Detecting Middlebox Interference on Applications

Shan Huang

Félix Cuadrado, Steve Uhlig

School of Electronic Engineering and Computer Science

Submitted for the degree of Master of Philosophy to the University of London

July 2017

Abstract

Middleboxes are widely used in today's Internet, especially for security and performance. Middleboxes classify, filter and shape traffic, therefore interfering with application behaviour and performing new network functions for end hosts. Recent studies have uncovered and studied middleboxes in different types of networks.

In order to understand the middlebox interference on traffic flows and explore the involved ASes, our methodology relies on a client-server architecture, to be able to observe both directions of the middlebox interaction. Meanwhile, probing with increasing TTL values provides us chances to inspect behaviour of middleboxes hop by hop.

Implementing our methodologies, we exploit a large-scale proxy infrastructure *Luminati*, to detect HTTP-interacting middleboxes across the Internet. We collect a large-scale dataset from vantage points distributed in nearly 10,000 ASes across 196 countries. Our results provide abundant evidence for middleboxes deployed across more than 1000 ASes. We observe various middlebox interference in both directions of traffic flows, and across a wide range networks, including mobile operators and data center networks.

Contents

List of Figures

List of Tables

Nomenclature

AS Autonomous System

AWS Amazon Web Services

BSD Berkeley Software Distribution

DNS Domain Name System

DNSSEC Domain Name System Security Extensions

DPI Deep Packet Inspection

ECN Explicit Congestion Notification

GFC Great Firewall of China

HTTP Hypertext Transfer Protocol

HTTPS HTTP Secure

IDS Intrusion Detection System

IP Internet Protocol

IPS Intrusion Prevention System

ISP Internet Service Provider

LAN Local Area Network

MTU Maximum Transmission Unit

NAT Network Address Translation

NXDOMAIN Non-Existent Domain

OONI Open Observatory of Network Interference

P2P Peer to Peer

PKI Public Key Infrastructure

SSL Secure Sockets Layer

TBIT TCP Behaviour Interference Tool

TCP Transmission Control Protocol

TLD Top Level Domain

TLS Transport Layer Security

TTL Time to Live

UDP User Datagram Protocol

URL Uniform Resource Locator

VPN Virtual Private Network

VPS Virtual Private Server

WAN Wide Area Network

Chapter 1

Introduction

1.1 Introduction

Middleboxes such as firewalls, load balancing switches and Deep Packet Inspection (DPI) boxes have been a major part in today's network infrastructure. A middlebox could be defined as any intermediary network device performing functions other than standard functions of an IP router on datagram between two end hosts [1]. Currently, the main goals driving the deployment of middleboxes are security (to enhance the visibility of network traffic and enable the enforcement of security policies) [2, 3, 4, 5] and performance enhancement (through traffic shaping, caching and transparent proxying) [2, 6, 7, 8].

In comparison with other Internet devices, such as switches and routers, middleboxes are complex, as they operate on packets from network layer to application layer at line rate. Middleboxes interfere with end-to-end packet transmission, application functionality, and restricting or preventing end host applications from performing properly [1, 9]. In general, the middlebox interference can be categorized into three types. First, middleboxes intentionally drop or filter packets as policies [10, 11]. For example, network administrators filter P2P file sharing traffic to avoid the legal implications of copyrighted files [12]. Second, middleboxes modify packet contents [13, 10, 5]. Some web proxies modify HTTP headers to control meta information between client and server (e.g., cache preferences). Finally, in order to perform other functions as net-

work policies, middleboxes inject forged packets, e.g., for blocking purposes. For instance, the Great Firewall of China (GFC) blocks some certain sites by injecting spoofed DNS responses, and causes intrusional damage in terms of Internet censorship [14]. As a summary, middleboxes have the ability to inspect and manipulate traffic flows, interfere with their end-to-end behaviour, and not only have positive impact on network traffic but also affect the behaviours of end host applications.

1.2 Motivation

Middleboxes are widely used in various types of networks. Indeed, according to a survey of 57 enterprise network administrators, it was concluded that there are probably as many middleboxes as routers inside the network [2]. Also, the survey of edge-network behaviour [15] showed the evidence of middlebox traffic manipulation in common ISPs. In addition, lots of governments use middleboxes to implement network censorship [16, 17, 18]. GFC is the most complex middlebox system that operates with numerous protocols ([19, 20, 21, 22, 23]). The governments of Iran, Yemen, Tunisia and Sudan use commercial software in state-controlled servers, and censor all outgoing flows [17, 24, 11]. It is claimed that middleboxes are distributed in everywhere of today's network. However, we have limited knowledge about where and how middleboxes interfere with traffic flows now.

At the same time, Internet traffic is changing (e.g., HTTPS accounts for a significant portion of Internet traffic now [25]). Considering the complexity of middleboxes, applications and network traffic, we argue that methodologies are needed to detect and analyse middlebox interference for different kinds of applications.

In this report, we analyse the potential impart of middleboxes in today's network and explain the reasons why we care about middleboxes; we summary the lessons learnt from prior work, and introduce our own measurement methodologies to detect and locate middlebox interference on traffic flows; we show the measurement results in a commercial platform which provides vantage points all over the world, and illustrate middlebox behaviour in different kinds of network.

1.3 Contribution

Our contributions are twofold. First, we introduce our methodology to detect and locate middlebox interference based on a client-server architecture. We also explain how to use the *Luminati* platform to run large-scale measurements and get a global dataset from our measurements.

Second, based on our methodology, we find evidence for a significant amount of middlebox interference on both directions of the traffic flows in different networks. According to our dataset, we observe a wide variety of injected HTTP headers in HTTP messages, some known and some never reported before. Surprisingly, we even observe new headers that are only added by mobile networks and cloud platforms.

Overall, we find that injected HTTP headers expose the presence of multiple types of middleboxes across diverse networks and most of middleboxes are at the edge of networks. Further, the interference on HTTP responses often reveals the corresponding functions of the middleboxes, such as proxying, caching, URL filtering, and WAN optimization.

This report is structured as follows: we present the introduction and motivation in this session. Afterwards, we review the status of current studies in Chapter 2. We present the measurement design process, and explain the methodology in Chapter 3. We present our study result in Chapter 4. At the end, in Chapter 5 and Chapter 6, we discuss our potential work and make a conclusion for this report.

Chapter 2

Literature Study

Middleboxes play a crucial role in the modern Internet infrastructure [24, 10, 2]. The growing benefits of using middleboxes and unknown middlebox interference on network traffic flows motivated researchers in technical and regulatory community to examine middleboxes from various aspects. In this chapter, we review some studies to understand the presence of middleboxes, the motivations of measuring behaviour of middleboxes, and essential methodologies of detecting middleboxes.

2.1 Middleboxes Taxonomy

The word "middlebox" was introduced by Lixia Zhang to describe new phenomenon in Internet. Afterwards, B.Carpenter defined middlebox in the work [1], "a middlebox is any intermediary device performing functions other than the normal, standard functions of an IP router on the datagram path between a source host and destination host." Meanwhile, a middlebox should not be clarified as a physical device necessarily, it could be virtual and interpreted as a network function.

As we know, the classical Internet architecture is based on hourglass model [26] and end-to-end principle [27]. In the old architecture, the only boxes between two end hosts are IP routers,

which provide the function of forwarding packets purely [28]. However, insertion of middleboxes breaks old network model. Middleboxes not only spread new functions throughout the network, but also bring new challenges for network communication sessions.

Furthermore, middleboxes are complex, they operate at network, transparent, upper layers, and even a mixture. RFC3234 [1] classifies middleboxes in a multidimensional taxonomy and gives examples to describe characterisations of them. For example, NAT, firewall, IDS/IPS, load balancer and web proxy are all examples for middleboxes, and they work across different layers for diverse purposes.

2.2 Middleboxes Today

Middleboxes are widely used and play an important role in different kinds of networks [9]. Moreover, the behaviour of middleboxes is complicated, while the impact on applications is poorly understood. In this subsection, we describe some prior work, state current deployments of middleboxes, and illustrate our motivations in details.

2.2.1 Middleboxes in Enterprise Networks

Sherry *et al.* [2] in 2012 presented a detailed study of middlebox deployments. In this study, the researchers conducted a survey of 57 enterprise network administrators, including the number of middleboxes deployed, personnel dedicated to middleboxes, and challenges faced in administering related middleboxes. Based on the survey, they undertook a systematic exploration of requirements and design spaces for outsourcing middleboxes, implemented *APLOMB*, a practical service for outsourcing enterprise middlebox processing to the cloud.

This work is the first large-scale survey of middlebox deployments in research community. Their dataset illustrates that typical enterprise network is a complex ecosystem of firewalls, IDSes, web proxies, and other network devices. The number of middleboxes in each enterprise is comparable to its number of traditional switches and routers. All of these are vital evidences to show middleboxes are widely used in enterprise network and affect the Internet infrastructure. On the other hand, the design of *APLOMB* aims to reduce cost of middlebox infrastructure, while redirecting traffic through *APLOMB* may cost bandwidth, affect application performance and bring

security challenges at the same time. For instance, as shown in evaluation part in this work, three common applications (HTTP, BitTorrent, VoIP) suffer little when their traffic is redirected through *APLOMB*. In summary, moving middleboxes to cloud solves the problems caused by cost, management complexity, and failures of middlebox infrastructure. But it is needed to understand how middleboxes deal with real traffic flows and take the middlebox interference into account.

2.2.2 Middleboxes in ISP Networks

Kreibich *et al.* [15] presented *Netalyzr*, a network measurement and debugging service that evaluates the properties of Internet access and connectivity in ISP networks. In this study, they have recorded 130,000 runs of the system from 99,000 different public IP addresses, constructing a large-scale picture of many facets of Internet edge behaviour (outgoing port filtering, hidden in-network HTTP caches, DNS manipulations, NAT behaviour, path MTU issues, IPv6 support, and access-modem buffer capacity) as well as tracking each behaviour's technological evolution over time.

The dataset of this work consists sessions from 6884 organizations across 186 countries, covering most ISP networks. The results reveal some systemic problems, such as difficulties with fragmentation, the unreliability of path MTU discovery, restrictions on DNSSEC deployment. Also, it shows evidence of DNS NXDOMAIN wildcarding, HTTP proxying and caching. Though, the original usage of *Netalyzr* was for Internet users to obtain analysis of their Internet connectivity, their findings illustrate some behaviour of middleboxes. For instance, some NATs and firewalls are DNS-aware devices, some HTTP proxies open new connections to end servers and inject new headers to HTTP requests.

The researchers also used *Netalyzr* in recent study of cellular networks [29, 30], showing that 13% (of 299 mobile operators) were manipulating headers [29], and part of these manipulating headers were modified by middleboxes for user privacy, security and network operations. At the same time, *Netalyzr* detected the presence of transparent HTTP proxies in 58% of cellular sessions (covering 296 operators in 119 countries), describing how proxies modified or blocked DNS and HTTP traffic [30].

Xu [31] undertook an analysis of proxy behaviour in 2015, and demonstrated how transparent web proxies interacted with HTTP traffic in four major US cellular carriers. With full control over the client devices and servers, researchers explored proxy properties through various experiment configurations, varying parameters (e.g. content to fetch), socket properties (e.g. server IP address or port), HTTP configuration (including modified headers), and even adjust network conditions.

Based on experiment results, researchers exhibited the application-level features of web proxies, such as HTTP caching, HTTP redirection, connection and so on. More specifically, there were three key conclusions characterizing transparent web proxies in ISPs. First, each carrier implements proxying policies differently, thus the speed and quality of downloaded contents for users are affected. Second, the split connections improve performance for larger flows (up to 45%), while they have negligible impact on small ones (less than 100KB). Last, proxies interpose on connections to almost all popular web sites, but Google's YouTube traffic could bypasses T-Mobile proxies, which maybe due to a special arrangements for certain content. All of these findings notice us the behaviour of web proxies is diverse, and their impact on performance for web workloads is varied. Moreover, end users may remain unaware of the existence of web proxy typically, we are motivated that it is necessary to explore large-scale and in-depth measurements to understand behaviour of middleboxes and their impact on applications.

The Open Observatory of Network Interference (OONI) [32] has processed some network measurements which aim to detect internet censorship, traffic manipulation and other signs of surveillance since 2012. The OONI project is under the Tor project, collecting millions of network tests across more than 90 countries. The researchers published the testing methodology to identify HTTP Header Field Manipulation and the collected HTTP headers on the website. According to the published data, we observed that lots of manipulated HTTP headers were injected by proxies, caches and other network middleboxes.

2.2.3 Middleboxes and Internet Censorship

Recent years, Internet censorship is achieved by using nation-wide middleboxes in some countries (e.g. China, Yemen, Sudan). The work [11] is a survey on Internet censorship detection which explains censoring mechanisms in detail. In this work, the researchers gave a charac-

terization of current censoring systems, proposed related analysis and discussion of censorship detection techniques. From this work, we find lots of evidence to describe the usage of middleboxes in censorship, such as, IP blocking, TCP reset injection, DNS injection, transparent proxy, HTTP warning page injection and so on. Additionally, the performance impact (e.g. increased packet loss and delay) made by middleboxes attracts us to detect and analyse middleboxes from multiple aspects.

Aryan *et al.* presented a network measurement about Internet censorship in Iran [33]. In this paper, the researchers investigated technical mechanisms and mapped the network topology of censorship infrastructure. This work highlights the fact that censorship in Iran relies heavily on centralized equipments (middleboxes). The main mechanisms which are used for censoring in Iran are IP filtering, DNS NXDOMAIN packet injection, DNS hijacking, HTTP warning page injection and TCP connection throttling.

Great Firewall of China (GFC) is a well-known middlebox system, featuring IP blocking, keyword filtering, DNS hijacking and so on [16]. Different from other detection work, Xu *et al.* [20] explored the location of Intrusion Detection System (IDS) devices of GFC placed for keyword filtering as AS and router level. The algorithm of this work was sending probe packets that contain known blacklist keywords with increasing TTL values. As opposed to previous work, the researchers did not use top websites as targeted servers, the selected servers from diverse provinces in China to find more filtering devices.

In short, this work gives an explanation of China's AS-level Internet topology. ASes in this work are categorized into two types, one is border AS which is directly peered with foreign ones, the others is internal AS. Not surprisingly, findings show that most of filtering devices belong to border ASes expect two internal filtering interfaces. These findings state the middlebox distribution in network topology, and provide better understanding for Chinese national-scale intrusion detection system.

2.3 Detecting Method

Internet has changed significantly now. For example, the increasing personalization of web service has led to a number of clients and servers adopting HTTPS [34]. The work [34] is a study of HTTPS which categorizes and quantifies the cost of 'S' in HTTPS. By collecting data from large ISPs, this work claims that 50% of web traffic flows today are secure. In other words, HTTPS is widely used in web services. However, encryption provides confidentiality and authentication to end users, but it renders all transparent and explicit middleboxes (e.g. web proxy, DPI) ineffective at the same time. Moreover, HTTPS affects the protocol-related performance for applications, e.g., increasing latency, negative impact on battery life. To manage the trade off between security and network functions on the path, we need to detect and understand middleboxes firstly.

In this section, we describe and review prior work which introduces the detecting methods of middleboxes. As shown in Table 2.1, we summary and compare detecting methods with different criteria. In brief, The work ([35, 10, 13]) detected middlebox interference on IP and TCP protocols in different kinds of networks, especially illustrated packet header modifications in each layer. The one [7] employed a range of detectors at transport and application layers for identifying proxies specifically. The new tools of middlebox detection were also designed, and implemented in both computer [5] and mobile device versions [36].

2.3.1 Interactions Between Transport Protocols and Middleboxes

The first detailed analysis of interactions between protocols (especially transport protocols) and middleboxes was published in 2004 [35]. The researchers used active measurement to assess the behaviour of middleboxes, illustrated the fact that performance of protocol mechanisms in Internet was quite different from theory.

This work used and extended the methodology from prior work [37] on TCP Behaviour Inference Tool (TBIT). TBIT is a tool that characterizes TCP behaviour of a remote web server. TBIT establishes a TCP connection with remote host, uses raw IP sockets to send segments, and sets up BSD filter to deliver incoming packets to TBIT analysing process. However, the detecting process of TBIT is limited by network conditions to some degree. For instance, the packet losses

may lead the test wrongly reported. In this work, the researchers took each test specifically, and used large set of web servers to make tests as robust as possible. In the section of results, this work detected middlebox interference on Explicit Congestion Notification (ECN), Path MTU Discovery, IP options, TCP options, and examined whether all of these headers could be used safely.

As the first work to measure middlebox interference, this work makes a positive beginning to understand behaviour and effect of middleboxes on basic TCP headers (emphasizing the end-to-end network principle is broken). While, it leaves an open question that how middleboxes affect the *performance* of transport protocols or applications. More active tests with application traffic flows may be useful to investigate other middlebox interference.

2.3.2 Middlebox Interference on TCP Extensions

In 2011, Honda *et al.* [13] developed a tool (*TCPExposure*) made of a client and a server to determine whether middleboxes affect the performance of TCP option headers across diverse networks. In particular, this work clarifies the limitations on TCP extensions imposed by middleboxes, discusses some questions about TCP options in real network. For example, what will and will not pass through middleboxes, whether there are modifications in TCP option headers or how middleboxes affect the transmission of TCP flows.

To answer these questions, the researchers conducted a study from 142 networks in 24 countries, including cellular, WiFi and wired networks.

The initiator (acting like a TCP client) and the responder (a TCP server) run tests to trigger on-path middlebox interference. The initiator transmitted crafted segments that included bytes indicating commands in their payloads. The responder replied with a segment contains both received and returned headers in payloads for identifying manipulations. Additionally, the generated traffic flows were mimicked with new TCP extensions, and examined destination ports were varied.

As a summary, this work gives an example to measure middlebox impact on TCP options systematically. First, it examines the middlebox interference on TCP layer across diverse networks (public, private, residential, commercial and academic networks). The satisfying conclusions are

middleboxes are widely used for implementing functions for network layer, and middleboxes do not affect the use of TCP options. Second, the idea of controlling both ends of the path is significant for middlebox detection, as it provides the access to generate, track and analyse traffic flows. Final, it guides the future design of TCP extensions, attracting the attentions of researchers to think about compatibility of middleboxes in network deployment. However, there are some other details needed to be discussed. Since main mimicked traffic flows in experiment were SYN/SYNACK TCP segments, this method could not detect the middlebox interference on performance of applications. Differently, the tool *HICCUPS*([38]) integrated middlebox detection into common TCP connection process and presented relative tests with application layer payloads. In addition, middleboxes interfere with different protocols. It is better to create a high-level taxonomy of the testing tool, not only be used for specific protocols.

2.3.3 Middlebox Detection in Cellular Network

Today's growing cellular traffic demand leads to great challenges for cellular network carriers to provide users good network performance, and protect mobile devices from potential attacks [10]. The goal of this study is to gain insight into middleboxes' policies, specific NAT and firewall in cellular networks, and make targeted improvement for mobile applications.

In methodology part, the data was collected from about 400 users in 107 carriers across 6 continents. Researchers introduced *NetPiculet*, a tool comprised of client software running on mobile devices inside cellular networks and a dedicated server in Internet. The client and server sent end-to-end probes to infer middleboxes' policies and quantify their impact based on traces.

This work is the first large-scale system that effectively discovers cellular network policies. It illustrates a diversity of NAT and firewall polices and explains important implications for network carriers as well as mobile application developers. It provides methodologies and results to understand NAT and firewall in cellular network. However, although the target of this work is to detect middleboxes, the detected policies only cover NAT and firewall in cellular network. Middleboxes are much more complex and diverse, and therefore require considering wider interactions.

2.3.4 Proxy Detection

As an extensive measurement of *Netalyzr*, the work [7] presented the results of experiment probing for presence of web proxies by establishing HTTP connections from end-user browsers to custom web servers under researchers' control. Running *Netalyzr*, 14% of clients show evidence of HTTP proxying through one or more tests.

The *Netalyzr* tool uses a java applet to drive the bulk of test communication with their programmed servers. A Java applet is typically embedded inside a web page and runs in the context of a browser [39]. Precisely, java applets run automatically within most major web browsers, engage in raw TCP and UDP flows to arbitrary port and come with intrinsic security guarantees for users. These advantages provide sufficient flexibility to conduct measurement tests and a simple enough interface for unsophisticated users run it (giving access to a much larger user population). When the user visits *Netalyzr* website, browser downloads and executes the applet. The applet conducts test connections to various measurement servers. Test results and raw network traffic traces in servers are stored for later analysis.

More specifically, this work detected web proxies based on two main strategies: TCP termination and packet rewriting. A proxy employing TCP termination separates TCP connection with the controlled server, relaying the content stream from both endpoints. To indicate the presence of a TCP-terminating proxy, they used non-responsive server to send a RST packet to all incoming requests. If *Netalyzr* clients make a connection on port 80 successfully, it provides evidence that there is a TCP-terminating proxy on path. On the other hand, a packet-rewriting proxy may modify traffic or inject additional traffic as flows through it. By checking the received HTTP responses, modifications in HTTP headers, HTML documents and transfer encoding in a manner can presence the packet-rewriting proxy.

In addition, this work added a new test to point the location of TCP-terminating proxy. For any port on which previous test flagged the presence of a terminating proxy, *Netalyzr* client attempted a TCP connection to a *traceroute server* which conducted a traceroute to respond client using SYN-ACK packets. The traceroute terminated upon receiving TCP handshake's pure ACK, rather than an ICMP "TTL exceeded" response. The hop which returned last ICMP time-exceeded reply is the closest hop near the TCP-terminated middleboxes.

Furthermore, the researchers repeated requests with different cache-control headers to check and compare the received contents, detecting the behaviours of caching, transcoding and other performance of HTTP.

As a summary, this work detects the presence of proxies, identifies the functions, and introduces a TCP-terminating proxy location technique based on traceroute. However, the detection of this work is based on TCP termination and packet manipulation. It is hard to detect transparent proxy which does not open a new connection or modify packet contents. Making use of traceroute is a key contribution to locate proxies, but the locating method in this work is only designed for TCP-terminating proxy. In fact, the locating method only works for transport layer. At the country level, they find that some use proxies for nation-wide censorship, such as Bahrain, Singapore, Lebanon, the United Arab Emirates and so on. Although it could be evidence to prove the Internet censorship is used in these countries, only controlling server side and focusing on behaviour of proxies are not enough to explain the Internet censorship accurately. Overall, this work is a empirical study of web proxy, while we need more general methodologies to detect and locate all kinds of middleboxes.

2.3.5 Traceroute-style Detecting Tool

Detal *et al.* proposed *tracebox* [5], an extension of the traceroute tool, that is capable of detecting middlebox interference. Same with traceroute, tracebox sends probes with increasing TTL values and waiting for ICMP time-exceeded replies. Differently, traceroute uses quoted packet inside of ICMP replies to identify intermediate routers, tracebox uses it to infer modifications applied on probes by intermediate middleboxes and pinpoint where a given modification takes place. The researchers deployed tracebox on PlanetLab [40], using 72 machines as vantage points. Each vantage point had a target list of 5000 domains built with top 5,000 Alexa web sites. They found some vantage points removed or changed TCP options at the very first hop. Also, the modification of TCP sequence number occurred on the path of two experiment cases.

In brief, it is a key contribution of making use of ICMP time-exceeded replies to detect and locate middleboxes. However, detecting methods are limited by the types of ICMP replies. When IPv4 router receives an IPv4 packet whose TTL is going to expire, it returns an ICMPv4 time-exceeded message that contains offending packets. According to RFC792, the returned ICMP

packet should quote the IP header of original packet and the first 64 bits of payload of this packet [41]. Different with this, RFC1812 [42] recommends to quote the entire IP packet in the returned ICMP, but this recommendation has only been implemented on some routers. When tracebox receives ICMP replies with full quoted original packet, it is able to detect any changes in each layer. From the result of PlanetLab use case, the number of router that replies with a full ICMP is too few to cover all paths.

On the other hand, the presence of middleboxes is inferred only by packet modification. As we mentioned before, the behaviour of middleboxes and application traffic flows are both complex. The interference could not only be modification, but also filtering or injection. Tracebox could detect the modification of bytes in each packet, but it is not enough for detecting other middlebox interference. In addition, lacking of controlling in server side, tracebox merely explores modification performed by upstream middleboxes.

TraceboxAndroid [36] is a *proof-of-concept* measurement application for Android mobile devices implementing the tracebox algorithm. It is a tool for smartphones to infer middlebox behaviour in WiFi and cellular networks. Although this tool has same limitations as tracebox, the results list more impact and understanding of mobile devices (e.g. middlebox TCP option blocking).

Craven *et al.* proposed TCP HICCUPS [38] in 2014 to reveal packet header manipulation (specific for TCP header) to both sides of a TCP connection. To detect modifications, HICCUPS hashed packet headers and spread the resulting hash into three fields of the original headers (in case a change to any header field). Running HICCUPS across a distributed and diversity set of paths, the researchers discovered a wide variety of behaviours in both forwarding and receiving directions, but as displayed in the tool name, it purely works for TCP traffic flows.

2.4 Summary and Discussion

As a summary, we explain the definition of middlebox, illustrate the reasons that why we care about middleboxes, and discuss existing detecting methods in prior work. Middleboxes are intermediary devices that performing new network functions other than forwarding packets. While, the insertion of middleboxes brings new challenges, such as compatibility of protocols, privacy

of network traffic and so on. On the other hand, middleboxes are extensively deployed for security and performance enhancement in networks, and interfere with traffic flows in varied ways. All of these reveal that middleboxes are important in current Internet and affect the behaviour of applications, then methodologies are needed to detect and measure them.

Table 2.1: Detecting Method Comparison

Method	Client Root Access	Server Root Access	Locating	Support Protocols	Multiple Interference
TBIT [35]	✓	×	×	IP,TCP	×
TcpExposure [13]	✓	✓	×	IP,TCP	✓
NetPiculet [10]	✓	✓	×	IP,TCP	✓
Netalyzr [7]	×	✓	✓	HTTP	×
Tracebox [5]	✓	×	✓	IP,TCP	×
HICCUPS [38]	✓	✓	×	IP,TCP	×

As shown in Table 2.1, we discuss prior methodologies from different aspects. Learnt from them and achieve our own motivations, we propose our methodologies in three parts. First, both end hosts should be controlled for generating, capturing, and analysing traffic flows. Second, client and server are programmed for working as applications, the traffic flows should contain payloads of application layers. Last, detecting methods are based on different interference of middleboxes (filtering, injection, modification), and could be used for large-scale measurement generally.

Chapter 3

Methodology

Our work aims to detect and locate the behaviour of middleboxes, especially measuring the middlebox interference on applications. In this chapter, we explain the methodologies of our work conscientiously.

Our methods adopt a client-server architecture, allowing to generate probe packets from clients and servers, compare the expected traffic exchanged between end hosts and the one actually received (Section 3.1). Also, we describe three types of middlebox interference, illustrating the detecting process with controlling of both end hosts (Section 3.2). Different with prior work, our probe packets are crafted with application layer payloads that attempt to expose middlebox interference with application traffic flows. In Section 3.3, we describe our application probes, explaining the process of samples. As followed, we describe our traceroute-like probing with increasing TTL values, illustrating how to locate the approximate position range of the interfering middlebox by these methods. Finally, we describe how to adopt our methodologies with *Luminati*, the commercial Peer-to-Peer (P2P)-based HTTP/S proxy service, launching crafted HTTP requests across all over the world.

3.1 Client-Server Architecture

Our methods control both client and server sides, making use of root access to generate, capture and compare traffic flows. As shown in Figure 3.1, we treat the middlebox as a black box, and detect the interference by checking the differences between exchanged traffic flows and original probes. Client-server architecture provides visibility of the network between end hosts, observing the difference between upstream and downstream traffic. Different from bit modification detection, e.g. tracebox [5], our end hosts are programmable to generate probes from varied layers, make connections, and respond to queries as running real applications. Filtering, injection, and modification are inferred by comparing the traces from client and server.

The code of client makes up of lots of bash commands to make connection, send queries and capture traces at the same time. It is compatible to run from terminals of the Linux system machines. Based on multiple application layer protocols, the server listens on particular ports and returns responses as the format of varied application servers, such as DNS resolver, web server and so on.

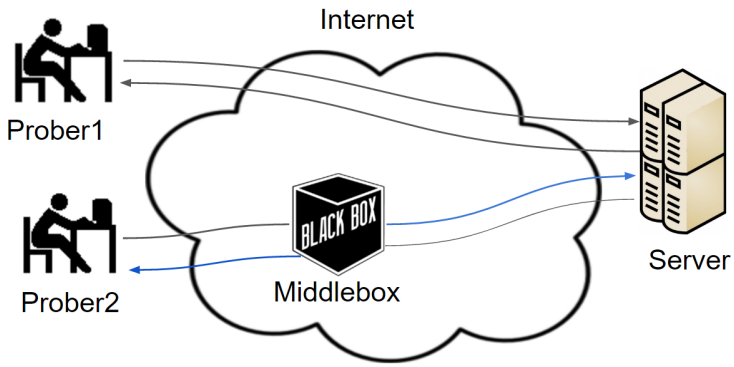

Figure 3.1: Client-Server Architecture

3.2 Detecting Process

Our methods detect three types of interference: filtering, injection and modification. In this subsection, we describe the interference with different use cases, explain the process of detection and implementation with diagrams, and finally discuss how to identify the interference by analysing captured traces.

3.2.1 Filtering

Theoretically, one end host should receive the original probes from source host. If not, the probe may have been lost or filtered by some middleboxes. As shown in Figure 3.2, the middlebox drops the packets for purpose when packets go through it.

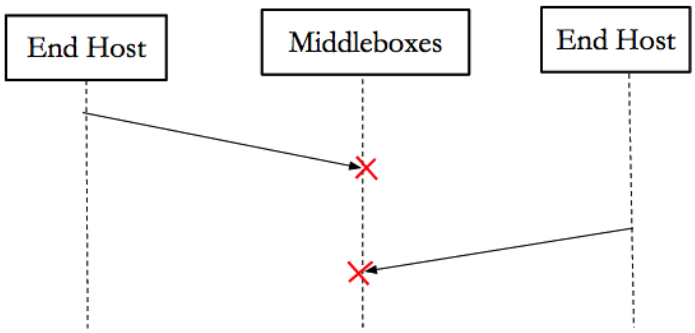

Figure 3.2: Diagram for Filtering

A firewall is an important example of network middleboxes that filters traffic flows for network security and network control. Nowdays, there are three types of use cases for filtering. First, firewalls or any other middleboxes drop packets based on IP addresses and TCP ports. As described in prior chapter, IP or port blocking is the earliest filtering mechanism and used for Internet censorship in China, Brazil and so on [20, 43, 11].

Second, middleboxes operate up to the transport layer of OSI model, the state of connection is used for making judgements for filtering. In principle, this kind of filtering should follow the policies of TCP. However, as observed in the work [10], the stateful firewall policies in some cellular networks usually do not strictly follow the TCP specification.

Furthermore, middleboxes also operate filtering in application layer. The triggers of filtering are inside payloads of application layer and middleboxes need understand the applications and related protocols. For example, URL filtering and keyword filtering are achieved by checking application payloads (e.g. HTTP request target, content of HTTP response), implementing Internet censorship, copyright protection and other web traffic control. In addition, middleboxes are getting ”wider” or ”deeper” inspection at application stack [44]. For instance, intrusion detection system (IDS) are developed in today’s network for malicious activity or policy violations.

In our work, we control both client and server sides, we detect the filtering by mapping the

original probes and received packets. To take into account the effect of random packet loss, we send the probes several times, repeating the experiments for multiple samples. Also, we send probes that do not attempt to trigger middlebox interference to verify that the exposed middlebox behaviour is indeed caused by specific payloads.

3.2.2 Injection

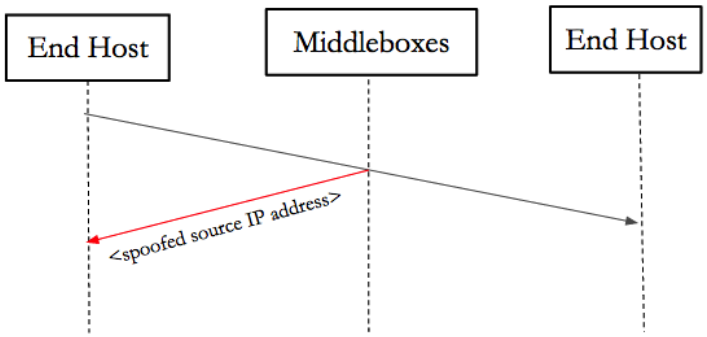

Figure 3.3: Diagram for Injection

As shown in Figure 3.3, middleboxes may inject fake packets to end hosts with spoofed source IP addresses. The injected packets usually interferes with network connection for blocking, affecting the performance of applications. For example, GFC returns forged DNS responses to clients for blocking some blacklisted websites, causing collateral damage and affecting communication beyond the censored networks when outside DNS traffic traverses censored links [14]. Moreover, forged TCP RST packets are deployed by some ISPs to manage P2P traffic, as well as by GFC to censor communication of in and out traffic flows [45]. Also, some organizations insert HTTP response with warning html page to notice clients the network filtering or network control.

The injected packets are crafted by middleboxes. The source IP addresses are forged with the destination IP of original probes, pretending the standard communication with end hosts. We compare the traces from both end hosts, finding the extra packets which are not sent from end hosts, but injected by middleboxes. Indeed, some middleboxes inject HTTP response with warning html page payload to notice the client about the traffic filtering. Since we control client and server, we could infer the filtering and injection at the same time, exposing the full process of middlebox interference.

3.2.3 Modification

Packet manipulation is a main kind of middlebox interference for shaping traffic flows. Middleboxes may remove headers in different layers, inject new headers to carry more information of middleboxes or connection, modify the value of headers and response payloads for new functions.

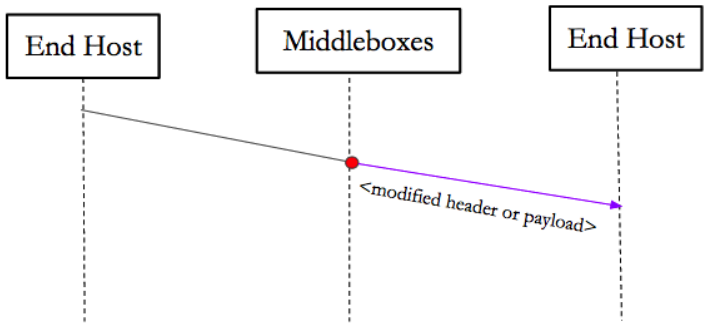

Figure 3.4: Diagram for Modification

We find lots of examples for packet manipulation in today's network. First, some middleboxes manipulate the IP and TCP headers, affecting the compatibility between middleboxes and network protocols. For example, the study [13] developed a measurement methodology to evaluate middlebox behaviour relating to TCP headers, especially for TCP extensions. From the results, we are warned that do not assume sequence numbers arrive unmodified, which will affect the performance of TCP option or TCP connection. In addition, proxies are common in today's networks, and strips TCP options in different ASes.

Second, by manipulating the headers and contents of application traffic flows across regions and networks, more communication meta and user's private information are exposed and tracked through middleboxes [46].

Furthermore, the Chinese censorship inserts additional attacking scripts to outgoing HTTP responses, engineering a denial-of-service attack on GreatFire.org, an organization dedicated to resisting China's censorship [16]. By making use of packet manipulation, Chinese government turns GFC into a weapon (China's Great Cannon) which coopts arbitrary computers outside of China to achieve Chinese policies.

To detect the modification in each packet, we decode the traces as each protocol defined, compar-

ing the difference between original probes and received packets from clients and servers. Also, by correlating the different types of modifications, we could infer the presence of middleboxes, explore the functions and effects of these middleboxes in varied networks.

3.3 Probe

3.3.1 HTTP Probe

World Wide Web (WWW) is accessed through HTTP, dictating the format of exchanging web contents. Since HTTP provides a vital infrastructure for users exchanging information and communicating with each other, it has changed Internet significantly.

Currently, the implementation of HTTP is diverse: the user agents of HTTP traffic are not only the general-purpose browsers, but also other household appliances, command-line programs, mobile applications and so on. Likewise, common HTTP original servers are not only large public websites, but also other home automation units, configurable networking components, office machines, advertisement selectors, third party servers and video-delivery platforms [47]. Moreover, lots of middleboxes interact with HTTP traffic, serving numerous roles, from performance enhancement to access control to network censorship [7]. Therefore, we craft our probes as HTTP messages, simulate the work process of HTTP for detecting related middlebox interference.

Theoretically, a HTTP message consists of a message header and an optional message body, separated by a blank line [48]. As shown in Figure 3.5, the message header of HTTP request contains request line and request headers [49]. The request line includes the request method, URL link and the version of HTTP. As discussed in Section 3.2, these contents of HTTP requests contain information about traffic flows, and may be triggers of some middlebox interference. Also, HTTP response contains the status line and other response headers. In general, HTTP request/response headers are used to pass additional communicate meta information between clients and servers. The headers are classified by their usage category, such as authentication, cache, client hints, conditionals and so on, representing the related functions of middleboxes. At the same time, the HTTP payload constitutes the content of the message, returning the queried documents.

In our method, the client makes TCP connection with our owned server firstly. After making

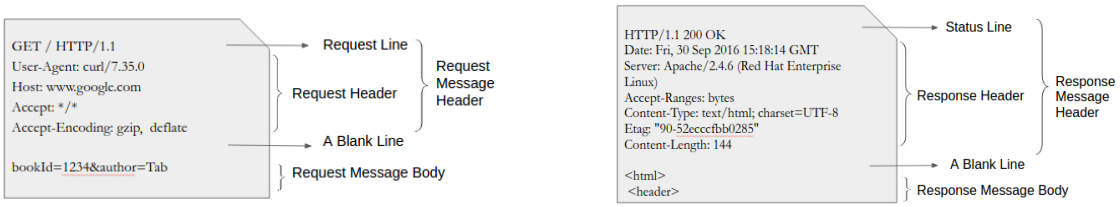

Figure 3.5: HTTP Message Format

TCP connection successful, the client sends HTTP request, the server returns HTTP response to client as followed. By making the TCP connection, we could achieve the full process of sending and responding HTTP messages, which may trigger more middlebox interference about state. For instance, application-layer filter is a kind of stateful middleboxes. It typically operates on live TCP connections. Instead of filtering all packets directly, the end hosts are allowed to make TCP connections. After examining the payloads of application traffic flows, the application-layer middlebox filters traffic flows or terminates live TCP connections by related policies and states.

At the client side, we launch HTTP requests by using the command-line program curl. Three default request headers are issued: Host, User-Agent and Accept. To detect the behaviour of middleboxes for implementing Internet censorship, we craft the value of ”Host” header with blacklisted domain names (Table 3.1). For our own server, we instantiate the server in Amazon EC2 instance, using the default Apache configuration with the exception of disabling all caching (using the Cache-Control header). To avoid ethical issue, the payload of HTTP response is a simple ”Hello World” HTML page (Figure 3.7).

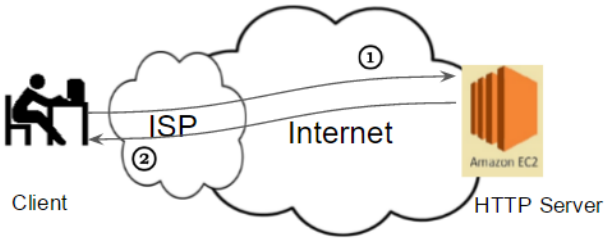

Figure 3.6: Detecting Process of HTTP Traffic Flows

The analysing process is based on the captured traces. We compare the traffic flows with traces from client and server sides, checking the filtering and modification of HTTP request and response. At the same time, we detect TCP reset packet injection or other interference about connection state by finding the additional TCP reset packets from middleboxes.

```
<HTML>
 <HEADER>
  <TITLE>
   A SMALL HELLO
  </TITLE>
 </HEADER>
<BODY>
 <H1>Hi</H1>
  <P>This is very minimal helloworld.TYPE1 RESPONSE</P>
</BODY>
</HTML>
```

Figure 3.7: The Source Code of HTML Page in HTTP Response

3.3.2 TLS probe

Security is more and more important on the Internet [50]. TLS and SSL are cryptographic protocols that provide means for achieving security at the transport layer [51, 52]. Currently, TLS is the most widely used as the secure transport layer below HTTP, running the secure version of HTTP, HTTPS [50, 34]. A large number of websites, such as Gmail, Facebook or even YouTube use TLS as the secure transport layer to transmit encrypted data to protect the user information, for example, 50% of YouTube streaming flows are over HTTPS [34]. Furthermore, some middleboxes today operate TLS/SSL traffic [53], performing ”man in the middle” (attack) on the connection for Internet censorship purpose [54, 55, 56]. We use the TLS handshake message to detect whether there is fake certificate injection or any other interference on TLS traffic flows.

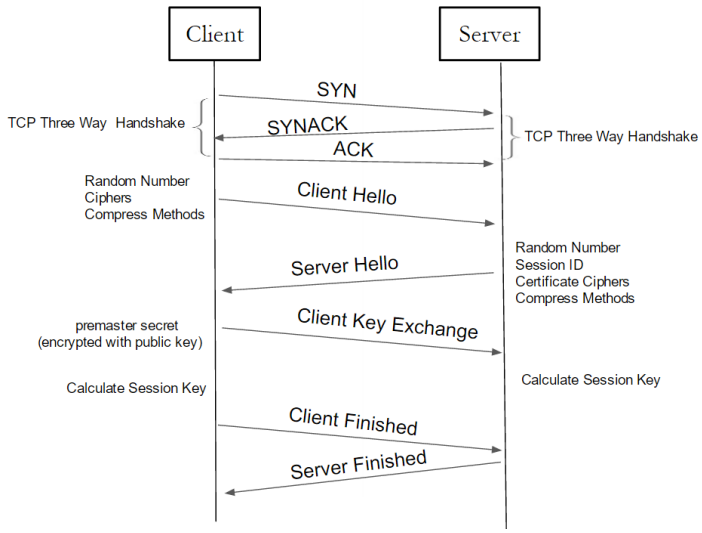

Figure 3.8: The Process of TLS Handshake

Each TLS connection begins with a handshake between client and server. As shown in Figure 3.8, the client asks the Public Key Infrastructure (PKI) suite from the authenticated server

and generates cryptographic keys to create a secure channel for encrypted data transmission.

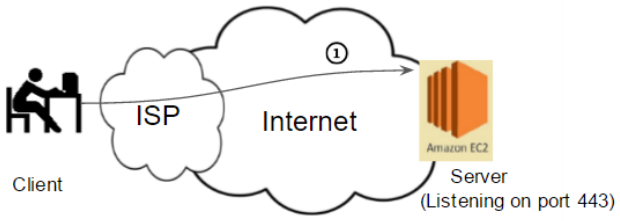

Figure 3.9: Detecting Process of TLS Traffic Flows

In theory, TLS negotiation [57] includes four steps. First, client sends a ”Client Hello” message to list cryptographic information for the server. Then, server responds to client with a ”Server Hello” message which returns the server’s certificate. Getting the public key from the server’s certificate, the client creates a random Pre-Master Secret. Based on the Pre-Master Secret, the server and client generate the new session keys for encrypting application data. If any middlebox modifies or injects fake certificate or key on the path, the handshake will fail and the transmission will not safe any more. As mentioned in [25], because of TLS and HTTPS, all network functionalities must reside at the endpoints. However, it may be not optimal or even possible for the networks with middleboxes. In order to understand the behaviour of middleboxes on encrypted data, we plan to measure how middleboxes interfere with TLS traffic flows firstly.

To check the process of TLS negotiation, our client sends a ”Client Hello” message to our controlled server (Figure 3.9). The server listens on port 443 to wait for the TLS handshake message. The server does not respond the handshake message, any received TLS handshake message in client side is injected by middleboxes. Since the extension section in Client Hello Message has the Server Name field, we craft it with our blacklisted host names to check any middlebox interference which is related to Internet censorship.

3.3.3 DNS Probe

Currently, DNS is the most widely used service on the Internet [58]. In theory, most of Internet devices make connection and access websites based on IP addresses. While, it is quite harder for users to remember the exact IP address than the name of a website. DNS provides the function of converting domain name to IP address. Before connecting to a website, users need resolve domain name to IP address by accessing DNS. DNS is the fundamental step for end users accessing

web-hosting services, and affects the performance of many Internet applications.

In general, end user firstly sends DNS query to a resolver, which may be operated by their ISP, wireless carrier or other third party providers. If the resolver does not have the answer, the resolver queries to root nameserver, TLD nameserver and authoritative DNS server to get the DNS record. The resolver returns the DNS record to end user, and stores the record in its local cache. If anyone else requests the same domain name again, and the TTL of the local cache is not expired, the resolver will return the record to the user directly. Otherwise, the resolver will query to the authoritative DNS server again. In other words, the resolver is important to DNS service for returning response, the local cache in the resolver can become poisoned if it contains an incorrect entry. For example, the GFC inspects DNS queries near the ISP's boundary routers for sensitive domain keywords and injecting forged DNS responses. Unfortunately, the local DNS resolver saves incorrect caches and returns forged response to all queried users. The DNS requests are not only from inside censored networks, but also from outside censored network. The injected forged responses affect communication beyond the censored networks when outside DNS traffic traverses censored links, causing large scale collateral damage across the world [14, 59].

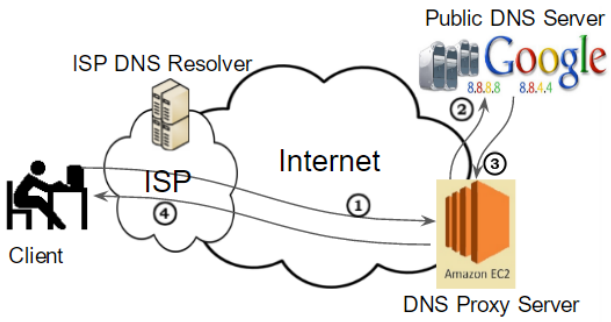

Figure 3.10: Detecting Process of DNS Traffic Flows

In our method, we set up a DNS proxy server to capture, compare the DNS queries and responses to detect middlebox interference on DNS traffic flows. Figure 3.10 shows the process of our detection. Instead of querying to ISP DNS resolver, our client queries to our own controlled DNS proxy server which is running in Amazon Elastic Compute Cloud (EC2) instance [60].

Our DNS proxy server forwards DNS query to Google public DNS service to get the correct DNS response which returns actual IP addresses for the queried domain name. To defend against spoofing attacks that can "poison" a name server's cache and return malicious IP address, Google has implemented several recommended solutions to help guarantee the authenticity of the re-

sponses [61]. At last, our DNS proxy server returns correct response to our clients. The client and DNS proxy server are both controlled, the analysis is based on decoding and comparing traces from both sides.

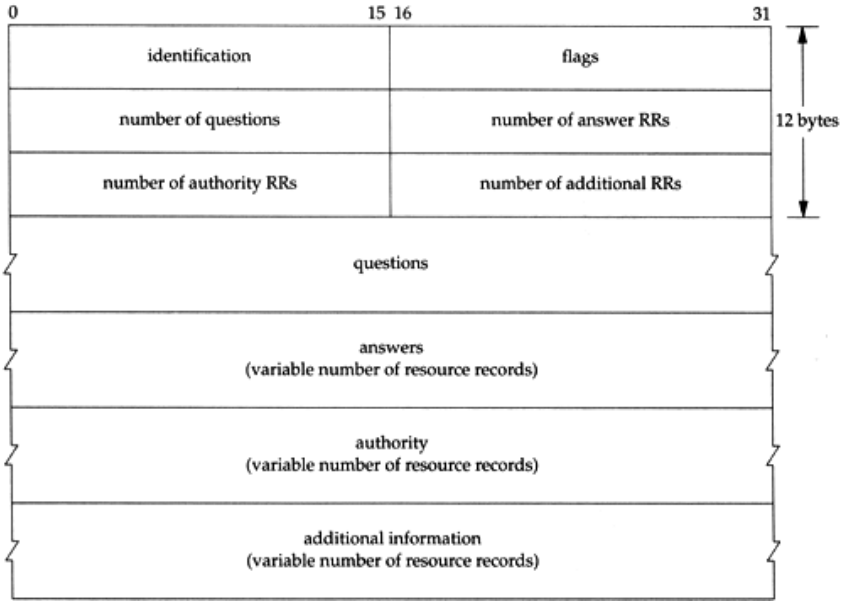

Figure 3.11: DNS Message Format

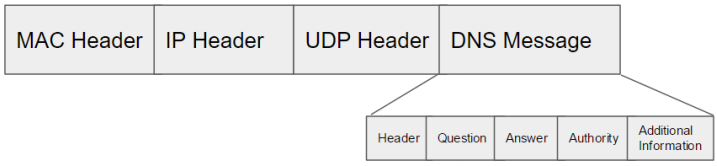

Figure 3.12: A Small Ethernet II Frame Contains DNS message

As shown in Figure 3.12, DNS protocol uses UDP as transport layer protocol because of its small overhead. Without making connection firstly, UDP is much faster than TCP. The DNS message could be divided into five sections: DNS header, the question for the name server, the answer for the query, the records for authoritative server and additional information that may be used (Figure 3.11).

DNS header contains 12 bytes in total, assigning an identifier for the DNS message as DNS ID; specifying which of the remaining sections are present, whether the message is a query or a response, a standard query or other type query in flags; pointing out the number of question and answer entries. We use DNS ID to map the original query and received query, and detect the packet filtering or manipulation by comparing captured traces.

Different with DNS response, DNS query only has question section. Question section contains three fields: the queried domain name, type of the query and the class of query. In our method, we use type A query to resolve the IPv4 address into domain name.

The answer, authority, and addition sections all share the same resource record format [62, 63]. Each resource record contains the domain name for this resource record, the type and class of resource record, and the time duration that resource record may be cached. Fake responses with spoofed resource records may be injected to the user, and the resolved IP address in the resource record may be modified [59]. We decode the traces as DNS protocol, and compare each field to identify the interference.

As discussed in Section 3.2.2, the queried domain name in DNS protocol is the trigger for middlebox interference. In details, we select 47 host names (Table 3.1) to craft different DNS queries, triggering the packet filtering or packet injection, especially the behaviour of middleboxes which are related to Internet censorship, copyright protection, and network security. The host names are listed from Internet censorship paper ([64, 11]) and online news about blocked websites in different countries ([65, 66, 67, 68]).

File Hosting		
www.buzzfeed.com	narod.ru	ipicture.ru
thepiratebay.sx	thepiratebay.org	www.primewire.ag
fileserver.com	newzbin.es	Movie4K.to
torrentfreak.com	imgur.com	fc2.com
nm.ru	fenopy.eu	bittorrent.am
h33t.com	kat.ph	torlock.com
firstrow1.eu		
Proxy and VPN		
NinjaProxy.com	www.sodahead.com	pirateproxies.net
www.hotspotshield.com	www.turbohide.com	sur.ly
bayproxy.pw		
Porn Website		
www.pornhub.com	hardsextube.com	10gay.com
redtube.com	pornru.net	young12.nm.ru
teen-sweet.com	younger18.com	
News/Search/Bet Blocking Website		
facebook.com	youtube.com	twitter.com
barenakedislam.wordpress.com	www.similarsites.com	friendlyatheist.com
www.loonwatch.com	www.foxnews.com	newsfilter.org
en.wikipedia.org	archive.org	wordpress.com
bet365.com		

Table 3.1: Blacklist of Host Names

In our early work, we implemented the methods of DNS probes in PlanetLab platform [40]. Our methods detected two types of foged DNS responses and the corresponding results confirmed the interference of *injection*, as reported in prior work [14, 69]. Limited by the number of available instances in PlanetLab, our detection results about DNS-interacting middleboxes are all related to GFC, which has been already studied in prior work [14, 69, 11, 64]

3.4 Middlebox Locating

Currently, middleboxes are located by probing with different TTL values. As mentioned in tracebox methodology [5], when an IPv4 router receives an IPv4 packet whose TTL is going to expire, it returns an ICMPv4 time-exceeded reply contain the offending packet. The differences between original probes and offending packets reveal the behaviour of middleboxes on the path.

3.4.1 Locating Three Types of Middlebox Interference

Inspired by tracebox, we develop traceroute-like locating methods with our client-server architecture, comparing the original probe, the quoted offending packet in the returned ICMP time-exceeded reply and the received one in server side, inferring the approximate hop range of three types of middlebox interference. While, some routes do not return ICMP time-exceeded replies. Also, some middleboxes modify or filter the replies again. Using our methods which control both client and server sides is essential to locate varied interference. In this subsection, we explain how to locate three types of middlebox interference specifically. Furthermore, we discuss the limitation of locating in next subsection 3.4.2.

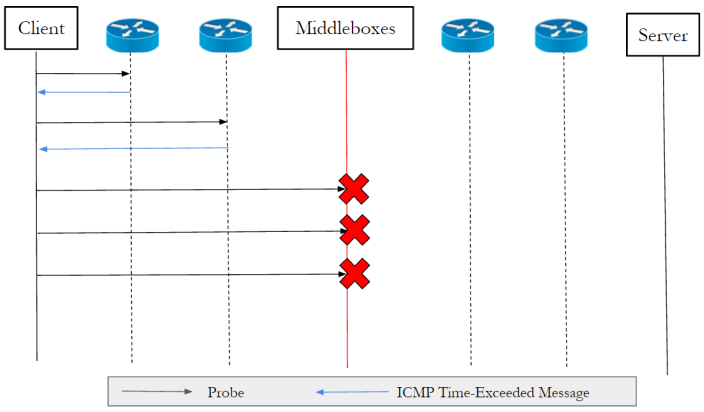

Figure 3.13: Locating Process of Packet Filtering

As shown in Figure 3.13, the probes are sent with increasing TTL values. The client receives multiple ICMP time-exceeded replies to describe the path. However, after the middlebox filters the original probes, the received ICMP time-exceeded replies could not cover the whole path between client and server. The last hop in described path is the approximate location of where a given filtering takes place.

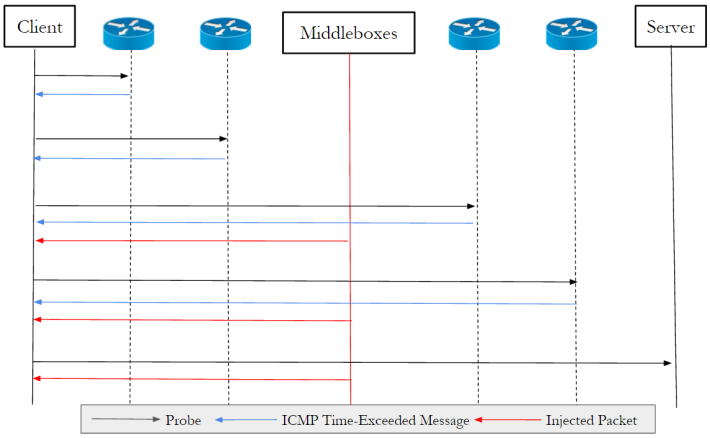

Figure 3.14: Locating Process of Packet Injection

The probes with small TTL values will not trigger the injection packet (shown in Figure 3.14). From the received ICMP time-exceeded replies and injected packets, we identify the last hop before the probe triggers injected packet. At the same time, we could identify the first hop after the probe passes though middleboxes and triggers injected packet. These two hops describe the approximate location of middleboxes.

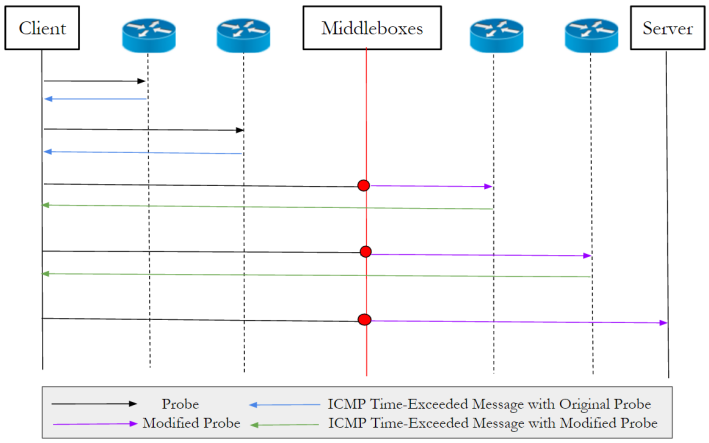

Figure 3.15: Locating Process of Packet Modification

If the original probe is modified by middleboxes, the offending packet inside the ICMP time-exceeded replies are different. We locate the range of middlebox by checking each received

ICMP time-exceeded replies, discovering the involved networks deeper and deeper.

3.4.2 Discussion

To locate the middlebox, we check each quoted packets in ICMP time-exceeded replies, and point out the last hop before modification and the first hop after crossing the middlebox. However, these traceroute-like locating methods are limited with the received ICMP time-exceeded replies.

First, some routers do not return the ICMP time-exceeded replies, especially inside private network. The distance between middleboxes and identified location may be more than one hop.

Second, the type of returned ICMP reply affects our identification. According to RFC792 [41], the returned ICMP time-exceeded reply should quote the IP header and the next eight bytes of original packet. RFC1812 [42] suggests to quote the entire IP packet in ICMP time-exceeded reply, but this recommendation has not been widely implemented. The more contents of original probes quoted in the ICMP replies, the more interference we could locate.

At last, as we discussed before, middleboxes are complex. There may be multiple middleboxes between two routers. With our methods, we could figure out the hop range of middleboxes from upstream and downstream directions, trying to identify the types of involved networks, but, it is hard to point out the accurate number and location of each middlebox .

3.5 Methodology with Luminati

Since we aim to measuring vantage points in different networks across the world, we implement our methodology with *Luminati*, based on the Hola network, to launch HTTP requests across the Internet. In this section, we describe how to develop our methodology with Luminati platform, detecting the presence of middleboxes through their interaction with HTTP requests and answers. We keep adopting the client-server architecture, with control on both sides of the end-to-end flow. All probes sent and received are collected and kept for further analysis. Note that the detail description of our client-server methodology has been described in Section 3.1.

3.5.1 Hola and Luminati

Hola is a P2P VPN service, which allows users to route traffic over a large number of country peers, from nearly 280 countries. These country peers run on users' machines, therefore based on a variety of devices, e.g., laptops, mobile devices, and distributed across various types of networks. In practice however, Hola forwards traffic via super proxies located in a few countries (e.g., the UK or the USA), instead of going though each country peer.

To get full advantage of the vantage points from the Hola proxy network, one needs to rely on *Luminati*. *Luminati* is a paid HTTP/S service that is based on the Hola network. *Luminati* forwards users' traffic via Hola country peers, not the specific super proxy, therefore providing a much larger set of vantage points from which to launch probes. Furthermore, *Luminati* enables users to select country peers in the Hola network. For instance, the *Luminati* users could launch multiple requests from same country peer by adding the same *-session-number* parameter to their subsequent requests. The session number could be a random number, and *Luminati* will keep forwarding the subsequent requests via the same country peer, for a period of 60 seconds only. After these 60 seconds, *Luminati* chooses another country peer, potentially different from the prior one. Also, even though in principle a country peer from the chosen country will be used, there is no guarantee that this is actually going to be the case.

In the received HTTP responses, *Luminati* super proxies add two headers, for logging and debugging the selected country peer (**X-Hola-Timeline-Debug** and **X-Hola-Unblocker-Debug**), carrying the identifier **zID** for the selected country peer. By recording the **zID**, we can check whether our probes are forwarded via the same country peer, even if the country peer has changed its IP address in the meantime.

3.5.2 Measurement Methodology

3.5.2.1 Available Country Peers

Although *Luminati* provides diverse country peers across the Internet, a user is only able to select country peers at the granularity of an AS or a country, without indication about the network type inside which the country peer is. *Luminati* does not enable a controlled selection of the ingress nodes, and selects the available country peers randomly. As we aim to probe from as many

networks as possible in our measurements, we launch HTTP requests via as many as possible country peers. For each country among the 275 available countries of the measurements, we keep sending probes using random session numbers.

```
curl --proxy zproxy.luminati.io:22225 \
     --proxy-user $USERNAME-country-$CC-session-$SESSION:$PASSWORD \
     "http://lumtest.com/myip.json"
```

Figure 3.16: Sample Request of Luminati.

Figure 3.16 shows the shell scripts to launch HTTP requests with *Luminati*. The parameter *zproxy.luminati.io* could be replace by the IP address of one particular super proxy, and the port number is fixed (22225). The country code parameter (CC) and the random session number are used for selecting available country peers in different countries and separating country sources of traffic. More examples which explain how to work with *Luminati* in Java, Python, Ruby and Perl are displayed on the official website [70].

We launch sets of 500 requests for a given country at a time. If for a given country, after a set of 500 requests, we observe 5 duplicate country peers, we stop probing this country for a while[1].

3.5.2.2 DNS resolution

In principle, *Luminati* allows users to set the DNS resolution by adding the **-dns-remote** parameter to the request. This DNS resolution can be done at two different places. The first place where the DNS resolution can be done is at the super proxies, which send the DNS query to Google's public DNS service. The second place is at the country peer, which sends the DNS query to its local DNS resolver. To ensure that our probes are forwarded by country peers, and not the super proxies, we forward the original HTTP GET request directly to the country peer, and ask for the DNS resolution to be also done by the country peer. Figure 3.17 illustrates the process of launching HTTP requests using the *Luminati* platform. For all requests, we select the same super proxy from the United Kingdom, using all available country peers.

3.5.2.3 Crafted Probes and Answers

With our methodology, we control both the client and server sides, sending crafted probes (HTTP requests) and responses (HTTP responses). Unfortunately, *Luminati* does not provide direct ac-

[1]But we guarantee that we launch a total of 1000 requests in any given country in our measurements, even if a single country peer is available.

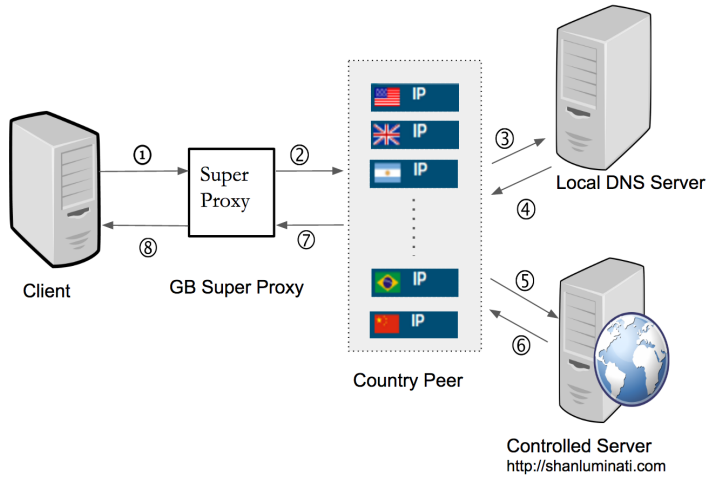

Figure 3.17: Luminati-based Probing Methodology.

cess to the country peers for probes to be sent, but only allows requests to be forwarded through them. Figure 3.18 shows the traffic exchange process between country peers and our controlled server. We treat country peers as our original clients, and detect middlebox interference on the path between the country peer and the server. As shown in Figure 3.18, the actual origin of the probes is one of the available *Luminati* country peers acting as client, and the target of the probes is a server under our control, located in Ireland Amazon cloud data center, running the server-side of our methodology.

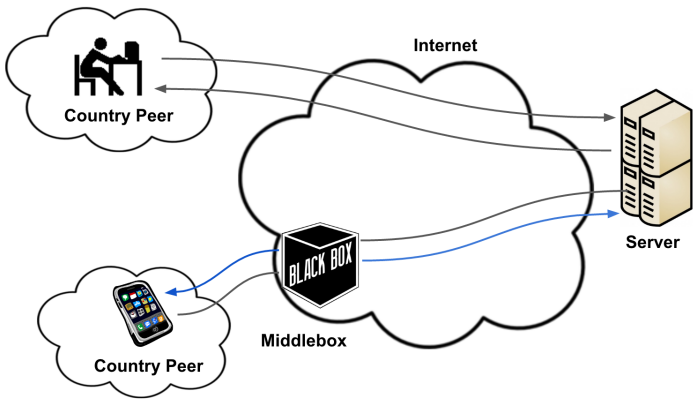

Figure 3.18: Traffic Flow between Country Peer and Target Server.

We set up measurement clients, operating in Amazon cloud instances in Ireland. We launch HTTP requests to our target server, via the *Luminati* country peers[2], with 7 default request headers: Host, User-Agent, Accept, Accept-Charset, Accept-Encoding, Keep-Alive and Connection.

[2]As we rely on the *Luminati* proxy peers, the location of the clients is technically irrelevant for the middlebox detection.

Table 3.2: Default HTTP Headers.

	Name	Default Value
Request Header	Host	shanluminati.com
	Accept-Charset	utf-8
	Accept	*/*
	User-Agent	curl/7.35.0
	Accept-Encoding	gzip
	Connection	keep-alive
	Keep-Alive	max=20, timeout=10
Response Header	Accept-Ranges	bytes
	Connection	keep-alive
	Keep-Alive	max=20,timeout=10
	Content-Type	text/html; charset=UTF-8
	Date	Mon, 15 Aug 2016 16:59:35 GMT
	Etag	90-52ecccfbb0285
	Content-Length	144
	Last-Modified	Thu, 24 Mar 2016 15:07:56 GMT
	Server	Apache/2.4.6 (Red Hat Enterprise Linux)

The default values of these headers are shown in Table 3.2.

In Figure 3.19, we show the sample script of launching HTTP requests from client side. As described before, the requests are fowarded to super proxies firstly. We choose to do the DNS resolution by country peers and insert the parameter *-dns-remote* in the requests. The default HTTP headers are setted by parameter *-H*, which is the option of curl to specify HTTP headers.

```
USERNAME=lum-customer-CUSTOMER-zone-YOURZONE
PASSWORD=YOURPASS

PORT="22225"
SESSION=$RANDOM
CC="cn"

curl --proxy zproxy.luminati.io:$PORT \
     --proxy-user $USERNAME-country-$CC-dns-remote-session-$SESSION:$PASSWORD \
     -H 'Host: shanholaserver.com' \
     -H 'Accept-Encoding: gzip, deflate' \
     -H 'Connection: keep-alive' \
     -H 'Accept: */*' \
     -H 'keep-alive: max=10, timeout=10' \
     -H 'Accept-Charset: utf-8' \
     "http://shanluminati.com"
```

Figure 3.19: Sample Request of Luminati.

Our target server is operating in an Amazon cloud instance, located in Ireland as well[3]. The server acts as a standard Apache HTTP server, waits for the TCP connection and GET request, and returns the HTTP response with a crafted payload. As we are forwarding requests through

[3]Only in the case of requests launched from country peers in Ireland, we rely on an Amazon cloud instance in the US as target server to ensure our probes travel through the Internet, not inside the Amazon infrastructure.

other users' machines, to avoid potential ethical problems for these users, we only respond to requests with a simple "Hello World" HTML page with 9 default response headers.

```
#!/usr/bin/env python
import socket
#import pcap
#import dpkt
import sys
import time
from thread import *
#from scapy.all import *

server_socket=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind(('',80))

server_socket.listen(1500)


##########Response to the request###################
def clientthread(c,addr,payloadlist):
        data=c.recv(12400)

        if((data[0]=='G') & (data[1]=='E')& (data[2]=='T')):
                c.send(payloadlist[1])
        time.sleep(1)
        c.close()



#########Craft the payload of HTTP response#########
payloadlist={}
f=open('/home/ec2-user/luminati_server1_1.txt'%m,'r')
a=f.read()
payloadlist[1]=a
f.close()



########Waiting for connection####################
while 1:
        c,addr=server_socket.accept()
        print addr[0]
        start_new_thread(clientthread,(c,addr,payloadlist,))
server_socket.close()
```

Figure 3.20: Sample Script of HTTP Server.

Figure 3.20 shows the sample python script which is operating in Amazon cloud instance. We craft a sample payload of HTTP response in a text file in advance. After making the connection successfully, we send the crafted response back to the client. In addition, we import the *Scapy* library in the scripts for the next location step, sending the probes with increasing TTL values.

Although not every country peer would return us with a response, we compared all received HTTP messages with the sent probes. As explained in Section 3.5.1, the super proxy of *Luminati* adds two headers (**X-Hola-Timeline-Debug** and **X-Hola-Unblocker-Debug**) to HTTP response, illustrating the information of selected country peer. These two headers do not affect or overwrite other original headers. When comparing HTTP responses, we do not take these two headers into consideration, but investigate other headers.

3.5.3 Locating Methodology with Luminati

Inspired by tracebox, we try to locate the involved middleboxes hop by hop. Since we can not have access to Luminati country peers, our current process of probing is unidirectional, from

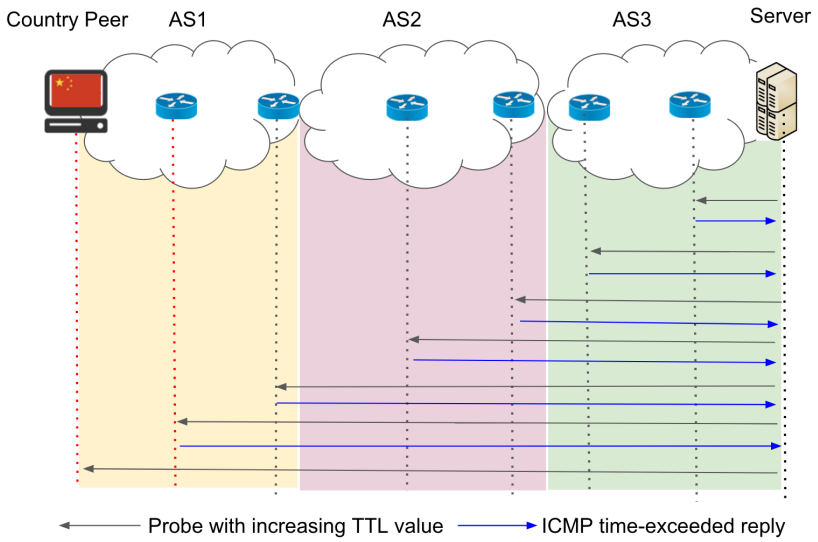

Figure 3.21: Diagram of Probing with Increasing TTL Values

server to country peers (shown in Figure 3.21). The servers are both in Amazon cloud platform, one is located in Ireland data center, the other is located in US for probing the country peers distributed in Ireland.

In principle, the last hop which returns ICMP time-exceeded reply should be in the same AS of country peers. Meanwhile, if the middleboxes are located near client side, the last ICMP reply may also come from same AS of country peers. On the other hand, if the last ICMP reply comes from different AS of country peer, we infer that there may be TCP-terminating middleboxes between two end hosts, and the locations of middleboxes are in the middle or close to server side.

Similar with our prior experiments, our probes are crafted with HTTP response payloads. To implement the response with increasing TTL values, we develop our python scripts with the *Scapy* library, and send our responses to the country peers directly (without making connections). To mark the TTL value for each probe, we craft the IP identification in IP datagram with the same value of its TTL.

3.6 Summary

Lacking the access to the network devices inside networks, it is very hard to have the visibility to observe what happens in the middle. Our methods aim to mimic the transmission of traffic

flows, describing the behaviour of middleboxes on the path, analysing middlebox interference and locating the approximate location of involved middleboxes.

As discussed in Section 2.4 and Table 2.1, we learn and develop our methodology from prior detecting methods. Inspired by *TcpExposure*, *NetPiculet* and *HICCUPS*, our methods control both client and server sides, which allow to compare the traces between two end hosts (client and server) at the same time. The difference between traces illustrates the middlebox interference, explaining the effect on the state of connection, and how middlebox breaks the end-to-end model. Differs from these three prior methods, our probe packets are crafted as our design, providing the freedom to launch different kinds of queries to mimic varies kinds of traffic flows. In addition, we detect the packet manipulation in both direction (from client to server and server to client) independently, avoiding multiple manipulation by the same middlebox. The tool *Netalyzr* provides different types of network functionality tests, giving us lots of ideas about making large-scale measurement. While, our detection is done by both upstream and downstream directions with larger dataset. *Tracebox* is a traceroute-like tool to identify packet modifications performed by upstream middleboxes. It is a seminal work in the area of middlebox interference. Our methodology extends this work with application traffic flows, detecting middlebox interference on applications, especially exploring the middlebox behaviour triggered by application payloads. In addition, with the help of traceroute-like probing, we check the middlebox interference hop by hop, locate the range of approximate position of middlebox, exploring the types of networks inspected.

On the other hand, the process of locating middlebox depends on the returned ICMP time-exceeded replies. As we discussed in Section 3.4.2, not all the routers return ICMP replies. It is hard to describe each hop in the whole path. Also, some ICMP replies only quote the first 64 bits of original packets, which can not be used for detecting and locating the manipulation of application layer payloads. Indeed, it is hard to point out the location where each interference occurred, our methods locate the approximate hop range of involved middleboxes.

Chapter 4

Results

To sample middlebox interference across the Internet, we want the probes to be sent through a physical infrastructure distributed across the world. However, the infrastructure used to send the probes should provide significant and as representative as possible vantage points, i.e., beyond a purely academic one such as PlanetLab. Indeed, PlanetLab is not suitable for our middlebox study. Most of PlanetLab instances are distributed inside research labs with unrestricted Internet access, and the usage of middleboxes in academic networks is limited. We have used our methodology on the PlanetLab infrastructure as well, but hardly found any middlebox interference this way, only a few non-representative instances of middleboxes. Therefore, we use the commercial Peer-to-Peer (P2P)-based HTTP/S proxy service, *Luminati*, based on the Hola network, to launch HTTP requests across the Internet.

In this Chapter, we explain what we get by using *Luminati* platform to run large-scale measurements (Section 4.1.1). Also, based on our analysis and results, we show evidence for a significant amount of middlebox interference on both directions of traffic flows in different kinds of networks (Section 4.1.2 and Section 4.1.3). In addition, we investigate more information about the location of involved middleboxes by launching traceroute-like probes (Section 4.2).

4.1 Middlebox Interference on HTTP Messages

Middleboxes are widely used in today's Internet, especially for security and performance. Middleboxes classify, filter and shape traffic, therefore interfering with application performance and performing new network functions for end hosts. In this part of results, we provide evidence for middleboxes deployed across more than 1000 ASes. We observe various middlebox interference in both directions of traffic flows, and across a wide range networks, including mobile operators and data center networks. In details, we examine middlebox interference on HTTP requests in Section 4.1.2 and describe response manipulation in Section 4.1.3. Furthermore, for each type of manipulation, we not only study the variety of headers affected, but also try to infer the type of network in which the manipulations take place.

4.1.1 Dataset

We now give a brief introduction of our collected dataset. We kept launching requests through different country peers for a period of 5 days, from September 29 until October 3 2016. Table 4.1 shows the number of the IP addresses seen through the requests and responses, their AS and country. In Table 4.2, we investigate the AS coverage based on the classification provided by Dimitropoulos et al. [71]. Despite the fine-grained nature of this classification, nearly 67% of the ASes we observe are unclassified (last row of Table 4.2). Also, the classifier abstains from classifying some ASes, when the information about them is not sufficient (labeled as "Missing Sufficient Information" in Table 4.2).

Table 4.1: Overview of Dataset

	Requests	**Responses**
# of IPs	401,746	372,603
# of ASes	10,060	9,634
# of countries	196	196

For requests, we rely on the source IP address of the received request at the server. This IP address will either be the one from the *Luminati* country peer or a TCP-terminating middlebox located between the country peer and our server. *Luminati* adds the IP address of the selected country peer to the response header, and therefore the IP address we use for the response in our dataset is one of the selected country peers (not of a middlebox). All the corresponding ASes and

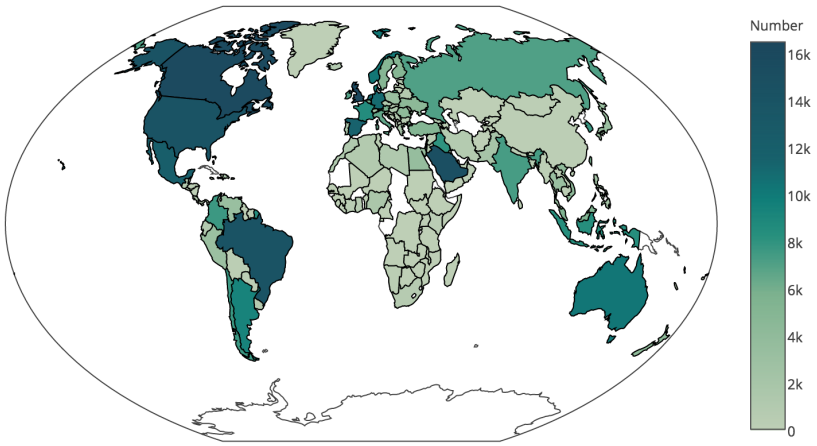

Figure 4.1: Number of IP Addresses per Country (max 16433).

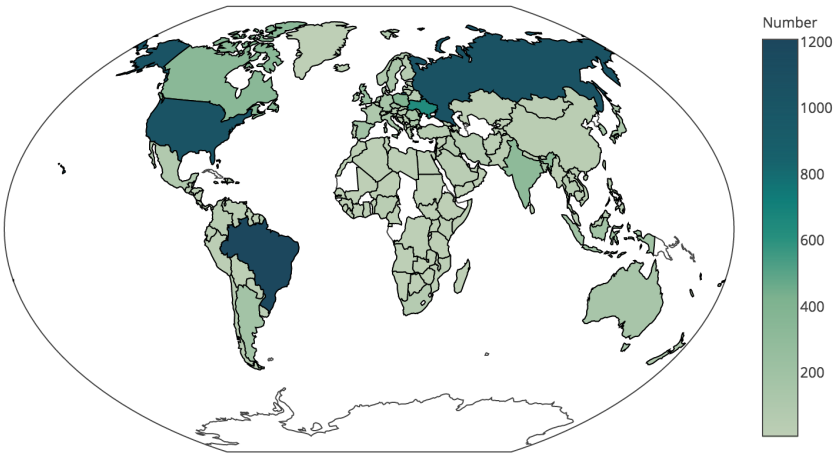

Figure 4.2: Number of ASes per Country (max 1200).

Table 4.2: AS Classification

AS Classification	**Requests (# of ASes)**	**Responses (# of ASes)**
Customer	927	887
University	324	319
Internet Exchange	3	3
Network Information Center	43	43
Tier 1	39	38
Tier 2	1631	1611
Missing Sufficient Information	197	198
Unclassified	6896	6536

countries are inferred from these IP addresses. We provide the distribution of selected country peers in Figure 4.1 and Figure 4.2.

4.1.2 Request Header Injection

In the upstream direction, i.e., on the path the HTTP request takes towards the server, we see a variety of new headers injected by middleboxes. These headers mostly relate to common network functions, such as proxying, caching, filtering and load balancing. When comparing the injected headers from the requests with those from responses, we see a wider diversity of different headers being manipulated in responses. We also observe that most of the manipulated response headers relate to proxies or caches that inject new headers into requests. Though the sheer numbers do not constitute conclusive evidence, this may indicate that middleboxes affecting the upstream direction (requests) are actually a subset of those affecting the downstream direction (responses). Given that middleboxes are stateful devices that see both directions of the traffic flows, it is natural to expect a significant overlap between manipulations done in both directions of the traffic.

4.1.2.1 Proxies/Caches Request Header Injection

Table 4.3: Injected Request Headers Related to Proxy or Cache Functions.

	Injected header	**# of ASes**	**# of countries**	**Note**
Proxy-Related	Via	695	117	Via: 1.1 rcdn9-cd1-dmz-wsa-1.cisco.com:80
	X-Forwarded-For	535	106	X-Forwarded-For: 192.168.2.157
	X-Proxy-ID	178	58	X-Proxy-ID: 2004304525
	X-IMForwards	30	20	X-IMForwards: 20
	Max-Forwards	5	4	Max-Forwards: 10
	Client-IP	5	7	Client-IP: 10.224.164.34
	Client-ip	3	2	Client-ip: 192.168.23.5
	X-BlueCoat-Via	49	9	X-BlueCoat-Via: fb09b83d12ade53b
	CUDA_CLIIP	19	11	CUDA_CLIIP: 172.16.20.138
	X-IWS-Via	7	6	X-IWS-Via: 1.1 51066FAS (IWSS)
	X-IWSaaS-Via	1	1	X-IWSaaS-Via: 1.1 scannerdy-an-20-3012-a-pro-18293387:8080 (IWSaaS)
	X-RBT-Optimized-By	2	2	X-RBT-Optimized-By: LGEPS-PC-ACC-3070M-A (RiOS 8.6.2c) SC
	RVBD-CSH	1	1	RVBD-CSH: ::ffff:172.25.80.199
	RVBD-SSH	1	1	RVBD-SSH: ::ffff:172.17.12.199
	Surrogate-Capability	8	5	Surrogate-Capability:srv015.guape.zigdigital.com.br
	X-Tinyproxy	1	1	X-Tinyproxy: 10.192.9.79
	X-If-Via	1	1	X-If-Via: 1.1 i-FILTER84982
Cache-Related	Cache-Control	750	106	Cache-Control: max-stale=0
	Pragma	4	3	Pragma: no-cache
	X-Loop-Control	35	2	X-Loop-Control: 151.233.132.133
	If-Modified-Since	24	13	If-Modified-Since: Thu, 24 Mar 2016 15:07:56 GMT
	If-None-Match	21	11	If-None-Match: ”90-52ecccfbb0285”

In Table 4.3, we list all instances of injected headers corresponding to proxies and caches. For each header instance, we also provide the number of ASes and countries of the possible location of the injection. As previously mentioned in our methodology, we infer the AS and country of the source IP address of the received requests. This IP address will either be the one from

the *Luminati* country peer or from a middlebox located between the country peer and our server. Therefore, even though it is not the definitive location of the middlebox, it will be typically at the edge of the Internet given the vantage points provided by *Luminati*. From how often these headers are observed in different networks, we get a measure of the popularity of these two important network functions overall. Meanwhile, we check the values of the injected headers (see examples provided in the last column of Table 4.3). We find that the values of the headers are consistent with the names of the headers, reflecting the related network functions played by the corresponding middlebox.

The most frequently injected request header in our dataset is **Cache-Control**. This header sets specific directives for cached copies, and is seen in about 7.5% of all ASes we sample in our measurements. The next most popular injected header is **Via**, injected by proxies to inform end points of its presence, sometimes also adding information about the name and version of the middlebox. We observed the **Via** header across 695 ASes in 117 countries. Middleboxes do more than tell their functions. They also add private information about the end-point originating the HTTP request, as from the **X-Forwarded-For** header that carries the IP address of the original client. Doing this is surprising, if the intended usage of proxy is to provide anonymity for end users, since adding the IP address of the original client defeats the very purpose of proxying, by revealing to the server the originator of the query. The next most popular injected header is **X-Proxy-ID**, seen in 178 ASes across 58 countries, which carries the identifiers of the proxies.

Injected HTTP headers also reveal a significant number of vendor-specific middleboxes. For example, **X-IWS-Via** and **X-IWSaaS-Via** are headers added by Trend Micro middleboxes, running the InterScan Web Security service. InterScan Web Security (IWS) is a software appliance that dynamically protects traffic flows on Internet gateway [72]. Another expected header is **X-IWSaaS-Via**, from the Amazon cloud instance inside a Japanese data center. Beside the typical functions of proxying and caching, we find headers related to services such as private IP mapping (**X-Tinyproxy**), traffic flows filtering (**X-If-Via**) and WAN optimization (**RVBD-CSH** and **RVBD-SSH**). Although not very common, these instances provide evidence of the diversity of roles played by middleboxes in today's Internet, way beyond the usual functions such as caching and proxying.

As shown in Figure 4.3, most of involved ASes are Tier-2 or customer networks, supporting our

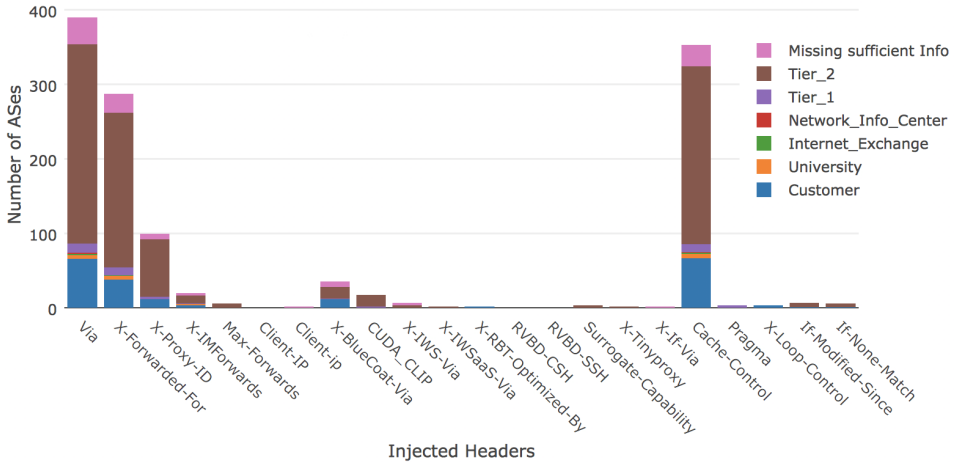

Figure 4.3: AS Classification of Injected Request Header

expectation that the middleboxes are generally located at the edge of networks.

4.1.2.2 *Non-standard Request Header Injection*

So far, we have studied injected headers for which the function of the middlebox is straightforward, because the header is well known or the name strongly suggests its function. This is not always the case unfortunately. When the header does not tell us its purpose, we try to guess its function based on its name or its network.

Mobile devices/networks request header injection In Table 4.4, we list a set of headers for which we could guess the purpose. We also provide the name of the networks from which the requests come, and the countries of these networks.

We observe that some headers are used to expose information about cellular networks and the mobile devices that use these networks. For example, **x-gateway** and **o2gw-id** are specially used in the O2 (GB) mobile network, to carry the ID and location of gateways [29]. **X-Roaming** and **X-RAT-TYPE** are both injected to describe the radio technology used in a particular mobile network.

From either the name of the header, or the network where the middlebox is located, it is obvious that the headers shown in Table 4.4 are used for mobile devices and mobile networks. Unexpectedly, looking at the content of the headers reveals information about the mobile users, which is visible to the server-side. Unfortunately, despite the huge number of vantage points provided

Table 4.4: Request Headers Exposing Mobile Network and Device Information.

Injected Header	# of ASes	# of countries
x-gateway	1 (O2-ONLINE)	1 (GB)
o2gw-id	1	1
X-Nokia-Gateway-Id	1 (MIRS)	1 (IL)
X-Nokia-MaxDownlikBitrate	1	1
X-Nokia-MaxUplinkBitrate	1	1
X-Nokia-BEARER	1	1
X-Roaming	1 (MTNSS)	1 (SS)
X-RAT-TYPE	1	1
MSISDN	2 (ZAIN, Jordan Tele)	2 (IQ,JO)
msisdn	2 (AXIATA,globacom)	2 (BD,NG)
mpesaMsisdn	1 (VODAFONE)	1 (RO)

by *Luminati*, we only observed this in 8 ASes, 4 of them could be classified. Two of them are customer networks, and the other two are Tier-2 networks. As much as *Luminati* provides vantage points at the edge of the Internet, few of these are actually located inside mobile networks, at least as seen from modified HTTP headers. Our results suggest that relying on vantage points located inside mobile networks is necessary to better sample middleboxes deployed across the Internet.

Remaining injected headers The remaining injected headers are either non-standard, or have names or values that make it challenging, though sometimes possible, to guess the purpose of the HTTP header or the function of the corresponding middlebox. We list these on Table 4.5. We provide the name, country of the network (when found) and example values.

Some injected headers nicely exemplify specific types of middlebox interference, such as URL filtering, real-time content filtering, or the specific networks where the middleboxes are deployed, e.g., mobile networks or cloud providers. For example, the **x-up-vfza-id** header is added by the VODACOM network, revealing the ID of the upstream gateway inside the mobile network. The headers with the prefix **X-TMCE** are added by Trend Micro middleboxes, and from looking at the AS number of the *Luminati* proxy, these headers are injected for the IWSaaS service inside the Amazon cloud platform. Similarly, the headers containing **FFI** and **HCF** are likely to be added by specific vendor devices. The **X-FCCKV2** header is added by middleboxes running the Fortinet software for traffic filtering [7]. Finally, despite its rather generic name, the **Server-Slot** header is added by a middlebox inside the French cloud computing company OVH. OVH offers

VPS, dedicated server and other web services, suggesting that the original request went through some web host running inside a cloud instance before reaching our target server.

Table 4.5: Remaining Injected Request Headers.

Injected Header	AS Num/ISP	Country	Note
x-up-vfza-id	1 (VODACOM-AS)	1 (ZA)	x-up-vfza-id: 65501
x-subscriber-info			x-subscriber-info: 10.139.195.196
cli	1 (QA-ISP)	1 (QA)	cli: 97433872509
imsi			imsi: 427012926009698
X-TMCE-GUID			X-TMCE-GUID: 48c1b6b0-4a2f-11e6-9c7d.....
X-TMCE-Token	1 (Amazon.com, Inc)	1 (JP)	X-TMCE-Token:48c1b6b0-4a2f-11e6-9c7d.....
X-TMCE-User			X-TMCE-User: %40ce-ac7caab8-74df-4bc4-ae2d.....
FFIClient			FFIClient: True
FFI-Authenticate	1 (NL-SOLCON)	1 (NL)	FFI-Authenticate: e78d964b-99db-4c70-88c5.....
FFI-AuthenticateUser			FFI-AuthenticateUser: enno
FFI-UrlToFilter			FFI-UrlToFilter: http://shanluminati.com/?TYPE1_nl_21408
HCFVer	1 (TTNET)	1 (TR)	HCFVer: 3.7.18
HCFType			HCFType: server
X-FCCKV2	2 (ENERGOTEL,TTSLMEIS)	2 (SK,IN)	X-FCCKV2: GAJ3kZcRPNFiiMihhS2K.....
X-Bloxx-Result	1 (DATAWEB B.V)	1 (NL)	X-Bloxx-Result: [201, 203, 250, 251, 254, 255, 260, 261, 266, 267, 401, 425, 3009]
Server-Slot	1 (OVH)	1 (FR)	Server-Slot:ovh01FR.openvpn.wifiprotector...
Referer	2 (NHN-AS,CHINA-UNICOM)	2 (KR,CN)	Referer: http://www.baidu.com/s?wd=www
X-delete-header	1 (CHINANET-BACKBONE)	1 (CN)	X-delete-header: gzip
Accept-Xncoding	1 (Bezeqint Internet Backbone)	1 (IL)	Accept-Xncoding: gzip
NCLIENT50	2 (VIA-NUMERICA,Hanyang University)	2 (FR,KR)	NCLIENT50: NCLIENT50
serialnumber	1 (INFOCLIP-AS)	1 (FR)	serialnumber: V2401625

4.1.2.3 Summary

Overall, our results from HTTP request header manipulation demonstrate the variety of different middleboxes deployed inside today's networks, and their many purposes and behaviours. From injected HTTP headers inside requests, we found that proxies and caches are the prevalent type of middlebox, and these are deployed world-wide, more than 100 countries. Despite relying on the *Luminati* probing infrastructure that does not particularly sample well mobile networks or cloud providers, we still found evidence of middleboxes in these networks. Finally, we observed multiple instances of vendor-specific middleboxes, which define their own HTTP headers, or of service-specific behaviours outside proxying and caching.

4.1.3 Response Header Manipulation

n this section, we explore HTTP response header manipulation, as well as web page blocking specifically. Before going into details of the response header manipulation, let us mention that most of the middlebox interference observed on the downstream part (by the client receiving the response) overlaps with the middlebox functions observed in the upstream direction. As previously mentioned, this makes sense as middleboxes typically are stateful devices, that see both directions of the traffic. However, this does not imply that HTTP header manipulation will take place in both directions for a given flow, or that the same headers will be affected. Therefore, one cannot expect that HTTP header manipulation in the two directions of the traffic will be similar, but at best consistent.

4.1.3.1 Response Header Injection

Proxies/Caches response header injection Expectedly, similar to the case of HTTP request headers, most instances of injected response headers relate to proxying and caching, such as the cache hit record, the age for the cached copies and proxy connection status.

Table 4.6: Response Header Injection.

	Injected Header	**# of ASes**	**# of countries**	**Note**
Cache-Related	X-Cache	519	105	X-Cache: MISS from localhost
	X-Cache-Lookup	401	99	X-Cache-Lookup: MISS from localhost:3128
	Age	216	53	Age: 0
	Cache-Control	206	76	Cache-Control: max-age=0,must-revalidate,no-cache,no-store
	X-CFLO-Cache-Result	48	5	X-CFLO-Cache-Result: TCP_MISS
	X-Loop-Control	22	2	X-Loop-Control: 5.202.228.198 179F97.....
	X-Cache-Full	11	1	X-Cache-Full: MISS from myauth.pirai.rj.gov.br
	Vary	7	6	Vary: *
	X-Cache-Debug	1	1	X-Cache-Debug: TCP_MISS/NODNS-IIP/-
	SPINE-CACHE	1	1	SPINE-CACHE: MISS
	ANIS-CACHE	1	1	ANIS-CACHE: MISS
Proxy-Related	Proxy-Connection:	128	52	Proxy-Connection: Keep-Alive
	X-Cnection	23	7	X-Cnection: close
	X-OSSProxy	19	16	X-OSSProxy: OSSProxy 1.3.337.376 (Build 337.376 Win32 en-us)(Apr 22 2016 15:45:25)
	X-Squid-Error	9	6	X-Squid-Error: ERR-READ-ERROR 104
Third Party Server	Set-Cookie	2	2	Set-Cookie: xodbpb=; Path=/; HttpOnly

As shown in Table 4.6, **X-Cache** is the most frequently added response header, observed from

519 ASes across 105 countries[1]. The next most popular, **X-Cache-Lookup** is observed in 401 ASes, nearly 4% of all ASes we observe. Both of them are used to handle cache implementation details.

Surprisingly, we find the header **Set-Cookie** injected in some of our responses, while the server should be adding it, not a middlebox. Although we could not identify the host that actually sets these cookies, the injection implies the existence of a third-party server (or a middlebox) responsible for such an injection. Though we do not see the third-party actually tracking the browsing behaviour of the client, the existence of such a third-party constitutes a privacy risk for end-users who are unlikely to be aware of its presence.

Compared to the injected request headers, we see less information about the unique user or gateway is injected in the response headers by the middleboxes. We observe 12 injected request headers that carry the information about the original user (private IP address) and the name or identification of proxies. Only two injected response headers record the cache hit results, carrying information about caches on the path. Although upstream and downstream traffic flows are likely to cross the same middleboxes, the middlebox interference we observe in both directions of the traffic is different. More private information about subnets or clients is added to requests compared to responses.

Remaining response header injection Similar to the request header situation, Table 4.7 shows the non-standard injected response headers. Again, in such cases we need to guess the purpose of the header. From our inference, it appears that most of these injected headers carry information related to content filtering and identification of middleboxes in different networks. However, we did not find any specific network function that would generally apply in these cases. For instance, **X-IS-ELAPSED** and **X-IS-FILTER** are injected in the same request, but from the values of these two headers we could not infer their function. From their name, we guess they are likely to be injected for filtering. Headers such as those with the **X-Nokia** prefix, or **X-Android**, are injected by the Android operating system, and therefore related to middleboxes located in wireless or mobile networks. The **Client-Date**, **Client-Peer** and **Client-Response-Num** headers are injected by SmarTone, the mobile network operator in Hong Kong. This shows

[1]Different from the case of requests, for responses we rely on the IP address of the country peer to infer the AS number and country of this header modification.

that consistently with the upstream case, we see evidence of middleboxes in mobile networks from the downstream direction of the traffic.

Table 4.7: Injected Response Headers Requiring Inference.

Injected Header	**# of ASes**	**# of countries**
X-IS-ELAPSED	2	1
X-IS-FILTER	2	1
X-Android-Selected-Protocol	1	1
X-Android-Response-Source	1	1
Client-Date	1	1
Client-Peer	1	1
Client-Response-Num	1	1
X-Bst-Request-Id	3	4
X-Bst-Info	3	4
X-WS-PAC	3	4
Warning	14	11
Mime-Version	11	8
Location	10	9
Content-Language	6	5
X-Vitruvian	6	5
X-TurboPage	4	5
Refresh	2	1

4.1.3.2 Response Header Modification and Removal

For response headers, we also observe header removals (Table 4.9) and value modifications (Table 4.8). Though we do not have explicit evidence about the type of middlebox in these cases, a large portion of the ASes for these headers overlap with those involved in the **Via**, **Cache-Control** and **X-Forwarded-For** headers in the requests. For example, as shown in Table 4.8, 77% of ASes for which **Accept-Range** modifications occur overlap with the ASes involved in request header injection. This suggests that these modifications and removals are actually done by the same middleboxes in both directions.

Overlapping ASes also give us the opportunity to look at cases where the ASes from the requests and responses differ. Indeed, when the IP address seen in the request received by the server differs from the IP address seen in the response (identifying the country peer), this means that the former IP address belongs to a TCP-terminating middlebox. We therefore count such IP addresses (1025), ASes (168), and countries (55) where these are located. Unfortunately, these statistics provide us only with a very poor lower bound on the number of middleboxes and networks observed, compared to the evidence from the HTTP header manipulation. Indeed, from

the sheer HTTP manipulation we observed, we found evidence of middleboxes in 1011 ASes from the requests, and in 1023 ASes for responses.

Some response header modifications and removals may affect the end-to-end performance. For example, some proxies modify or remove the value of the **Accept-Ranges** header, to disable byte serving. As byte serving allows the server to partially deliver the content, modifying the value of **Accept-Ranges** can very well affect the content transfer. Also, the removal of the **Last-Modified** and **Etag** headers may affect the updating of cached copies.

Table 4.8: Modified Response Headers (with AS overlap).

Modified Header	**# of ASes**	**# of overlap ASes**	**# of countries**
Content-Length	191	108 (57%)	75
Accept-Ranges	61	47 (77%)	38
Content-Type	37	20 (54%)	24
Server	26	15 (58%)	17

Table 4.9: Removed Response Headers (with AS overlap).

Removed Header	**# of ASes**	**# of overlap ASes**	**# of countries**
Last-Modified	143	84 (59%)	65
Accept-Ranges	107	64 (71%)	50
Content-Length	85	52 (51%)	36
Etag	73	42 (58%)	39
Server	33	21 (64%)	20

4.1.3.3 Summary

All in all, the manipulations of HTTP responses confirm the diversity of network functions played by today's middleboxes. Similar to the case of request headers, most of the injected response headers are added by proxies and caches. As shown in Figure 4.4, the classification of ASes which inject new headers inside responses is quite similar to those that do so on the requests. Although the types of injected headers are different between requests and responses, the consistency in the trends in both directions of the traffic possibly indicate that the same middleboxes indeed affect both directions of the traffic. From the downstream part of the traffic, we observed header manipulations that may potentially negatively impact end-to-end performance. Finally, we also observed instances of page blocking, exposing the URL filtering function of middleboxes.

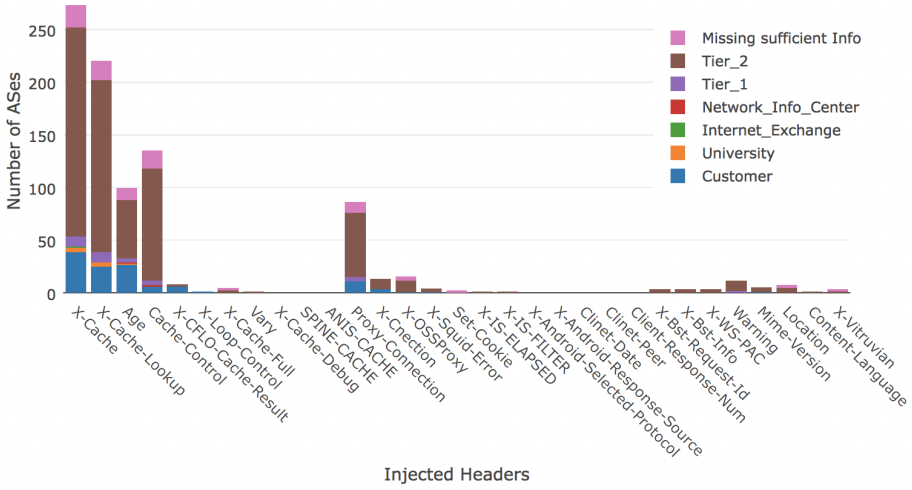

Figure 4.4: AS Classification of Injected Response Header

4.1.4 Conclusion

In our results, we studied the presence of multiple types of HTTP-interfering middleboxes in today's networks. We observed a large number and variety of injected headers, demonstrating the prevalence of HTTP-interfering middleboxes across the Internet. The main middlebox functions identified from these interactions are proxying, caching, filtering and NAT. Despite the limited number of instances, we also observed HTTP header manipulation done by mobile networks and cloud providers, providing evidence for widespread deployment of middleboxes in the Internet.

4.2 Middlebox Location

To investigate the location of involved middleboxes, we launch traceroute-like probes to pinpoint the approximate hop range where the interference takes place. From prior section, we already got the IP lists of all the available country peers. To maximize the usage of our data and explore the ASes deeper, we extended our work to send crafted HTTP responses with increasing TTL values from our servers to all available country peers.

4.2.1 Information about Last Returned ICMP Reply

Analysing the received ICMP time-exceeded replies, we compared the ASes of available country peers and last hops which returned ICMP reply back to our servers. Also, to investigate more about the last ICMP replies and involved ASes, we figured out the types of ASes with same

classifier [71], which are used for our *Luminati* dataset.

Table 4.10: Comparing ASes between Country Peers and Last Hops

	Total number	**Last hop in same AS**	**Last hop in different AS**
# of IPs	349,617	229,955 (65.77%)	119,662
# of ASes	9450	5749 (60.83%)	5734 (2033 are overlapped)

Table 4.11: AS Classifications of Last Hops

AS Classification	**# of ASes**
Customer	612
University	147
Internet Exchange	5
Network Information Center	40
Tier 1	40
Tier 2	1429
Missing Sufficient Information	159
Unclassified	3952

In total, we sent traceroute-like probes to 372,603 IP addresses which are distributed in 9634 ASes. While, since some IP addresses of last hops could not map to ASes (the mapped result is null), we got the IP addresses of last hops for 349,617 probes successfully, and nearly 66% of probes could receive ICMP replies from same ASes of target country peers (shown in Table 4.10). Furthermore, we classified the corresponding ASes of last hops which returned last ICMP time-exceeded replies. As shown in Table 4.11, most of last hops are distributed at the edge of networks.

4.2.2 Observation of Response Header Injection

Although it is hard to use *Luminati* to locate middleboxes, we explored the injected response headers again by checking the ASes of last hops to explore ASes between servers and country peers. From Section 4.1, we already analysed the IP addresses and corresponding ASes of country peers which were injected new response headers. We counted each country peer as one sample, compared the ASes of country peers and last hops, inferring the approximate place that where the header injection took place.

As shown in Figure 4.5, most of last hops of samples are in the same ASes of country peers. The manipulation may take place in same ASes of country peers, and the location of middleboxes may be close to client side. In contract, some cache related headers may be injected from different

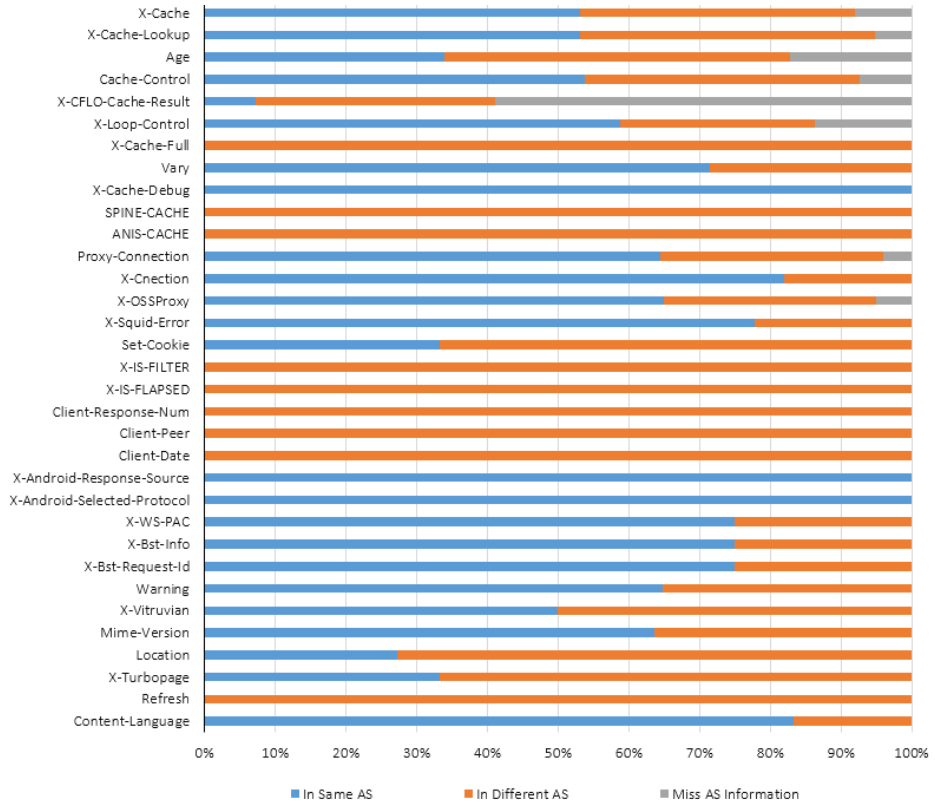

Figure 4.5: ASes Comparison between Last Hops and Involved Country Peers

ASes. For example, SPINE-CACHE and ANIS-CACHE are injected by vendor-specific caches, and the last received ICMP replies are all from different ASes. Although the number of samples is only 4, we inferred that these two types of vendor-specific caches may be TCP-terminating middleboxes, located in the middle between clients and servers.

4.2.3 Summary

In conclusion, we explored the IP addresses and corresponding ASes of last hops which returns ICMP time-exceeded replies between country peers and servers. In our results, nearly 66% of samples receive ICMP replies from hops which are close to country peers. For the rest of samples, we guess that there may be middleboxes on the path which open new TCP connections with servers. However, our current work could not locate the actual position of middleboxes, and the results are more or less affected by returned ICMP time-exceeded replies . In order to probe and explore more deeply on the paths, more methodologies are needed for locating middleboxes and investigating ASes in the middle.

Chapter 5

Discussion

In brief, we describe our methodologies and provide an abundant insight on HTTP-interacting middleboxes. According to our results, we believe that our methodologies will be useful for understanding the middlebox interference on traffic flows, inferring and identifying the involved networks. However, when we tried to measure the middleboxes on large scale, we noticed some limitations on our methodologies, samples and dataset. In this chapter, we explain these limitations and discuss how to extend our work in the future work.

5.1 Limitations on Methodology

5.1.1 Discussion about Client-Server Architecture

As mentioned in Section 3.1, our methodologies control both end hosts to adopt a client-server architecture, generating and capturing traffic flows in both directions. As the paths between two end hosts are complicated, the locations of clients and servers affect the behaviour of middleboxes. Futhermore, some middleboxes are reported to be stateful devices, the directions of traffic flows also affect the middlebox interference. However, our current work did not cover the selection of clients and servers. If we could control more vantage instances in the future, we could investigate more behaviour of middleboxes.

In order to run the application traffic flows as we designed, we need the root permission to access vantage points. However, it is a challenge for sampling across the Internet. Although we could make use of the virtual machines in some commercial cloud infrastructures, such as Amazon Web Service (AWS) , Google Cloud Platform (GCP) and so on, the number and distribution of instances are not enough for our large-scale measurement. For the purpose of extending our work, we need design other methodologies to run scripts, or collect more volunteers in varied networks to detect different types of middlebox interference.

Presently, we treat all middleboxes as one blackbox in our client-server model. While, it is possible that multiple middleboxes are located between two end hosts. In addition, one middlebox could operate from network layer to application layer. It is hard to figure out the accurate number or individual behaviour of each middlebox by using our current methodologies.

5.1.2 Limitations on Application Probe

Implemented with *Luminati*, we reproduced HTTP traffic flows and detected HTTP-interacting middleboxes in our results. We believe that our methodologies will be suitable to measure the further performance of applications in next step. However, the reproduced traffic flows are limited. Considering the ethical issues when we reproduce traffic flows from users, the probes are limited to be standard application traffic messages without any triggers of filtering, blocking or other behaviours of Internet censorship.

One the other hand, Internet traffic is changing (e.g., HTTPS accounts for a significant percentage of application traffic flows now), and the corresponding policies of middleboxes are evolving. HTTP traffic flows are not enough to investigate the behaviour of middleboxes. We need extend our work to HTTPS, understand how middleboxes interfere with the encrypted traffic flows and investigate the corresponding effect on the performance of HTTPS.

5.1.3 Limitations on Traceroute-Like Locating

Making good use of ICMP messages is a key method to locate middleboxes hop by hop. However, the traceroute-like locating methods are limited with the returned ICMP time-exceeded replies. As described in Section 3.4.2, some routers do not return the ICMP time-exceeded

replies, and some middleboxes modify the returned replies again in the downstream direction. We could not illustrate the accurate behaviour of middleboxes at each hop, and the result of location is rounded to approximate hop range where the interference takes place.

In addition, the types of returned ICMP replies also affect our detecting and locating the manipulation of traffic flows. According to RFC792 [41], the returned ICMP time-exceeded reply should quote the IP header and the next eight bytes of original packet. Meanwhile, RFC1812 [42] suggested to quote the entire IP packet in ICMP time-exceeded reply, but this recommendation has not been widely implemented. The more contents of original probes quoted in the ICMP replies, the more manipulation we could detect.

5.2 Limitations on Luminati

Thanks to *Luminati*, we could be able to launch HTTP requests and receive responses from more than 350,000 available country peers distributed in nearly 10,000 ASes. According to our dataset and results, the samples of Luminati are from different types of networks, which provide enough evidences to show the presence of middleboxes. However, since the super proxies and all country peers are based on Hola network, the measurement crawler does not have root access to available country peers. As discussed in prior section, the locations of clients and servers affect our detection. With *Luminati*, we could not select or change the location of clients. Since crawler only receive HTTP responses from super proxy, it is not possible for us to confirm how the traffic flows are filtered or dropped between super proxy and country peer. Also, the probes with increasing TTL values could not be reproduced from the country peers, which means that we could not locate the middleboxes from upstream direction.

Furthermore, *Luminati* controls the selection of country peers. Although *Luminati* provides different types of users (e.g. mobile device user, desktop user), we do not have the right to make choices. Also, the number of available country peers is varied with time. If we could launch our requests in a longer term, our results could cover more ASes and present more types of middleboxes inside networks. Overall, in order to make use of millions of country peers from *Luminati*, we need to better estimate our samples, value the sheer number of middleboxes deployed across the Internet in the future.

5.3 Limitations on AS Classification

In our results, we investigated the involved AS by classifying the types of ASes. We experimented with three classifiers. CAIDA's AS ranking [73] is an up-to-date classifier which provides great coverage of ASes. However, the categories of ASes are quite coarse, and there are only three categories: Transit, Content and Enterprise. Compared with CAIDA AS ranking, the classifier provided by Dhamdhere [74] and the one provided by Dimitropoulos [71] have lower recall, but the classification is a slightly finer-grained. In our work, we used the classifier provided by Dimitropoulos, which offered similar recall with [74], but with a finer grained classification. Whilst, there are nearly 67% of ASes are unclassified in our result. Therefore, we argue that better classification must be developed to investigate the involved ASes, and give better understand of behaviour of middleboxes in varied networks.

5.4 Summary

In summary, we discuss the limitations of our work from different aspects of methodology, sample collection and data classification. Middleboxes affect the traffic flows on both directions of traffic flows, and the types of interference are not only packet manipulation, but also filtering and injection. We need adopt a client-server architecture to generate and capture traffic flows from both end hosts. However, these requirements make sampling harder. To do the large-scale measurement, we need design more methods to collect volunteers or samples across Internet to count and locate middleboxes in the future. Also, we should include the selection of servers in our future work, detecting other behaviours by changing locations of clients and servers or directions of traffic flows.

Luminati is an important platform for us to collect samples. Although we could not control the selection of these country peers and do not have the access to these vantage points, the significant scale of samples provide lots of valuable data to analyze. Furthermore, since *Luminati* provides HTTPS traffic flows at the same time and HTTPS plays an increasingly significant role in our network. We could extend our methods and make use of millions of country peers in *Luminati* to understand middlebox interference on HTTPS in the future.

Chapter 6

Conclusion

In this report, we firstly give an introduction to middleboxes from the definition in theory to the behaviour in varied networks; present our motivations to illustrate why it is important to study the middleboxes and measure the effect of interference on traffic flows. We describe our methodologies and explain the differences between other related work, emphasising the reasons that why we need our methods. Deploying our methodologies in a commercial P2P network *Luminati*, we find the presence of middleboxes in different networks, manipulating traffic flows for implementing more network functions, enhancing the visibility of network traffic flows and enable the enforcement of security policies. At the same time, detected middleboxes provide us changes to infer and investigate the involved ASes.

As a new kind of device in today's network, middleboxes break the end-to-end network model, split connections between two end hosts, interfere with the traffic flows, and provide functions for security and performance improvement. However, as middleboxes and networks are all complicated, we have the limited knowledge about what happens in the middle and how the interference affects the performance of applications. It is critical to detect and measure the behaviour of middlboxes and have a deeper understanding of types of networks inspected.

In our work, we introduce the methodology to detect middlebox interference based on a client-server architecture. The clients and servers are both controlled and programmable to reproduce

traffic flows with different application payloads. Furthermore, adopting a client-server architecture provides us abilities to generate and capture exchanged traffic flows in both directions. To better implement our methods and conduct a large-scale dataset, we introduce the commercial platform *Luminati* platform and explain how to develop our methodology with it. Thanks to *Luminati*, we find evidence for a significant amount of middlebox interference on the traffic flows in different networks. Our results demonstrate the presence of varied HTTP-interacting middleboxes in nearly 1,000 ASes across 196 countries We observe a wide variety of injected HTTP headers in HTTP requests, some known and some never reported before. Surprisingly, we even observe new headers that are only added by mobile networks and cloud platforms. Overall, we find that injected headers expose the presence of multiple types of middleboxes across diverse networks. Further, the interference on HTTP responses often reveals the corresponding functions of the middleboxes, such as proxying, caching, URL filtering, and WAN optimization. Analysing the types of involved ASes and probing with the increasing TTL valuses responses, we find that most of interference happends at the edge of network.

As discussed in Chapter 5, *Luminati* and our methodologies have some limitations. In the future, we need extend our methodologies with the selection of clients or servers, change the traffic directions between two end hosts, and try to investigate the stateful behaviour of middleboxes. At the same time, more methodologies are needed for sampling and collecting volunteers in varied networks, launching probes from multiple types of ASes, giving a better estimate of the number of middleboxes in the Internet, as well as their actual impact. Since *Luminati* has a large-scale country peers to forward HTTP/HTTPS messages, we plan to make good use of HTTPS messages, explore the middlebox interference on encrypted traffic. Last but not least, our current work focuses on the interference and behaviour of middleboxes, we should extend our measurement to examine the effect on application performance, such as detecting the impact on packet loss, delay and other performance factors.

References

[1] S. W. Brim and B. E. Carpenter, "Middleboxes: Taxonomy and Issues." RFC 3234, Mar. 2013.

[2] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *ACM SIGCOMM 2012 Conference, SIGCOMM '12, Helsinki, Finland - August 13 - 17, 2012*, pp. 13–24.

[3] "Guide to intrusion detection and prevention systems(idps)." `http://ecinetworks.com/wp-content/uploads/bsk-files-manager/86_SP800-94.pdf`.

[4] N. Freed, "Behavior of and Requirements for Internet Firewalls." RFC 2979, Nov. 2015.

[5] G. Detal, B. Hesmans, O. Bonaventure, Y. Vanaubel, and B. Donnet, "Revealing middlebox interference with tracebox," in *Proceedings of the 2013 Internet Measurement Conference, IMC 2013, Barcelona, Spain, October 23-25, 2013*, pp. 1–8.

[6] "Wan optimization." `https://en.wikipedia.org/wiki/WAN_optimization`.

[7] N. Weaver, C. Kreibich, M. Dam, and V. Paxson, "Here be web proxies," in *Passive and Active Measurement - 15th International Conference, PAM 2014, Los Angeles, CA, USA, March 10-11, 2014*, pp. 183–192.

[8] "Network caching technologies." `http://docwiki.cisco.com/wiki/Network_Caching_Technologies#Proxy_Servers`.

[9] "Middlebox." `https://en.wikipedia.org/wiki/Middlebox`.

[10] Z. Wang, Z. Qian, Q. Xu, Z. M. Mao, and M. Zhang, "An untold story of middleboxes in cellular networks," in *Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Toronto, ON, Canada, August 15-19, 2011*, pp. 374–385.

[11] G. Aceto and A. Pescapè, "Internet censorship detection: A survey," *Computer Networks*, vol. 83, pp. 381–421, 2015.

[12] M. Dischinger, A. Mislove, A. Haeberlen, and P. K. Gummadi, "Detecting bittorrent blocking," in *Proceedings of the 8th ACM SIGCOMM Internet Measurement Conference, IMC 2008, Vouliagmeni, Greece, October 20-22, 2008*, pp. 3–8.

[13] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend tcp?," in *Proceedings of the 11th ACM SIGCOMM Internet Measurement Conference, IMC '11, Berlin, Germany, November 2-, 2011*, pp. 181–194.

[14] Anonymous, "The collateral damage of internet censorship by dns injection," *SIGCOMM Comput. Commun. Rev.*, vol. 42, pp. 21–27, June 2012.

[15] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson, "Netalyzr: Illuminating the edge network," in *Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference, IMC 2010, Melbourne, Australia - November 1-3, 2010*, pp. 246–259.

[16] "Chinas great cannon." `https://citizenlab.org/2015/04/chinas-great-cannon/`.

[17] B. Warf, "Geographies of global internet censorship," *GeoJournal*, vol. 76, pp. 1–23, 2011.

[18] J. Dalek, B. Haselton, H. Noman, A. Senft, M. Crete-Nishihata, P. Gill, and R. J. Deibert, "A method for identifying and confirming the use of URL filtering products for censorship," in *Proceedings of the 2013 Internet Measurement Conference, IMC 2013, Barcelona, Spain, October 23-25, 2013*, pp. 23–30.

[19] P. Winter and S. Lindskog, "How china is blocking tor," *CoRR*, vol. abs/1204.0447, 2012.

[20] X. Xu, Z. M. Mao, and J. A. Halderman, "Internet censorship in china: Where does the filtering occur?," in *Passive and Active Measurement - 12th International Conference, PAM 2011, Atlanta, GA, USA, March 20-22, 2011*, pp. 133–142.

[21] "Conceptdoppler: A weather tracker for internet censorship," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pp. 352–365, 2007.

[22] X. Chu, "Complete GFW Rulebook for Wikipedia plus Comprehensivve List for Websites, IPs, IMDB and AppStore." `https://docs.google.com/file/d/0B8ztBERe_FUwLWxUX0laeWF3aE0/edit`, 2013. [Online; accessed 25-December-2013].

[23] R. Ensafi, D. Fifield, P. Winter, N. Feamster, N. Weaver, and V. Paxson, "Examining how the great firewall discovers hidden circumvention servers," in *Proceedings of the 2015 Internet Measurement Conference*, IMC '15, pp. 445–458, 2015.

[24] P. Gill, M. Crete-Nishihata, J. Dalek, S. Goldberg, A. Senft, and G. Wiseman, "Characterizing web censorship worldwide: Another look at the opennet initiative data," *TWEB*, vol. 9, no. 1, pp. 4:1–4:29, 2015.

[25] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. R. Rodríguez, and P. Steenkiste, "Multi-context TLS (mctls): Enabling secure in-network functionality in TLS," in *Proc. ACM SIGCOMM*, 2015.

[26] S. Akhshabi and C. Dovrolis, "The evolution of layered protocol stacks leads to an hourglass-shaped architecture," in *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, 2011.

[27] D. Clark, "The design philosophy of the darpa internet protocols," *SIGCOMM Comput. Commun. Rev.*, vol. 18, pp. 106–114, Aug. 1988.

[28] B. Carpenter, "Architectural Principles of the Internet," RFC 1958, Internet Engineering Task Force, June 1996.

[29] N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, and V. Paxson, "Header enrichment or isp enrichment?: Emerging privacy threats in mobile networks," in *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, HotMiddlebox '15, 2015.

[30] N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, N. Weaver, and V. Paxson, "Beyond the radio: Illuminating the higher layers of mobile networks," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '15, pp. 375–387, 2015.

[31] X. Xu, Y. Jiang, T. Flach, E. Katz-Bassett, D. R. Choffnes, and R. Govindan, "Investigating transparent web proxies in cellular networks," in *Passive and Active Measurement - 16th*

International Conference, PAM 2015, New York, NY, USA, March 19-20, 2015, pp. 262–276.

[32] "Open observatory of network interference." https://ooni.torproject.org/nettest/.

[33] S. Aryan, H. Aryan, and J. A. Halderman, "Internet censorship in iran: A first look," in *3rd USENIX Workshop on Free and Open Communications on the Internet, FOCI '13, Washington, D.C., USA, August 13, 2013*.

[34] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. M. Munafò, K. Papagiannaki, and P. Steenkiste, "The cost of the "s" in HTTPS," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, CoNEXT 2014, Sydney, Australia, December 2-5, 2014*, pp. 133–140.

[35] A. Medina, M. Allman, and S. Floyd, "Measuring interactions between transport protocols and middleboxes," in *Proceedings of the 4th ACM SIGCOMM Internet Measurement Conference, IMC 2004, Taormina, Sicily, Italy, October 25-27, 2004*, pp. 336–341.

[36] V. Thirion, K. Edeline, and B. Donnet, "Tracking middleboxes in the mobile world with traceboxandroid," in *Traffic Monitoring and Analysis - 7th International Workshop, TMA 2015, Barcelona, Spain, April 21-24, 2015*, pp. 79–91.

[37] J. Pahdye and S. Floyd, "On inferring tcp behavior," vol. 31, pp. 287–298, Aug. 2001.

[38] R. Craven, R. Beverly, and M. Allman, "A middlebox-cooperative TCP for a non end-to-end internet," in *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*, pp. 151–162.

[39] https://en.wikipedia.org/wiki/Applet.

[40] "Planetlab." https://www.planet-lab.org/.

[41] "Internet Control Message Protocol." RFC 792, Mar. 2013.

[42] F. Baker, "Requirements for IP Version 4 Routers." RFC 1812, Mar. 2013.

[43] Y. Xu, "Internet Censorship Around the World," tech. rep., Feb. 2016.

[44] https://en.wikipedia.org/wiki/Firewall_(computing).

[45] N. Weaver, R. Sommer, and V. Paxson, "Detecting forged TCP reset packets," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA, 8th February - 11th February 2009*.

[46] G. Tyson, S. Huang, F. Cuadrado, I. Castro, V. C. Perta, A. Sathiaseelan, and S. Uhlig, "Exploring HTTP header manipulation in-the-wild," in *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*.

[47] R. T. Fielding and J. F. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing." RFC 7230, June 2014.

[48] "The hypertext transfer protocol (http) - request." `http://researchhubs.com/post/computing/web-application/http-request.html`.

[49] "Introductin to http basic." `https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html`.

[50] G. Apostolopoulos, V. G. J. Peris, and D. Saha, "Transport layer security: How much does it really cost?," in *Proceedings IEEE INFOCOM '99, The Conference on Computer Communications, Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, The Future Is Now, New York, NY, USA, March 21-25, 1999*, pp. 717–725.

[51] "Transport layer security." `https://en.wikipedia.org/wiki/Transport_Layer_Security`.

[52] T. Dierks, "The Transport Layer Security (TLS) Protocol Version 1.2." RFC 5246, Oct. 2015.

[53] J. Jarmoc, "Ssl/tls interception proxies and transitive trust." `https://media.blackhat.com/bh-eu-12/Jarmoc/bh-eu-12-Jarmoc-SSL_TLS_Interception-WP.pdf`.

[54] "Ooni: Open observatory of network interference," in *Presented as part of the 2nd USENIX Workshop on Free and Open Communications on the Internet*, USENIX, 2012.

[55] R. Holz, T. Riedmaier, N. Kammenhuber, and G. Carle, "X.509 forensics: Detecting and localising the SSL/TLS men-in-the-middle," in *Computer Security - ESORICS 2012 - 17th*

European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012, pp. 217–234.

[56] R. Holz, L. Braun, N. Kammenhuber, and G. Carle, "The ssl landscape: A thorough analysis of the x.509 pki using active and passive measurements," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pp. 427–444, 2011.

[57] "An overview of the ssl or tls handshake." `http://www.ibm.com/support/knowledgecenter/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm`.

[58] A. Chaudhry, A. Madhavapeddy, C. Rotsos, R. Mortier, A. Aucinas, J. Crowcroft, S. P. Eide, S. Hand, A. W. Moore, and N. Vallina-Rodriguez, "Signposts: End-to-end networking in a world of middleboxes," in *SIGCOMM*, pp. 83–84, ACM, 2012.

[59] O. Farnan, A. Darer, and J. Wright, "Poisoning the well: Exploring the great firewall's poisoned dns responses," in *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, WPES '16, pp. 95–98, ACM, 2016.

[60] "Amazon ec2 - virtual server hosting." `https://aws.amazon.com/?nc2=h_lg`.

[61] "Introduction: Dns security threats and mitigations." `https://developers.google.com/speed/public-dns/docs/security`.

[62] P. Mockapetris, "Domain names — implementation and specification." Internet RFC 1035, November 1987.

[63] "Domain names - concepts and facilities." RFC 1034, Nov. 1987.

[64] G. Aceto, A. Botta, A. Pescapè, N. Feamster, M. F. Awan, T. Ahmad, and S. B. Qaisar, "Monitoring internet censorship with UBICA," in *Traffic Monitoring and Analysis - 7th International Workshop, TMA 2015, Barcelona, Spain, April 21-24, 2015. Proceedings*, pp. 143–157.

[65] "List of websites blocked in the united kingdom." `https://en.wikipedia.org/wikiList_of_websites_blocked_in_the_United_Kingdom`.

[66] "The pirate bay blockade in spain: Spanish court orders isps to block tpb." `http://www.ibtimes.com.au/pirate-bay-blockade-spain-spanish-court-orders-isps-block-tpb-1434455`.

[67] "Pirate bay and four more torrent sites get blocked in italy." http://www.zdnet.com/article/pirate-bay-and-four-more-torrent-sites-get-blocked-in-italy/.

[68] "Websites blocked in mainland china." https://en.wikipedia.org/wiki/Websites_blocked_in_mainland_China.

[69] Anonymous, "Towards a comprehensive picture of the Great Firewall's DNS censorship," in *Free and Open Communications on the Internet*, USENIX, 2014.

[70] "Business proxy network." https://luminati.io/.

[71] X. Dimitropoulos, D. Krioukov, G. Riley, and k. claffy, "Revealing the Autonomous System Taxonomy: The Machine Learning Approach," in *Passive and Active Network Measurement Workshop (PAM) Adelaide, 2006*.

[72] "Interscan web security." http://www.trendmicro.co.uk/products/interscan-web-security/index.html?gclid=CjwKEAiAjcDBBRCJxouz9fWHynwSJADaJg9BQ5DPaywxLrCeWP8Av7_dH56go6dep0fLbzOECkpPthoCt1Pw_wcB#why-its-better.

[73] "As rank: As ranking." http://as-rank.caida.org/.

[74] A. Dhamdhere and C. Dovrolis, "Twelve Years in the Evolution of the Internet Ecosystem," vol. 19, Sep 2011.