookup***. As a proof of concept, the internet virtual machine was selected as the target knowing that the ***DNS response*** will exceed the ***512 bytes of a UDP frame***. Consequently, the DNS client ***retransmitted*** the DNS query over a ***TCP*** connection. The results were verified in the same fashion as the first experiment and the captured network traffic is available in AppendixJ.

4.8 Tor Experimental Network Implementation

This section heavily relies on the ***technical skills*** gained through the development of an isolated network infrastructure (3.7) and the ***knowledge gained*** from the literature review (Chapter 2). It discusses the integration of Tor software in the Netkit-ng ***filesystem*** and utilises the third network version (***WAN with DNS***) as a basis for the configuration and deployment of ***Tor clients***, ***relays*** and ***directory authorities***. Following, additional functionality and capabilities are added to the produced ***Tor virtual network***, generating the fifth and ***final*** network version and marking the end of the development phase. The section concludes with the implementation, analysis and evaluation of experiments in the isolated Tor experimental network.

4.9 Netkit-ng Filesystem Update

Unfortunately, the ***Tor*** software, which is an essential part of the second phase of development, is ***not part of*** in the default Netkit-ng filesystem installation. However, Netkit-ng provides a special network interface, named ***TAP,*** that allows a virtual machine to ***bridge*** a network connection to the host operating system and connect to the internet, with the purpose of updating and installing new software packages. Nonetheless, the host operating system does not maintain a connection to the internet. An alternative solution is to download the source code of Tor (TheTorProject, 2017) and ***build*** it using the ***make*** command within each virtual machine. However, the Tor software depends on ***libevent-2.0-5***, ***tor-geoipdb*** and ***torsocks*** software packages that are not included in the source code of Tor.

Having in mind the ***time*** limitations of the project, it was deemed ***easier*** to enable the ***internet connection*** to the host operating system and use the ***TAP interface*** to ***download*** and ***install*** all required packages. Upon completion of the Netkit-ng filesystem update and installation of new packages the internet connection to the host operating system was ***disabled*** again.

Initially, the command "***vstart vm1 --eth0=tap,10.0.0.1,10.0.0.2 --mem=512 --no-cow"*** was used to start a single Netkit-ng virtual machine, named ***vm1***. The virtual machine was allocated ***512 MB*** of RAM and was connected through the ***eth0*** network interface with IP ***10.0.0.2*** to the ***nk_tap_parallel*** network interface created in the host operating system with IP ***10.0.0.1.*** The argument ***--no-cow*** enables ***vm1*** to write to the Netkit-ng filesystem, meaning that any modifications made on run time will persist and any virtual machines that will later use the modified Netkit-ng filesystem will adopt these changes. At this point it is worth mentioning that ***if more than one*** Netkit-ng virtual machine uses the ***--no-cow*** command at the same time, there is a risk of ***filesystem corruption***, which can lead to ***unexpected errors***.

The following actions were performed on the ***vm1*** virtual machine aiming to update the existing software packages and install the Tor software. Initially, the command ***apt-get update*** was executed unsuccessfully with an error message "***… could not resolve host***", which is usually related to an invalid DNS configuration or an inaccessible Linux URL repository. Consequently, the ***8.8.8.8*** and ***8.8.4.4*** IP addresses, which are the Google's public DNS servers, are imported to the ***resolv.conf*** text file (line 1-2). Additionally, the URL address of ***Debian Wheezy repository*** in the UK was used to replace (lines 3-4) the contents of ***source.list*** text file (LinuxConfig.org, 2014).

Continuing, the command ***apt-get update*** was executed for the second time and terminated with an error message ***"there is no key available for the following IDs: 7638D0442B90D010"***. According to the Debian documentation, digital signing of all software packages in Debian is mandatory from 2013 onwards, in order to protect against forgeries (Debian.org, 2013). The command "***apt-get install debian-keying***" downloads all ***OpenPGP*** keys, which belongs to all authorised Debian developers. Alternatively, the command ***apt-key*** can be used to download a specific public key if the user knows the PGP fingerprint (line 5), which in this case is provided in the error message of ***apt-get update***. Finally, the ***vm1*** virtual machine

was ***updated*** and ***upgraded*** and the ***Tor*** package, along with ***libevent-2.0-5***, ***tor-geoipdb*** and ***torsocks*** dependency packages were installed (line 6-7).

```
1: echo "nameserver 8.8.8.8" > /etc/resolv.conf
2: echo "nameserver 8.8.4.4" >> /etc/resolv.conf
```

```
3: echo "deb http://ftp.uk.debian.org/debian/ wheezy main contrib non-
   free" > etc/apt/source.list
4: echo "deb-src http://ftp.uk.debian.org/debian/ wheezy main contrib
   non-free" >> etc/apt/source.list
```

```
5: apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
KEY_ID 7638D0442B90D010
6: apt-get update && apt-get upgrade && apt-get autoremove
7: apt-get install tor
```

At this point, it is worth mentioning, that during the upgrade of the existing Netkit-ng packages, it was observed that a number of available services were ***started automatically***. Examining this unexpected behaviour, the command ***service --status-all*** was executed in order to check which services were active. Surprisingly enough, there were thirteen additional services running. The ***vm1*** was shut down and restarted, however all thirteen services were restarted during the Netkit-ng boot up script. Using the command ***ls /etc/rc*.d***, which lists the services that have been specified to start during boot up, it was discovered that the state of all upgraded services was changed from ***K (kill)*** to ***S (Start)***.

In order to prevent the execution of unnecessary services, the command ***update-rc.d*** was used to remove these services from the ***Upstart*** list (line 11-23), with the only exception being the ***Tor*** service, which was specified to start with ***priority 2 3 6*** (James Hunt and Byrum, 2014).

```
10: update-rc.d tor start default
11: update-rc.d -f pdns-recursor remove
12: update-rc.d -f dnsmasq remove
13: update-rc.d -f apache2 remove
14: update-rc.d -f proftp remove
15: update-rc.d -f openvpn remove
16: update-rc.d -f guagga remove
17: update-rc.d -f ipsec remove
18: update-rc.d -f squid remove
19: update-rc.d -f ssh remove
20: update-rc.d -f bind9 remove
21: update-rc.d -f dbus remove
```

```
22: update-rc.d -f proftpd remove
23: update-rc.d -f openbsd-inetd remove
```

4.10 Version 4: Tor Integration

The fourth network version was built on top of the ***WAN with DNS***, which is the final outcome of the development of an isolated network. As the project progressed, less emphasis was given to the virtual aspect of the network and the focus shifted to the configuration and deployment of services, particularly Tor. Besides the implementation of Tor, the fourth network version aimed at addressing a number of deficiencies that had been identified during previous development phases, such as the amount of allocated ***virtual memory*** (section 4.5.2) in each virtual machine and the ***even number*** of ***Tor Directory Authority*** virtual machines. Figure 4.7 illustrates the network topology of the fourth virtual network version and provides additional network information such as subnet names, IP ranges and active network interfaces.

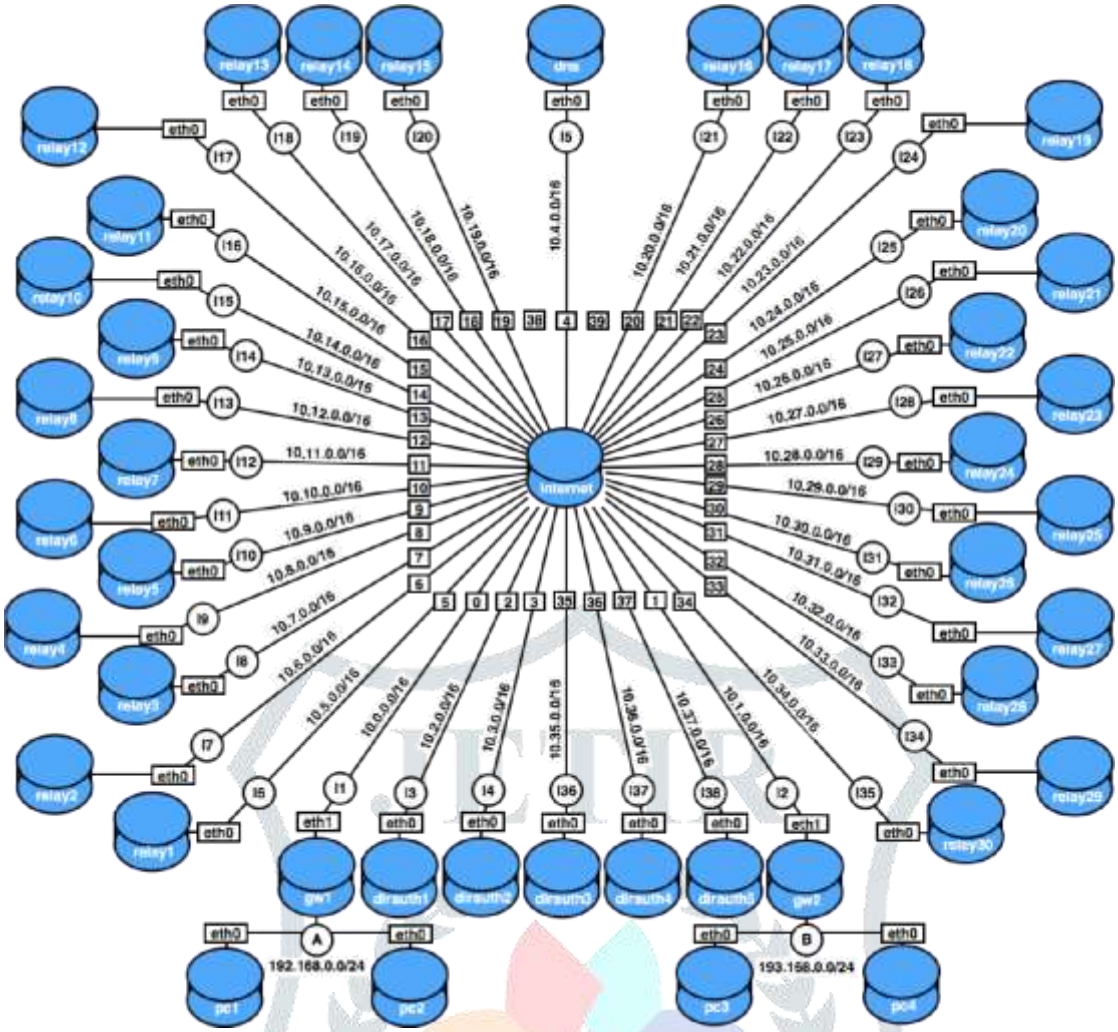


Figure 4.7 DNS server evaluation – tcpdump and vdump deployment

4.10.1 Development

The network topology of the fourth network version does not differ significantly from the third (WAN with DNS). Tor directory authorities participate in a ***voting procedure*** in order to ***agree*** on the basis of ***consensus***. To avoid ***conflicts*** and to ***guarantee*** that the voting will not result in a ***draw***, it is crucial that the number of active directory authorities is ***odd***. Consequently, ***three*** virtual machines (***dirauth3***, ***dirauth4*** and ***dirauth5)*** were added to the network, in order to comply with this rule. The number of added virtual machines was selected at ***random*** with the only criterion being the employment of an odd number of directory authorities.

Also, all virtual machines were allocated 128MB of virtual ***RAM*** (was 32 MB) in order to improve their overall ***performance*** and considering that, upon completion of the fourth network version, the majority of virtual machines will act as ***Tor network entities***. Additional memory (256 MB - 512 MB) was allocated to ***dns***, ***internet*** and all ***Directory Authority*** virtual machines, since they were expected to provide increased

functionality compared to other network entities. The memory allocation was performed via the ***lab.conf*** file is available in Appendix K.

Furthermore, the fourth network version utilised a ***Shared*** folder, which provided similar functionality with the ***shared.startup*** text file. This subdirectory contained a directory named ***Scripts,*** that was made available inside every virtual machine ***(on the root directory ("/"))*** and contained three bash scripts. The following subsection discusses the configuration of Tor ***clients***, ***relays*** and ***directory authorities*** and explains the functionality provided by each bash script located in the ***/Scripts*** directory of each virtual machine.

4.10.2 Configuration of Tor Directory Authorities, Relays and Clients

Every Directory Authority has a long-term ***authority identity key*** (***extracted from authority certificate)*** which is normally stored offline and is used to certify shorter-lived ***authority signing keys*** these are stored online and used by the Directory Authority to sign ***votes*** and ***consensus*** documents. The command ***tor-gencert*** (line 1) can be used to generate the ***certificate*** and the two ***private keys***, if the Tor directory ***protocol*** of the Directory Authority is set to ***v3 (0.2.0 version and later)*** through the ***/etc/tor/torrc file***. A fourth ***private key***, named ***secret onion key*** exists in all ***Tor entities***. ***Tor directory authorities*** and ***relays*** use the ***secret onion key*** to generate ***session keys***, but for ***Tor clients*** this is used as an ***encryption key*** during the establishment of a Tor circuit.

```
1: tor-gencert --create-identity-key -m 12 -a $dir_ip:7000 \
     -i /var/lib/tor/keys/authority_identity_key \
     -s /var/lib/tor/keys/authority_signing_key \
     -c /var/lib/tor/keys/authority_certificate
```

As discussed in chapter 2 and illustrated in ***Table 2.1***, the real Tor network employs the use of nine Tor directory authorities. The ***IP addresses*** and associated ***public keys*** of all nine Tor directory authorities are ***hardcoded*** in the Tor ***source code***. As a result, the five Directory Authority ***virtual*** machines included in the ***fourth*** network version were ***not authorised*** to act as directory authorities. However, a Tor user is able to run its own private Directory Authority by ***downgrading the priority*** of the other nine hardcoded directory authorities. The configuration requires the extraction of two ***cryptographic fingerprints*** which are then imported into the ***/etc/tor/torrc*** text file along with the ***IP address***, ***port*** and ***running protocol*** of the customised Tor Directory Authority. The first fingerprint was extracted automatically from the ***secret id***

key (generated with secret onion key), using the ***tor*** command (line 2) and the second fingerprint was found inside the ***authority certificate***. Table 4.19 summarises all cryptographic keys used by Tor directory authorities and provide a brief overview of their functionality

```
2: tor --list-fingerprint --orport 1 \      --
dirserver "x 127.0.0.1:1 \
ffffffffffffffffffffffffffffffffffffffff" \
     --datadirectory /var/lib/tor/
```

Table 4.19 Cryptographic keys used by Tor directory authorities

NO	KEY NAME	PURPOSE
1.	authority_identity_key	long term key to sign authority certificate
2.	authority_signing_key	medium-term key (12 months) to sign directory information
3.	authority_certificate	document signed by authority identity key to certify authority signing key
4.	secret_id_key	long-term key to sign router descriptor and TLS certificates
5.	secret_onion_key	medium-term key used to establish a circuit and negotiate ephemeral (session) keys
6.	secret_onion_key_ntor	short-term key for handshake
7.	fingerprint	fingerprint of the identity key

The next step of deploying customised Tor Directory Authorities, relays and clients is the modification of the /***etc/tor/torrc*** text file. Although, there are many ***attributes*** that can be customised in the ***torrc text file (TorProject, 2017h)***, for the purposes of this project and based on the desirable behaviour and network functionality, only a ***subset*** (***42 in total***) of the attributes were used. Appendix L outlines the selected Tor attributes and discusses their functionality. The ***torrc*** files of ***dirauth1, relay1 and pc1*** virtual machines is available in Appendix M.

Since the values of some attributes in the ***torrc*** text file depend on the ***network role*** of the respective virtual machine, the Netkit-ng ***Shared*** folder cannot be used to upload the file as in such case, and it would be the same on all virtual machines.

Initially, an attempt was made through the ***shared.startup*** file. The idea was to use the commands ***tor-gencert*** and ***tor*** in two functions declared in the ***shared.startup*** file and ***called*** during the boot up time of each virtual machine. However, there were two problems with this approach. First, each time the network will boot up, every ***Directory Authority*** will generate a new ***authority certificate*** which will require the extraction of its ***cryptographic fingerprint*** as this is necessary to exist in all ***torrc text files***. The fingerprints will first need to be imported into ***the torrc files*** of all ***Directory Authorities*** and also stored separately in the host operating system in order to be made available during the boot up script of the remaining virtual machines. In addition, the ***parallel*** boot up of Directory Authority virtual machines would have to be ***disabled*** as it could ***corrupt*** the text file used to store the fingerprints in the host operating system. Although there are many complications and risks with this approach, it significantly increases the overall network ***automation*** and ***robustness***. However, there was a second problem with this approach which could not be overcome. The ***tor-gencert*** command requires the import of a password in order to generate the ***authority certificate***. Although ***it is not*** a recommended and secure practice, the argument ***--passphrase-fd*** can be used to read the password from a file. However, there is currently a bug (Fossies, 2017) with the argument ***--passphrase-fd***, with a bug-fix on Tor ***version 0.2.7.3*** (Tor version 0.2.4.27 is the latest in Debian wheezy repositories).

Having in mind the above complications, a different approach was adopted. At first, three bash scripts (***dirauth_gen_keys.sh***, ***relay_gen_keys.sh***, ***client_gen_key.sh***) were uploaded to all virtual machines using the ***Shared*** folder and the command ***lstart*** was used to start the fourth network version. The ***dirauth_gen_keys.sh*** script was executed in each ***Directory Authority*** virtual machine and generate all necessary ***Tor keys***, as illustrated in Table 4.20. However, any modification in Netkit-ng virtual machines does not persist during shutdown. Consequently, the same script was used to copy the

/var/lib/tor/fingerprint, ***/var/lib/tor/keys*** and ***/etc/tor/torrc*** to the ***/hostlab/$name***, where

$name is the name of the virtual machine. The script completed execution by ***extracting*** the ***authority certificate fingerprint*** and the ***onion fingerprint*** and storing them in a ***dirauth_fingerprints.txt*** text file, located inside the Netkit-ng lab directory. Similar functionality and actions were performed by the ***relay_gen_keys.sh*** script in each Tor relay virtual machine and by ***client_gen_key.sh*** script in each Tor client, with the ***exception*** of generating authority certificates (this is unnecessary for Tor relays and clients). Then, the fourth network version was stopped using the command ***lhalt***. At this point, it is worth mentioning that the ***passphrase*** for the command ***tor-gencert***, used in the ***dirauth_gen_keys.sh*** script, was imported manually.

Following the completion of the above steps, each Netkit-ng virtual machine folder, located in the lab directory, contained a set of cryptographic keys used by Tor and a ***torrc text file***. Finally, the cryptographic fingerprints of ***Directory Authorities***, located in the ***dirauth_fingerprints.txt*** text files, were used to ***manually*** configure the individual torrc text files of each virtual machine. The ***dirauth_gen_keys.sh***, ***relay_gen_keys.sh*** and ***client_gen_key.sh*** scripts are available in Appendix K.

4.10.3 Data Collection

The data presented in Tables 4.20 – 4.21 was acquired based on the set of network evaluation criteria (section 4.4) and was used to compare and evaluate the fourth virtual network versions with the third. Due to the rather large number of virtual machines, the complete tables of data is presented in Appendix N, O and P. Also, the routing paths, presented in Table 4.20, list only the three newly added virtual machines, since nothing else had changed from the previous version.

Table 4.20. Version 4: Tor Integration – routing paths

NO	VIRTUAL MACHINE	IP	NETWORK INTERFACE	DEFAULT GATEWAY
1.	dirauth3	10.35.0.1	eth0	10.35.0.254
2.	dirauth4	10.36.0.1	eth0	10.36.0.254
3.	dirauth5	10.37.0.1	eth0	10.37.0.254
4.	internet	10.35.0.254 10.36.0.254 10.37.0.254	eth35 eth36 eth37	10.35 .0.1 10.36.0.1 10.37.0.1

Table 4.21. Version 4: Tor Integration – running services

NO	VIRTUAL MACHINE	RUNNING SERVICES
1.	All virtual machines	bind9, dbus, ebtables, lxc, resolvconf, rpcbind, rsyslog, ssh
	dns, gw1, gw2, internet	tor

Table 4.22. Version 4: Tor Integration – portability

	DIRECTORIES / FILES	STORAGE ALLOCATION
1.	Lab directory (7 items)	1.8 MB
2.	netkit-ng (827 items)	2.1 GB = 2 100 000 KB
TOTAL	-	2 101 800 KB

Table 4.23. Version 4: Tor integration – speed

	VM	VM BOOT UP TIME (min: sec. millisec)	NETWORK BOOT TIME (min: sec. millisec)
1.	dns	00:20.88	00:20.88
2.	internet	00:54.87	01:15.75
……	……	……	……
33.	relay30	00:14.76	02:41.14
……	……	……	……
43.	pc4	00:22.47	03:18.15

Table 4.24. Version 4: Tor Integration – hardware limitations

	VIRTUAL MACHINE	MEMORY ALLOC. ON RUN TIME	STORAGE ALLOC. ON RUN TIME	CPU USAGE ON / SYSTEM RUN TIME
1.	dirauth1	277 MB	772 KB	0.7% – 1.3%
……	……	……	……	……
10.	pc1	149 MB	448 KB	0.7% – 1.3%
……	……	……	……	……
TOTAL	-	7936 MB ≈ 7.9 GB	24152 KB ≈ 24.1 MB	30.1% – 55.9%

Table 4.25. Version 4: Tor Integration – Open Ports

	VM	OPEN PORT	PROTOCOL	SERVICE
……	……	……	……	……
6.	pc1 - pc4	0.0.0.0:5000 19*.168.0.10*:53 127.0.0.1:9011	tcp tcp tcp	tor named tor
7.	dirauth1 - dirauth5	0.0.0.0:5000 0.0.0.0:7000 10.*.0.1:53	tcp tcp tcp	tor tor named
8.	relay1 - relay30	10.*.0.1:53 0.0.0.0:5000	tcp tcp	named tor

4.10.4 Network Evaluation

Although the ***network size*** of the fourth virtual network version ***did not*** substantially increase, there are still major differences when compared with previous versions. The ***update*** and ***installation*** of new software packages, the allocation of ***additional memory*** to all virtual machines and the ***configuration*** and ***deployment*** of the Tor Directory Authorities, relays and clients significantly increase the overall

complexity of the network infrastructure. Besides the evaluation and comparison of the collected data versus the third virtual network (***WAN with DNS***), this subsection aims at analysing a number of ***unexpected system behaviours*** and providing a concise, justifiable evaluation of interesting network and system characteristics. The last goal of this subsection is to identify and highlight possible network or system improvements, which could be implemented during the fifth and final version of development.

The results obtained regarding network ***routing paths***, ***portability*** and ***speed*** are identical to the third network version ***(WAN with DNS).*** Despite the fact that the Netkit-ng ***filesystem*** was updated and additional packages were installed, the storage allocation remained at approximately ***2.1 GB.*** The fourth network version required an additional ***41 seconds*** to boot up (3:18) compared with the third network version (02:37), which was expected since three additional virtual machines were added and the ***tor service*** was initiated from the ***Upstart scripts***.

As discussed during the Netkit-ng filesystem update, a number of services were ***automatically started*** and ***imported*** to the ***Upstart scripts***. The command ***updaterc.d*** was used to remove them and the command ***service --status-all*** was used to verify their removal. However, based on the ***open ports*** and ***running services*** obtained during the data collection, it can be assumed that the ***rpcbind***, ***bind9***, ***ssh*** and ***dbus*** services were still running on all virtual machines. In addition, the ***Tor service*** was started on all virtual machines. Nevertheless, the command ***service --status-all*** could not recognise that the ***tor service*** was running, with the exception of ***gw1***, ***gw2***, ***internet*** and ***dns*** virtual machines.

Although there is no clear explanation why the ***Tor service*** was not listed as running in the majority of virtual machines, it could still be ***verified*** through open ports (***netstat***), operating system processes (***top***) and upstart listing (***ls /etc/rc*.d***) that the ***tor service*** was indeed running. Having in mind that Netkit-ng uses a modified version of the ***Debian Wheezy*** operating system, it can be speculated that both ***systemd*** and ***init.d*** initialization tools were used in all virtual machines. This is due to the 2013 migration of ***Debian*** to ***systemd*** from ***Upstart***, preserving the ***Upstart*** and ***SysV*** initialization tools in order to guarantee ***backward compatibility***. Consequently, running ***service --status-all*** was listing services that used a more traditional initialization script. For instance, the ***torrc text file*** had not been modified in the ***dns***, ***gw1***, ***gw2*** and ***internet*** virtual machines, resulting in the use of ***@CONFDIR@/torrc-defaults script***, which contained a

default Tor configuration (this is used if the ***/etc/tor/torrc*** text file is not modified). Similarly, the remaining virtual machines did not list ***tor*** as a ***running service*** through the ***service*** command, since their ***torrc text files*** had been modified, resulting in the execution of a native initialization script. The speculation that ***more than one initialization tools*** were used by Netkit-ng virtual machines could be verified through the use of ***locate*** command (line 1-4).

```
1: updated
2: locate upstart
3: locate systemd
4: locate init.d
```

The most interesting but at the same time ***troubling results*** concern the total memory allocation of the fourth network version. Specifically, the consumption of ***RAM*** by each Netkit-ng virtual machine ranged between ***149 MB to 534 MB***, with an overall ***RAM*** allocation of approximately ***7.9 GB***. Having in mind that the host operating system (Ubuntu) was allocated ***4 GB of RAM*** and that the laptop (MacBook Pro) had a total of ***16GB*** available, the amount of allocated ***RAM*** by the fourth network version was extremely high. However, one must bear in mind that Netkit-ng virtual machines would be allowed to access an overall RAM of ***7.9 GB,*** only in case there was a need for it.

Nonetheless, as a result there could be a significant amount of ***memory swapping***, meaning that there would be a ***constant data exchange*** between the ***RAM*** and the ***hard disk***, since the system required more ***RAM*** than the available amount. However, the fact that ***memory swapping*** would happen in an already virtualized environment (Ubuntu), makes its analysis an ***extremely complex task*** that requires deep knowledge of the virtualization system in use (***Parallels for Mac***). ***Memory swapping*** is normally executed by the operating system ***kernel*** or, in the case of virtualized environments by the ***hypervisor*** process. Although the concept of ***memory swapping*** is considered outside the project boundaries, it is a critical aspect of the virtual network infrastructure, as it can severely ***affect*** the system performance.

As discussed in section 4.4, each Netkit-ng virtual machine contains ***three sub-processes*** that are responsible for the execution of Netkit-ng modules such as the creation of a virtual ***network hub*** and the ***I/O*** in the allocated ***.disk file***. However, as it can be observed through the ***htop*** tool, the only difference between those three sub-processes is the ***process id***. Based on these similarities, a hypothesis could be made: if a Netkit-ng virtual machine could operate with ***only one*** of these three sub-processes and having

in mind that they access ***the same memory block***, then the overall ***network consumption of RAM*** could be reduced. Due to the ***lack of documentation*** regarding the processes required by Netkitng virtual machines to operate, two ***experiments*** were conducted to verify this ***hypothesis***.

The goal of the first experiment was to ***validate*** the current ***Netkit-ng configuration*** in the host operating system (Ubuntu) and ***reject*** the possibility of a ***mistake*** during the preliminary steps of the project (section 4.1 - 4.2). A ***new virtual machine*** was built using an ***Ubuntu 18.04*** .iso file and the Netkit-ng was ***downloaded*** and ***configured***, following the steps discussed in section 4.2. The command ***vstart vm1*** was used to initiate a Netkitng virtual machine and the command ***htop*** was used to check the associated running processes. As expected, the ***vm1 virtual machine*** was using the ***same six processes*** discussed in section 4.4. The same experiment was conducted on an ***Ubuntu 18.04.3*** and an ***Ubuntu 18.04*** with identical results. ***Based on the first experiment, the configuration of Netkit-ng was considered valid.***

The second experiment aimed at ***checking*** whether a Netkit-ng virtual machine was ***able to operate*** with only one of the three sub-processes running. The command ***vstart vm1*** was used to initiate a Netkit-ng virtual machine. Then, the ***process ids*** of the three subprocesses were extracted from the ***htop command*** and the command ***kill*** was used to terminate ***two*** of the three running processes. The ***vm1*** virtual machine continued to run ***without any interruption*** or ***error message***. However, when the command ***ping*** was used in ***vm1*** the virtual machine ***froze***. When the last remaining process was killed the vm1 virtual machine ***crashed***.

In order to better understand the functionality of each of the three running processes, vm1 was booted up again and the command ***strace <pid>*** was used to debug the three processes. According to the output of ***strace***, one of the running processes was managing the I/O operations of vm1 while another process was responsible for networking operations. No functionality or strace output was observed regarding the third process. The same experiment was conducted several times to verify the results. The only attained observation made, was that the process with the lowest pid was always the one that was idle. The results were further verified by conducting the same experiment in the ***internet***, ***relay1*** and ***dns*** virtual machines of the fourth virtual network. Based on the outcomes of the second experiment, the hypothesis that a

Netkit-ng virtual machine can operate with ***only one*** of these three sub-processes was rejected. However, the purpose of the third sub-process (the one with the highest pid number) remained unclear.

4.10.5 Tor Configuration Evaluation

By design, Tor generates a number of useful information and log files depending on the network role of the system (client, relay, Directory Authority), with the majority of this log files located in ***/var/lib/tor/*** and ***/var/log/tor/***. The goal of this subsection is to analyse the configuration of Tor based on the information contained in the ***info.log*** text files located in ***dirauth1***, ***relay1*** and ***pc1.*** The specific files are selected because they contain the ***most detailed information*** regarding the ***actions*** and ***processes*** performed by each ***Tor entity***. Due to the substantial size of each ***info.log text file*** (***4000 lines in 18 minutes***), only a sample version***,*** obtained from the ***dirauth1*** virtual machine is provided in Appendix Q.

Following the initialization of the fourth network version, the tor service was started. Initially, ***Tor relay*** and ***Tor client*** virtual machines attempted to connect to one of the ***nine real Tor Directory Authorities*** in order to ***fetch*** the latest version of ***consensus***. This is due to the fact that a ***cache-consensus*** text file was downloaded, along with the Tor package, during the execution of ***apt-get install tor.*** Once all ***fetch consensus*** requests returned an ***http status code 404 ("Not Found")***, the tor service recognised the addition of the ***five new Tor Directory Authorities*** located in the ***/etc/tor/torrc*** text file and attempted to match the cryptographic ***fingerprints*** with existing records in the ***cached_ consensus*** text file. As expected, the matching process failed, since the five new Tor Directory Authorities did not exist in the ***cached_ consensus***. As a result, all ***Tor relays*** and ***Tor clients*** attempted to initiate a ***TLS session*** with any of the five ***reachable Tor Directory Authorities*** in order to ***authenticate*** their ***fingerprints***. This procedure was ***repeated continuously*** until an authentication and a new consensus file was received from at least ***three of the five Tor Directory Authorities.***

At the same time, the five Tor Directory Authority virtual machines after the failed attempts to connect to the ***nine real Tor Directory Authorities***, they started a process of authenticating between them by establishing ***TLS sessions*** and ***authenticating*** the ***fingerprints*** located in the ***/etc/tor/torrc*** text file. Once this authentication procedure was completed all five Tor Directory Authority virtual machines started ***requesting*** and ***collecting*** relay and client ***descriptor files*** in order to ***generate*** a new consensus document.

The process of generating a new consensus file took approximately ***one minute*** to complete. Once all five Tor Directory Authority virtual machines created their own ***version of consensus***, a ***voting procedure*** was initiated between them (section 2.6.4), which took ***approximately nine minutes*** to complete. Upon completion, an ***identical consensus*** document was ***published*** on all five ***Tor Directory Authority virtual machines***. Then, all ***Tor relay*** virtual machines attempted to ***fetch the consensus document***, a task that took approximately ***three minutes***. The last Tor entities that obtained a copy of consensus, ***fifteen minutes*** after the initialization of the fourth network version, were the ***Tor clients***. At this point in time, the ***Tor clients*** were in a position to ***start*** the process of building Tor circuits. The consensus document was ***refreshed*** every ***fifteen*** minutes, by repeating the same processes.

4.11 Version 5: Tor Experimental Network

The ***fifth*** and ***final*** network version aims at addressing a number of ***deficiencies*** and ***problems*** that have been identified during the previous network and tor configuration evaluation. Upon completion, it also aims at producing an ***isolated Tor experimental network*** that incorporates the majority of ***capabilities*** and ***processes*** of the ***real Tor network***. To this extent, ***two*** additional virtual machines, which act as ***webservers***, were added to the virtual network in order to provide another interface for experimentation. This subsection concludes with the incorporation of ***Object Oriented Programming*** principles (OOP) in the ***shared.startup*** script and the implementation of the ***kill_duplicates.sh script***, with the objectives of ***improving*** the overall network ***efficiency***, ***lowering*** the amount ***computational power*** required during boot up and ***reducing*** the overall ***memory*** consumption. Figure 4.8 illustrates the network topology, IP ranges and running services of the produced Tor experimental network.

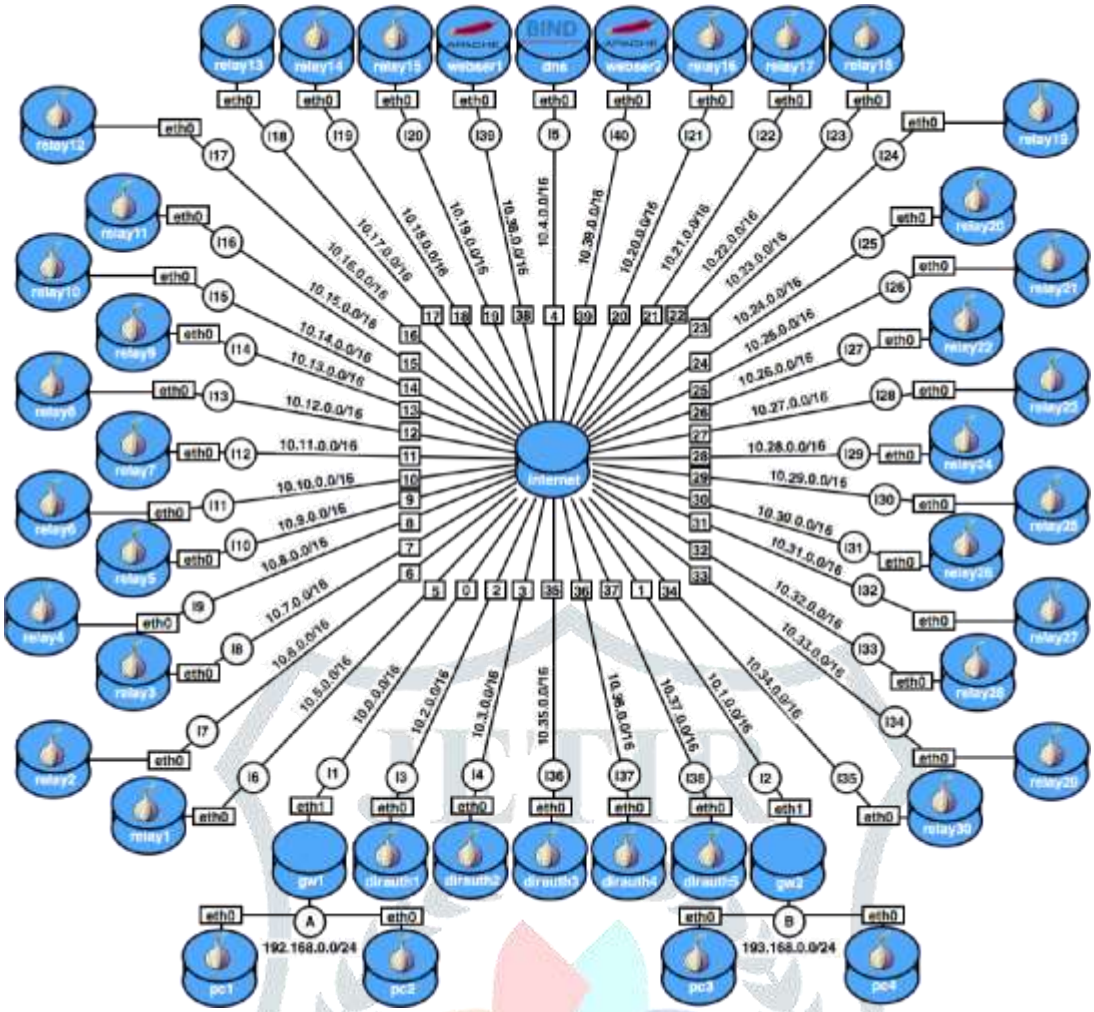


Figure 4.8 Tor Experimental Networks

4.11.1 Development

Initially, two virtual machines ***(webser1, webser2)*** were added to the virtual network and an ***Apache web service*** was started using the command ***/etc/init.d/apache2***. The two ***webservers*** were connected directly to the ***internet*** virtual machine but they were not part of the virtual Tor network, aiming at providing an additional experimentation interface. For instance, ***pc1*** virtual machine could send an ***http request*** to the Apache web server running on ***port 80*** of the ***webser1*** virtual machine either through the ***internet*** virtual machine or using the ***Onion Proxy*** running on ***localhost port 9011*** of ***pc1*** virtual machine.

Besides the addition of the two ***webservers***, the main focus of this development activity was to improve the overall network efficiency by ***eliminating*** unnecessary services and ***optimising*** existing functionality. To this extent, a number of software packages, such as ***rpcbind***, ***dnsmasq***, ***pdns_recursor***, ***dbus***, ***quagga*** and ***wireshark-common*** were ***permanently removed*** from the Netkit-ng filesystem, in order to reduce the overall filesystem size. Also, the command ***update-rc.d "package-name" disable | remove*** was used to ***disable*** and then ***remove*** all software packages configured to start on boot up through the ***Upstart***

initialization tool, with the exception of ***Debian Wheezy*** and ***Netkitng*** dependent packages, is listed and explained in Appendix R. At this point, it is important to ***highlight*** that the ***tor service*** was also removed from the Upstart services, in order to prevent its execution on virtual machines that were not part of the virtual Tor network like ***gw1***, ***gw2***, ***dns*** and ***internet***.

Consequently, the function ***initiating_tor()***, declared in the ***shared.startup*** script was executed in all Tor-configured virtual machines, to start the tor service. To further optimize the ***shared.startup*** code quality, ***repeated code fractions*** were wrapped in functions and the ***nested if*** statements were replaced by a ***switch statement.***

Having in mind the analysis of the latest network evaluation, regarding the ***total memory consumption*** of the fourth network version and based on the conclusions of the ***two experiments*** (1. verification of Netkit-ng ***configuration***, 2. ***killing*** of duplicate virtual machine processes), a ***kill_duplicates.sh*** script was produced. This script was able to identify all ***running*** Netkit-ng virtual machines, extract the ***process id*** of the ***three*** identical ***sub-processes*** and then ***kill*** the one with the highest pid number. The final version of ***shared.startup*** and ***kill_duplicates.sh*** is available in Appendix S.

It is important to mention that despite the fact that the execution of ***kill_duplicates.sh script,*** reduced the number of ***running operating system processes*** and the allocation of relevant resources, the script ***was not actually used*** in the following data collection and evaluation processes. In general, the termination of any operating system sub-processes without knowing their exact purpose is considered a highly insecure practice and should be avoided.

4.11.2 Data Collection

The data presented below was captured based on the network evaluation criteria (section 4.4) and was used to compare and evaluate the produced Tor Experimental environment with the previous network versions. Due to the increased number of virtual machines, the complete data of Table 31 (speed) are presented in Appendix T, and only a selected number of machines are presented in this subsection. Due to the similarity of results with the fourth network version, network routing paths (Table 4.26) and hardware limitations (Table 4.30) consider only the two newly added virtual machines (webser1, webser2).

Table 4.26. Version 5: Tor Experimental Network – routing paths

NO	VIRTUAL MACHINE	IP	NETWORK INTERFACE	DEFAULT GATEWAY
1.	webser1	10.38.0.1	eth0	10.38.0.254
2.	webser2	10.39.0.1	eth0	10.39.0.254
3.	internet	10.38.0.254 10.39.0.254	eth38 eth39	10.38.0.1 10.39.0.1

Table 4.27. Version 5: Tor Experimental Network – running services

NO	VIRTUAL MACHINE	RUNNING SERVICES
1.	All virtual machines except dns, gw1, gw2, internet, webser1, webser2	tor
2.	dns	bind9
3.	webser1, webser2	appache2

Table 4.28. Version 5: Tor Experimental Network – portability

	DIRECTORIES / FILES	STORAGE ALLOCATION
1.	Lab directory (7 items)	1.8 MB
2.	netkit-ng (827 items)	2.1 GB = 2 100 000 KB
TOTAL	-	2 101 800 KB

Table 4.29. Version 5: Tor Experimental Network – speed

	VM	VM BOOT UP TIME (min: sec. millisec)	NETWORK BOOT TIME (min: sec. millisec)
1.	dns	00:15.30	00:15.30
……	……	……	……
43.	pc2	00:17.94	
43.	pc3	00:17.94	02:39.30
43.	pc4	00:17.94	

Table 4.30. Version 5: Tor Experimental Network – hardware limitations

	VIRTUAL MACHINE / SYSTEM	MEMORY ALLOC. ON RUN TIME	STORAGE ALLOC. ON RUN TIME	CPU USAGE ON RUN TIME
1.	webser1	149 MB	460 KB	0.7% – 1.3%
2.	webser2	149 MB	450 KB	0.7% – 1.3%
3.	Network Hop I39	7552 KiB ≈ 7.7 MB	-	0%
4.	UML Switch I39	2172 KiB ≈ 2.2 MB	-	0%
5.	Network Hop I40	7552 KiB ≈ 7.7 MB	-	0%
6.	UML Switch I40	2172 KiB ≈ 2.2 MB	-	0%
……	……	……	……	……
TOTAL 25 MB 55.9%	-	8.1 GB	2506 2 KB ≈	30.1% –

Table 4.31. Version 5: Tor Experimental Network – Open Ports

	VM	OPEN PORT	PROTOCOL	SERVICE
1.	relay1 - relay30	0.0.0.0:5000	tcp	tor
2.	pc1 - pc4	127.0.0.1:9011	tcp tcp	tor tor
3.	dirauth1 - dirauth5	0.0.0.0:5000 0.0.0.0:7000	tcp tcp	tor tor
4.	webser1, webser2	:::80	tcp6	apache2
5.	dns	127.0.0.1:953 127.0.0.1:53 ::1:953 :::53 10.4.0.230:53	tcp tcp tcp6 tcp6 / udp6 tcp / udp	rndc rndc rndc rndc named
6.	dirauth1 - dirauth5 relay1 - relay30	0.0.0.0: *	udp	tor

4.11.3 Network Evaluation

The network routing paths, portability and hardware limitation results, obtained during the latest data collection, are very similar to the data collected during the fourth network version. Consequently, this subsection focuses on the analysis and overall evaluation of the running services and open ports.

As discussed in the evaluation of the fourth network version (***Tor Integration***), the ***bind9***, ***dbus***, ***rpcbind***, ***ssh*** and ***tor*** services were running on all virtual machines. However, there was no real need for the use of ***dbus***, ***rpcbind*** and ***ssh*** in any of the virtual machines. Also, the ***bind9*** service should have been used only by the dns virtual machine, since it was allocated the role of a ***DNS server*** for the entire network infrastructure. Finally, there was a small number of virtual machines that ran the ***tor service*** without being part of the virtual Tor network. In order to address these ***deficiencies***, appropriate actions were taken during the development of the fifth virtual network version. As a result, the number of running services in all virtual machines of the fifth network version, ***were significantly reduced***, as illustrated in Table 4.27.

In addition, the ***Tor clients*** of the fourth network version were configured via the ***/etc/tor/torrc*** text file to run the ***onion proxy software*** on the ***localhost port 9011,*** which allowed them to redirect their traffic in the Tor network. Also, the ***OrPort*** functionality was enabled via ***port 5000***. However, ***OrPort*** mostly used by ***Tor relays*** or Tor clients wishing to run a ***Tor hidden service.*** Since neither of the two cases is valid, the attribute ***OrPort 5000*** was ***removed*** from the ***/etc/tor/torrc*** text file of all Tor client virtual machines (pc1 - pc4). As a result, the Tor clients of the fifth network version used only ***port 9011*** (Table 4.31).

4.12 Tor Experimental Network Evaluation

This subsection concludes the second development and evaluation phase of the project and aims to highlight the most important characteristics of the produced Tor Experimental Network. Particular attention is given to unexpected system behaviours and peculiar network aspects. This subsection completes with the design, implementation and analysis of experiments which are used as a proof of concept.

4.12.1 Important Observations

Based on what has been discussed in Chapter 2 (The Onion Router) and according to the Tor documentation (TorProject, 2017k), the ***UDP protocol*** is not used in the Tor network. However, it was observed using the command ***netstat -tu*** that the tor service, running in all Tor relays and Tor Directory Authorities in the produced Tor experimental network, had ***randomly selected a port*** (30000 and above) listed as ***UDP***. According to the Tor tickets ***#8219*** (elgo, 2017) and ***#965*** (Sebastian, 2014), earlier versions of Tor used to allow Tor relays to perform DNS queries (hence the selection of a random UDP port) in

order to test ***their own DNS resolution*** (even though they ***shouldn't be willing*** to perform DNS queries for Tor clients) and ***latest version of Tor*** maintains a random UDP port in all ***Tor exit relays***, as they might need to perform a ***DNS query*** on behalf of a Tor client and then return the results to the Tor client through the Tor circuit.

Having in mind the above, additional ***tests*** and ***significant observations*** were made. Before discussing these observations, one must be aware that a ***Tor relay*** can be assigned ***up to*** six flags: (i) Valid, (ii) Guard, (iii) Exit, (iv) Fast, (v) Stable and (vi) Running, all listed in the consensus document. Initially, it was observed through the newly generated and shared ***consensus*** document that both the ***Exit*** and ***Guard*** flags were assigned to ***all Tor relays***, meaning that the specific Tor relays could act either as an ***Entry*** or an ***Exit*** node. Figure 4.9 illustrates the use of the ***listing.sh*** script to extract the information from the consensus file of the dirauth5 virtual machine.

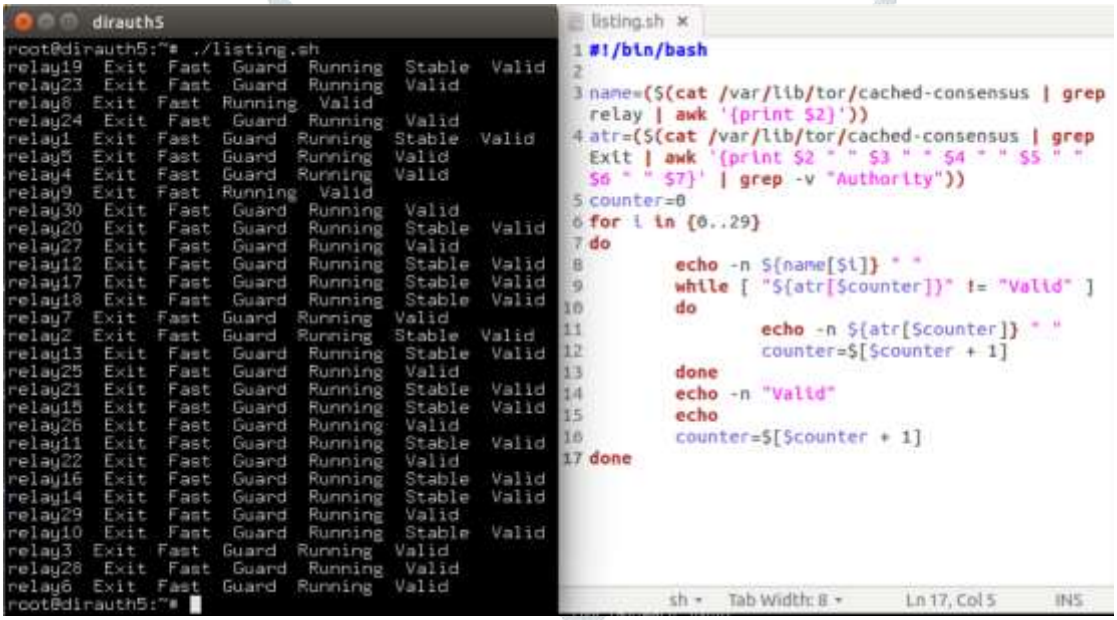


Figure 4.9 Extraction of Tor relay flags from consensus file

The files ***/var/lib/tor/router-stability*** and ***/var/lib/tor/v3-status-votes*** can be used to provide the reasoning behind the decision of the Tor Directory Authorities to assign the ***Exit*** and ***Guard*** flags to all running Tor relays. As discussed in section 2.7, in order for a Tor relay to be an ***Entry node*** it needs to have an ***up-time*** greater than 12,5%, provide at least ***250 KB/s bandwidth*** and have a ***WFU*** (Weighted Fractional Uptime) of 98%. However, the produced Tor experimental network ***does not contain "Tor Bandwidth***

Authorities" (Stem, 2017), which are responsible for taking periodic measurements of all Tor relays and share them with the Tor Directory Authorities. As a result, the ***bandwidth*** and ***WFU*** of all Tor relays are

set to ***Unmeasured = 1***. In the real Tor network, any Tor relay with ***Unmeasured=1*** is removed from the consensus. However, since the attribute ***TestingTorNetwork 1*** is included in all ***torrc text files***, this is an acceptable behaviour. Similarly, a Tor relay can be assigned the flag ***Exit*** if it fulfils the above mentioned Guard requirements and is ***allowed exits*** to at least two of the ports ***80*** (HTTP), ***443*** (standard TCP port which use SSL) and ***6667*** (IRC service). Since the attribute ***FirewallPorts*** is not specified in any of the ***/etc/tor/torrc*** text files and the command ***ExitPolicy "IP"*** does not specify a range of acceptable exit ports (e.g. reject *:80), all Tor relays of the produced Tor Experimental Network are eligible to be Exit nodes. As a result of all the above, the Tor Directory Authorities set both the flags ***Exit*** and ***Guard*** for all Tor relays.

4.12.2 Proof of Concept - Traffic Capturing

A number of actions and experiments were conducted in order to verify the ***validity*** and ***correctness*** of the produced Tor Experimental Network. The main objective of the experiments was to ***verify the correctness of the Tor workflow*** and to ***test*** that a Tor ***Client*** was able to build a Tor ***circuit*** and ***communicate*** via the Tor network with an ***Apache*** webserver. Furthermore, the experiments aimed at ***identifying*** whether there were ***any complications*** due to the ***Exit*** and ***Guard*** flags set for all Tor relays.

The virtual machine ***pc4*** was selected as Tor client and the ***webser1*** as the Apache webserver. In order to identify the Tor ***exit*** node which was selected by the ***pc4*** virtual machine**,** tcpdump was deployed in the ***webser1*** virtual machine. The command ***curl -socks5 127.0.0.1:9011 http:/10.38.0.1*** was used to send an http request from ***pc4*** via the ***Onion Proxy*** (localhost 9011) to the webserver. As illustrated in Figure 4.9, the webserver received the request from the ***IP 10.28.0.1 (relay24)***.

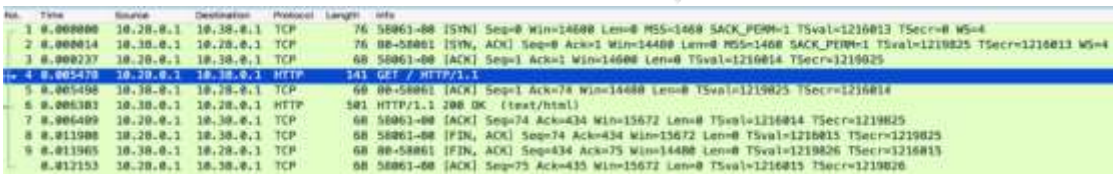


Figure 4.10 HTTP request from Tor Exit relay24 to webserver

In order to identify the Tor middle relay, tcpdump was deployed on ***relay24*** and a second ***http request*** was sent from ***pc4*** to the ***webserver***. As illustrated in Figure 4.10, the ***relay24*** virtual machine received a series of encrypted packets from ***IP 10.9.0.1 (relay5).*** It is worth mentioning that the size of encrypted packets was ***611 bytes*** since, besides the ***512 bytes*** of each Tor cell (section 2.10), the Tor cells are wrapped in the headers of three additional protocols: ***TLS***, ***IP*** and ***Ethernet***.

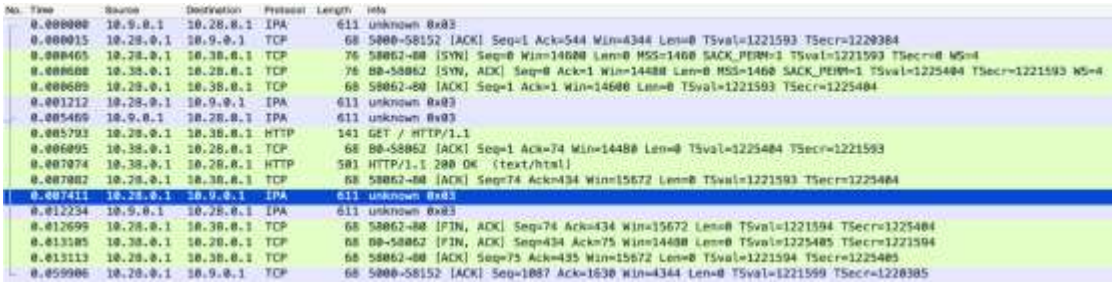
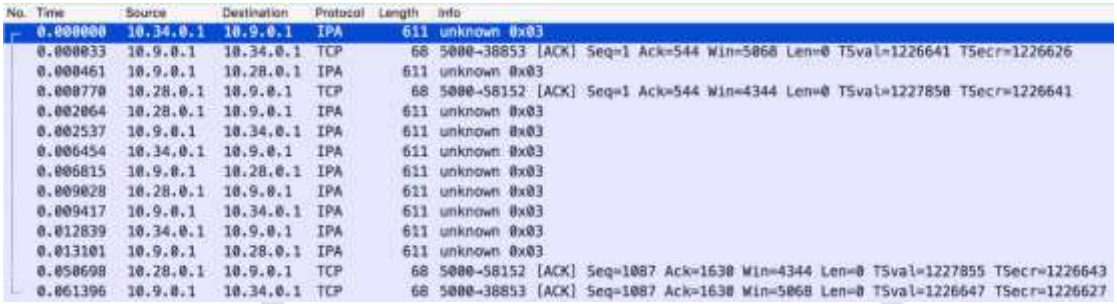


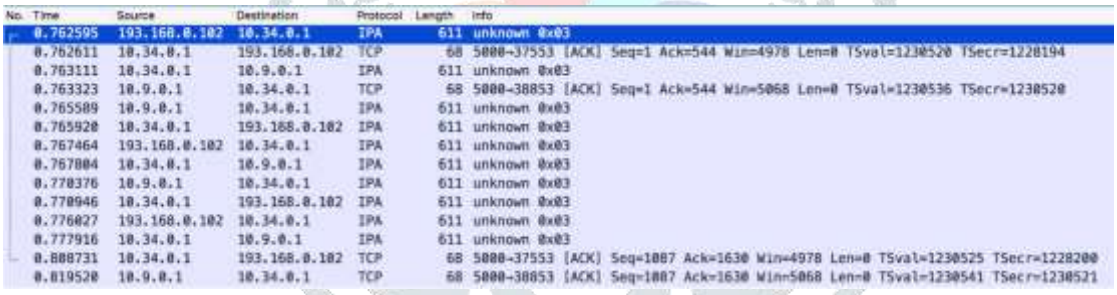
Figure 4.11 Tor Exit relay communication with webserver and Tor middle relay Next, tcpdump was deployed on ***relay5 (Tor middle relay)*** to identify the Tor Entry relay. Figure 4.11 illustrates the encrypted communication originating from the Tor entry relay with IP 10.34.0.1 (relay30) and forwarded to the exit relay with IP 10.28.0.1 (relay24).



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.34.0.1	10.9.0.1	IPA	611	unknown 0x03
2	0.000033	10.9.0.1	10.34.0.1	TCP	68	5000→38853 [ACK] Seq=1 Ack=544 Win=5068 Len=0 TSval=1226641 TSecr=1226626
3	0.000461	10.9.0.1	10.28.0.1	IPA	611	unknown 0x03
4	0.000770	10.28.0.1	10.9.0.1	TCP	68	5000→58152 [ACK] Seq=1 Ack=544 Win=4344 Len=0 TSval=1227850 TSecr=1226641
5	0.002064	10.28.0.1	10.9.0.1	IPA	611	unknown 0x03
6	0.002537	10.9.0.1	10.34.0.1	IPA	611	unknown 0x03
7	0.006454	10.34.0.1	10.9.0.1	IPA	611	unknown 0x03
8	0.006815	10.9.0.1	10.28.0.1	IPA	611	unknown 0x03
9	0.009028	10.28.0.1	10.9.0.1	IPA	611	unknown 0x03
10	0.009417	10.9.0.1	10.34.0.1	IPA	611	unknown 0x03
11	0.012839	10.34.0.1	10.9.0.1	IPA	611	unknown 0x03
12	0.013101	10.9.0.1	10.28.0.1	IPA	611	unknown 0x03
13	0.058698	10.28.0.1	10.9.0.1	TCP	68	5000→58152 [ACK] Seq=1087 Ack=1630 Win=4344 Len=0 TSval=1227855 TSecr=1226643
14	0.061396	10.9.0.1	10.34.0.1	TCP	68	5000→38853 [ACK] Seq=1087 Ack=1630 Win=5068 Len=0 TSval=1226647 TSecr=1226627

Figure 4.12 Traffic passing through the Tor middle relay5

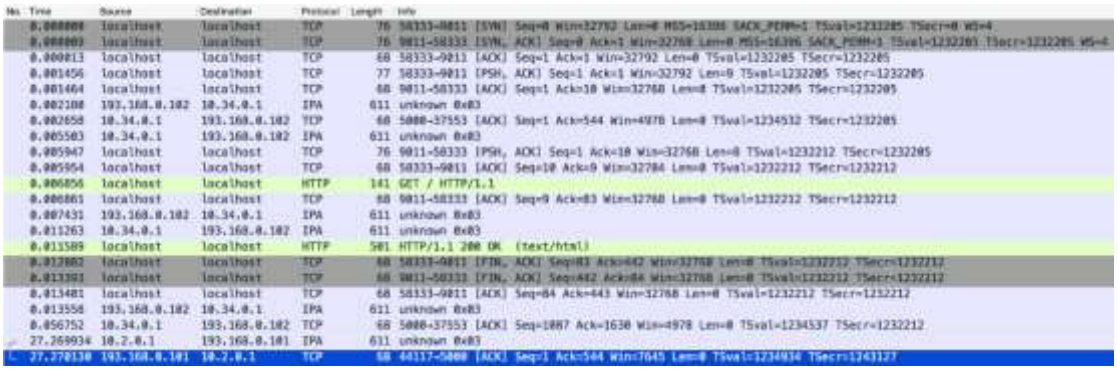
The results were verified by observing the traffic in relay30 as illustrated in Figure 4.13.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.762595	193.168.0.102	10.34.0.1	IPA	611	unknown 0x03
2	0.762611	10.34.0.1	193.168.0.102	TCP	68	5000→37553 [ACK] Seq=1 Ack=544 Win=4978 Len=0 TSval=1230520 TSecr=1228194
3	0.763111	10.34.0.1	10.9.0.1	IPA	611	unknown 0x03
4	0.763323	10.9.0.1	10.34.0.1	TCP	68	5000→38853 [ACK] Seq=1 Ack=544 Win=5068 Len=0 TSval=1230536 TSecr=1230520
5	0.765589	10.9.0.1	10.34.0.1	IPA	611	unknown 0x03
6	0.765920	10.34.0.1	193.168.0.102	IPA	611	unknown 0x03
7	0.767464	193.168.0.102	10.34.0.1	IPA	611	unknown 0x03
8	0.767804	10.34.0.1	10.9.0.1	IPA	611	unknown 0x03
9	0.770376	10.9.0.1	10.34.0.1	IPA	611	unknown 0x03
10	0.770946	10.34.0.1	193.168.0.102	IPA	611	unknown 0x03
11	0.776027	193.168.0.102	10.34.0.1	IPA	611	unknown 0x03
12	0.777916	10.34.0.1	10.9.0.1	IPA	611	unknown 0x03
13	0.888731	10.34.0.1	193.168.0.102	TCP	68	5000→37553 [ACK] Seq=1087 Ack=1630 Win=4978 Len=0 TSval=1230525 TSecr=1228200
14	0.819520	10.9.0.1	10.34.0.1	TCP	68	5000→38853 [ACK] Seq=1087 Ack=1630 Win=5068 Len=0 TSval=1230541 TSecr=1230521

Figure 4.13 Traffic passing through the Tor Entry relay30

Finally, tcpdump was used on the Tor client (pc4) to ***double check*** the validity of the results and ***verify*** the ***redirection*** of the ***http request*** through the ***Onion Proxy*** as illustrated in Figure 4.14



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	localhost	localhost	TCP	76	58333→9011 [SYN] Seq=0 Win=32792 Len=0 MSS=16396 SACK_PERM=1 TSval=1232205 TSecr=0 WS=4
2	0.000003	localhost	localhost	TCP	76	9011→58333 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=16396 SACK_PERM=1 TSval=1232205 TSecr=1232205 WS=4
3	0.000013	localhost	localhost	TCP	68	58333→9011 [ACK] Seq=1 Ack=1 Win=32792 Len=0 TSval=1232205 TSecr=1232205
4	0.001456	localhost	localhost	TCP	77	58333→9011 [PSH, ACK] Seq=1 Ack=1 Win=32792 Len=9 TSval=1232205 TSecr=1232205
5	0.001464	localhost	localhost	TCP	68	9011→58333 [ACK] Seq=1 Ack=10 Win=32768 Len=0 TSval=1232205 TSecr=1232205
6	0.002108	193.168.0.102	10.34.0.1	IPA	611	unknown 0x03
7	0.002658	10.34.0.1	193.168.0.102	TCP	68	5000→37553 [ACK] Seq=1 Ack=544 Win=4978 Len=0 TSval=1234532 TSecr=1232205
8	0.005503	10.34.0.1	193.168.0.102	IPA	611	unknown 0x03
9	0.005947	localhost	localhost	TCP	76	9011→58333 [PSH, ACK] Seq=1 Ack=10 Win=32768 Len=8 TSval=1232212 TSecr=1232205
10	0.005954	localhost	localhost	TCP	68	58333→9011 [ACK] Seq=10 Ack=9 Win=32784 Len=0 TSval=1232212 TSecr=1232212
11	0.006856	localhost	localhost	HTTP	141	GET / HTTP/1.1
12	0.006861	localhost	localhost	TCP	68	9011→58333 [ACK] Seq=9 Ack=83 Win=32768 Len=0 TSval=1232212 TSecr=1232212
13	0.007431	193.168.0.102	10.34.0.1	IPA	611	unknown 0x03
14	0.011263	10.34.0.1	193.168.0.102	IPA	611	unknown 0x03
15	0.011589	localhost	localhost	HTTP	501	HTTP/1.1 200 OK (text/html)
16	0.012882	localhost	localhost	TCP	68	58333→9011 [FIN, ACK] Seq=83 Ack=442 Win=32768 Len=0 TSval=1232212 TSecr=1232212
17	0.013381	localhost	localhost	TCP	68	9011→58333 [FIN, ACK] Seq=442 Ack=84 Win=32768 Len=0 TSval=1232212 TSecr=1232212
18	0.013481	localhost	localhost	TCP	68	58333→9011 [ACK] Seq=84 Ack=443 Win=32768 Len=0 TSval=1232212 TSecr=1232212
19	0.013556	193.168.0.102	10.34.0.1	IPA	611	unknown 0x03
20	0.056752	10.34.0.1	193.168.0.102	TCP	68	5000→37553 [ACK] Seq=1087 Ack=1630 Win=4978 Len=0 TSval=1234537 TSecr=1232212
21	27.269934	10.2.0.1	193.168.0.101	IPA	611	unknown 0x03
22	27.270130	193.168.0.101	10.2.0.1	TCP	68	44117→5000 [ACK] Seq=1 Ack=544 Win=7045 Len=0 TSval=1234934 TSecr=1243127

Figure 4.14 Traffic redirection through the onion proxy

Based on the obtained results, the Tor ***circuit*** constructed by the virtual machine pc4, which is a Tor client, is ***relay30*** => ***relay5*** => ***relay24***. Also, as can be observed in Figures 4.6 to 4.14, the traffic passing through the ***Tor middle*** and ***Tor entry*** nodes is encrypted as expected, despite the fact that both relays could be used as Tor Exit relays.

4.12.3 Proof of Concept - Controller Interface

From version 0.1.0.11 onwards Tor provides to Tor clients an alternative way of managing Tor circuits, known as ***Controller Interface***. According to the Tor documentation, a controller is a ***python-based*** program that connects to the Tor clients and allows them to send control commands such as ***getinfo***, ***setevents*** and ***setconf*** (Mathewson, 2005). Consequently, the attribute ***ControlPort 12345*** was set in all ***/etc/tor/torrc*** text files in order to allow the deployment of the Tor Controller Interface. The connection could have been made through the use of ***netcat*** or ***telnet*** utilities.

However, in order to increase security and ***prevent unauthorised access*** to the service, Tor requires the use of two additional attributes: (i) ***HashedControlPassword*** which contains the hash value of a password, generated by ***tor --hash-password "password"*** and is located in the ***/etc/tor/torrc*** file and (ii) the use of a ***CookieAuthentication*** attribute which generates an authentication cookie located at ***/run/tor/control.authcookie.*** However, for the purposes of this experiment the attribute ***CookieAuthentication*** was set to ***0***, meaning that the Controller Interface would not request a password during the establishment of the connection.

Next, the command ***telnet localhost 12345*** was used in pc1 virtual machine to connect to the Tor controller interface. ***AUTHENTICATE*** followed by space was used to authenticate to the service and then the command ***getinfo circuit-status*** was used to request the randomly select and verified Tor circuits of the pc4 virtual machine. Figure 4.15 illustrates the procedure of connecting to the Tor controller interface and requesting circuit information.

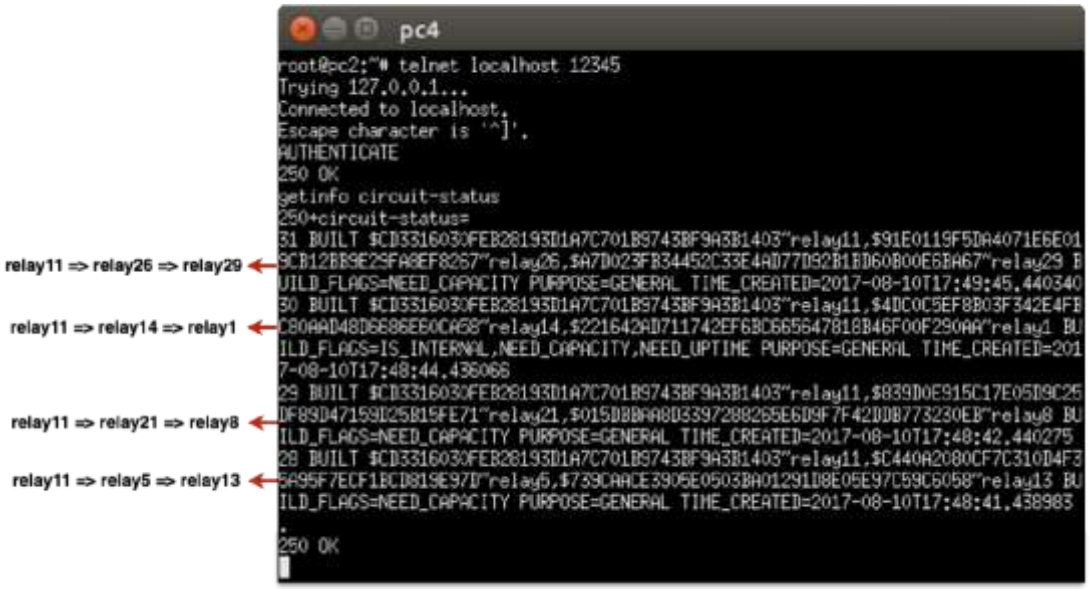


Figure 4.15 Requesting Tor circuit information via the Tor controller interface

In conclusion, the results obtained during the ***traffic capturing experiment*** and the use of Tor ***controller interface*** do not indicate ***any complication*** or issues in the overall Tor workflow of the Tor experimental network. As expected, there was an ***encrypted communication*** within the Tor network and the Tor exit relay ***decrypted*** the traffic and ***propagated*** it to the public network. In addition, a number of different ***Tor circuits*** were constructed from each ***Tor client*** and all Tor traffic was ***redirected*** through the ***Onion Proxy***, preventing any leakage of information. Lastly, the ***UDP protocol*** was not used by any of the Tor entities.

4.13 Discussion and Limitations

This section discusses the outcome of the last development phase by highlighting the key characteristics of the created isolated Tor experimental network in comparison with the knowledge gained from the literature review. The section begins by outlining the importance of online privacy and anonymity with an emphasis in the ways that Tor seeks to provide it. The section then proceeds with a brief overview of the research methodology and research approach. Emphasis is also placed in summarising the two practical phases of the project and its research implications. The section concludes by discussing the system limitations.

The increasing collection of personally identifiable information has become an issue of major value and importance, significantly undermining the fundamental end user right to online privacy. Tor seeks to address the issues surrounding online privacy and provide a means that can enhance end user online anonymity. This is achieved through encrypted, tunneled traffic, propagated through a number of randomly selected relay servers that are located around the world, until the traffic exits to the public internet.

Nonetheless, online privacy enhancing tools, such as Tor, require a significant amount of thorough experimentation and service evaluation in order to successfully assess the risks of cyber-attacks and collect accurate, scientific network data, in an ethically appropriate manner. Therefore, the development of an isolated Tor experiment network can provide an exploratory environment that can be used for secure testing and system evaluation.

Furthermore, the combination of experimental and literature based, research approaches, increased the project's quality and took advantage of the theoretical knowledge gained before initiating the design and implementation of a virtual network infrastructure. The establishment of a strong network foundation and the sequential network development, analysis and evaluation approach contributed to a concise, effective and scientifically verified outcome that incorporated the main principles and capabilities of Tor.

Specifically, the produced virtual system was built on top of a Wide Area Network that was initially based on a Local Area Network. The implementation of a Domain Name Server and the deployment of two Apache webservers, presented a major stepping stone towards the reproduction of a real-world network. The subsequent configuration and enactment of Tor directory authorities, relays and clients expanded the capabilities and functionalities of the overall infrastructure significantly. The fifth and final network version consisted of a total of forty-five virtual machines, each having its own distinct network role. Another major aspect of the network development procedure was the creation of X.509 digital certificates along with the extraction and distribution of cryptographic fingerprints to all virtual Tor entities.

4.14 Summary of Development Results

In order to provide a concise, efficient and scientifically verified virtual network infrastructure, the development phase of the study followed a sequential network development, analysis and evaluation approach. A set of network evaluation criteria were established and defined prior to the development phase, aiming to establish means of comparison between different network versions. The network evaluation criteria consisted of routing paths, open port, running services, network portability, speed and hardware limitations. The remaining of this subsection summarises the results of the five virtual network versions, as follows:

- **Routing Paths**

Due to the manual creation and distribution of X.509 digital certificates, the correlation of cryptographic fingerprints between Tor entities and their assigned IP addresses and to better manage the development environment, static IP addresses were used. Specifically, virtual machines that were directly connected to the internet virtual machine were allocated an IP address that belonged to a different /16 network (10.0-39.0.1-254/16).

In addition, all directly connected virtual machines (to the internet) utilized only their eth0 network interface, with the exception of gw1 and gw2 virtual machines that maintained a second network interface connected to two individual /24 subnets (192193.168.0.0/24). Also, the internet virtual machine was set as the default gateway for all /16 subnets besides the LAN A and B that used their own local router gateway (gw1, gw2). The selected routing paths, IP addresses and network topology did not cause any issues or concerns in any of the five virtual networks.

- **Open Port / Connections**

The two first network versions focused on the design and deployment of the virtual infrastructure rather than the integration of network services. Subsequently, virtual machines of the first (LAN) and second (WAN) network versions did not maintained any open ports. During the development of the third network version (WAN with DNS) a DNS server was configured to run on ports 53 (named) and 953 (rndc), using the bind9 package.

After the Netkit-ng file system modification that took place during the preliminary development steps of the fourth network version (Tor Integration), the tor service was integrated and configured to the virtual infrastructure. As a result, all virtual machines that had been allocated a Tor role (relay, directory authority) maintained the port 5000 (OrPort). Port 7000 (DirPort) was also used by the Tor Directory Authorities. In addition, Tor clients were configured to run an onion proxy on the localhost port 9011. Finally, two apache webservers running on port 80, were deployed during the development of the fifth network version (Tor experimental network).

- **Running Services**

After the Netkit-ng filesystem modification that took place during the preliminary development steps of the fourth network version (Tor Integration), a number of unexpected services such as dbus, rpcbind, ssh and bind9 were running. However, these deficiencies have been addressed during the development phased of the fifth network version. As a result, the produced Tor experimental network utilizes only the tor, bind9 and appache2 services.

- **Portability**

Despite the Netkit-ng filesystem modification, the network portability remained stable in all network version at approximately 2.1 GB.

- **Speed**

The network size and complexity of third network version differed substantially (40 virtual machines, 37 subnets) when compared to the first (3 virtual machines, 1 subnet) and second (10 virtual machines, 7 subnets) network versions. Nonetheless, the third network version required approximately 2:38 minutes to boot up, which was a major improvement considering that the second network version, which consisted of only 10 virtual machines, required 2:09 minutes. Since the network size of the fourth and fifth versions was similar with the third, the speed results were similar (2:39.30).

- **Hardware Limitations**

The second network version consumed 355.2 MB of RAM which was approximately three times more than the first (95.7 MB). However, the memory consumption of the third network version was measured at 1.7 GB of RAM, which exceeded any expectations.

Besides the abnormal memory consumption, the third network version required a significant amount of computational power that ranged between 28% to 52%, compared with the former two network version that required 1.8% to 3.9% and 6% to 13% respectively. The consumption of RAM by each Netkit-ng virtual machine in the fourth and fifth network versions ranged between 149 MB to 534 MB, with an overall RAM allocation of approximately 7.9 - 8.1 GB. Having in mind that the host operating system (Ubuntu) was allocated 4 GB of RAM and that the laptop (MacBook Pro) had a total of 16GB available, the amount of allocated RAM was extremely high.

4.15 Research Implications

Aided by the light-weighted, open source nature of the selected simulation environment (Netkit-ng) and the well documented configuration practices, the produced Tor experimental network is able to contribute to a variety of cyber security domains.

The achieved system scalability, limited only by the available hardware system resources, contributed towards the creation of a secure, isolated environment that can be used for the conduction of research experiments. Also, the ability to alter, update or remove software packages allow the investigation and exploitation of past and current vulnerabilities, aiming at further understanding and analysing the issues leading to an attack. Additionally, the ownership of the entire Tor network allows the examination and analysis of time and traffic correlation attacks that are not feasible in the real Tor network.

The system can also be used as a method of teaching in the context of security architecture, cryptography, network protocols and secure communications. Students can experiment and learn in a secure and controlled fashion about the network design principles and encryption capabilities of one of the most prevailing anonymity system. In a similar fashion, the developed system can be used by law enforcement and junior forensic examiners as a training tool in order to enhance their knowledge and confidence in extracting Tor-related forensic artefacts.

Last but not least, the adoption, design and implementation of a similar network infrastructure within the boundaries of an enterprise, can impose a major defensive barrier regarding industrial espionage and in addressing particularly the issues surrounding insider threats.

4.16 Limitations

As with any piece of research, there will always be room for improvement, no matter how much time or resources are allocated in the research effort. This section outlines and briefly discusses a number of observed system limitations, as follows:

- **Dynamic Scalability**

Despite its substantial network complexity, the produced Tor experimental network does not employ means of integrating additional virtual machines in a dynamic way. Although the deployment of additional

network entities is considered a fairly easy and well documented process, no means of dynamic generation are provided, since the majority of the network infrastructure has been manually created.

- **Allocation of Resources**

This research selected to use the Netkit-ng simulation tool as the development environment due to its low demand in hardware resources. Despite the fact that there is a substantial number of virtual machines in the produced virtual network, the amount of memory finally allocated (7,9 GB of RAM) exceeded the initial project expectations.

- **Tor Bandwidth Authorities**

The lack of Tor bandwidth authority servers and the resulting inability of Tor Directory Authority servers to obtain valid bandwidth statistics and measurements, resulted in the allocation of Exit and Guard flags to all Tor relays. Although this did not affect the overall functionality of the network (section 5.4.2, 5.4.3), it is considered a system limitation as it deviates from the real Tor network functionality.

- **Simulation Interface**

The produced system does not employ a simulation interface that can be used to observe the status and operations of each network entity. The increased number of virtual machines makes the overall management and control of the virtual infrastructure complex. Netkit-ng supports the deployment of "headless" virtual machines, meaning that the specified virtual machine will not be allocated a terminal interface, but can rather be operated through an ssh tunnel. However, this action is considered inefficient as it can create unnecessary network overheads when combined with the Tor traffic.

- **Static Allocation of Cryptographic Keys**

The process of generating and distributing X.509 digital certificates was performed manually (section 5.2.2), since the current configuration requires the existence of Tor Directory Authority fingerprints in all torch configuration files. Consequently, a manual configuration of all Tor entities is necessary, if one decides to generate his/her own certificates.

CHAPTER FIVE

CONCLUSION AND FUTURE WORK

This chapter summarizes the results regarding each research objective and provides a concise, justifiable and evidence-based answer to the research question. It also discusses future work directions.

5.1 Research Outputs

Each research objective carried its own importance towards addressing a significant part of the research question, as follows:

1a. To develop an isolated network infrastructure

The first part of this objective was accomplished through practical experimentation and the utilisation of Netkit-ng network simulation environment. The followed detailed and well documented development process contributed significantly to the achievement of the research outcomes. The initial design, development and deployment of a virtual Local Area Network, followed by its expansion into a Wide Area Network produced an adequate isolated network infrastructure. Finally, the integration and configuration of a Domain Name Server fulfilled the overall expectations of the objective and delivered a virtual network infrastructure that was used as the basis for the subsequent research objectives.

1b. To integrate Tor in the produced isolated network infrastructure

The second part of this objective was also achieved through practical experimentation which relied heavily on the outcomes of the previous objective. The produced virtual network infrastructure was used as a foundation for the integration of Tor. Significant emphasis was given to the configuration of Tor Directory Authorities, relays and clients in accordance with the knowledge gained from the conducted literature review. The relevant research activity concluded with the delivery of a concise, effic ient and isolated Tor experimental network.

2. To analyze and understand in depth the Tor capabilities regarding its implementation, network architecture, cryptographic controls and design.

This objective was achieved through a comprehensive and thorough analysis of the literature. Due to the substantial breadth of the relevant knowledge domain, the research focuses on the areas of network

architecture, cryptographic controls and system design. The literature review began with a background research regarding the original Tor project (mid-1990s) and proceeded to the examination of the 2nd generation Onion Router (2004). An emphasis was given in the design advances and network architecture updates, surrounding the themes of Perfect Forward Secrecy, Tor proxy software and TCP streaming.

The literature review continued by looking at the functionality, capabilities and means of digital communications between (i) Tor Directory Authorities, (ii) entry, middle and exit relays, and (iii) Tor clients. Significant emphasis was also given to the creation and establishment of Tor circuits and the encryption capabilities and algorithms that operate within the Tor network. The literature concluded with an overview of the current state of Tor.

3. To critically evaluate the performance of the developed isolated Tor experimental network and define its limitations

The third and last objective of the research was fulfilled through detailed analysis and intensive experimentation. A set of network evaluation criteria, consisting of routing paths, open port, running services, network portability and hardware limitations, were established and defined prior to the development phase. These evaluation criteria were used for the full duration of the research to assess each network version and to determine the deficiencies that needed to be addressed in subsequent versions. Wherever necessary, additional experiments were conducted to augment the validity, reliability and repeatability of the system and provide a comprehensive view of the system limitations.

5.2 Conclusion

The accomplishment of the research objectives answered the research question:

To what extent can the concepts and principles of Tor help in minimizing privacy concerns in online interactions?

This research activity led to the development of an isolated experimental network that incorporates the fundamental principles and core capabilities offered by Tor. The developed system provides the means for the creation and establishment of Tor circuits that can be used by Tor clients to access webserver content anonymously. The system also includes its own Tor Directory Authorities that receive network statics from Tor relays, perform a voting procedure and publish a new consensus document every fifteen minutes. In

addition, all network communications between Tor entities are encrypted using elliptic curve cryptography and the data streams are transmitted over the TLS protocol. Finally, the system allows for the management and control of Tor entities through the Control Port interface.

Aided by the light-weighted, open source nature of the selected simulation environment (Netkit-ng) and the well documented configuration practices, the produced Tor experimental network can also be used as a method of teaching in the context of security architecture, cryptography, network protocols, secure communications and digital forensics.

5.3 Future work

The outputs of this research activity contributed to the creation of a solid research foundation that could be used as starting point for future research directions. The remaining of this section presents and discusses future research recommendations, as follows:

5.3.1 Dynamic Network Scalability

The development of a software-based controller could further increase the flexibility and scalability of the system. Software features could include the dynamic configuration of Tor entities, the creation and segregation of additional subnets and the automatic deployment of firewalls. Figure 7 illustrates a UML structural model diagram of the recommended software enhancements.

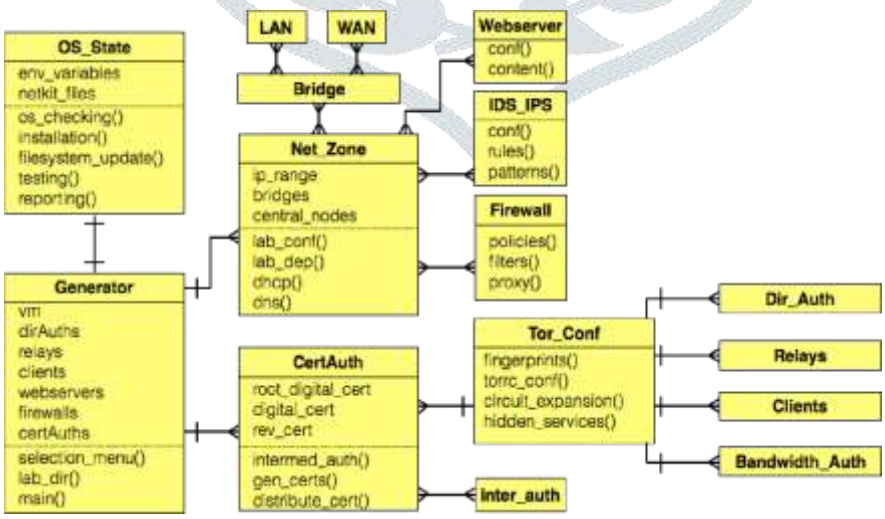


Figure 5.1 UML structural model diagram for a dynamic Tor network

5.3.2. Hidden Services

A hidden service is a special Tor protocol that allows Tor clients to offer a variety of services such as email servers, web publishing and instant messaging within the boundaries of the Tor network. In addition, the Tor project with the collaboration of the Debian team published in 2013 a new, open source operating system, named Tails (TorProject, 2013) that supports the use of messaging applications such as Google Talk, IRC and Bonjour, utilizing the hidden service protocol. Future research can focus either on the deployment and analysis of Tor hidden services on virtual Tor clients or the development of messaging application that utilise the capabilities of Tor hidden services.

5.3.3 System Interface

The increased number of virtual machines makes the overall management and control of the developed virtual infrastructure complex. Future research activity could be developing a software module that will collect statistics, measurements and log files from all virtual machines and summarize their status in a graphical interface.

5.3.4 Network Zoning

The produced system does not take into account the network latency that might occur during real world network communication. To cater for this, the system could be divided into different network zones, having their own central nodes. The deployment of firewalls in each network zone and the deliberate creation of bandwidth bottlenecks could be used to introduce substantial network latency, able to represent a more realistic network infrastructure.

5.3.5 Certificate Authorities

The possible deployment of master and intermediate Certificate Authorities within the Tor experimental network can provide another layer of security to the overall infrastructure and enable the use of https from Tor clients. As a result, Tor clients will have the capability to maintain end-to-end encryption, since the traffic exiting the Tor network will remain encrypted.

APPENDICES

Appendix A Consensus Example

```
1.  network-status-version 3
2.  vote-status consensus
3.  consensus-method 25
4.  valid-after 2017-06-17 02:00:00
5.  fresh-until 2017-06-17 03:00:00
6.  valid-until 2017-06-17 05:00:00
7.  voting-delay 300 300
8.  client-versions
    0.2.4.27,0.2.4.28,0.2.4.29,0.2.5.12,0.2.5.13,0.2.5.14,0.2.6.11,0.2.
    6.12,0.2.7.6,0.2.7.7,0.2.7.8,0.2.8.9,0.2.8.10,0.2.8.11,0.2.8.12,0.2
    .8.13,0.2.8.14,0.2.9.9,0.2.9.10,0.2.9.11,0.3.0.2-alpha,0.3.0.3alpha,0.3.0.4-
    rc,0.3.0.5-rc,0.3.0.6,0.3.0.7,0.3.0.8,0.3.1.1alpha,0.3.1.2-alpha,0.3.1.3-
    alpha
9.  server-versions
    0.2.4.27,0.2.4.28,0.2.4.29,0.2.5.12,0.2.5.13,0.2.5.14,0.2.6.11,0.2.
    6.12,0.2.7.6,0.2.7.7,0.2.7.8,0.2.8.9,0.2.8.10,0.2.8.11,0.2.8.12,0.2
    .8.13,0.2.8.14,0.2.9.9,0.2.9.10,0.2.9.11,0.3.0.7,0.3.0.8,0.3.1.2alpha,0.3.1.3
    -alpha
10. known-flags Authority Bad Exit Exit Fast Guard HSDir NoEdConsensus Running
    Stable V2Dir Valid
11. recommended-client-protocols Cons=1-2 Desc=1-2 DirCache=1 HSDir=1 HSIntro=3
    HSRend=1 Link=4 LinkAuth=1 Microdesc=1-2 Relay=2
12. recommended-relay-protocols Cons=1-2 Desc=1-2 DirCache=1 HSDir=1 HSIntro=3
    HSRend=1 Link=4 LinkAuth=1 Microdesc=1-2 Relay=2
13. required-client-protocols Cons=1-2 Desc=1-2 DirCache=1 HSDir=1 HSIntro=3
    HSRend=1 Link=4 LinkAuth=1 Microdesc=1-2 Relay=2
14. required-relay-protocols Cons=1 Desc=1 DirCache=1 HSDir=1 HSIntro=3 HSRend=1
    Link=3-4 LinkAuth=1 Microdesc=1 Relay=1-2
15. params CircuitPriorityHalflifeMsec=30000 NumDirectoryGuards=3
    NumEntryGuards=1 NumNTorsPerTAP=100 Support022HiddenServices=0
    UseNTorHandshake=1 UseOptimisticData=1 bwauthpid=1 cbttestfreq=10
    pb_disablepct=0 usecreatefast=0
16. shared-rand-previous-value 8 ot0icPHUE9JyVZDhYJT+Kf6mNrN+BLXjcbxbqvnpeBI=
17. shared-rand-current-value 8
    SBn5MkltbK+Kmv2fGqAMPvw/bTVPCvy91Cujd2fHZ2Q=
18. dir-source dannenberg 0232AF901C31A04EE9848595AF9BB7620D4C5B2E
    dannenberg.torauth.de 193.23.244.244 80 443
19. contact Andreas Lehner
20. vote-digest F0F0268B314581443873B630265B2831F2C4BA28
21. dir-source tor26 14C131DFC5C6F93646BE72FA1401C02A8DF2E8B4
    86.59.21.38 86.59.21.38 80 443
22. contact Peter Palfrader
23. vote-digest 111CD2BCDF9BBE6F46731E24B6FE0A07B1BE80E2
24. dir-source longclaw 23D15D965BC35114467363C165C4F724B64B4F66
    199.254.238.53 199.254.238.53 80 443
25. contact Riseup Networks <collective at riseup dot net> -
    1nNzekuHGGzBYRzyjfjFEfeisNvxkn4RT
26. vote-digest 96C5D4FA3AD0622CCEB97542E2DE66B3DB81DAFB
27. dir-source maatuska 49015F787433103580E3B66A1707A00E60F2D15B
    171.25.193.9 171.25.193.9 443 80
28. contact 4096R/1E8BF34923291265 Linus Nordberg <linus@nordberg.se>
29. vote-digest 9EED4BB69522E2E819BBECD28AF0FB30E780BEC1
30. dir-source moria1 D586D18309DED4CD6D57C18FDB97EFA96D330566
    128.31.0.34 128.31.0.34 9131 9101
31. contact 1024D/28988BF5 arma mit edu
32. vote-digest 3C6923EBF5B0C3CC7B1C3037202537096245F57E
33. dir-source dizum E8A9C45EDE6D711294FADF8E7951F4DE6CA56B58
    194.109.206.212 194.109.206.212 80 443
34. contact 1024R/8D56913D Alex de Joode <adejoode@sabotage.org>
35. vote-digest 89E6FD95EFAC0718FDC990D7FF424E6FAD5A7DBD
36. dir-source gabelmoo ED03BB616EB2F60BEC80151114BB25CEF515B226
    131.188.40.189 131.188.40.189 80 443
37. contact 4096R/261C5FBE77285F88FB0C343266C8C2D7C5AA446D Sebastian Hahn
    <tor@sebastianhahn.net> - 12NbRAjAG5U3LLWETSF7fSTcdaz32Mu5CN
38. vote-digest EE1A0BF77B6764BC6CBE15B4049C08BD801A4CF6
39. dir-source Faravahar EFCBE720AB3A82B99F9E953CD5BF50F7EEFC7B97
```

```
    154.35.175.225 154.35.175.225 80 443
40. contact 0x0B47D56D Sina Rabbani (inf0) <sina redteam net>
41. vote-digest FC8389E36A9E4E06168B5E9EAAEB9A043ABAD515
42. r seele AAoQ1DAR6kkoo19hBAX5K0QztNw TbvUJlQyYJahdnJK3SmSRR3DG0g
    2017-06-16 09:11:31 67.161.31.147 9001 0
43. s Running Stable V2Dir Valid
44. v Tor 0.2.9.10
45. pr Cons=1-2 Desc=1-2 DirCache=1 HSDir=1 HSIntro=3 HSRend=1-2 Link=1-4
    LinkAuth=1 Microdesc=1-2 Relay=1-2
46. w Bandwidth=29
47. p reject 1-65535
48. r gfythfh AA3G2vHNskh5QB5Un5afxywdaQ0 tjeH1dDu00VCTuF4DjMhF+PcgCg
    2017-06-16 23:16:53 176.31.126.144 10927 19219
49. s Fast HSDir Running Stable V2Dir Valid
50. v Tor 0.2.7.6
51. pr Cons=1-2 Desc=1-2 DirCache=1 HSDir=1 HSIntro=3 HSRend=1 Link=1-4 LinkAuth=1
    Microdesc=1-2 Relay=1-2
52. w Bandwidth=945
53. p reject 1-65535
54. r CalyxInstitute14 ABG9JIWtRdmE7EFZyI/AZuXjMA4
    2q3KLdpi1/4YE6kQO7PMvJUOflc 2017-06-16 13:19:02 162.247.72.201 443 80
55. s Exit Fast Guard HSDir Running Stable V2Dir Valid
56. v Tor 0.3.0.7
57. pr Cons=1-2 Desc=1-2 DirCache=1 HSDir=1-2 HSIntro=3-4 HSRend=1-2 Link=1-4
    LinkAuth=1,3 Microdesc=1-2 Relay=1-2
58. w Bandwidth=10100
59. p accept 20-23,43,53,79-
    81,88,110,143,194,220,389,443,464,531,543544,554,563,636,706,749,873,902-
    904,981,989-
    995,1194,1220,1293,1500,1533,1677,1723,1755,1863,2082-2083,2086-
    2087,2095-2096,2102-2104,3128,3389,3690,4321,4643,5050,5190,5222-
    5223,5228,5900,6660-6669,6679,6697,8000,8008,8074,8080,8087-
    8088,8332-8333,8443,8888,9418,9999-
    10000,11371,12350,19294,19638,23456,33033,64738
60. r Neldoreth ABUk3UA9cp8I9+XXeBPvEnVs+o0 Ucsx2DFMSNG/F+maBw02fvOwlHs
    2017-06-16 09:58:18 185.13.39.197 443 80
61. s Fast Guard HSDir Running Stable V2Dir Valid
62. v Tor 0.2.9.11
63. pr Cons=1-2 Desc=1-2 DirCache=1 HSDir=1 HSIntro=3 HSRend=1-2 Link=1-4
    LinkAuth=1 Microdesc=1-2 Relay=1-2
64. w Bandwidth=8120
65. p reject 1-65535
66. r MansikkaMaki04 ABpoc34PraISaJvGDXJn/1XrZyI iDh3K/LuJ3Ua27FATo4lnTIeK5E 2017-
    06-16 13:13:58 35.185.111.102 9001 9030
67. s Fast HSDir Running Stable V2Dir Valid
    ……
    ……
    ……
```

Appendix B Version 2: WAN Configuration Files

lab.conf

```
1: LAB_DESCRIPTION="WAN"
2: LAB_VERSION="2"
3: LAB_AUTHOR="Abdullahi Modibbo" 4:
LAB_EMAIL="modibbo04@gmail.com" 5:
6: gw1[0]=A
7: pc1[0]=A
8: pc2[0]=A 9:
10: gw2[0]=B
11: pc3[0]=B
12: pc4[0]=B 13:
14: gw1[1]=I1 15: internet[0]=I1
16:
17: gw2[1]=I2 18: internet[1]=I2
19:
20: dirauth1[0]=I3
```

```
21: internet[2]=I3 22:
23: dirauth2[0]=I4
24: internet[3]=I4 25:
26: dns[0]=I5
27: internet[4]=I5
```

pc1.startup

```
1: ifconfig eth0 192.168.0.101/24
2: route add default gw 192.168.0.254
```

pc2.startup

```
1: ifconfig eth0 192.168.0.102/24
2: route add default gw 192.168.0.254
```

pc3.startup

```
1: ifconfig eth0 193.168.0.101/24
2: route add default gw 193.168.0.254
```

pc4.startup

```
1: ifconfig eth0 193.168.0.102/24
2: route add default gw 193.168.0.254
```

gw1.startup

```
1: ifconfig eth0 192.168.0.254/24
2: ifconfig eth1 10.0.0.1/16
3: route add default gw 10.0.0.254
```

gw2.startup

```
1: ifconfig eth0 193.168.0.254/24
2: ifconfig eth1 10.1.0.1/16
3: route add default gw 10.1.0.254
```

dirauth1.startup

```
1: ifconfig eth0 10.2.0.1/16
2: route add default gw 10.2.0.254
```

dirauth2.startup 1: ifconfig eth0

```
10.3.0.1/16
2: route add default gw 10.3.0.254
```

dns.startup

```
1: ifconfig eth0 10.4.0.230/16
2: route add default gw 10.4.0.254
```

internet.startup 1: ifconfig eth0

```
10.0.0.254/16
2: ifconfig eth1 10.1.0.254/16
3: ifconfig eth2 10.2.0.254/16
4: ifconfig eth3 10.3.0.254/16
5: ifconfig eth4 10.4.0.254/16 6:
7: route add -net 192.168.0.0/24 gw 10.0.0.1
8: route add -net 193.168.0.0/24 gw 10.1.0.1
9: route add -net 10.2.0.0/16 gw 10.2.0.1
10: route add -net 10.3.0.0/16 gw 10.3.0.1
11: route add -net 10.4.0.0/16 gw 10.4.0.230
```

Appendix C Additional Netkit-ng Services

1. console-screen.sh
2. hwclock.sh
3. #mountfs-bootclean.sh
4. #mountnfs.sh
5. #setkey
6. bootmisc.sh
7. checkfs.sh
8. checkroot-boot.sh
9. freeradius
10. killprocs
11. kmod
12. mountall-bootclean.sh
13. mountall.sh
14. mountdevsubfs.sh
15. mountkernfs.sh
16. mtab.sh
17. netkit-mount-modules-dir
18. netkit-phase1
19. netkit-phase2
20. netkit-welcome
21. networking
22. pdns-recursor
23. pppd-dns 24. guagga
25. rc.local
26. rdnssd
27. screen-cleanup
28. sendsigs
29. udev-mtab
30. umountfs
31. umountndfs.sh
32. umountroot
33. x12tpd

Appendix D Version 3: WAN with DNS Configuration Files

```
shared.startup 1: #get machine
name
2: machine_name=$(echo $(uname -n))  3:
4: if [ "$machine_name" = "dns" ]; then
5:          echo "Initializing dns specific
            configuration"

6:          ip addr add 10.4.0.230/16 dev eth0

7:          ip link set up dev eth0

8:          ip route add default via 10.4.0.254

9:          echo "Initializing DNS Server"

10:         /etc/init.d/bind9 start

11:         echo "DNS SERVER STARTED"

12:         echo "Checking for open ports"

13:         netstat -tulpn | grep :53

14: elif [ "$machine_name" = "internet" ]; then
15:         ifconfig eth0 10.0.0.254/16

16:         ifconfig eth1 10.1.0.254/16

17:         ifconfig eth2 10.2.0.254/16

18:         ifconfig eth3 10.3.0.254/16

19:         ifconfig eth4 10.4.0.254/16

20:         ifconfig eth5 10.5.0.254/16

21:         ifconfig eth6 10.6.0.254/16

22:         ifconfig eth7 10.7.0.254/16

23:         ifconfig eth8 10.8.0.254/16

24:         ifconfig eth9 10.9.0.254/16

25:         ifconfig eth10 10.10.0.254/16

26:         ifconfig eth11 10.11.0.254/16

27:         ifconfig eth12 10.12.0.254/16

28:         ifconfig eth13 10.13.0.254/16

29:         ifconfig eth14 10.14.0.254/16

30:         ifconfig eth15 10.15.0.254/16

31:         ifconfig eth16 10.16.0.254/16

32:         ifconfig eth17 10.17.0.254/16

33:         ifconfig eth18 10.18.0.254/16
```

```
34:         ifconfig eth19 10.19.0.254/16

35:         ifconfig eth20 10.20.0.254/16

36:         ifconfig eth21 10.21.0.254/16

37:         ifconfig eth22 10.22.0.254/16

38:         ifconfig eth23 10.23.0.254/16

39:         ifconfig eth24 10.24.0.254/16

40:         ifconfig eth25 10.25.0.254/16

41:         ifconfig eth26 10.26.0.254/16

42:         ifconfig eth27 10.27.0.254/16

43:         ifconfig eth28 10.28.0.254/16

44:         ifconfig eth29 10.29.0.254/16

45:         ifconfig eth30 10.30.0.254/16

46:         ifconfig eth31 10.31.0.254/16

47:         ifconfig eth32 10.32.0.254/16

48:         ifconfig eth33 10.33.0.254/16

49:         ifconfig eth34 10.34.0.254/16

50:         route add -net 192.168.0.0/24 gw
            10.0.0.1

51:         route add -net 193.168.0.0/24 gw
            10.1.0.1

52:         route add -net 10.2.0.0/16 gw 10.2.0.1

53:         route add -net 10.3.0.0/16 gw 10.3.0.1

54:         route add -net 10.5.0.0/16 gw 10.5.0.1

55:         route add -net 10.6.0.0/16 gw 10.6.0.1

56:         route add -net 10.7.0.0/16 gw 10.7.0.1

57:         route add -net 10.8.0.0/16 gw 10.8.0.1

58:         route add -net 10.9.0.0/16 gw 10.9.0.1

59:         route add -net 10.10.0.0/16 gw
            10.10.0.1

60:         route add -net 10.11.0.0/16 gw
            10.11.0.1

61:         route add -net 10.12.0.0/16 gw
            10.12.0.1

62:         route add -net 10.13.0.0/16 gw
            10.13.0.1

63:         route add -net 10.14.0.0/16 gw
            10.14.0.1
```

```
64:         route add -net 10.15.0.0/16 gw
            10.15.0.1

65:         route add -net 10.16.0.0/16 gw
            10.16.0.1

66:         route add -net 10.17.0.0/16 gw
            10.17.0.1

67:         route add -net 10.18.0.0/16 gw
            10.18.0.1

68:         route add -net 10.19.0.0/16 gw
            10.19.0.1

69:         route add -net 10.20.0.0/16 gw
            10.20.0.1

70:         route add -net 10.21.0.0/16 gw
            10.21.0.1

71:         route add -net 10.22.0.0/16 gw
            10.22.0.1

72:         route add -net 10.23.0.0/16 gw
            10.23.0.1

73:         route add -net 10.24.0.0/16 gw
            10.24.0.1

74:         route add -net 10.25.0.0/16 gw
            10.25.0.1

75:         route add -net 10.26.0.0/16 gw
            10.26.0.1

76:         route add -net 10.27.0.0/16 gw
            10.27.0.1

77:         route add -net 10.28.0.0/16 gw
            10.28.0.1

78:         route add -net 10.29.0.0/16 gw
            10.29.0.1

79:         route add -net 10.30.0.0/16 gw
            10.30.0.1

80:         route add -net 10.31.0.0/16 gw
            10.31.0.1

81:         route add -net 10.32.0.0/16 gw
            10.32.0.1

82:         route add -net 10.33.0.0/16 gw
            10.33.0.1

83:         route add -net 10.34.0.0/16 gw
            10.34.0.1
```

```
84:  elif [ "$machine_name" = "gw1" ]; then
85:          ifconfig eth0 192.168.0.254/24
86:          ifconfig eth1 10.0.0.1/16
87:          route add default gw 10.0.0.254
88:  elif [ "$machine_name" = "gw2" ]; then
89:          ifconfig eth0 193.168.0.254/24
90:          ifconfig eth1 10.1.0.1/16
91:          route add default gw 10.1.0.254
92:  elif [ "$machine_name" = "dirauth1" ]; then
93:          ifconfig eth0 10.2.0.1/16
94:          route add default gw 10.2.0.254
95:  elif [ "$machine_name" = "dirauth2" ]; then
96:          ifconfig eth0 10.3.0.1/16
97:          route add default gw 10.3.0.254
98:  elif [ "$machine_name" = "pc1" ]; then
99:          ifconfig eth0 192.168.0.101/24
100:         route add default gw 192.168.0.254
101: elif [ "$machine_name" = "pc2" ]; then
102:         ifconfig eth0 192.168.0.102/24
103:         route add default gw 192.168.0.254
104: elif [ "$machine_name" = "pc3" ]; then
105:         ifconfig eth0 193.168.0.101/24
106:         route add default gw 193.168.0.254
107: elif [ "$machine_name" = "pc4" ]; then
108:         ifconfig eth0 193.168.0.102/24
109:  route add default gw 193.168.0.254
110: elif [ "$machine_name" = "relay1" ]; then
111:  ifconfig eth0 10.5.0.1/16
112:  route add default gw 10.5.0.254
113: elif [ "$machine_name" = "relay2" ]; then
114:  ifconfig eth0 10.6.0.1/16
115:  route add default gw 10.6.0.254
116: elif [ "$machine_name" = "relay3" ]; then
117:    ifconfig eth0 10.7.0.1/16
118:    route add default gw 10.7.0.254
```

```
elif [ "$machine_name" = "relay4" ]; then
   ifconfig eth0 10.8.0.1/16
   route add default gw 10.8.0.254
elif [ "$machine_name" = "relay5" ]; then
   ifconfig eth0 10.9.0.1/16
   route add default gw 10.9.0.254
elif [ "$machine_name" = "relay6" ]; then
   ifconfig eth0 10.10.0.1/16
   route add default gw 10.10.0.254
elif [ "$machine_name" = "relay7" ]; then
   ifconfig eth0 10.11.0.1/16
   route add default gw 10.11.0.254
elif [ "$machine_name" = "relay8" ]; then
   ifconfig eth0 10.12.0.1/16
   route add default gw 10.12.0.254
elif [ "$machine_name" = "relay9" ]; then
   ifconfig eth0 10.13.0.1/16
   route add default gw 10.13.0.254
elif [ "$machine_name" = "relay10" ]; then
   ifconfig eth0 10.14.0.1/16
   route add default gw 10.14.0.254
elif [ "$machine_name" = "relay11" ]; then
   ifconfig eth0 10.15.0.1/16
   route add default gw 10.15.0.254
elif [ "$machine_name" = "relay12" ]; then
   ifconfig eth0 10.16.0.1/16
   route add default gw 10.16.0.254
elif [ "$machine_name" = "relay13" ]; then
   ifconfig eth0 10.17.0.1/16
   route add default gw 10.17.0.254
elif [ "$machine_name" = "relay14" ]; then
   ifconfig eth0 10.18.0.1/16
   route add default gw 10.18.0.254
elif [ "$machine_name" = "relay15" ]; then
   ifconfig eth0 10.19.0.1/16
   route add default gw 10.19.0.254
elif [ "$machine_name" = "relay16" ]; then
   ifconfig eth0 10.20.0.1/16
   route add default gw 10.20.0.254
elif [ "$machine_name" = "relay17" ]; then
   ifconfig eth0 10.21.0.1/16
   route add default gw 10.21.0.254
elif [ "$machine_name" = "relay18" ]; then
   ifconfig eth0 10.22.0.1/16
   route add default gw 10.22.0.254
elif [ "$machine_name" = "relay19" ]; then
   ifconfig eth0 10.23.0.1/16
   route add default gw 10.23.0.254
elif [ "$machine_name" = "relay20" ]; then
   ifconfig eth0 10.24.0.1/16
   route add default gw 10.24.0.254
elif [ "$machine_name" = "relay21" ]; then
   ifconfig eth0 10.25.0.1/16
   route add default gw 10.25.0.254
elif [ "$machine_name" = "relay22" ]; then
   ifconfig eth0 10.26.0.1/16
   route add default gw 10.26.0.254
elif [ "$machine_name" = "relay23" ]; then
   ifconfig eth0 10.27.0.1/16
   route add default gw 10.27.0.254
elif [ "$machine_name" = "relay24" ]; then
   ifconfig eth0 10.28.0.1/16
   route add default gw 10.28.0.254
elif [ "$machine_name" = "relay25" ]; then
   ifconfig eth0 10.29.0.1/16
   route add default gw 10.29.0.254
elif [ "$machine_name" = "relay26" ]; then
   ifconfig eth0 10.30.0.1/16
   route add default gw 10.30.0.254
```

```
188: elif [ "$machine_name" = "relay27" ]; then
189:    ifconfig eth0 10.31.0.1/16
190:    route add default gw 10.31.0.254
191: elif [ "$machine_name" = "relay28" ]; then
192:    ifconfig eth0 10.32.0.1/16
193:    route add default gw 10.32.0.254
194: elif [ "$machine_name" = "relay29" ]; then
195:    ifconfig eth0 10.33.0.1/16
196:    route add default gw 10.33.0.254
197: elif [ "$machine_name" = "relay30" ]; then
198:    ifconfig eth0 10.34.0.1/16
199:    route add default gw 10.34.0.254
200: fi
201: echo "Configuring /etc/resolv.conf file"
202: echo "search torlab.test." > /etc/resolv.conf
203: echo "nameserver 10.4.0.230" >> /etc/resolv.conf
204: echo "Restarting networking service"
205: service networking stop 206: service networking
start
207: echo "Testing DNS server"
208: dig torlab.test
209: host dns.torlab.test
```

lab.conf

```
1: LAB_DESCRIPTION="TOR NETWORK SIMULATOR"
2: LAB_VERSION="3"
3: LAB_AUTHOR="Abdullahi Modibbo" 4:
LAB_EMAIL="modibbo04@gmail.com" 5:
6: #subnets
7: gw1[0]=A
8: pc1[0]=A
9: pc2[0]=A 10:
11: gw2[0]=B
12: pc3[0]=B
13: pc4[0]=B 14:
15: #links subnets to internet
16: gw1[1]=I1 17: internet[0]=I1
18:
19: gw2[1]=I2 20: internet[1]=I2
21:
22: dirauth1[0]=I3
23: internet[2]=I3 24:
25: dirauth2[0]=I4
26: internet[3]=I4 27:
28: dns[0]=I5 29: internet[4]=I5
30:
31: relay1[0]=I6 32:
internet[5]=I6 33:
34: relay2[0]=I7 35:
internet[6]=I7 36:
37: relay3[0]=I8 38:
internet[7]=I8 39:
40: relay4[0]=I9 41:
internet[8]=I9 42:
43: relay5[0]=I10 44:
internet[9]=I10 45:
46: relay6[0]=I11 47:
internet[10]=I11 48:
49: relay7[0]=I12 50:
internet[11]=I12 51:
52: relay8[0]=I13 53:
internet[12]=I13 54:
55: relay9[0]=I14 56:
internet[13]=I14 57:
58: relay10[0]=I15 59:
internet[14]=I15 60:
61: relay11[0]=I16 62:
internet[15]=I16 63:
64: relay12[0]=I17 65:
internet[16]=I17 66:
```

```
67: relay13[0]=I18
68: internet[17]=I18
69
70: relay14[0]=I19 71:
internet[18]=I19 72:
73: relay15[0]=I20 74:
internet[19]=I20 75:
76: relay16[0]=I21 77:
internet[20]=I21 78:
79: relay17[0]=I22 80:
internet[21]=I22 81:
82: relay18[0]=I23 83:
internet[22]=I23 84:
85: relay19[0]=I24 86:
internet[23]=I24 87:
88: relay20[0]=I25 89: internet[24]=I25 90:
91: relay21[0]=I26 92:
internet[25]=I26 93:
94: relay22[0]=I27 95:
internet[26]=I27 96:
97: relay23[0]=I28 98:
internet[27]=I28 99:
100: relay24[0]=I29 101:
internet[28]=I29 102:
103: relay25[0]=I30 104:
internet[29]=I30 105:
106: relay26[0]=I31 107:
internet[30]=I31 108:
109: relay27[0]=I32 110:
internet[31]=I32 111:
112: relay28[0]=I33 113:
internet[32]=I33 114:
115: relay29[0]=I34 116:
internet[33]=I34 117:
118: relay30[0]=I35
119: internet[34]=I35
```

Appendix E DNS Configuration Files

named.conf.local *1: zone*

```
"torlab.test" {
2:      type master;
3:          file "/etc/bind/db.test.torlab";
4: };
5: zone "192.in-addr.arpa" {
6:          type master;
7:          file "/etc/bind/db.192";
8: };
9: zone "193.in-addr.arpa" {
10:          type master;
11:          file "/etc/bind/db.193";
12: };
13: zone "10.in-addr.arpa" {
14:          type master;
15:          file "/etc/bind/db.10";
16: };
```

named.conf.options

```
1: options {
2:  directory "/var/cache/bind";
3:  forwarders {
4:      8.8.8.8;
5:      8.8.4.4;
6:  };
7:  forward only;
```

```
8:  recursion yes;
9:  allow-query {any;};
10:  auth-nxdomain no;
11:  listen-on-v6 {any;};
12: };  db.test.torlab
1: $TTL       8h
2: @ IN SOA     torlab.test.    root.torlab.test. (
3:           1         ; serial

4:           8h          ; refresh

5:           2h          ; retry

6:           1w          ; expire

7:           0           ; negative cache
             ttl

8: )
9: ;
10: ; nameserver(s) 11: @ IN    NS
 dns.torlab.test.
12:
13: ; ip's pointing to host
14: dns          IN    A     10.4.0.230

15: dirauth1 IN   A     10.2.0.1

16: dirauth2 IN   A     10.3.0.1

17: pc1          IN    A 192.168.0.101

18: pc2          IN    A 192.168.0.102

19: pc3          IN    A 193.168.0.101

20: pc4          IN    A 193.168.0.102

21: relay1       IN    A     10.5.0.1

22: relay2       IN    A     10.6.0.1

23: relay3       IN    A     10.7.0.1

24: relay4       IN    A     10.8.0.1

25: relay5       IN    A     10.9.0.1

26: relay6       IN    A     10.10.0.1

27: relay7       IN    A     10.11.0.1

28: relay8       IN    A     10.12.0.1

29: relay9       IN    A     10.13.0.1

30: relay10      IN    A     10.14.0.1

31: relay11      IN    A     10.15.0.1

32: relay12      IN    A     10.16.0.1

33: relay13      IN    A     10.17.0.1

34: relay14      IN    A     10.18.0.1

35: relay15      IN    A     10.19.0.1
```

```
36: relay16       IN    A     10.20.0.1
37: relay17       IN    A     10.21.0.1
38: relay18       IN    A     10.22.0.1
39: relay19       IN    A     10.23.0.1
40: relay20       IN    A     10.24.0.1
41: relay21       IN    A     10.25.0.1
42: relay22       IN    A     10.26.0.1
43: relay23       IN    A     10.27.0.1
44: relay24       IN    A     10.28.0.1
45: relay25       IN    A     10.29.0.1
46: relay26       IN    A     10.30.0.1
47: relay27       IN    A     10.31.0.1
48: relay28       IN    A     10.32.0.1
49: relay29       IN    A     10.33.0.1
50: relay30       IN    A     10.34.0.1
51: ; gateways
52: gw1      IN    A     192.168.0.254
53:  IN      A     10.0.0.1
54:
55: gw2      IN    A     193.168.0.254
56:  IN      A     10.1.0.1
57: ; universal
58: internet IN    A     10.0.0.254
59:          IN    A     10.1.0.254
60:          IN    A     10.2.0.254
61:          IN    A     10.3.0.254
62:          IN    A     10.4.0.254
63:          IN    A     10.5.0.254
64:          IN    A     10.6.0.254
65:          IN    A     10.7.0.254
66:          IN    A     10.8.0.254
67:          IN    A     10.9.0.254
68:          IN    A     10.10.0.254
69:          IN    A     10.11.0.254
```

```
70:         IN    A     10.12.0.254

71:         IN    A     10.13.0.254

72:         IN    A     10.14.0.254

73:         IN    A     10.15.0.254

74:         IN    A     10.16.0.254

75:         IN    A     10.17.0.254

76:         IN    A     10.18.0.254

77:         IN    A     10.19.0.254

78:         IN    A     10.20.0.254

79:         IN    A     10.21.0.254

80:         IN    A     10.22.0.254

81:         IN    A     10.23.0.254

82:         IN    A     10.24.0.254

83:         IN    A     10.25.0.254

84:         IN    A     10.26.0.254

85:         IN    A     10.27.0.254

86:         IN    A     10.28.0.254

87:         IN    A     10.29.0.254

88:         IN    A     10.30.0.254

89:         IN    A     10.31.0.254

90:         IN    A     10.32.0.254

91:         IN    A     10.33.0.254

92:         IN    A     10.34.0.254
```

db.10 1: $TTL 8h
2: @ IN SOA torlab.test. root.torlab.test. (
3: 1 ; serial

4: 8h ; refresh

5: 2h ; retry

6: 1w ; expire

7: 0 ; negative cache
ttl

8:)

9: ;
10: ; dns's nameservers
11: @ IN NS dns.torlab.test.
12: ;
13: ; ip's pointing to host domains
14: ; network ID : "I1" ->>10.0.0.0/16
15: 1.0.0 IN PTR gw1.torlab.test.

```
16: 254.0.0   IN    PTR internet.torlab.test.
17: ; network ID : "I2" ->>10.1.0.0/16
18: 1.0.1     IN    PTR gw2.torlab.test.
19: 254.0.1   IN    PTR internet.torlab.test.
20: ; network ID : "I3" ->>10.2.0.0/16
21: 1.0.2     IN    PTR dirauth1.torlab.test. 22: 254.0.2     IN
 PTR internet.torlab.test.
23: ; network ID : "I4" ->>10.3.0.0/16
24: 1.0.3     IN    PTR dirauth2.torlab.test. 25: 254.0.3     IN
 PTR internet.torlab.test.
26: ; network ID : "I5" ->>10.4.0.0/16 27: 230.0.4     IN
 PTR dns.torlab.test.
28: 254.0.4   IN    PTR internet.torlab.test.
29: ; network ID : "I6" ->>10.5.0.0/16
30: 1.0.5     IN    PTR relay1.torlab.test.
31: 254.0.5   IN    PTR internet.torlab.test.
32: ; network ID : "I7" ->>10.6.0.0/16
33: 1.0.6     IN    PTR relay2.torlab.test.
34: 254.0.6   IN    PTR internet.torlab.test.
35: ; network ID : "I8" ->>10.7.0.0/16
36: 1.0.7     IN    PTR relay3.torlab.test.
37: 254.0.7   IN    PTR internet.torlab.test.
38: ; network ID : "I9" ->>10.8.0.0/16
39: 1.0.8     IN    PTR relay4.torlab.test.
40: 254.0.8   IN    PTR internet.torlab.test.
41: ; network ID : "I10" ->>10.9.0.0/16
42: 1.0.9     IN    PTR relay5.torlab.test.
43: 254.0.9   IN    PTR internet.torlab.test.
44: ; network ID : "I11" ->>10.10.0.0/16 45: 1.0.10     IN
 PTR relay6.torlab.test.
46: 254.0.10 IN     PTR internet.torlab.test.
47: ; network ID : "I12" ->>10.11.0.0/16 48: 1.0.11     IN
 PTR relay7.torlab.test.
49: 254.0.11 IN     PTR internet.torlab.test.
50: ; network ID : "I13" ->>10.12.0.0/16 51: 1.0.12     IN
 PTR relay8.torlab.test.
52: 254.0.12 IN     PTR internet.torlab.test.
53: ; network ID : "I14" ->>10.13.0.0/16
54: 1.0.13    IN    PTR relay9.torlab.test.
55: 254.0.13 IN     PTR internet.torlab.test.
56: ; network ID : "I15" ->>10.14.0.0/16 57: 1.0.14     IN
 PTR relay10.torlab.test.
58: 254.0.14 IN     PTR internet.torlab.test.
59: ; network ID : "I16" ->>10.15.0.0/16 60: 1.0.15     IN
 PTR relay11.torlab.test.
61: 254.0.15 IN     PTR internet.torlab.test.
62: ; network ID : "I17" ->>10.16.0.0/16 63: 1.0.16     IN
 PTR relay12.torlab.test.
64: 254.0.16 IN     PTR internet.torlab.test.
65: ; network ID : "I18" ->>10.17.0.0/16 66: 1.0.17     IN
 PTR relay13.torlab.test.
67: 254.0.17 IN     PTR internet.torlab.test.
68: ; network ID : "I19" ->>10.18.0.0/16 69: 1.0.18     IN
 PTR relay14.torlab.test.
70: 254.0.18 IN     PTR internet.torlab.test.
71: ; network ID : "I20" ->>10.19.0.0/16 72: 1.0.19     IN
 PTR relay15.torlab.test.
73: 254.0.19 IN     PTR internet.torlab.test.
74: ; network ID : "I21" ->>10.20.0.0/16 75: 1.0.20     IN
 PTR relay16.torlab.test.
76: 254.0.20 IN     PTR internet.torlab.test.
77: ; network ID : "I22" ->>10.21.0.0/16 78: 1.0.21     IN
 PTR relay17.torlab.test.
79: 254.0.21 IN     PTR internet.torlab.test.
80: ; network ID : "I23" ->>10.22.0.0/16 81: 1.0.22     IN
 PTR relay18.torlab.test.
82: 254.0.22 IN     PTR internet.torlab.test.
83: ; network ID : "I24" ->>10.23.0.0/16 84: 1.0.23     IN
 PTR relay19.torlab.test.
```

```
85: 254.0.23 IN     PTR internet.torlab.test.
86: ; network ID : "I25" ->>10.24.0.0/16 87: 1.0.24     IN
 PTR relay20.torlab.test.
88: 254.0.24 IN     PTR internet.torlab.test.
89: ; network ID : "I26" ->>10.25.0.0/16 90: 1.0.25     IN
 PTR relay21.torlab.test.
91: 254.0.25 IN     PTR internet.torlab.test.
92: ; network ID : "I27" ->>10.26.0.0/16 93: 1.0.26     IN
 PTR relay22.torlab.test.
94: 254.0.26 IN     PTR internet.torlab.test.
95: ; network ID : "I28" ->>10.27.0.0/16 96: 1.0.27     IN
 PTR relay23.torlab.test.
97: 254.0.27 IN     PTR internet.torlab.test.
98: ; network ID : "I29" ->>10.28.0.0/16 99: 1.0.28     IN
 PTR relay24.torlab.test.
100: 254.0.28       IN    PTR internet.torlab.test. 101: ; network
ID : "I30" ->>10.29.0.0/16
102: 1.0.29  IN PTR relay25.torlab.test.
103: 254.0.29       IN    PTR internet.torlab.test. 104: ; network
ID : "I31" ->>10.30.0.0/16
105: 1.0.30  IN PTR relay26.torlab.test.
106: 254.0.30       IN    PTR internet.torlab.test. 107: ; network
ID : "I32" ->>10.31.0.0/16
108: 1.0.31  IN PTR relay27.torlab.test.
109: 254.0.31       IN    PTR internet.torlab.test. 110: ; network
ID : "I33" ->>10.32.0.0/16
111: 1.0.32  IN PTR relay28.torlab.test.
112: 254.0.32       IN    PTR internet.torlab.test. 113: ; network
ID : "I34" ->>10.33.0.0/16
114: 1.0.33  IN PTR relay29.torlab.test. 115: 254.0.33      IN    PTR
internet.torlab.test. 116: ; network ID : "I35" ->>10.34.0.0/16
117: 1.0.34         IN    PTR relay30.torlab.test.
118: 254.0.34       IN    PTR internet.torlab.test.
 db.192 1: $TTL    8h
2: @ IN      SOA     torlab.test.    root.torlab.test. (
3:          1       ; serial

4:          8h        ; refresh

5:          2h        ; retry

6:          1w        ; expire

7:          0         ; negative cache
            ttl

8:          )

9: ;
10: ; dns's nameservers
11: @ IN      NS    dns.torlab.test.
12: ;
13: ; ip's pointing to host domains

14: ; network ID : "A" ->>192.168.0.0./24
15: 101.0.168     IN    PTR   pc1.torlab.test.

16: 102.0.168     IN    PTR   pc2.torlab.test.

17: 254.0.168     IN    PTR   gw1.torlab.test.

 db.193 1: $TTL    8h
2: @ IN      SOA     torlab.test.    root.torlab.test. (
3:          1        ; serial

4:          8h         ; refresh
```

```
5:          2h          ; retry

6:          1w          ; expire

7:          0           ; negative cache
            ttl

8:          )

9: ;
10: ; dns's nameservers
11: @ IN      NS    dns.torlab.test.
12: ;
13: ; ip's pointing to host domains
14: ; network ID : "B" ->>193.168.0.0./24
15: 101.0.168     IN    PTR   pc3.torlab.test.

16: 102.0.168     IN    PTR   pc4.torlab.test.

17: 254.0.168     IN    PTR   gw2.torlab.test.
```

Appendix F Version 3: WAN with DNS – Routing Paths

NO	VIRTUAL MACHINE	IP	NETWORK INTERFACE	DEFAULT GATEWAY
1.	gw1	192.168.0.254 10.0.0.1	eth0 eth1	- 10.0.0.254
2.	pc1	192.168.0.101	eth0	192.168.0.254
3.	pc2	192.168.0.102	eth0	192.168.0.254
4.	gw2	193.168.0.254 10.1.0.1	eth0 eth1	- 10.1.0.254
5.	pc3	192.168.0.101	eth0	193.168.0.254
6.	pc4	192.168.0.102	eth0	193.168.0.254
7.	dirauth1	10.2.0.1	eth0	10.2.0.254
8.	dirauth2	10.3.0.1	eth0	10.3.0.254
9.	dns	10.4.0.230	eth0	10.4.0.254
10.	relay1	10.5.0.1	eth0	10.5.0.254
11.	relay2	10.6.0.1	eth0	10.6.0.254
12	relay3	10.7.0.1	eth0	10.7.0.254
13.	relay4	10.8.0.1	eth0	10.8.0.254
14.	relay5	10.9.0.1	eth0	10.9.0.254
15.	relay6	10.10.0.1	eth0	10.10.0.254
16.	relay7	10.11.0.1	eth0	10.11.0.254
17.	relay8	10.12.0.1	eth0	10.12.0.254
18.	relay9	10.13.0.1	eth0	10.13.0.254
19.	relay10	10.14.0.1	eth0	10.14.0.254
20.	relay11	10.15.0.1	eth0	10.15.0.254
21.	relay12	10.16.0.1	eth0	10.16.0.254

22.	relay13	10.17.0.1	eth0	10.17.0.254
23.	relay14	10.18.0.1	eth0	10.18.0.254
24.	relay15	10.19.0.1	eth0	10.19.0.254
25.	relay16	10.20.0.1	eth0	10.20.0.254
26	relay17	10.21.0.1	eth0	10.21.0.254
27	relay18	10.22.0.1	eth0	10.22.0.254
28	relay19	10.23.0.1	eth0	10.23.0.254
29	relay20	10.24.0.1	eth0	10.24.0.254
30.	relay21	10.25.0.1	eth0	10.25.0.254
31.	relay22	10.26.0.1	eth0	10.26.0.254
32.	relay23	10.27.0.1	eth0	10.27.0.254
33.	relay24	10.28.0.1	eth0	10.28.0.254
34.	relay25	10.29.0.1	eth0	10.29.0.254
35.	relay26	10.30.0.1	eth0	10.30.0.254
36.	relay27	10.31.0.1	eth0	10.33.0.254
37.	relay28	10.32.0.1	eth0	10.32.0.254
38.	relay29	10.33.0.1	eth0	10.33.0.254
39.	relay30	10.34.0.1	eth0	10.34.0.254
40.	internet	10.0.0.254 10.1.0.254 10.2.0.254 10.3.0.254 10.4.0.254 10.5.0.254 10.6.0.254 10.7.0.254 10.8.0.254 10.9.0.254 10.10.0.254 10.11.0.254 10.12.0.254	eth0 eth1 eth2 eth3 eth4 eth5 eth6 eth7 eth8 eth9 eth10 eth11 eth12	10.0.0.1 10.1.0.1 10.2.0.1 10.3.0.1 10.4.0.230 10.5.0.1 10.6.0.1 10.7.0.1 10.8.0.1 10.9.0.1 10.10.0.1 10.11.0.1 10.12.0.1

40.	internet	10.13.0.254 10.14.0.254 10.15.0.254 10.16.0.254 10.17.0.254 10.18.0.254 10.19.0.254 10.20.0.254 10.21.0.254 10.22.0.254 10.23.0.254 10.24.0.254 10.25.0.254 10.26.0.254 10.27.0.254 10.28.0.254 10.29.0.254 10.30.0.254 10.31.0.254 10.32.0.254 10.33.0.254 10.34.0.254	eth13 eth14 eth15 eth16 eth17 eth18 eth19 eth20 eth21 eth22 eth23 eth24 eth25 eth26 eth27 eth28 eth29 eth30 eth31 eth32 eth33 eth34	10.13.0.1 10.14.0.1 10.15.0.1 10.16.0.1 10.17.0.1 10.18.0.1 10.19.0.1 10.20.0.1 10.21.0.1 10.22.0.1 10.23.0.1 10.24.0.1 10.25.0.1 10.26.0.1 10.27.0.1 10.28.0.1 10.29.0.1 10.30.0.1 10.31.0.1 10.32.0.1 10.33.0.1 10.34.0.1

Appendix G Version 3: WAN with DNS – Speed

	VM	VM BOOT UP TIME (min: sec. millisec)	NETWORK BOOT TIME (min: sec. millisec)
1.	dns	00:18 .05	00:18 .05
2.	internet	00:49.5 0	01:07.55
3.	dirauth2	00:09.30	
4.	gw1	00:09.30	
5.	gw2	00:09.30	01:16.85
6.	dirauth1	00:09.30	
7.	relay1	00:09.30	
8.	relay10	00:07.36	
9.	relay11	00:07.36	
10.	relay12	00:07.36	01:24.21
11.	relay13	00:07.36	
12.	relay14	00:07.36	
13.	relay15	00:10.47	
14.	relay16	00:10.47	
15.	relay17	00:10.47	01:34.68
16.	relay18	00:10.47	
17.	relay19	00:10.47	
18.	relay2	00:09.33	
19.	relay20	00:09.33	
20.	relay21	00:09.33	01:44.01
21.	relay22	00:09.33	
22.	relay23	00:09.33	
23.	relay24	00:09.47	01:53.48
24.	relay25	00:09.47	

25.	relay26	00:09.47	
26.	relay27	00:09.47	01:53.48
27.	relay28	00:09.47	
28.	relay29	00:10.20	
29.	relay3	00:10.20	
30.	relay30	00:10.20	02:0 3.68
31.	relay4	00:10.20	
32.	relay5	00:10.20	
33.	relay6	00:12.28	
34.	relay7	00:12.28	
35.	relay8	00:12.28	02:15.96
36.	relay9	00:12.28	
37.	pc1	00:12.28	
38.	pc2	00:22.01	
39.	pc3	00:22.01	02:37.97
40.	pc4	00:22.01	

Appendix H Version 3: WAN with DNS – Hardware Limitations

	VIRTUAL MACHINE / SYSTEM	MEMORY ALLOC. ON RUN TIME	STORAGE ALLOC. ON RUN TIME	CPU USAGE ON RUN TIME
1.	dirauth1	264156 KiB ≈ 33.8 MB	700 KB	0.7% – 1.3%
2.	dirauth2	264156 KiB ≈ 33.8 MB	704 KB	0.7% – 1.3%
3.	dns	264156 KiB ≈ 33.8 MB	792 KB	0.7% – 1.3%
4.	gw1	264156 KiB ≈ 33.8 MB	700 KB	0.7% – 1.3%
5.	gw2	264156 KiB ≈ 33.8 MB	700 KB	0.7% – 1.3%
6.	internet	264156 KiB ≈ 33.8 MB	760 KB	0.7% – 1.3%
7.	pc1	264156 KiB ≈ 33.8 MB	684 KB	0.7% – 1.3%
8.	pc2	264156 KiB ≈ 33.8 MB	696 KB	0.7% – 1.3%
9.	pc3	264156 KiB ≈ 33.8 MB	704 KB	0.7% – 1.3%
10.	pc4	264156 KiB ≈ 33.8 MB	692 KB	0.7% – 1.3%
11.	relay1	264156 KiB ≈ 33.8 MB	696 KB	0.7% – 1.3%
12.	relay2	264156 KiB ≈ 33.8 MB	700 KB	0.7% – 1.3%
13.	relay3	264156 KiB ≈ 33.8 MB	704 KB	0.7% – 1.3%
14.	relay4	264156 KiB ≈ 33.8 MB	700 KB	0.7% – 1.3%

15.	relay5	264156 KiB ≈ 33.8 MB	700 KB	0.7% – 1.3%
16.	relay6	264156 KiB ≈ 33.8 MB	704 KB	0.7% – 1.3%
17.	relay7	264156 KiB ≈ 33.8 MB	700 KB	0.7% – 1.3%
18.	relay8	264156 KiB ≈ 33.8 MB	700 KB	0.7% – 1.3%
19.	relay9	264156 KiB ≈ 33.8 MB	704 KB	0.7% – 1.3%

20.	relay10	264156 KiB ≈ 33.8 MB	696 KB	0.7% – 1.3%
21.	relay11	264156 KiB ≈ 33.8 MB	696 KB	0.7% – 1.3%
22.	relay12	264156 KiB ≈ 33.8 MB	704 KB	0.7% – 1.3%
23.	relay13	264156 KiB ≈ 33.8 MB	700 KB	0.7% – 1.3%
24.	relay14	264156 KiB ≈ 33.8 MB	700 KB	0.7% – 1.3%
25.	relay15	264156 KiB ≈ 33.8 MB	704 KB	0.7% – 1.3%
26.	relay16	264156 KiB ≈ 33.8 MB	696 KB	0.7% – 1.3%
27.	relay17	264156 KiB ≈ 33.8 MB	696 KB	0.7% – 1.3%
28.	relay18	264156 KiB ≈ 33.8 MB	700 KB	0.7% – 1.3%
29.	relay19	264156 KiB ≈ 33.8 MB	704 KB	0.7% – 1.3%
30.	relay20	264156 KiB ≈ 33.8 MB	700 KB	0.7% – 1.3%
31.	relay21	264156 KiB ≈ 33.8 MB	700 KB	0.7% – 1.3%
32.	relay22	264156 KiB ≈ 33.8 MB	700 KB	0.7% – 1.3%
33.	relay23	264156 KiB ≈ 33.8 MB	704 KB	0.7% – 1.3%
34.	relay24	264156 KiB ≈ 33.8 MB	704 KB	0.7% – 1.3%
35.	relay25	264156 KiB ≈ 33.8 MB	696 KB	0.7% – 1.3%
36.	relay26	264156 KiB ≈ 33.8 MB	708 KB	0.7% – 1.3%
37.	relay27	264156 KiB ≈ 33.8 MB	704 KB	0.7% – 1.3%
38.	relay28	264156 KiB ≈ 33.8 MB	704 KB	0.7% – 1.3%
39.	relay29	264156 KiB ≈ 33.8 MB	704 KB	0.7% – 1.3%
40.	relay30	264156 KiB ≈ 33.8 MB	704 KB	0.7% – 1.3%
41.	37 Network Hops	279424 KiB ≈ 284.9 MB	-	0%
42.	37 UML Switches	80364 KiB ≈ 81.4 MB	-	0%
TOTAL	-	1718.3 MB ≈ 1.7 GB	28164 KB ≈ 28.1 MB	28% – 52%

Appendix I DNS Evaluation Experiment – DNS Query over UDP

No. Time Source Destination Protocol Length Info

1 0.000198000 192.168.0.101 10.4.0.230 DNS 83 Standard query 0x1650 PTR 230.0.4.10.in-addr.arpa

Frame 1: 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface 0

Ethernet II, Src: 66:58:4a:dd:5a:63 (66:58:4a:dd:5a:63), Dst: ca:8f:13:ab:59:4f (ca:8f:13:ab:59:4f)

Internet Protocol Version 4, Src: 192.168.0.101 (192.168.0.101), Dst: 10.4.0.230 (10.4.0.230)

User Datagram Protocol, Src Port: 60988 (60988), Dst Port: 53 (53)

Domain Name System (query)

No. Time Source Destination Protocol Length Info

2 0.003032000 10.4.0.230 192.168.0.101 DNS 142 Standard query response 0x1650 PTR dns.torlab.test

Frame 2: 142 bytes on wire (1136 bits), 142 bytes captured (1136 bits) on interface 0

Ethernet II, Src: ca:8f:13:ab:59:4f (ca:8f:13:ab:59:4f), Dst: 66:58:4a:dd:5a:63 (66:58:4a:dd:5a:63)

Internet Protocol Version 4, Src: 10.4.0.230 (10.4.0.230), Dst: 192.168.0.101 (192.168.0.101)

User Datagram Protocol, Src Port: 53 (53), Dst Port: 60988 (60988)

Domain Name System (response)

Appendix J DNS Evaluation Experiment – DNS Query over TCP

```
No.     Time           Source                Destination           Protocol Length Info

1       0.000000000    192.168.0.101         10.4.0.230            DNS      80     Standard query
0xabde  A internet.torlab.test

2       0.001073000    10.4.0.230            192.168.0.101         DNS      544    Standard query
response 0xabde  A 10.9.0.254 A 10.10.0.254 A 10.11.0.254 A 10.12.0.254 A 10.13.0.254 A
10.14.0.254 A 10.15.0.254 A 10.16.0.254 A 10.17.0.254 A 10.18.0.254 A 10.19.0.254 A 10.20.0.254
A 10.21.0.254 A
10.22.0.254 A 10.23.0.254 A 10.24.0.254 A 10.25.0.254 A 10.26.0.254 A 10.27.0.254 A 10.28.0.254
A
10.29.0.254 A 10.30.0.254 A 10.31.0.254 A 10.32.0.254 A 10.33.0.254 A 10.34.0.254 A 10.0.0.254
A 10.1.0.254 A 10.2.0.254

3       0.008856000    192.168.0.101         10.4.0.230            TCP      74     46777→53 [SYN]
Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=1068024 TSecr=0 WS=4

4       0.009357000    10.4.0.230            192.168.0.101         TCP      74     53→46777 [SYN,
ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=1079954 TSecr=1068024 WS=4

5       0.009419000    192.168.0.101         10.4.0.230            TCP      66     46777→53 [ACK]
Seq=1 Ack=1 Win=14600 Len=0 TSval=1068024 TSecr=1079954

6       0.010823000    192.168.0.101         10.4.0.230            DNS      106    Standard query
0x048a  A internet.torlab.test

7       0.011514000    10.4.0.230            192.168.0.101         TCP      66     53→46777 [ACK]
Seq=1 Ack=41 Win=14480 Len=0 TSval=1079954 TSecr=1068024

8       0.011879000    10.4.0.230            192.168.0.101         DNS      700    Standard query
response 0x048a  A 10.10.0.254 A 10.11.0.254 A 10.12.0.254 A 10.13.0.254 A 10.14.0.254 A
10.15.0.254 A 10.16.0.254 A 10.17.0.254 A 10.18.0.254 A 10.19.0.254 A 10.20.0.254 A 10.21.0.254
A 10.22.0.254 A
10.23.0.254 A 10.24.0.254 A 10.25.0.254 A 10.26.0.254 A 10.27.0.254 A 10.28.0.254 A 10.29.0.254
A
10.30.0.254 A 10.31.0.254 A 10.32.0.254 A 10.33.0.254 A 10.34.0.254 A 10.0.0.254 A 10.1.0.254 A
10.2.0.254 A 10.3.0.254 A 10.4.0.254 A 10.5.0.254 A 10.6.0.254 A 10.7.0.254 A 10.8.0.254 A
10.9.0.254

9       0.011940000    192.168.0.101         10.4.0.230            TCP      66     46777→53 [ACK]
Seq=41 Ack=635 Win=15868 Len=0 TSval=1068034 TSecr=1079954
10      0.015096000    192.168.0.101         10.4.0.230            TCP      74     56029→53 [SYN]
Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=1068034 TSecr=0 WS=4

11      0.015101000    192.168.0.101         10.4.0.230            TCP      66     46777→53 [FIN,
ACK] Seq=41 Ack=635 Win=15868 Len=0 TSval=1068034 TSecr=1079954

12      0.015724000    10.4.0.230            192.168.0.101         TCP      74     53→56029 [SYN,
ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=1079954 TSecr=1068034 WS=4

13      0.015959000    192.168.0.101         10.4.0.230            TCP      66     56029→53 [ACK]
Seq=1 Ack=1 Win=14600 Len=0 TSval=1068034 TSecr=1079954

14      0.016126000    192.168.0.101         10.4.0.230            DNS      106    Standard query
0xd273  AAAA internet.torlab.test

15      0.016263000    10.4.0.230            192.168.0.101         TCP      66     53→46777 [FIN,
ACK] Seq=635 Ack=42 Win=14480 Len=0 TSval=1079954 TSecr=1068034

16      0.016322000    192.168.0.101         10.4.0.230            TCP      66     46777→53 [ACK]
Seq=42 Ack=636 Win=15868 Len=0 TSval=1068034 TSecr=1079954

17      0.016627000    10.4.0.230            192.168.0.101         TCP      66     53→56029 [ACK]
Seq=1 Ack=41 Win=14480 Len=0 TSval=1079954 TSecr=1068034

18      0.018086000    10.4.0.230            192.168.0.101         DNS      147    Standard query
response 0xd273

19      0.018131000    192.168.0.101         10.4.0.230            TCP      66     56029→53 [ACK]
Seq=41 Ack=82 Win=14600 Len=0 TSval=1068034 TSecr=1079954

20      0.018665000    192.168.0.101         10.4.0.230            TCP      74     49886→53 [SYN]
Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=1068034 TSecr=0 WS=4

21      0.018964000    10.4.0.230            192.168.0.101         TCP      74     53→49886 [SYN,
ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=1079955 TSecr=1068034 WS=4
```

22 0.019050000 192.168.0.101 10.4.0.230 TCP 66 49886→53 [ACK] Seq=1 Ack=1 Win=14600 Len=0 TSval=1068034 TSecr=1079955

23 0.019138000 192.168.0.101 10.4.0.230 TCP 66 56029→53 [FIN, ACK] Seq=41 Ack=82 Win=14600 Len=0 TSval=1068034 TSecr=1079954

24 0.019595000 192.168.0.101 10.4.0.230 DNS 106 Standard query 0xf3b1 MX internet.torlab.test

25 0.019916000 10.4.0.230 192.168.0.101 TCP 66 53→49886 [ACK] Seq=1 Ack=41 Win=14480 Len=0 TSval=1079955 TSecr=1068034

26 0.020288000 10.4.0.230 192.168.0.101 TCP 66 53→56029 [FIN, ACK] Seq=82 Ack=42 Win=14480 Len=0 TSval=1079955 TSecr=1068034

27 0.020340000 192.168.0.101 10.4.0.230 TCP 66 56029→53 [ACK] Seq=42 Ack=83 Win=14600 Len=0 TSval=1068035 TSecr=1079955

28 0.020855000 10.4.0.230 192.168.0.101 DNS 147 Standard query response 0xf3b1

29 0.020891000 192.168.0.101 10.4.0.230 TCP 66 49886→53 [ACK] Seq=41 Ack=82 Win=14600 Len=0 TSval=1068035 TSecr=1079955

30 0.021578000 192.168.0.101 10.4.0.230 TCP 66 49886→53 [FIN, ACK] Seq=41 Ack=82 Win=14600 Len=0 TSval=1068035 TSecr=1079955

31 0.022264000 10.4.0.230 192.168.0.101 TCP 66 53→49886 [FIN, ACK] Seq=82 Ack=42 Win=14480 Len=0 TSval=1079955 TSecr=1068035

32 0.022359000 192.168.0.101 10.4.0.230 TCP 66 49886→53 [ACK] Seq=42 Ack=83 Win=14600 Len=0 TSval=1068035 TSecr=1079955

Appendix K Version 4: Tor Integration Configuration Files

lab.conf

```
1: LAB_DESCRIPTION="TOR NETWORK SIMULATOR"
2: LAB_VERSION="4"
3: LAB_AUTHOR="Abdullahi Modibbo"
4: LAB_EMAIL="modibbo04@gmail.com"
5: #memory allocation
6: pc1[mem]=128
7: pc2[mem]=128
8: pc3[mem]=128
9: pc4[mem]=128
10: gw1[mem]=128
11: gw2[mem]=128
12: dns[mem]=256
13: internet[mem]=512
14: dirauth1[mem]=256
15: dirauth2[mem]=256
16: dirauth3[mem]=256
17: dirauth4[mem]=256
18: dirauth5[mem]=256
19: relay1[mem]=128
20: relay2[mem]=128
21: relay3[mem]=128
22: relay4[mem]=128
23: relay5[mem]=128
24: relay6[mem]=128
25: relay7[mem]=128
26: relay8[mem]=128
27: relay9[mem]=128
28: relay10[mem]=128
29: relay11[mem]=128
30: relay12[mem]=128
31: relay13[mem]=128
32: relay14[mem]=128
33: relay15[mem]=128
34: relay16[mem]=128
35: relay17[mem]=128
36: relay18[mem]=128
37: relay19[mem]=128
38: relay20[mem]=128
39: relay21[mem]=128
40: relay22[mem]=128
41: relay23[mem]=128
42: relay24[mem]=128
43: relay25[mem]=128
44: relay26[mem]=128
45: relay27[mem]=128
46: relay28[mem]=128
47: relay29[mem]=128
48: relay30[mem]=128
49: #subnets
50: gw1[0]=A
51: pc1[0]=A
52: pc2[0]=A
53: gw2[0]=B
54: pc3[0]=B
55: pc4[0]=B
56: #links subnets to internet
57: gw1[1]=I1
58: internet[0]=I1
59: gw2[1]=I2
60: internet[1]=I2
61: dirauth1[0]=I3
62: internet[2]=I3
63: dirauth2[0]=I4
64: internet[3]=I4
```

```
65: dns[0]=I5
66: internet[4]=I5
67: relay1[0]=I6
68: internet[5]=I6
69: relay2[0]=I7
70: internet[6]=I7
71: relay3[0]=I8
72: internet[7]=I8
73: relay4[0]=I9
74: internet[8]=I9
75: relay5[0]=I10
76: internet[9]=I10
77: relay6[0]=I11
78: internet[10]=I11
79: relay7[0]=I12
80: internet[11]=I12
81: relay8[0]=I13
82: internet[12]=I13
83: relay9[0]=I14
84: internet[13]=I14
85: relay10[0]=I15
86: internet[14]=I15
87: relay11[0]=I16
88: internet[15]=I16
89: relay12[0]=I17
90: internet[16]=I17
91: relay13[0]=I18
92: internet[17]=I18
93: relay14[0]=I19
94: internet[18]=I19
95: relay15[0]=I20
96: internet[19]=I20
97: relay16[0]=I21
98: internet[20]=I21
99: relay17[0]=I22
100: internet[21]=I22
101: relay18[0]=I23
102: internet[22]=I23
103: relay19[0]=I24
104: internet[23]=I24
105: relay20[0]=I25
106: internet[24]=I25
107: relay21[0]=I26
108: internet[25]=I26
109: relay22[0]=I27
110: internet[26]=I27
111: relay23[0]=I28
112: internet[27]=I28
113: relay24[0]=I29
114: internet[28]=I29
115: relay25[0]=I30
116: internet[29]=I30
117: relay26[0]=I31
118: internet[30]=I31
119: relay27[0]=I32
120: internet[31]=I32
121: relay28[0]=I33
122: internet[32]=I33
123: relay29[0]=I34
124: internet[33]=I34
125: relay30[0]=I35
126: internet[34]=I35
127: dirauth3[0]=I36
128: internet[35]=I36
129: dirauth4[0]=I37
130: internet[36]=I37
131: dirauth5[0]=I38
132: internet[37]=I38
```

Scripts/dirauth_gen_keys.sh *1:*
```
#!/bin/bash 2:
3: #get the ip
4: dir_ip=$(echo $(ifconfig | grep Bcast | cut -d ':' -f2 | cut -d ' ' -f1))
5: machine_name=$(echo $(uname -n)) 6:
7: #create dir to store tor keys 8: mkdir
/var/lib/tor/keys 9:
10: #generating certificates
11: echo "Generating Certificates"
12:tor-gencert --create-identity-key -m 12 -a $dir_ip:7000 \
 -i /var/lib/tor/keys/authority_identity_key \
 -s /var/lib/tor/keys/authority_signing_key \  -c
/var/lib/tor/keys/authority_certificate 13:
14: #generating onion keys
15: echo "Generating fingerprints"
16: tor --list-fingerprint --orport 1 \
  --dirserver "x 127.0.0.1:1
ffffffffffffffffffffffffffffffffffffffff" \   --
datadirectory /var/lib/tor/ 17:
18: #copying files to the host machine
19: cp -R /var/lib/tor/keys /hostlab/$machine_name/var/lib/tor/
20: cp /var/lib/tor/fingerprint /hostlab/$machine_name/var/lib/tor/ 21: cp
/etc/tor/torrc /hostlab/$machine_name/etc/tor/ 22:
23: #extracting dirauth fingerprints and importing them to a file located in the
    host
24: echo "Copying keys to host maching"
25: echo "Machine name: $machine_name" >> /hostlab/dirauth_fingerprints.txt
26: fingerprint1=$(echo $(cat /var/lib/tor/keys/authority_certificate | grep
    fingerprint | cut -d ' ' -f2))
27: echo "Certificate fingerprint is: $fingerprint1" echo
"Certificate fingerprint is: $fingerprint1" >>
/hostlab/dirauth_fingerprints.txt 28:
29: fingerprint2=$(echo $(cat /var/lib/tor/fingerprint | cut -d ' ' f2))
30: echo "Onion Fingerprint is: $fingerprint2"
31: echo "Onion Fingerprint is: $fingerprint2" >>
    /hostlab/dirauth_fingerprints.txt
32: echo "-------------------------------------------------------------
    ---------" >> /hostlab/dirauth_fingerprints.txt
```

Scripts/relay_gen_keys.sh *1:*
```
#!/bin/bash 2:
3: #get the ip
4:dir_ip=$(echo $(ifconfig | grep Bcast | cut -d ':' -f2 | cut -d ' ' -f1))
5: machine_name=$(echo $(uname -n)) 6:
7: #generating onion keys
8: echo "Generating fingerprints"
9: tor --list-fingerprint --orport 1 \
  --dirserver "x 127.0.0.1:1
ffffffffffffffffffffffffffffffffffffffff" \   --
datadirectory /var/lib/tor/ 10:
11: #copying files to the host machine
12: cp /var/lib/tor/fingerprint /hostlab/$machine_name/var/lib/tor/ 13:
14: #extracting dirauth fingerprints and importing them to a file located in the
    host
15: echo "Copying fingerprint to host maching"
16: echo "Machine name: $machine_name with IP: $dir_ip" >>
    /hostlab/relay_fingerprints.txt
17: fingerprint=$(echo $(cat /var/lib/tor/fingerprint | cut -d ' ' f2))
18: echo "Onion Fingerprint is: $fingerprint"
19: echo "Onion Fingerprint is: $fingerprint" >> /hostlab/relay_fingerprints.txt
20: echo "-------------------------------------------------------------
    --" >> /hostlab/relay_fingerprints.txt
```

Scripts/client_gen_key.sh *1:*
```
#!/bin/bash 2:
3: #get the ip
4: dir_ip=$(echo $(ifconfig | grep Bcast | cut -d ':' -f2 | cut -d ' ' -f1))
5: machine_name=$(echo $(uname -n)) 6:
7: #generating onion keys
```

```
8: echo "Generating fingerprints"
9: tor --list-fingerprint --orport 1 \
  --dirserver "x 127.0.0.1:1
ffffffffffffffffffffffffffffffffffffffff" \   --
datadirectory /var/lib/tor/ 10:
11: #copying files to the host machine
12: cp /var/lib/tor/fingerprint /hostlab/$machine_name/var/lib/tor/ 13:
14: #extracting dirauth fingerprints and importing them to a file located in the
    host
15: echo "Copying fingerprint to host maching"
16: echo "Machine name: $machine_name with IP: $dir_ip" >>
    /hostlab/client_fingerprints.txt
17: fingerprint=$(echo $(cat /var/lib/tor/fingerprint | cut -d ' ' f2))
18: echo "Onion Fingerprint is: $fingerprint"
19: echo "Onion Fingerprint is: $fingerprint" >> /hostlab/client_fingerprints.txt
20: echo "------------------------------------------------------------" >>
    /hostlab/client_fingerprints.txt
```

Appendix L Torrc Text Files

dirauth1 virtual machine ->>> /etc/tor/torrc

```
1: TestingTorNetwork 1
2: DataDirectory /var/lib/tor
3: RunAsDaemon 1
4: ConnLimit 60
5: Nickname dirauth1
6: PidFile /var/lib/tor/pid
7: Log notice file /var/lib/tor/notice.log
8: Log info file /var/lib/tor/info.log
9: ProtocolWarnings 1
10: SafeLogging 0
11: DisableDebuggerAttachment 0
12: DirAuthority dirauth1 orport=5000 no-v2 hs
    v3ident=34AA3B1524B616FFE8629DEBA048E4A8593D8934 10.2.0.1:7000
    CA23E608D2012F40F482A8FCEF6EE87D5F5300AF
13: DirAuthority dirauth2 orport=5000 no-v2 hs
    v3ident=A40A681ACD17E86AF7D04F022801444C87C84DFA 10.3.0.1:7000
    6F081D8E1D73B744E5B78C3646635A2231B6ACFF
14: DirAuthority dirauth3 orport=5000 no-v2 hs
    v3ident=F6534ED79E9A9B7D958EF5FCB7C6D6A89AED8B1B 10.35.0.1:7000
    100ADF0F0D15F2C5911ABAE024FA830FB9202F94
15: DirAuthority dirauth4 orport=5000 no-v2 hs
    v3ident=CD75A6CC850FA37241C94543956A077ABD60B3B0 10.36.0.1:7000
    4DFD720526C21C246676329DA90E1DAEEC8CAF2F
16: DirAuthority dirauth5 orport=5000 no-v2 hs
    v3ident=6966E61FB5EB175818A4F23C790649BE3D6B9AD2 10.37.0.1:7000
    206F927649F6BA585E9B82459B875F1650B2CE80
17: SocksPort 0
18: OrPort 5000
19: Address 10.2.0.1
20: DirPort 7000
21: ExitPolicy accept 192.168.0.0/24:*
22: ExitPolicy accept 193.168.0.0/24:*
23: ExitPolicy accept 10.0.0.0/16:*
24: ExitPolicy accept 10.1.0.0/16:*
25: ExitPolicy accept 10.2.0.0/16:*
26: ExitPolicy accept 10.3.0.0/16:*
27: ExitPolicy accept 10.4.0.0/16:*
28: ExitPolicy accept 10.5.0.0/16:*
29: ExitPolicy accept 10.6.0.0/16:*
30: ExitPolicy accept 10.7.0.0/16:*
31: ExitPolicy accept 10.8.0.0/16:*
32: ExitPolicy accept 10.9.0.0/16:*
33: ExitPolicy accept 10.10.0.0/16:*
34: ExitPolicy accept 10.11.0.0/16:*
35: ExitPolicy accept 10.12.0.0/16:*
36: ExitPolicy accept 10.13.0.0/16:*
37: ExitPolicy accept 10.14.0.0/16:*
38: ExitPolicy accept 10.15.0.0/16:*
39: ExitPolicy accept 10.16.0.0/16:*
40: ExitPolicy accept 10.17.0.0/16:*
41: ExitPolicy accept 10.18.0.0/16:*
42: ExitPolicy accept 10.19.0.0/16:*
43: ExitPolicy accept 10.20.0.0/16:*
44: ExitPolicy accept 10.21.0.0/16:*
45: ExitPolicy accept 10.22.0.0/16:*
46: ExitPolicy accept 10.23.0.0/16:*
47: ExitPolicy accept 10.24.0.0/16:*
48: ExitPolicy accept 10.25.0.0/16:*
49: ExitPolicy accept 10.26.0.0/16:*
50: ExitPolicy accept 10.27.0.0/16:*
51: ExitPolicy accept 10.28.0.0/16:*
52: ExitPolicy accept 10.29.0.0/16:*
53: ExitPolicy accept 10.30.0.0/16:*
54: ExitPolicy accept 10.31.0.0/16:*
55: ExitPolicy accept 10.32.0.0/16:*
```

```
56: ExitPolicy accept 10.33.0.0/16:*
57: ExitPolicy accept 10.34.0.0/16:*
58: ExitPolicy accept 10.35.0.0/16:*
59: ExitPolicy accept 10.36.0.0/16:*
60: ExitPolicy accept 10.37.0.0/16:*
61: ExitPolicy accept 10.38.0.0/16:*
62: ExitPolicy accept 10.39.0.0/16:*
63: ExitPolicy accept [::1]:*
64: IPv6Exit 1
65: AuthoritativeDirectory 1
66: V3AuthoritativeDirectory 1
67: ContactInfo dirauth1@test.test
68: ExitPolicy reject *:*
69: TestingV3AuthInitialVotingInterval 300
70: TestingV3AuthInitialVoteDelay 20
71: TestingV3AuthInitialDistDelay 20
```

relay1 virtual machine ->>> /etc/tor/torrc

```
1: TestingTorNetwork 1
2: DataDirectory /var/lib/tor
3: RunAsDaemon 1
4: ConnLimit 60
5: Nickname relay1
6: PidFile /var/lib/tor/pid
7: Log notice file /var/lib/tor/notice.log
8: Log info file /var/lib/tor/info.log
9: ProtocolWarnings 1
10: SafeLogging 0
11: DisableDebuggerAttachment 0
12: DirAuthority dirauth1 orport=5000 no-v2 hs
    v3ident=34AA3B1524B616FFE8629DEBA048E4A8593D8934 10.2.0.1:7000
    CA23E608D2012F40F482A8FCEF6EE87D5F5300AF
13: DirAuthority dirauth2 orport=5000 no-v2 hs
    v3ident=A40A681ACD17E86AF7D04F022801444C87C84DFA 10.3.0.1:7000
    6F081D8E1D73B744E5B78C3646635A2231B6ACFF
14: DirAuthority dirauth3 orport=5000 no-v2 hs
    v3ident=F6534ED79E9A9B7D958EF5FCB7C6D6A89AED8B1B 10.35.0.1:7000
    100ADF0F0D15F2C5911ABAE024FA830FB9202F94
15: DirAuthority dirauth4 orport=5000 no-v2 hs
    v3ident=CD75A6CC850FA37241C94543956A077ABD60B3B0 10.36.0.1:7000
    4DFD720526C21C246676329DA90E1DAEEC8CAF2F
16: DirAuthority dirauth5 orport=5000 no-v2 hs
    v3ident=6966E61FB5EB175818A4F23C790649BE3D6B9AD2 10.37.0.1:7000
    206F927649F6BA585E9B82459B875F1650B2CE80
17: SocksPort 0
18: OrPort 5000
19: Address 10.5.0.1
20: ExitPolicy accept 192.168.0.0/24:*
21: ExitPolicy accept 193.168.0.0/24:*
22: ExitPolicy accept 10.0.0.0/16:*
23: ExitPolicy accept 10.1.0.0/16:*
24: ExitPolicy accept 10.2.0.0/16:*
25: ExitPolicy accept 10.3.0.0/16:*
26: ExitPolicy accept 10.4.0.0/16:*
27: ExitPolicy accept 10.5.0.0/16:*
28: ExitPolicy accept 10.6.0.0/16:*
29: ExitPolicy accept 10.7.0.0/16:*
30: ExitPolicy accept 10.8.0.0/16:*
31: ExitPolicy accept 10.9.0.0/16:*
32: ExitPolicy accept 10.10.0.0/16:*
33: ExitPolicy accept 10.11.0.0/16:*
34: ExitPolicy accept 10.12.0.0/16:*
35: ExitPolicy accept 10.13.0.0/16:*
36: ExitPolicy accept 10.14.0.0/16:*
37: ExitPolicy accept 10.15.0.0/16:*
38: ExitPolicy accept 10.16.0.0/16:*
39: ExitPolicy accept 10.17.0.0/16:*
40: ExitPolicy accept 10.18.0.0/16:*
41: ExitPolicy accept 10.19.0.0/16:*
```

```
42: ExitPolicy accept 10.20.0.0/16:*
43: ExitPolicy accept 10.21.0.0/16:*
44: ExitPolicy accept 10.22.0.0/16:*
45: ExitPolicy accept 10.23.0.0/16:*
46: ExitPolicy accept 10.24.0.0/16:*
47: ExitPolicy accept 10.25.0.0/16:*
48: ExitPolicy accept 10.26.0.0/16:*
49: ExitPolicy accept 10.27.0.0/16:*
50: ExitPolicy accept 10.28.0.0/16:*
51: ExitPolicy accept 10.29.0.0/16:*
52: ExitPolicy accept 10.30.0.0/16:*
53: ExitPolicy accept 10.31.0.0/16:*
54: ExitPolicy accept 10.32.0.0/16:*
55: ExitPolicy accept 10.33.0.0/16:*
56: ExitPolicy accept 10.34.0.0/16:*
57: ExitPolicy accept 10.35.0.0/16:*
58: ExitPolicy accept 10.36.0.0/16:*
59: ExitPolicy accept 10.37.0.0/16:*
60: ExitPolicy accept 10.38.0.0/16:*
61: ExitPolicy accept 10.39.0.0/16:*
62: ExitPolicy accept [::1]:*
63: IPv6Exit 1
```

pc1 virtual machine ->>> /etc/tor/torrc

```
1: TestingTorNetwork 1
2: DataDirectory /var/lib/tor
3: RunAsDaemon 1
4: ConnLimit 60
5: Nickname pc1
6: PidFile /var/lib/tor/pid
7: Log notice file /var/lib/tor/notice.log
8: Log info file /var/lib/tor/info.log
9: ProtocolWarnings 1
10: SafeLogging 0
11: DisableDebuggerAttachment 0
12: DirAuthority dirauth1 orport=5000 no-v2 hs
    v3ident=34AA3B1524B616FFE8629DEBA048E4A8593D8934 10.2.0.1:7000
    CA23E608D2012F40F482A8FCEF6EE87D5F5300AF
13: DirAuthority dirauth2 orport=5000 no-v2 hs
    v3ident=A40A681ACD17E86AF7D04F022801444C87C84DFA 10.3.0.1:7000
    6F081D8E1D73B744E5B78C3646635A2231B6ACFF
14: DirAuthority dirauth3 orport=5000 no-v2 hs
    v3ident=F6534ED79E9A9B7D958EF5FCB7C6D6A89AED8B1B 10.35.0.1:7000
    100ADF0F0D15F2C5911ABAE024FA830FB9202F94
15: DirAuthority dirauth4 orport=5000 no-v2 hs
    v3ident=CD75A6CC850FA37241C94543956A077ABD60B3B0 10.36.0.1:7000
    4DFD720526C21C246676329DA90E1DAEEC8CAF2F
16: DirAuthority dirauth5 orport=5000 no-v2 hs
    v3ident=6966E61FB5EB175818A4F23C790649BE3D6B9AD2 10.37.0.1:7000
    206F927649F6BA585E9B82459B875F1650B2CE80
17: SocksPort 0
18: OrPort 5000
19: Address 192.168.0.101
```

Appendix M Version 4: Tor Integration – Speed

	VM	VM BOOT UP TIME (min: sec. millisec)	NETWORK BOOT TIME (min: sec. millisec)
1.	dns	00:20.88	00:20.88
2.	internet	00:54.87	01:15.7 5
3.	dirauth1	00:09.99	
4.	dirauth2	00:09.99	
5.	dirauth3	00:09.99	01:25.74
6.	dirauth4	00:09.99	
7.	dirauth5	00:09.99	
8.	gw1	00:12. 07	
9.	gw2	00:12.07	
10.	relay1	00:12.07	01:37.81
11.	relay10	00:12.07	
12.	relay11	00:12.07	
13.	relay12	00:11.35	
14.	relay13	00:11.35	
15.	relay14	00:11.35	01:49.16
16.	relay15	00:11.35	
17.	relay16	00:11.35	
18.	relay17	00:12.65	
19.	relay18	00:12.65	
20.	relay19	00:12.65	02:01.81
21.	relay2	00:12.65	
22.	relay20	00:12.65	
23.	relay21	00:11.87	02:13.68
24.	relay22	00:11.87	
25.	relay23	00:11.87	
26.	relay24	00:11.87	
27.	relay25	00:11.87	

28.	relay26	00:12.70	
29.	relay27	00:12.70	
30.	relay28	00:12.70	02:26.38
31.	relay29	00:12.70	
32.	relay3	00:12.70	
33.	relay30	00:14.76	
34.	relay4	00:14.76	
35.	relay5	00:14.76	02:41.14
36.	relay6	00:14.76	
37.	relay7	00:14.76	
38.	relay8	00:14.44	
39.	relay9	00:14.44	
40.	pc1	00:14.44	02:55.68
41.	pc2	00:14.44	
42.	pc3	00:14.44	
43.	pc4	00:22.47	03:18.15

Appendix N Version 4: Tor Integration – Hardware Limitations

	VIRTUAL MACHINE	MEMORY ALLOC. ON /	STORAGE ALLOC. ON SYSTEM	CPU USAGE ON RUN TIME
1.	dirauth1	277 MB	772 KB	0.7% – 1.3%
2.	dirauth2	277 MB	760 KB	0.7% – 1.3%
3.	dirauth3	277 MB	804 KB	0.7% – 1.3%
4.	dirauth4	277 MB	764 KB	0.7% – 1.3%

5.	dirauth5	277 MB	768 KB	0.7% – 1.3%
6.	dns	277 MB	520 KB	0.7% – 1.3%
7.	gw1	149 MB	460 KB	0.7% – 1.3%
8.	gw2	149 MB	464 KB	0.7% – 1.3%
9.	internet	534 MB	524 KB	0.7% – 1.3%
10.	pc1	149 MB	448 KB	0.7% – 1.3%
11.	pc2	149 MB	472 KB	0.7% – 1.3%
12.	pc3	149 MB	504 KB	0.7% – 1.3%
13.	pc4	149 MB	416 KB	0.7% – 1.3%
14.	relay1	149 MB	568 KB	0.7% – 1.3%
15.	relay2	149 MB	552 KB	0.7% – 1.3%
16.	relay3	149 MB	520 KB	0.7% – 1.3%
17.	relay4	149 MB	548 KB	0.7% – 1.3%
18.	relay5	149 MB	512 KB	0.7% – 1.3%
19.	relay6	149 MB	568 KB	0.7% – 1.3%

20.	relay7	149 MB	520 KB	0.7% – 1.3%
21.	relay8	149 MB	520 KB	0.7% – 1.3%
22.	relay9	149 MB	532 KB	0.7% – 1.3%
23.	relay10	149 MB	548 KB	0.7% – 1.3%
24.	relay11	149 MB	560 KB	0.7% – 1.3%
25.	relay12	149 MB	552 KB	0.7% – 1.3%
26.	relay13	149 MB	556 KB	0.7% – 1.3%
27.	relay14	149 MB	576 KB	0.7% – 1.3%
28.	relay15	149 MB	548 KB	0.7% – 1.3%
29.	relay16	149 MB	552 KB	0.7% – 1.3%
30.	relay17	149 MB	564 KB	0.7% – 1.3%
31.	relay18	149 MB	552 KB	0.7% – 1.3%
32.	relay19	149 MB	556 KB	0.7% – 1.3%
33.	relay20	149 MB	560 KB	0.7% – 1.3%
34.	relay21	149 MB	552 KB	0.7% – 1.3%
35.	relay22	149 MB	544 KB	0.7% – 1.3%
36.	relay23	149 MB	564 KB	0.7% – 1.3%
37.	relay24	149 MB	548 KB	0.7% – 1.3%
38.	relay25	149 MB	544 KB	0.7% – 1.3%
39.	relay26	149 MB	556 KB	0.7% – 1.3%
40.	relay27	149 MB	560 KB	0.7% – 1.3%
41.	relay28	149 MB	552 KB	0.7% – 1.3%

42.	relay29	149 MB	544 KB	0.7% – 1.3%
43.	relay30	149 MB	548 KB	0.7% – 1.3%
44.	40 Network Hop s	302080 KiB ≈ 308 MB	-	0%
45.	40 UML Switches	86880 KiB ≈ 88 MB	-	0%
TOTAL	-	7936 MB ≈ 7.9 GB	24152 KB ≈ 24.1 MB	30.1% – 55.9%

Appendix O Version 4: Tor Integration – Open Ports

	VM	OPEN PORT	PROTOCOL	SERVICE
1.	All virtual machines	127.0.0.1:953 127.0.0.1:53 0.0.0.0:111 0.0.0.0:22 ::1:953 :::111 :::53 :::22 0.0.0.0:749 :::749	tcp tcp tcp / udp tcp tcp6 tcp6 / udp6 tcp6 / udp6 tcp6 udp udp6	named named rpcbind sshd named rpcbind named sshd rpcbind rpcbind
2.	dns	10.4.0.230:53 127.0.0.1:9050	tcp / udp tcp	named tor
3.	gw1	192.168.0.254:53 127.0.0.1:9050	tcp/udp tcp	named tor
4.	gw2	193.168.0.254:53 127.0.0.1:9050	tcp/udp tcp	named tor
5.	internet	10.*.0.254:53 127.0.0.1:9050	tcp tcp	named tor
6.	pc1 - pc4	0.0.0.0:5000 19*.168.0.10*:53 127.0.0.1:9011	tcp tcp tcp	tor named tor
7.	dirauth1 - dirauth5	0.0.0.0:5000 0.0.0.0:7000 10.*.0.1:53	tcp tcp tcp	tor tor named
8.	relay1 - relay30	10.*.0.1:53 0.0.0.0:5000	tcp tcp	named tor

Appendix P Dirauth1 Reduced info.log

Sept 10 12:01:42.000 [notice] Tor 0.2.4.27 (git-412e3f7dc9c6c01a) opening new log file.

...

Sept 10 12:01:43.000 [warn] Configured authority type does not match authority type in DirAuthority list. Adjusting. (100 v 108)

Sept 10 12:01:43.000 [info] init_keys(): adding my own v3 cert

Sept 10 12:01:43.000 [info] trusted_dirs_load_certs_from_string(): Adding downloaded certificate for directory authority dirauth1 with signing key 3BBC0676EEC0FD8A2C80BC00256203BCAFDC1267

Sept 10 12:01:43.000 [info] cell_ewma_set_scale_factor(): Disabled cell_ewma algorithm because of value in Default value

Sept 10 12:01:43.000 [notice] Parsing GEOIP IPv4 file /usr/share/tor/geoip.

Sept 10 12:01:43.000 [notice] Parsing GEOIP IPv6 file /usr/share/tor/geoip6.

*Sept 10 12:01:44.000 [notice] Configured to measure statistics. Look for the *-stats files that will first be written to the data directory in 24 hours from now.*

Sept 10 12:01:44.000 [info] mark_my_descriptor_dirty(): Decided to publish new relay descriptor: config change

Sept 10 12:01:44.000 [info] configure_nameservers(): Parsing resolver configuration in '/etc/resolv.conf'

Sept 10 12:01:44.000 [info] eventdns: Parsing resolv.conf file /etc/resolv.conf

Sept 10 12:01:44.000 [info] eventdns: Added nameserver 10.4.0.230:53 as 0x40611d88

Sept 10 12:01:44.000 [info] eventdns: Setting maximum allowed timeouts to 16

Sept 10 12:01:44.000 [info] eventdns: Setting timeout to 10 Sept 10 12:01:44.000 [info] Bootstrapped 0%: Starting.

Sept 10 12:01:44.000 [info] trusted_dirs_load_certs_from_string(): Adding cached certificate for unrecognized directory authority with signing key 28EB6C6635F1DB90BB60AF8B43F1676E892BB30B

Sept 10 12:01:44.000 [info] trusted_dirs_load_certs_from_string(): Adding cached certificate for unrecognized directory authority with signing key 109A865D7DBE58367C120353CBE9947EE263695A

Sept 10 12:01:44.000 [info] trusted_dirs_load_certs_from_string(): Adding cached certificate for unrecognized directory authority with signing key 6C29F33BA94B5F9D8EC0EE9EB7144BDFF876AA52

Sept 10 12:01:44.000 [info] trusted_dirs_load_certs_from_string(): Adding cached certificate for unrecognized directory authority with signing key 02958C9BE542EEA43130E54FE6634B6E874B196D

Sept 10 12:01:44.000 [info] trusted_dirs_load_certs_from_string(): Adding cached certificate for unrecognized directory authority with signing key 8738EBDF44CD72B25FE20F8901290E0573950A56

Sept 10 12:01:44.000 [info] trusted_dirs_load_certs_from_string(): Adding cached certificate for unrecognized directory authority with signing key EFB790BA79E84FE5ED90B4C1E6022904B0E9C478

Sept 10 12:01:44.000 [info] trusted_dirs_load_certs_from_string(): Adding cached certificate for unrecognized directory authority with signing key E861D5367EE5A469892D3FE6B2A25218FBA133FC

Sept 10 12:01:44.000 [info] trusted_dirs_load_certs_from_string(): Adding cached certificate for unrecognized directory authority with signing key 45735762F32794C2286FEA914EE0322F04E0F36B

Sept 10 12:01:44.000 [info] trusted_dirs_load_certs_from_string(): Adding cached certificate for unrecognized directory authority with signing key 9439CFDCFD0FCF41E6501BCD2C0151E68C30B77F

Sept 10 12:01:44.000 [info] read_file_to_str(): Could not open "/var/lib/tor/cached-consensus": No such file or directory

Sept 10 12:01:44.000 [info] read_file_to_str(): Could not open "/var/lib/tor/unverified-consensus": No such file or directory

Sept 10 12:01:44.000 [warn] Consensus includes unrecognized authority 'dannenberg' at dannenberg.torauth.de:80 (contact Andreas Lehner; identity 0232AF901C31A04EE9848595AF9BB7620D4C5B2E)

Sept 10 12:01:44.000 [warn] Consensus includes unrecognized authority 'tor26' at 86.59.21.38:80 (contact Peter Palfrader; identity 14C131DFC5C6F93646BE72FA1401C02A8DF2E8B4)

Sept 10 12:01:44.000 [warn] Consensus includes unrecognized authority 'longclaw' at 199.254.238.53:80 (contact Riseup Networks <collective at riseup dot net> - 1nNzekuHGGzBYRzyjfjFEfeisNvxkn4RT; identity 23D15D965BC35114467363C165C4F724B64B4F66)

……

Appendix Q Debian Wheezy and Netkit-ng Depended Upstart Packages

LEVEL	NAME	DEPENDANCY	FUNCTIONALITY
rc0.d	-	-	-
	killprocs	Debian	manage / terminate specified pid's
rc1.d	motd	Debian	OS welcome message
	bootlogs	Debian	OS error logs
	single	Debian	start single-mode user
	netkit-phase1	Netkit	machine_name.startup

rc2.d	rsyslog	Debian	fast system log-processing
rc3.d	rsync	Debian	file synchronization and transferring
rc4.d	rc.local	Debian	add Upstart scripts
	netkit-phase2	Netkit	shared.startup
	rmnologin	Debian	managing system state
	-	-	-
rc5.d	-	-	-

…….

Appendix R Version 5: Tor Experimental Network - Configuration Files

kill_duplicates.sh *1:*

```
#!/bin/bash 2:
3: #get netkit-ng virtual machine names
4: vm_names=($(ps ax | grep netkit | grep "name=" | uniq -u | awk '{print
   $7}' | grep -v "/home" | cut -d'=' -f2 | sort --unique)) 5:
6: for i in "${vm_names[@]}" #loop through all vm names
7: do
8:      #Get all processes ids of $i
9: pid=($(ps ax | grep -v "bin/sh" | grep -v "xterm" | grep -v "Ss+"
              | grep $i | grep -v "grep" | grep -v "dnsmasq" | awk '{print $1}'))
10: echo "================================"
11: echo "killing pid" ${pid[0]} "of" $i
12: kill ${pid[0]} #kill pid with higher number
13: done
```

shared.startup *1: #get machine name*

```
2: machine_name=$(echo $(uname -n)) 3:
4: #configure the local dns server as the default and restart services 5:
conf_dns(){ 6:
7:      echo "Configuring /etc/resolv.conf file"
8:      echo "search torlab.test." > /etc/resolv.conf
9:      echo "nameserver 10.4.0.230" >> /etc/resolv.conf
10: echo "Restarting networking service"
11: service networking stop
12: service networking start
13: echo "Testing DNS server"
14: dig torlab.test
15: host dns.torlab.test 16: } 17:
18: #modifying key, fingerprint ownership permissions 19:
fixing_file_permissions(){ 20:
21:  chown debian-tor /var/lib/tor/fingerprint
22:  chown -R debian-tor /var/lib/tor/keys/
23: }
24: #starting appache server
25: initiating_webserver(){
26: /etc/init.d/apache2 start
26: netstat -tulpn | grep :80 28: } 29:
30: #starting tor service 31:
initiating_tor(){ 32:
33: service tor start
34: netstat -tulpn | grep tor 35: } 36:
37: #main configuration of network
38: case "$machine_name" in
39:  'dns')
```

```
40:     echo "Initializing dns specific configuration"
41:          ip addr add 10.4.0.230/16 dev eth0
42:          ip link set up dev eth0
43:          ip route add default via 10.4.0.254
44:          echo "Initializing DNS Server"
45:          /etc/init.d/bind9 start
46:          echo "DNS SERVER STARTED"
47:          echo "Checking for open ports"
48:          netstat -tulpn | grep :53
49:          ;;
50:    'internet')
51:          ifconfig eth0 10.0.0.254/16
52:          ifconfig eth1 10.1.0.254/16
53:          ifconfig eth2 10.2.0.254/16
54:          ifconfig eth3 10.3.0.254/16
55:          ifconfig eth4 10.4.0.254/16
56:          ifconfig eth5 10.5.0.254/16
57:          ifconfig eth6 10.6.0.254/16
58:          ifconfig eth7 10.7.0.254/16
59:          ifconfig eth8 10.8.0.254/16
60:          ifconfig eth9 10.9.0.254/16
61:          ifconfig eth10 10.10.0.254/16
62:          ifconfig eth11 10.11.0.254/16
63:          ifconfig eth12 10.12.0.254/16
64:          ifconfig eth13 10.13.0.254/16
65:          ifconfig eth14 10.14.0.254/16
66:          ifconfig eth15 10.15.0.254/16
67:          ifconfig eth16 10.16.0.254/16
68:          ifconfig eth17 10.17.0.254/16
69:          ifconfig eth18 10.18.0.254/16
70:          ifconfig eth19 10.19.0.254/16
```

```
71:         ifconfig eth20 10.20.0.254/16

72:         ifconfig eth21 10.21.0.254/16

73:         ifconfig eth22 10.22.0.254/16

74:         ifconfig eth23 10.23.0.254/16

75:         ifconfig eth24 10.24.0.254/16

76:         ifconfig eth25 10.25.0.254/16

77:         ifconfig eth26 10.26.0.254/16

78:         ifconfig eth27 10.27.0.254/16

79:         ifconfig eth28 10.28.0.254/16

80:         ifconfig eth29 10.29.0.254/16

81:         ifconfig eth30 10.30.0.254/16

82:         ifconfig eth31 10.31.0.254/16

83:         ifconfig eth32 10.32.0.254/16

84:         ifconfig eth33 10.33.0.254/16

85:         ifconfig eth34 10.34.0.254/16

86:         ifconfig eth35 10.35.0.254/16

87:         ifconfig eth36 10.36.0.254/16

88:         ifconfig eth37 10.37.0.254/16

89:         ifconfig eth38 10.38.0.254/16

90:         ifconfig eth39 10.39.0.254/16

91:         route add -net 192.168.0.0/24 gw 10.0.0.1

92:         route add -net 193.168.0.0/24 gw 10.1.0.1

93:         route add -net 10.2.0.0/16 gw 10.2.0.1

94:         route add -net 10.3.0.0/16 gw 10.3.0.1

95:         route add -net 10.5.0.0/16 gw 10.5.0.1

96:         route add -net 10.6.0.0/16 gw 10.6.0.1

97:         route add -net 10.7.0.0/16 gw 10.7.0.1

98:         route add -net 10.8.0.0/16 gw 10.8.0.1

99:         route add -net 10.9.0.0/16 gw 10.9.0.1

100:        route add -net 10.10.0.0/16 gw
            10.10.0.1
```

```
101:        route add -net 10.11.0.0/16 gw
            10.11.0.1

102:        route add -net 10.12.0.0/16 gw
            10.12.0.1

103:        route add -net 10.13.0.0/16 gw
            10.13.0.1

104:        route add -net 10.14.0.0/16 gw
            10.14.0.1

105:        route add -net 10.15.0.0/16 gw
            10.15.0.1

106:        route add -net 10.16.0.0/16 gw
            10.16.0.1

107:        route add -net 10.17.0.0/16 gw
            10.17.0.1

108:        route add -net 10.18.0.0/16 gw
            10.18.0.1

109:        route add -net 10.19.0.0/16 gw
            10.19.0.1

110:        route add -net 10.20.0.0/16 gw
            10.20.0.1

111:        route add -net 10.21.0.0/16 gw
            10.21.0.1

112:        route add -net 10.22.0.0/16 gw
            10.22.0.1

113:        route add -net 10.23.0.0/16 gw
            10.23.0.1

114:        route add -net 10.24.0.0/16 gw
            10.24.0.1

115:        route add -net 10.25.0.0/16 gw
            10.25.0.1

116:        route add -net 10.26.0.0/16 gw
            10.26.0.1

117:        route add -net 10.27.0.0/16 gw
            10.27.0.1

118:        route add -net 10.28.0.0/16 gw
            10.28.0.1

119:        route add -net 10.29.0.0/16 gw
            10.29.0.1

120:        route add -net 10.30.0.0/16 gw
            10.30.0.1

121:        route add -net 10.31.0.0/16 gw
            10.31.0.1

122:        route add -net 10.32.0.0/16 gw
```

```
            10.32.0.1

123:        route add -net 10.33.0.0/16 gw
            10.33.0.1

124:        route add -net 10.34.0.0/16 gw
            10.34.0.1

125:        route add -net 10.35.0.0/16 gw
            10.35.0.1

126:        route add -net 10.36.0.0/16 gw
            10.36.0.1

127:        route add -net 10.37.0.0/16 gw
            10.37.0.1

128:        route add -net 10.38.0.0/16 gw
            10.38.0.1

129:        route add -net 10.39.0.0/16 gw
            10.39.0.1

130:        conf_dns

131:        ;;

132: 'gw1')

133:        ifconfig eth0 192.168.0.254/24

134:        ifconfig eth1 10.0.0.1/16

135:        route add default gw 10.0.0.254

136:        conf_dns

137:        ;;

138: 'gw2')

139:        ifconfig eth0 193.168.0.254/24

140:        ifconfig eth1 10.1.0.1/16

141:        route add default gw 10.1.0.254

142:        conf_dns

143:        ;;

144: 'dirauth1')

145:        ifconfig eth0 10.2.0.1/16

146:        route add default gw 10.2.0.254

147:        conf_dns

148:        fixing_file_permissions

149:        initiating_tor

150:        ;;

151: 'dirauth2')
```

```
152:  ifconfig eth0 10.3.0.1/16

153:  route add default gw 10.3.0.254

154:  conf_dns

155:  fixing_file_permissions

156:  initiating_tor

157:  ;;

158: 'pc1')

159:  ifconfig eth0 192.168.0.101/24
```

```
160: route add default gw 192.168.0.254
161:    conf_dns
162:    initiating_tor
163:    ;;
164: 'pc2')
165:    ifconfig eth0 192.168.0.102/24
166: route add default gw 192.168.0.254
167:    conf_dns
168:    initiating_tor
169:    ;;
170: 'pc3')
171:    ifconfig eth0 193.168.0.101/24
172:   route add default gw 193.168.0.254
173:    conf_dns
174:    initiating_tor
175:    ;;
176: 'pc4')
177:    ifconfig eth0 193.168.0.102/24
178:   route add default gw 193.168.0.254
179:    conf_dns
180:    initiating_tor
181:    ;;
182: 'relay1')
183:    ifconfig eth0 10.5.0.1/16
184:    route add default gw 10.5.0.254
185:    conf_dns
186:    initiating_tor
187:    ;;
188: 'relay2')
189:    ifconfig eth0 10.6.0.1/16
190:    route add default gw 10.6.0.254
191:    conf_dns
192:    initiating_tor
193:    ;;
194: 'relay3')
195:    ifconfig eth0 10.7.0.1/16
196:    route add default gw 10.7.0.254
197:    conf_dns
198:    initiating_tor
199:    ;;
200: 'relay4')
```

```
201:    ifconfig eth0 10.8.0.1/16
202:    route add default gw 10.8.0.254
203:    conf_dns
204:    initiating_tor
205:    ;;
206: 'relay5')
207:    ifconfig eth0 10.9.0.1/16
208:    route add default gw 10.9.0.254
209:    conf_dns
210:    initiating_tor
211:    ;;
212: 'relay6')
213:    ifconfig eth0 10.10.0.1/16
214:    route add default gw 10.10.0.254
215:    conf_dns
216:    initiating_tor
217:    ;;
218: 'relay7')
219:    ifconfig eth0 10.11.0.1/16
220:    route add default gw 10.11.0.254
```

```
221:
222:
223:
224: 'relay8')
225:  ifconfig eth0 10.12.0.1/16
226:  route add default gw 10.12.0.254
227:  conf_dns
228:  initiating_tor
229:        ;;
230: 'relay9')
231:        ifconfig eth0 10.13.0.1/16
232:  route add default gw 10.13.0.254
233:        conf_dns
234:        initiating_tor
235:        ;;
236: 'relay10')
237:        ifconfig eth0 10.14.0.1/16
238:  route add default gw 10.14.0.254
239:        conf_dns
240:
241:        initiating_tor
242:        ;;
243: 'relay11')
244:        ifconfig eth0 10.15.0.1/16
245:  route add default gw 10.15.0.254
246:        conf_dns
247:        initiating_tor
248:        ;;
249: 'relay12')
250:        ifconfig eth0 10.16.0.1/16
251:  route add default gw 10.16.0.254
252:        conf_dns
253:        initiating_tor
254:        ;;
255: 'relay13')
256:        ifconfig eth0 10.17.0.1/16
257:  route add default gw 10.17.0.254
258:        conf_dns
259:        initiating_tor
260:        ;;
261: 'relay14')
262:        ifconfig eth0 10.18.0.1/16
263:  route add default gw 10.18.0.254
264:        conf_dns
265:        initiating_tor
266:        ;;
267: 'relay15')
268:        ifconfig eth0 10.19.0.1/16
269:  route add default gw 10.19.0.254
270:        conf_dns
271:        initiating_tor
272:        ;;
273: 'relay16')
274:  ifconfig eth0 10.20.0.1/16
275:  route add default gw 10.20.0.254
276:  conf_dns
277:  initiating_tor
278:  ;;
279: 'relay17')
280:  ifconfig eth0 10.21.0.1/16
281:  route add default gw 10.21.0.254 282:
283:
284:
285: 'relay18')
286:  ifconfig eth0 10.22.0.1/16
287:  route add default gw 10.22.0.254
288:  conf_dns
289:  initiating_tor
290:        ;;
```

```
291: 'relay19')
292:        ifconfig eth0 10.23.0.1/16
293:  route add default gw 10.23.0.254
294:        conf_dns
295:        initiating_tor
296:        ;;
297: 'relay20')
298:        ifconfig eth0 10.24.0.1/16
299:  route add default gw 10.24.0.254
300:        conf_dns
301:        initiating_tor
302:        ;;
303: 'relay21')
304:        ifconfig eth0 10.25.0.1/16
305:  route add default gw 10.25.0.254
306:        conf_dns
307:        initiating_tor
308:        ;;
309: 'relay22')
310:        ifconfig eth0 10.26.0.1/16
311:  route add default gw 10.26.0.254
312:        conf_dns
313:        initiating_tor
314:        ;;
315: 'relay23')
316:        ifconfig eth0 10.27.0.1/16
317:  route add default gw 10.27.0.254
318:        conf_dns
319:        initiating_tor
320:        ;;
321: 'relay24')
322:        ifconfig eth0 10.28.0.1/16 323:     route add default gw 10.28.0.254 324:
325:        conf_dns
326:        initiating_tor
327:        ;;
328: 'relay25')
329:        ifconfig eth0 10.29.0.1/16
330:  route add default gw 10.29.0.254
331:        conf_dns
332:        initiating_tor
333:        ;;
334: 'relay26')
335:  ifconfig eth0 10.30.0.1/16
336:  route add default gw 10.30.0.254
337:  conf_dns
338:  initiating_tor
339:  ;;
340: 'relay27')
341:  ifconfig eth0 10.31.0.1/16
342:  route add default gw 10.31.0.254 343:
344:
345:
346: 'relay28')
347:  ifconfig eth0 10.32.0.1/16
348:  route add default gw 10.32.0.254
349:  conf_dns
350:  initiating_tor
351:        ;;
352: 'relay29')
353:        ifconfig eth0 10.33.0.1/16
354:  route add default gw 10.33.0.254
355:        conf_dns
356:        initiating_tor
357:        ;;
358: 'relay30')
359:        ifconfig eth0 10.34.0.1/16
360:  route add default gw 10.34.0.254
361:        conf_dns
```

```
362:         initiating_tor
363:         ;;
364: 'dirauth3')
365:         ifconfig eth0 10.35.0.1/16
366:  route add default gw 10.35.0.254
368:         conf_dns
369:         fixing_file_permissions
370:         initiating_tor
371:         ;;
372: 'dirauth4')
373:         ifconfig eth0 10.36.0.1/16
374:  route add default gw 10.36.0.254
375:         conf_dns
376:         fixing_file_permissions
377:         initiating_tor
378:         ;;
379: 'dirauth5')
380:         ifconfig eth0 10.37.0.1/16
381:  route add default gw 10.37.0.254
382:         conf_dns
383:         fixing_file_permissions
384:         initiating_tor
385:         ;;
386: 'webser1')
387:         ifconfig eth0 10.38.0.1/16
388:  route add default gw 10.38.0.254
389:         conf_dns
390:         initiating_webserver
391:         ;;
392: 'webser2')
393:         ifconfig eth0 10.39.0.1/16
394:  route add default gw 10.39.0.254
395:         conf_dns
396:         initiating_webserver
397:         ;;
398: esac
```

Appendix S Version 5: Tor Experimental Network – Speed

	VM	VM BOOT UP TIME (min: sec. millisec)	NETWORK BOOT TIME (min: sec. millisec)
1.	dns	00:15.30	00:15.30
2.	internet	00:52.74	01:08.04
3.	dirauth1	00:07.39	
4.	dirauth2	00:07.39	
5.	dirauth3	00:07.39	01:15.43
6.	dirauth4	00:07.39	
7.	dirauth5	00:07.39	
8.	gw1	00:08.57	
9.	gw2	00:08.57	
10.	webser1	00:08.57	01:24.00
11.	webser2	00:08.57	
12.	relay1	00:08.57	
13.	relay10	00:09.66	
14.	relay11	00:09.66	
15.	relay12	00:09.66	01:33.66
16.	relay13	00:09.66	
17.	relay14	00:09.66	
18.	relay15	00:09.07	
19.	relay16	00:09.07	01:42.73
20.	relay17	00:09.07	

21.	relay18	00:09.07	
22.	relay19	00:09.07	
23.	relay2	00:08.95	01:51.68
24.	relay20	00:08.95	
25.	relay21	00:08.95	
26.	relay22	00:08.95	
27.	relay23	00:08.95	
28.	relay24	00:10.65	
29.	relay25	00:10.65	
30.	relay26	00:10.65	02:02.33
31.	relay27	00:10.65	
32.	relay28	00:10.65	
33.	relay29	00:07.99	
34.	relay3	00:07.99	
35.	relay30	00:07.99	02:10.32
36.	relay4	00:07.99	
37.	relay5	00:07.99	
38.	relay6	00:11.04	
39.	relay7	00:11.04	
40.	relay8	00:11.04	02:21.36
41.	relay9	00:11.04	
42.	pc1	00:11.04	
43.	pc2	00:17.94	
43.	pc3	00:17.94	02:39.30
43.	pc4	00:17.94	

References

ACQUISTI, A., BRANDIMARTE, L. & LOEWENSTEIN, G. 2015. Privacy and human behavior in the age of information. *Science,* 347**,** 509-514.

ADRIAN, D., BHARGAVAN, K., DURUMERIC, Z., GAUDRY, P., GREEN, M., HALDERMAN, J. A., HENINGER, N., SPRINGALL, D., THOMÉ, E. &
VALENTA, L. Imperfect forward secrecy: How Diffie-Hellman fails in practice.
Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015. ACM, 5-17.

AIRVPN. 2017. *Using AirVPN with Tor* [Online]. United States: AirVPN. Available: https://airvpn.org/tor/

AJZEN, I. 2002. Constructing a TPB questionnaire: Conceptual and methodological considerations.

AKHOONDI, M., YU, C. & MADHYASTHA, H. V. LASTor: A low-latency AS-aware Tor client. Security and Privacy (SP), 2012 IEEE Symposium on, 2012. IEEE, 476-490.

ALEXANDER, H. 2013. "North Korea's Twitter and Flickr accounts hacked by Anonymous," Telegraph (4 April), at http://www.telegraph.co.uk/news/worldnews/asia/northkorea/9972032/North-Koreas-Twitterand-Flickr-accounts-hacked-by-Anonymous.html,

ALLEGRETTA, C., RAMSEY, D. L., SCHULENBERG, B., CORSI, R., MALLACH, J., ROGOYSKI, A., SIEMBORSKI, R., FRYSINGER, M. & BENBENNICK, D. 2017. *The Gnu nano homepage* [Online]. US: nano-editor.org. Available: https://www.nano-editor.org/

ALMASY, S. 2013. "Two teens found guilty in Steubenville rape case," CNN (17 March), at http://edition.cnn.com/2013/03/17/justice/ohio-steubenville-case,

AL NABKI, M. W., FIDALGO, E., ALEGRE, E. & DE PAZ, I. 2017. Classifying Illegal Activities on Tor Network Based on Web Textual Contents.

ALSABAH, M., BAUER, K., ELAHI, T. & GOLDBERG, I. The path less travelled: Overcoming tor's bottlenecks with traffic splitting. International Symposium on Privacy Enhancing Technologies Symposium, 2013. Springer, 143-163.

ALSABAH, M. & GOLDBERG, I. PCTCP: per-circuit TCP-over-IPsec transport for anonymous communication overlay networks. Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, 2013. ACM, 349360.

ALSABAH, M. & GOLDBERG, I. 2016. Performance and security improvements for Tor: A survey. *ACM Computing Surveys (CSUR),* 49**,** 32.

AMBER RUDD, E. H. 2016. Investigatory Powers Act 2016. *In:* OFFICE, H. (ed.). UK parliament: Home Office.

ANTITREE. 2016. *Private Tor* [Online]. Github - Docker repositories. Available: https://hub.docker.com/r/antitree/private-tor/

ARMA. 2011. *Research problem: better guard rotation parameters* [Online]. TorBlog. Available: https://blog.torproject.org/blog/research-problem-better-guardrotation-parameters

ARTHUR, C. 2016. *Tesco cyber-raid raises serious questions over UK banks' security*
[Online]. United Kingdom: The Guardian. Available:
https://www.theguardian.com/business/2016/nov/12/tesco-cyber-theft-seriousquestions-bank-security.
ATLAS. 2017. *flag:authority* [Online]. Atlas. Available: https://atlas.torproject.org/ - search/flag:authority

BACKES, M., KATE, A. & MOHAMMADI, E. Ace: an efficient key-exchange protocol for onion routing. Proceedings of the 2012 ACM workshop on Privacy in the electronic society, 2012. ACM, 55-64.

BALL, J., BORGER, J. & GREENWALD, G. 2013. Revealed: how US and UK spy agencies defeat internet privacy and security. *The Guardian,* 6.

BAUER, K., MCCOY, D., GRUNWALD, D., KOHNO, T. & SICKER, D. Low-resource routing attacks against tor. Proceedings of the 2007 ACM workshop on Privacy in electronic society, 2007. ACM, 11-20.

BBC. 2015. *Details of UK website visits 'to be stored for year'* [Online]. BBC: United Kingdom. Available: http://www.bbc.co.uk/news/uk-politics-34715872

BBC. 2007. http://news.bbc.co.uk/2/hi/technology/6740075.stm

BERGSTRÖM, A. 2015. Online privacy concerns: A broad approach to understanding the concerns of different groups for different uses. *Computers in Human Behavior,* 53**,** 419-426.

BIGGS, H. 2010. *Healthcare research ethics and law: regulation, review and responsibility*, Routledge-Cavendish Abingdon.

BIRYUKOV, A., PUSTOGAROV, I., THILL, F. & WEINMANN, R.-P. Content and popularity analysis of Tor hidden services. Distributed Computing Systems Workshops (ICDCSW), 2014 IEEE 34th International Conference on, 2014. IEEE, 188-193.

BIRYUKOV, A., PUSTOGAROV, I. & WEINMANN, R.-P. Trawling for tor hidden services: Detection, measurement, deanonymization. Security and Privacy (SP), 2013 IEEE Symposium on, 2013. IEEE, 80-94.

BORISOV, N., DANEZIS, G., MITTAL, P. & TABRIZ, P. Denial of service or denial of security? Proceedings of the 14th ACM conference on Computer and communications security, 2007. ACM, 92-102.

BOUDREAU, M.-C., GEFEN, D. & STRAUB, D. W. 2001. Validation in information systems research: a state-of-the-art assessment. *MIS quarterly***,** 1-16.

BRYMAN, A. 2015. *Social research methods*, Oxford university press.

BYRNE, J. M. & KIMBALL, K. A. 2017. Inside the Darknet: Techno-Crime and criminal opportunity. *Criminal Justice Technology in the 21st centrury***,** 206.

CALEB, 2019 https://www.expressvpn.com/blog/best-browsers-for-privacy/.

CAI, X., ZHANG, X. C., JOSHI, B. & JOHNSON, R. Touching from a distance: Website fingerprinting attacks and defenses. Proceedings of the 2012 ACM conference on Computer and communications security, 2012. ACM, 605-616.

CANONICAL. 2017a. *Ubuntu* [Online]. Canonical Ltd. Available: https://www.ubuntu.com/about.

CANONICAL. 2017b. *Ubuntu Releases* [Online]. Canonical. Available: http://releases.ubuntu.com/.

CARTIGNY, J. 2014. *Netkit-ng* [Online]. Github, Inc. Available: https://netkitng.github.io/ [Accessed 30 June 2017].

CATALANO, D., FIORE, D. & GENNARO, R. Certificateless onion routing. Proceedings of the 16th ACM conference on Computer and communications security, 2009. ACM, 151-160.

CHAUM, D. L. 1981. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM,* 24**,** 84-90.

CHEN, C. 2020 https://www.privateinternetaccess.com/blog/the-firefox-browser-is-a-privacy-nightmare-on-desktop-and-mobile/

CHGANS. 2014. *tor-exit-notice* [Online]. GitHub. Available: https://github.com/chgans/tor-exit-notice.

CHONEY, S. 2015. *Microsoft announces new Cyber Defense Operations Center, Enterprise Cybersecurity Group* [Online]. https://blogs.microsoft.com/: Microsoft Available:

https://blogs.microsoft.com/firehose/2015/11/17/microsoftannounces-new-cyber-defense-operations-center-enterprise-cybersecurity-group/

CHOOPA, L. C. 2017. *This is a Tor Exit Router* [Online]. CHOOPA. Available: http://107.191.56.192/

CHRISTIN, N. Traveling the Silk Road: A measurement analysis of a large anonymous online marketplace. Proceedings of the 22nd international conference on World Wide Web, 2013. ACM, 213-224.

COHEN, H., FREY, G., AVANZI, R., DOCHE, C., LANGE, T., NGUYEN, K. & VERCAUTEREN, F. 2005. *Handbook of elliptic and hyperelliptic curve cryptography*, CRC press.

COHEN, A. 2012. "will we ever get strong intenet privacy"https://ideas.time.com/2012/03/05/will-we-ever-get-strong-internet-privacy-rules/?xid=gonewsedit

COMBS, G. 2017. *Wireshark* [Online]. Wireshark.org. Available: https://www.wireshark.org/ - news

CONSORTIUM, U. O. C. B.-I. S. 2016. *BIND9* [Online]. US: Internet Systems Consortium Available: https://www.isc.org/downloads/bind/doc/

COSTA, K., GJERMUNDRØD, H. & DIONYSIOU, I. privacyTracker: A Privacy-byDesign GDPR-Compliant Framework with Verifiable Data Traceability Controls. International Conference on Web Engineering, 2016. Springer, 3-15.

CRESWELL, J. W. & CLARK, V. L. P. 2007. Designing and conducting mixed methods research.

DAVENPORT, T. H. 2013. *Process innovation: reengineering work through information technology*, Harvard Business Press.

DEBIAN.ORG. 2013. *Package: debian-keyring (2013.04.21)* [Online]. Debian. Available: https://packages.debian.org/wheezy/debian-keyring

DeCew, J. 2006. "Privacy." In Stanford Encyclopedia of Philosophy. Stanford, CA: Metaphysics Research Lab, Center for the Study of Language and Information, Stanford University, 2006. http://plato.stanford.edu/entries/privacy/.

DELICHATSIOS, S. A; SONUYI, T. 2005, "Get to Know Google...Because They Know You", MIT, Ethics and Law on the Electronic Frontier, 6.805.

DINGLEDINE, R. 2011. *hellais/torspec* [Online]. GitHub. Available: https://github.com/hellais/torspec/blob/master/tor-spec.txt

DINGLEDINE, R., HOPPER, N., KADIANAKIS, G. & MATHEWSON, N. One fast guard for life (or 9 months). 7th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2014), 2014.

DINGLEDINE, R., MATHEWSON, N. & SYVERSON, P. 2004. Tor: The secondgeneration onion router. DTIC Document.

DURKLIN, P. 2019. https://nakedsecurity.sophos.com/2019/05/05/mozilla-bug-throws-tor-browser-users-into-chaos/

EASH, D. "Google Chrome Privacy Problems." Small Business - Chron.com, http://smallbusiness.chron.com/google-chrome-privacy-problems-28257.html.

EDMAN, M. & SYVERSON, P. AS-awareness in Tor path selection. Proceedings of the 16th ACM conference on Computer and communications security, 2009. ACM, 380-389.

ELAHI, T., BAUER, K., ALSABAH, M., DINGLEDINE, R. & GOLDBERG, I.

Changing of the guards: A framework for understanding and improving entry guard selection in Tor. Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society, 2012. ACM, 43-54.

ELGO. 2017. *#8215 Simple Relay: random unknown UDP port in listen mode* [Online]. torproject.org. Available: https://trac.torproject.org/projects/tor/ticket/8215

EVANS, N. S., DINGLEDINE, R. & GROTHOFF, C. A Practical Congestion Attack on Tor Using Long Paths. USENIX Security Symposium, 2009. 33-50.

FIELDER, A., PANAOUSIS, E., MALACARIA, P., HANKIN, C. & SMERALDI, F. 2016. Decision support approaches for cyber security investment. *Decision Support Systems,* 86**,** 13-23.

FIXSEN, D. L., NAOOM, S. F., BLASE, K. A. & FRIEDMAN, R. M. 2005. Implementation research: a synthesis of the literature.

FLICKER, S., TRAVERS, R., GUTA, A., MCDONALD, S. & MEAGHER, A. 2007. Ethical dilemmas in community-based participatory research: Recommendations for institutional review boards. *Journal of Urban Health,* 84**,** 478-493.

FORTE, A., ANDALIBI, N. & GREENSTADT, R. 2017. Privacy, anonymity, and perceived risk in open collaboration: a study of Tor users and Wikipedians. *PDF). Proceedings of Computer-Supported Cooperative Work and Social Computing (CSCW). Portland, OR. CSCW,* 17**,** 12.

FOSSIES. 2017. *Source code changes of the file "ChangeLog" of Tor* [Online]. United States: Fresh Open Source Software Archive. Available: https://fossies.org/diffs/tor/0.2.9.10_vs_0.3.0.4-rc/ChangeLog-diff.html

FOUNDATION, E. F. 2017. *Tor and HTTPS* [Online]. United States: Electronic Frontier Foundation. Available: https://www.eff.org/pages/tor-and-https.

FREEPTO. 2016. *AvANa-BBS/freepto-lb* [Online]. GitHub. Available: https://github.com/AvANa-BBS/freepto-lb..

GABERT, K. 2017. *TorStatus -Tor Network Status* [Online]. Tor: TorStatus. Available: http://torstatus.blutmagie.de/ -CustomDisplay.

GEDDES, J., JANSEN, R. & HOPPER, N. How low can you go: Balancing performance with anonymity in Tor. International Symposium on Privacy Enhancing Technologies Symposium, 2013. Springer, 164-184.

GEDDES, J., JANSEN, R. & HOPPER, N. IMUX: Managing tor connections from two to infinity, and beyond. Proceedings of the 13th Workshop on Privacy in the Electronic Society, 2014. ACM, 181-190.

GEIST, M. 2014. "The importance of online anonymity," Star (14 November), at http://www.thestar.com/business/2014/11/14/the_importance_of_online_anonymity_geist.html,

GELINAS, L., PIERCE, R., WINKLER, S., COHEN, I. G., LYNCH, H. F. & BIERER, B. E. 2017. Using social media as a research recruitment tool: Ethical issues and recommendations. *The American Journal of Bioethics,* 17**,** 3-14.

GELLMAN, B. & POITRAS, L. 2013. US, British intelligence mining data from nine US Internet companies in broad secret program. *The Washington Post,* 6.

GIRRY, K. T., OHZAHATA, S., WU, C. & KATO, T. 2015. Reducing Congestion in the Tor Network with Circuit Switching. *Journal of Information Processing,* 23**,** 589602.

GOLDBERG, I. On the security of the tor authentication protocol. International Workshop on Privacy Enhancing Technologies, 2006. Springer, 316-331.

GOLDSCHLAG, D., REED, M. & SYVERSON, P. Hiding routing information. Information Hiding, 1996. Springer, 137-150.

GOLDSCHLAG, D., REED, M. & SYVERSON, P. 1999. Onion routing. *Communications of the ACM,* 42**,** 39-41.

GOPAL, D. & HENINGER, N. Torchestra: Reducing interactive traffic delays over Tor. Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society, 2012. ACM, 31-42.

GORDON, L. A., LOEB, M. P., LUCYSHYN, W. & ZHOU, L. 2015. Increasing cybersecurity investments in private sector firms. *Journal of Cybersecurity,* 1**,** 317.

GOTTSCHALK JR, H. E., CALDWELL, M. & CARLETON, J. 2017. System and methods for identifying compromised personally identifiable information on the internet US Patent 20,170,053,369.

GREENWALD, G. 2013a. NSA collecting phone records of millions of Verizon customers daily. *The Guardian,* 6**,** 13.

GREENWALD, G. 2013b. XKeyscore: NSA tool collects' nearly everything a user does on the internet'. *The Guardian,* 31.

GREENWALD, G. 2014. *No place to hide: Edward Snowden, the NSA, and the US surveillance state*, Macmillan.

GREENWALD, G. & MACASKILL, E. 2013. NSA Prism program taps in to user data of Apple, Google and others. *The Guardian,* 7**,** 1-43.

GREIG, J. 2019. https://www.techrepublic.com/article/certificate-issue-disabling-add-ons-in-firefox-and-tor-browser-finally-fixed/?ftag=CMG-01-10aaa1b

HALL, R. 2008. *Applied social research: Planning, designing and conducting real-world research*, Macmillan Education AU.

HAMMOND, P. 2016. The UK will be one of the safest places in the world to do business, with a world-class cyber security industry and workforce thanks to a new plan underpinned by £1.9 billion of investment. United Kingdom: Cabinet Office, HM Treasury.

HANKERSON, D., MENEZES, A. J. & VANSTONE, S. 2006. *Guide to elliptic curve cryptography*, Springer Science & Business Media.

HARATY, R. A. & ZANTOUT, B. 2014. The TOR data communication system. *Journal of Communications and Networks,* 16**,** 415-420.

HAYES, J. & DANEZIS, G. 2015. Guard sets for onion routing. *Proceedings on Privacy Enhancing Technologies,* 2015**,** 65-80.

HE, G. F., ZHANG, T., MA, Y. Y. & FEI, J. X. Protecting User's Privacy from BrowserBased Attacks. Applied Mechanics and Materials, 2014. Trans Tech Publ, 941945.

HEATON, J. 2008. Secondary analysis of qualitative data: An overview. *Historical Social Research/Historische Sozialforschung***,** 33-45.

HEN, A. 2016. US defence department funded Carnegie Mellon research to break Tor. *The Guardian*, 25 February 2016.

HERRMANN, D., WENDOLSKY, R. & FEDERRATH, H. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïvebayes classifier. Proceedings of the 2009 ACM workshop on Cloud computing security, 2009. ACM, 31-42.

HEWSON, C. & STEWART, D. W. 2016. Internet research methods. *Cryptography Engineering.* Wiley Online Library.

HOFMANN, M. 2011. Why IP Addresses Alone Don't Identify Criminals. Available: https://www.eff.org/deeplinks/2011/08/why-ip-addresses-alone-dont-identifycriminals.

HOLLOWAY, I. & GALVIN, K. 2016. *Qualitative research in nursing and healthcare*, John Wiley & Sons.

HOPPER, N., VASSERMAN, E. Y. & CHAN-TIN, E. 2010. How much anonymity does network latency leak? *ACM Transactions on Information and System Security (TISSEC),* 13**,** 13.

HORSMAN, G. 2016. The challenges surrounding the regulation of anonymous communication provision in the United Kingdom. *Computers & Security,* 56**,** 151162.

HYKES, S. & DOCKER, I. 2017. *Docker* [Online]. Docker, Inc. Available: https://www.docker.com/company.

IBM. 2017. *Bringing big data to the enterprise* [Online]. IBM. Available: https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html.

IETF 1996. Address Allocation for Private Internets. US: Network Working Group.

IHONVBERE, D. 2011 "Privacy how to lock down google browser"https://blog.techprognosis.com/2011/07/08/privacy-how-to-lock-down-googles-chrome-browser.html

IOVATION 2013. iovation Finds 30 Percent of Transactions Conducted From Tor are Fraudulent: To Expose the Higher Risk Levels Associated with Anonymous Internet Transactions, iovation Launches New Feature. United States: Iovation.

JAMES HUNT & BYRUM, C. 2014. *Upstart Intro, Cookbook and Best Practises* [Online]. Canonical Inc. Available: http://upstart.ubuntu.com/cookbook/

JANSEN, R., GEDDES, J., WACEK, C., SHERR, M. & SYVERSON, P. F. Never Been KIST: Tor's Congestion Management Blossoms with Kernel-Informed Socket Transport. USENIX Security Symposium, 2014a. 127-142.

JANSEN, R. & HOPPER, N. 2012. Shadow - Proceedings of the 19th Symposium on Network and Distributed System Security (NDSS). United States: Internet Society.

JANSEN, R., JOHNSON, A. & SYVERSON, P. 2013. LIRA: Lightweight incentivized routing for anonymity. NAVAL RESEARCH LAB WASHINGTON DC.

JANSEN, R., MILLER, A., SYVERSON, P. & FORD, B. 2014b. From onions to shallots: Rewarding Tor relays with TEARS. NAVAL RESEARCH LAB WASHINGTON DC.

JANSEN, R., SYVERSON, P. F. & HOPPER, N. Throttling Tor Bandwidth Parasites. USENIX Security Symposium, 2012. 349-363.

JANSEN, R., TSCHORSCH, F., JOHNSON, A. & SCHEUERMANN, B. 2014c. The sniper attack: Anonymously deanonymizing and disabling the Tor network. OFFICE OF NAVAL RESEARCH ARLINGTON VA.

JARDINE, E. 2016. Tor, what is it good for? Political repression and the use of online anonymity-granting technologies. *new media & society***,** 1461444816639976.

JOHNSON, A., WACEK, C., JANSEN, R., SHERR, M. & SYVERSON, P. Users get routed: Traffic correlation on Tor by realistic adversaries. Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, 2013. ACM, 337-348.

JOY, B. 2016. *vi - screen-oriented (visual) display editor* [Online]. IEEE, The Open Group. Available: http://pubs.opengroup.org/onlinepubs/9699919799/utilities/vi.html

JOY, W., KEN GREER, MIKE ELLIS & PLACEWAY, P. 2009. *TCSH* [Online]. Apple Inc. Available: https://developer.apple.com/legacy/library/documentation/Darwin/Reference/Ma nPages/man1/tcsh.1.html.

JULIEN IGUCHI-CARTIGNY, JEAN-BAPTISTE MACHEMIE & BITARELLE, B. 2014. *uml_dump* [Online]. Italy: Github. Available: https://netkitng.github.io/umldump.html.

KATE, A., ZAVERUCHA, G. & GOLDBERG, I. Pairing-based onion routing. International Workshop on Privacy Enhancing Technologies, 2007. Springer, 95112.

KEIZER, G. 2011 https://www.computerworld.com/article/2507804/ftc-calls-out-google-s-chrome-over-do-not-track.html

KELLEY, S. 2017. *DNSMASQ* [Online]. Open Source. Available: http://www.thekelleys.org.uk/dnsmasq/doc.html

KERLINGER, F. N. & LEE, H. B. 1999. Foundations of behavioral research.

KOCH, R., GOLLING, M. & RODOSEK, G. D. 2016. How Anonymous Is the Tor Network? A Long-Term Black-Box Investigation. *Computer,* 49**,** 42-49.

KOTHARI, C. R. 2004. *Research methodology: Methods and techniques*, New Age International.

KUGLER, L. 2015. Online privacy: regional differences. *Communications of the ACM,* 58**,** 18-20.

KUMARI, S., KARUPPIAH, M., DAS, A. K., LI, X., WU, F. & GUPTA, V. 2017. Design of a secure anonymity-preserving authentication scheme for session initiation protocol using elliptic curve cryptography. *Journal of Ambient Intelligence and Humanized Computing***,** 1-11.

KWON, A., ALSABAH, M., LAZAR, D., DACIER, M. & DEVADAS, S. Circuit fingerprinting attacks: Passive deanonymization of tor hidden services. USENIX Security, 2015.

LANDAU, S. 2014. Highlights from making sense of Snowden, part II: What's significant in the NSA revelations. *IEEE Security & Privacy,* 12**,** 62-64.

LEE, R. M., ASSANTE, M. J. & CONWAY, T. 2016. Analysis of the cyber attack on the Ukrainian power grid. *SANS Industrial Control Systems*.

LEE, T. B. 2013. Five ways to stop the NSA from spying on you. *The Washington Post*.

LING, Z., LUO, J., WU, K. & FU, X. Protocol-level hidden server discovery. INFOCOM, 2013 Proceedings IEEE, 2013a. IEEE, 1043-1051.

LING, Z., LUO, J., YU, W., FU, X., JIA, W. & ZHAO, W. 2013b. Protocol-level attacks against Tor. *Computer Networks,* 57**,** 869-886.

LING, Z., LUO, J., YU, W., FU, X., XUAN, D. & JIA, W. A new cell counter based attack against tor. Proceedings of the 16th ACM conference on Computer and communications security, 2009. ACM, 578-589.

LINUXCONFIG.ORG. 2014. *Debian apt-get wheezy sources.list* [Online]. LinuxConfig.org. Available: https://linuxconfig.org/debian-apt-get-wheezysources-list.

LYON, G. F. 2017. *nmap* [Online]. US: nmap.org. Available: https://nmap.org/book/man.html.

MACKENZIE, D. 2010. *UNAME* [Online]. Free Software Foundation, Inc. Available: https://linux.die.net/man/1/uname.

MANSFIELD-DEVINE, S. 2014. Tor under attack. *Computer Fraud & Security,* 2014**,** 15-18.

MARKOFF, J. 2010. "Taking the mystery out of Web anonymity," New York Times (3 July), at http://www.nytimes.com/2010/07/04/weekinreview/04markoff.html

MARLINSPIKE, M. 2016. Facebook messenger deploys signal protocol for end to end encryption.

MARTIN, K. 2017. Everyday cryptography: fundamental principles and applications. *Oxford University Press***,** 672.

MATHEWSON, N. 2005. *Interfacing with Tor: Clients and Controllers* [Online]. TorProject. Available: https://svn.torproject.org/svn/torctl/trunk/doc/howto.txt

MATIC, S., TRONCOSO, C. & CABALLERO, J. 2017. Dissecting Tor Bridges: a Security Evaluation of Their Private and Public Infrastructures.

MCCOY, D., BAUER, K., GRUNWALD, D., KOHNO, T. & SICKER, D. Shining light in dark places: Understanding the Tor network. International Symposium on Privacy Enhancing Technologies Symposium, 2008. Springer, 63-76.

MCLACHLAN, J., TRAN, A., HOPPER, N. & KIM, Y. Scalable onion routing with torsk. Proceedings of the 16th ACM conference on Computer and communications security, 2009. ACM, 590-599.

MEANTERM. 2016. *The Clocks: Alarm Clock, World Clock* [Online]. US: Meanterm Inc. Available: https://itunes.apple.com/gb/app/the-clocks-alarm-clock-worldclock/id403684793?mt=8.

MENTOR, C. J. & CHOW, M. 2016. The Onion Router and the Darkweb.

MICROSOFT. 2012. *Netstat* [Online]. Microsoft. Available: https://technet.microsoft.com/en-us/library/ff961504(v=ws.11).aspx.

MITTAL, P., KHURSHID, A., JUEN, J., CAESAR, M. & BORISOV, N. Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. Proceedings of the 18th ACM conference on Computer and communications security, 2011a. ACM, 215-226.

MITTAL, P., OLUMOFIN, F. G., TRONCOSO, C., BORISOV, N. & GOLDBERG, I. PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval. USENIX Security Symposium, 2011b. 31-31.

MONTGOMERY, D. C. 2008. *Design and analysis of experiments*, John Wiley & Sons.

MOORE, W. B., WACEK, C. & SHERR, M. Exploring the potential benefits of expanded rate limiting in tor: Slow and steady wins the race with tortoise. Proceedings of the 27th Annual Computer Security Applications Conference, 2011. ACM, 207216.

MORIO, H. & BUCHHOLZ, C. 2009. How anonymous are you online? Examining online social behaviors from a cross-cultural perspective. *Ai & Society,* 23**,** 297307.

MOUTON, J. 2011. *How to succeed in your master's and doctoral studies: A South African*, Van Schaik Publishers.

MUHAMMAD, H. 2015. *htop* [Online]. Github, die.net. Available: https://linux.die.net/man/1/htop.

MURDOCH, S. J. & DANEZIS, G. Low-cost traffic analysis of Tor. Security and Privacy, 2005 IEEE Symposium on, 2005. IEEE, 183-195.

NARAINE, R. 2008. https://www.zdnet.com/article/extremely-severe-vulnerabilities-in-opera-browser/

NOWLAN, M. F., WOLINSKY, D. I. & FORD, B. Reducing Latency in Tor Circuits with Unordered Delivery. FOCI, 2013.

OVERLIER, L. & SYVERSON, P. Locating hidden servers. Security and Privacy, 2006 IEEE Symposium on, 2006. IEEE, 15 pp.-114.

PANCHENKO, A., NIESSEN, L., ZINNEN, A. & ENGEL, T. Website fingerprinting in onion routing based anonymization networks. Proceedings of the 10th annual ACM workshop on Privacy in the electronic society, 2011. ACM, 103-114.

PARALLELS. 2017. *Parallels Desktop for Mac* [Online]. United States: Parallels Inc. Available: http://www.parallels.com/products/desktop/.

PARLIAMENT, E. 2016. *General Data Protection Regulation - REGULATION (EU) 2016/679, Directive 95/46/EC* [Online]. European Union: European Parliament, Council of the European Union. Available: http://eur-lex.europa.eu/legalcontent/EN/TXT/?uri=OJ%3AL%3A2016%3A119%3ATOC

PETER TOBIAS, BERND ECKENFELS & WHITEHOUSE, S. 2013. *HOSTNAME* [Online]. Sourceforge. Available: http://man7.org/linux/manpages/man1/hostname.1.html [Accessed 31 July 2017].

PHIL BLUNDELL & ECKENFELS, B. 2014. *Route* [Online]. Sourceforge, Github: Sourceforge. Available: http://man7.org/linux/man-pages/man8/route.8.html

PIZZONIA, M. & RIMONDINI, M. 2016. Netkit: network emulation for education. *Software: Practice and Experience,* 46**,** 133-165.

PORTER, J. 2020 https://www.theverge.com/2020/2/25/21152335/mozilla-firefox-dns-over-https-web-privacy-security-encryption

PREIBUSCH, S. 2015. Privacy behaviors after Snowden. *Communications of the ACM,* 58**,** 48-55.

PRIVATORIA. 2017. *Hide & change IP in 2 clicks - Anonymous Proxy extension for your browser* [Online]. Privatoria. Available: https://privatoria.net/anonymous_proxy/

PRIVACY POLICY – "Privacy & Terms" https://policies.google.com/privacy/archive/20120301

RADIN, N. B. 2014. CURRENT REPORT Pursuant to Section 13 or 15(d) of the Securities Exchange Act of 1934 Date of Report (date of earliest event reported): October 2, 2014. *In:* COMMISSION, U. S. S. A. E. (ed.). United States: SEC.

RAJAGOPAL, R., TOSH, J. K., COX, F. L., NGUYEN, P. F. & CHAMPION, J. T. 2016. Disposable Browsers and Authentication Techniques for a Secure Online User Environment. Google Patents.

RASHID, A., BARON, A., RAYSON, P., MAY-CHAHAL, C., GREENWOOD, P. & WALKERDINE, J. 2013. Who am i? analyzing digital personas in cybercrime investigations. *Computer,* 46**,** 54-61.

REARDON, J. & GOLDBERG, I. Improving Tor using a TCP-over-DTLS tunnel. Proceedings of the 18th conference on USENIX security symposium, 2009. USENIX Association, 119-134.

REDHAT. 2017. *Red Hat Enterprise Linux 3: Reference Guide* [Online]. Red Hat Enterprise. Available: https://access.redhat.com/documentation/enUS/Red_Hat_Enterprise_Linux/3/html/Reference_Guide/s1-bind-rndc.html.

REUTERS. 2015. US marshals to auction 50,000 bitcoins seized from Silk Road. *The Guardian*, 18 February 2015.

RFC826 1982. An Ethernet Address Resolution Protocol - Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. Internet Engineering Task Force, Network Working Group.

RFC1034. 1987. *Domain Names - Concepts and Facilities* [Online]. Internet Engineering Task Force, Network Working Group. Available: https://tools.ietf.org/html/rfc1034.

RFC1035. 1987. *Domain Names - Implementations and Specifications* [Online]. Internet Engineering Task Force, Network Working Group. Available: https://tools.ietf.org/html/rfc1035.

RFC1123. 1989. *Requirements for Internet Hosts -- Application and Support* [Online]. Internet Engineering Task Force, Network Working Group. Available: https://tools.ietf.org/html/rfc1123.

RIMONDINI, M. 2006. Emulating Computer Networks with Netkit. *4th International Workshop on Internet Performance, Simulation,Monitoring, and Measurement.* Rome: Computer Networks Research Group.

RIMONDINI, M. 2010. *NETKIT* [Online]. Github: Computer Networks Research Group. Available: http://wiki.netkit.org/man/man7/netkit.7.html.

RYAN, T. P. & MORGAN, J. 2007. Modern experimental design. *Journal of Statistical Theory and Practice,* 1**,** 501-506.

SANGEETHA, K. & RAVIKUMAR, K. 2015. A novel traffic dividing and scheduling mechanism for enhancing security and performance in the tor network. *Indian Journal of Science and Technology,* 8**,** 689-694.

SAUNDERS, M. N. 2011. *Research methods for business students, 5/e*, Pearson Education India.

SCRIVANO, G. 2017. *GNU Wget* [Online]. GNU Operating System, Free Software Foundation. Available: https://www.gnu.org/software/wget/.

SEBASTIAN. 2014. *#965 Redo DNS tests when exit policy changes from reject *; avoid when policy changes to reject ** [Online]. torproject.org. Available: https://trac.torproject.org/projects/tor/ticket/965.

SHACKELFORD, S. & CRAIG, A. 2014. Beyond the New'Digital Divide': Analyzing the Evolving Role of National Governments in Internet Governance and Enhancing Cybersecurity.

SHANE, S. & SOMAIYA, R. 2013. New Leak Indicates Britain and US Tracked Diplomats. *New York Times,* 16**,** A7.

SHAVERS, B. & BAIR, J. 2016a. Chapter 2 - The Tor Browser. *Hiding Behind the Keyboard.* Boston: Syngress.

SHAVERS, B. & BAIR, J. 2016b. Chapter 6 - Cryptography and Encryption. *Hiding Behind the Keyboard.* Boston: Syngress.

SHELOR, M. 2013. *SHASUM* [Online]. Apple Inc. Available: https://developer.apple.com/legacy/library/documentation/Darwin/Reference/Ma nPages/man1/shasum.1.html.

SHERR, M., BLAZE, M. & LOO, B. Scalable link-based relay selection for anonymous routing. Privacy Enhancing Technologies, 2009. Springer, 73-93.

SHILLAIR, R., COTTEN, S. R., TSAI, H.-Y. S., ALHABASH, S., LAROSE, R. & RIFON, N. J. 2015. Online safety begins with you and me: Convincing Internet users to protect themselves. *Computers in Human Behavior,* 48**,** 199-207.

SHIRAZI, F., GOEHRING, M. & DIAZ, C. Tor experimentation tools. Security and Privacy Workshops (SPW), 2015 IEEE, 2015. IEEE, 206-213.

SIEBER, J. E. & TOLICH, M. B. 2013. *Planning ethically responsible research*, Sage.

SILVERSTONE, R. 2017. *Media, technology and everyday life in Europe: From information to communication*, Routledge.

SNADER, R. & BORISOV, N. A Tune-up for Tor: Improving Security and Performance in the Tor Network. ndss, 2008. 127.

SNADER, R. & BORISOV, N. 2011. Improving security and performance in the Tor network through tunable path selection. *IEEE Transactions on Dependable and Secure Computing,* 8**,** 728-741.

SOCKS-PROXY. 2017. *My IP Hide is better than Socks Proxy* [Online]. Free Proxy List. Available: https://www.socks-proxy.net/.

SOLOVE, D.J. 2006. "A Taxonomy of Privacy." University of Pennsylvania Law Review 154, no. 3 (January 2006): 477–560.

SOLOVE, D. J. 2008. "'I've Got Nothing to Hide' and Other Misunderstandings of Privacy." George Washington University Law School, Washington, DC, May 5, 2008. http:// papers.ssrn.com/sol3/papers.cfm?abstract_idp998565.

STAFFORD, A. 2012. "Nailing trolls when anonymity puts them beyond the law's reach," Age (12 September), at http://www.theage.com.au/opinion/society-and-culture/nailing-trolls-when-anonymityputs-them-beyond-the-laws-reach-20120911-25qfe.html.

STEM. 2017. *Stem Docs - Votes by Bandwith Authorities* [Online]. TorProject. Available: https://stem.torproject.org/tutorials/examples/votes_by_bandwidth_authorities.ht ml.

SUN, Y., EDMUNDSON, A., VANBEVER, L., LI, O., REXFORD, J., CHIANG, M. & MITTAL, P. RAPTOR: Routing Attacks on Privacy in Tor. USENIX Security, 2015. 271-286.

SYVERSON, P. 2013. Practical vulnerabilities of the tor anonymity network. *Advances in Cyber Security: Technology, Operation, and Experiences*, 60.

THETORPROJECT. 2017a. *Orbot: Proxy with Tor* [Online]. United States: Google Store. Available: https://play.google.com/store/apps/details?id=org.torproject.android&hl=en_GB.

THETORPROJECT. 2017b. *The Tor Project - Tor repository* [Online]. Github. Available: https://github.com/torproject/tor

TIGAS, M. 2017. *Onion Browser - Secure & Anonymous Web with T* [Online]. Apple iTunes. Available: https://itunes.apple.com/us/app/onionbrowser/id519296448?mt=8

TOR26. 2017. *Consensus* [Online]. Tor26 - IP Address 86.59.21.38. Available: http://86.59.21.38/tor/status-vote/current/consensus/.

TORBJORN GRANLUND, DAVID MACKENZIE, PAUL EGGERT & MEYERING, J. 2016. Disk Utility. *In:* FREE SOFTWARE FOUNDATION, I. (ed.). Free Software Foundation, Inc.

TORPROJECT. 2010a. *Bittorrent over Tor isn't a good idea* [Online]. TorProject. Available: https://blog.torproject.org/blog/bittorrent-over-tor-isnt-good-idea

TORPROJECT. 2010b. *Tips for Running an Exit Node* [Online]. TorProject. Available: https://blog.torproject.org/running-exit-node

TORPROJECT. 2014. *Tor Abuse Templates: Before You Start* [Online]. TorProject. Available: https://trac.torproject.org/projects/tor/wiki/doc/TorAbuseTemplates

TORPROJECT. 2016. *A New Bridge Authority* [Online]. TorProject. Available: https://blog.torproject.org/blog/new-bridge-authority

TORPROJECT. 2017a. *All Downloads* [Online]. United States: TheTorProject. Available: https://www.torproject.org/download/download.html.en

TORPROJECT. 2017b. *Anonymity Online - Protect your privacy. Defend yourself against network surveillance and traffic analysis* [Online]. United States: TheTorProject. Available: https://www.torproject.org/index.html.en

TORPROJECT. 2017c. *Configuring a Tor relay* [Online]. United States: TorProject. Available: https://www.torproject.org/docs/tor-doc-relay.html.en

TORPROJECT. 2017d. *Consensus Health* [Online]. TorProject. Available: https://consensus-health.torproject.org/

TORPROJECT. 2017e. *doc/GoodBadISPs* [Online]. TorProject. Available: https://trac.torproject.org/projects/tor/wiki/doc/GoodBadISPs

TORPROJECT. 2017f. *How to verify signatures for packages* [Online]. TorProject. Available: https://www.torproject.org/docs/verifying-signatures.html.en

TORPROJECT. 2017g. *ReducedExitPolicy* [Online]. TorProject. Available: https://trac.torproject.org/projects/tor/wiki/doc/ReducedExitPolicy

TORPROJECT. 2017h. *The Tor Manual* [Online]. United States: Canonical Inc. Available: http://manpages.ubuntu.com/manpages/trusty/man1/tor.1.html

TORPROJECT. 2017i. *Tor Metrics* [Online]. United States: TheTorProject. Available: https://metrics.torproject.org/userstats-relay-country.html].

TORPROJECT. 2017j. *Tor: Sponsors* [Online]. United States: TheTorProject. Available: https://www.torproject.org/about/sponsors.html.en.

TORPROJECT. 2017k. *TorSpec* [Online]. Github: The Tor Project. Available: https://gitweb.torproject.org/torspec.git/tree/torspec.txt?id=f9e111ead769d48441d99b52039ef0ccbd3f2c62-n1311 .

TORPROJECT, T. 2013. *Tails* [Online]. TorProject, Tails. Available: https://tails.boum.org/about/index.en.html.

TREPTE, S., TEUTSCH, D., MASUR, P. K., EICHER, C., FISCHER, M., HENNHÖFER, A. & LIND, F. 2015. Do people know about privacy and data protection strategies? Towards the "Online Privacy Literacy Scale"(OPLIS). *Reforming European data protection law.* Springer.

TRUSTE. 2015a. TRUSTe Privacy Index - UK 2015 Consumer Confidence Edition. Available: https://www.truste.com/resources/privacy-research/uk-consumerconfidence-index-2015/.

TRUSTE. 2015b. TRUSTe Privacy Index - US 2015 Consumer Confidence Edition. Available: https://www.truste.com/resources/privacy-research/us-consumerconfidence-index-2015/.

TRUSTE, N. C. S. A. 2016a. UK Consumer Privacy Index 2016. Available: https://www.truste.com/resources/privacy-research/ncsa-consumer-privacyindex-gb/ .

TRUSTE, N. C. S. A. 2016b. US Consumer Privacy Index 2016. Available: https://www.truste.com/resources/privacy-research/ncsa-consumer-privacyindex-us/.

TSCHORSCH, F. & SCHEUERMANN, B. Tor is unfair—And what to do about it. Local Computer Networks (LCN), 2011 IEEE 36th Conference on, 2011. IEEE, 432440.

TUNG, L. 2019. https://www.zdnet.com/article/mozillas-firefox-70-is-out-privacy-reports-reveal-whose-cookies-are-tracking-you/

ULRICH DREPPER, SCOTT MILLER & MADORE, D. 2010. *Sha256sum* [Online]. Free Software Foundation. Available: http://manpages.ubuntu.com/manpages/xenial/man1/sha256sum.1.html.

VAN JACOBSON, CRAIG LERES & MCCANNE, S. 2018. *TCPDUMP* [Online]. United States: tcpdump.org. Available: http://www.tcpdump.org/tcpdump_man.html.

VIECCO, C. 2008. UDP-OR: A fair onion transport design. *Proceedings of Hot Topics in Privacy Enhancing Technologies (HOTPETS'08)*.

VILLA, M. 2017. *I2P* [Online]. Google Store. Available: https://play.google.com/store/apps/details?id=net.i2p.android&hl=en_GB [Accessed 26 August 2019].

VIRTUALBOX. 2018. *VirtualBox* [Online]. Oracle. Available: https://www.virtualbox.org/wiki/Downloads.

VMWARE. 2018. *VMWare Station* [Online]. VMWare. Available: https://www.vmware.com/.

WACEK, C., TAN, H., BAUER, K. S. & SHERR, M. An Empirical Evaluation of Relay Selection in Tor. NDSS, 2013. 24-27.

WANG, T., BAUER, K., FORERO, C. & GOLDBERG, I. 2012. Congestion-aware path selection for Tor. *Financial Cryptography and Data Security*, 98-113.

WANG, T., CAI, X., NITHYANAND, R., JOHNSON, R. & GOLDBERG, I. Effective Attacks and Provable Defenses for Website Fingerprinting. USENIX Security Symposium, 2014. 143-157.

WANG, X., CHEN, S. & JAJODIA, S. Network flow watermarking attack on low-latency anonymous communication systems. Security and Privacy, 2007. SP'07. IEEE Symposium on, 2007. IEEE, 116-130.

WEINBERG, Z., WANG, J., YEGNESWARAN, V., BRIESEMEISTER, L., CHEUNG, S., WANG, F. & BONEH, D. StegoTorus: a camouflage proxy for the Tor anonymity system. Proceedings of the 2012 ACM conference on Computer and communications security, 2012. ACM, 109-120.

WINKLER, S. & ZEADALLY, S. 2015. An analysis of tools for online anonymity. *International Journal of Pervasive Computing and Communications,* 11**,** 436-453.

WINTER, P., ENSAFI, R., LOESING, K. & FEAMSTER, N. 2016. Identifying and characterizing Sybils in the Tor network. *arXiv preprint*.

WINTER, P., KÖWER, R., MULAZZANI, M., HUBER, M., SCHRITTWIESER, S., LINDSKOG, S. & WEIPPL, E. Spoiled onions: Exposing malicious Tor exit relays. International Symposium on Privacy Enhancing Technologies Symposium, 2014. Springer, 304-331.

WESTIN, A. 1967. "The Origins of Modern Claims to Privacy." In Privacy and Freedom, pp. 8–22. New York: Atheneum, 1967.

WOOD, C. P. 1996. *TRACEROUTE* [Online]. die.net. Available: https://linux.die.net/man/8/traceroute.

YANG, L. & LI, F. mTor: a multipath Tor routing beyond bandwidth throttling.
Communications and Network Security (CNS), 2015 IEEE Conference on, 2015. IEEE, 479-487.

YANOW, D. & SCHWARTZ-SHEA, P. 2015. *Interpretation and method: Empirical research methods and the interpretive turn*, Routledge.

ZHUO, J. 2010. "Where anonymity breeds contempt," New York Times (29 November), at http://www.nytimes.com/2010/11/30/opinion/30zhuo.html,

ZHOU, L., KONG, N., SHEN, S., SHENG, S. & SERVIN, A. 2015. Inventory and Analysis of WHOIS Registration Objects.